An Indexing System for Mobility-aware Information Management

Misael Mongiovì, Ambuj Singh, Xifeng Yan, Bo Zong
Department of Computer Science
University of California
Santa Barbara, CA 93106, US
{misael,ambuj,xyan,bzong}@cs.ucsb.edu

Konstantinos Psounis
Department of Electrical Engineering and Computer Science
University of Southern California
Los Angeles, CA 90089 US
kpsounis@usc.edu

ABSTRACT

We analyze the problem of minimizing the cost for satisfying information demand in a network of moving nodes, where the communication between nodes is subject to distance constraints and a costly communication link with an information source (central server or Internet). This problem has applications in several fields ranging from vehicular networks to space exploration, and it is NP-hard. We propose a novel indexing system that is able to reduce the search space and show that in practical cases, our system is able to find an optimal solution in a reasonable length of time. An extensive experimental analysis on large real and synthetic datasets shows that the proposed method responds in less than 10 seconds on datasets with millions of events and thousands of information requests, with an improvement of up to 100 times compared to the non-indexed solution.

1. INTRODUCTION

The analysis of trajectories of moving objects has interesting applications in many fields ranging from wildlife studies to traffic analysis [14]. An interesting application concerns the optimization of the information management in a scenario in which several moving objects are provided with mobile devices and can communicate among themselves subject to some distance constraints. The mobile devices form a communication network whose structure changes over time. This kind of networks have been studied by the networking community under the name of Intermittently Connected Mobile Networks (ICMN) [18], a special case of Delay Tolerant Networks (DTN) [1]. Past work on this field has been limited to the design of single-cast routing strategies that optimize the delay or the probability of delivery.

In a common scenario, moving nodes (e.g. people or vehicles) are provided with a device that supports two kinds of communication: a node-to-node communication (radio), usually non costly, and a centralized communication (cellular or satellite) that involves a cost. For example, consider a network of city buses, in which each bus is provided with a mobile device that receives updated information about traffic, weather and so on. Each device can obtain the updates by the cellular (e.g. UMTS) network (costly), but it is more convenient to share the information among various buses via node-to-node communication (non-costly). Another example considers soldiers or convoys that move following a specific strategy. They need to access certain information related to their location (e.g. satellite images). In this case, the only options available are satellite communication (highly costly) and node-to-node communication. Note that in both examples, the node routes as well as the information demands are known in advance.

The described scenarios have some important aspects that make it difficult to apply current approaches. First, the problem considers two communication networks in synergy, while current approaches are usually based on one single network. Second, it is based on multi-cast transmission, since the same information object needs to be delivered to many destinations. Third, it is mobility-aware, allowing one to take advantage of the available knowledge about the trajectories of moving nodes. Although some mobility-aware approaches exist [9, 8], they do not consider the case of multi-cast transmission. Last but not least, the amount of data to analyze may be huge, therefore an efficient solution is necessary.

In this paper we formulate the problem of optimizing the information demand in a network of moving nodes and call it *demand cover problem*. We prove that this problem is NP-hard and present a graph-based algorithm for it. In order to make this problem feasible on large datasets, we formulate it as a query processing problem and develop a novel indexing solution. Due to the index, we are able to solve each query in less than 10 seconds on datasets with millions of events and queries with thousands of information requests, with an improvement of up to 100 times compared to the non-indexed approach.

Our contribution can be summarized as follows:

- We define a novel problem (*demand cover*) that formalizes the problem of optimizing the information demand in a network of moving nodes. We prove that demand cover is NP-hard.
- We develop a compact graph-based representation of a *demand cover* instance. The graph formulation itself has interesting applications in other fields.
- We propose a novel indexing system for solving de-

VLDB '11, August 29- September 3, 2011, Seattle, WA

mand cover on graphs optimally. Our system uses a compact graph representation to efficiently locate a small portion of vertices that are relevant for achieving an optimal solution.

- We evaluate the proposed approach on two real and one synthetic datasets and show that an exact solution can be found in reasonable time in datasets with millions of events.

The paper is structured as follows: Sect. 2 describes related work. Sect. 3 defines the problem, provides a graph representation and a simple solution. Sect. 4 describes our indexing system for demand cover. Sect. 5 presents an extensive experimental analysis on real and synthetic datasets. Finally, Sect. 6 concludes the paper with some future directions.

2. RELATED WORK

2.1 Index structures for moving objects

Common indexing approaches for moving objects aim to perform spatio-temporal queries on databases of moving objects. They are usually based on TPR-Tree [22, 19], an adaptation of R^*-Tree that takes into account temporal features of moving objects. Other approaches are based on B^+-tree [4, 10]. Some approaches incorporate movement models in order to answer predictive queries [26, 20, 11]. Our indexing system differs substantially from previous approaches since our purpose is to manage information demand in a network of moving nodes. Although our approach assumes that the trajectories are known in advance, it can be integrated with existing predicting models to manage uncertainty.

2.2 Routing in Delay Tolerant Networks

Previous work on communication networks with mobile nodes has been focused on designing routing strategies that minimize the delay of message transmission or maximize the probability of delivery. Some approaches analyze the case in which the moving nodes have an uncertain behavior and hence heuristics need to be employed [18, 17], while other approaches focus on the case in which some knowledge about the future trajectories and the information demand is available [9, 8].

In [9], the authors consider the general case in which the loss of connectivity depends on various factors, including node mobility. They consider several cases, based on the level of knowledge available, from zero knowledge (no information about connectivity and information demand is available) to complete knowledge (connectivity, information demand and state of the network is known in advance). The full knowledge case is the closest to our work. In addition, the authors consider the bandwidth capacity and the storage capacity of each node. The traffic demand is given by a number of source-destination pairs representing messages. Since a fixed discretization interval introduces a problem of balance between quality and efficiency, the authors propose to sample the connectivity network based on a set of time intervals of variable length, that is proved to achieve an optimal result. They then use a Linear Programming (LP) method to find a solution and show that LP is able to find a solution only on very small scenarios (a few nodes and edges). In our work we analyze a multi-cast scenario with two kinds of communication and aim to minimize the communication cost. Moreover, we provide a solution that can be applied efficiently on large datasets.

2.3 Graph indexing

Graph indexing systems generally aim to solve problems as subgraph isomorphism [5, 25, 23], graph matching [15] or reachability [4, 12]. For subgraph isomorphism (and graph matching), a *preprocessing* off-line phase generates a data index, starting from a database of graphs or from a large network. When a query (generally a small graph) is requested, a *filtering* phase is performed, followed by the execution of a subgraph isomorphism algorithm on a reduced instance (*matching* phase). The filtering phase prunes unnecessary graphs or parts of them, with the purpose of reducing as much as possible the size of the input for the expensive matching phase. In some cases [5], the labels of the query and the target graph are substituted, so as to further reduce the matching time. The structure of the index is usually based on an inverted index of *features* (i.e. subgraphs, trees or paths) that is used to identify graphs of the database that are candidates to match the query. In this work, we use an analogous *preprocessing-filtering-optimization* scheme. The structure of the index, however, is completely different. Our index uses a compact graph representation based on disjoint path decomposition (Sect. 4) and a suitable set of labels to solve the demand cover problem.

Systems for reachability queries generally use compressed data structures to efficiently perform reachability tests. Some systems use chains [7] (generalization of paths) decomposition or path-tree [12] decomposition. The underlying idea is that if a vertex u of a chain (or a tree-of-paths) is reachable from another vertex v, all the vertices downstream in that chain are reachable from v. We use a similar idea to develop a compact index. However, our work differ from reachability indexes in many aspects. First, our system is designed to fast identify the regions of the graph that can reach a given destination instead of verifying the reachability between pairs. Second, we take into account the time constraints. Last but not least, we identify a small subset of vertices that is representative and allows to solve the demand cover problem optimally and with reasonable efficiency on large datasets (see Sect. 4).

3. PROBLEM DESCRIPTION

In the considered scenario, each moving node is provided with two types of wireless communication. Two nodes can communicate with each other through a non costly radio communication when they are at a distance lower than a certain threshold. Each node can also communicate with a central information source (Internet or a central server) through cellular or satellite communication. Depending on its location or other factors, each node needs to access certain data (*information objects*). Since typically the same information object needs to be accessed by several nodes, it is convenient to use the non costly radio communication to share objects among nodes when possible. However, the object sharing is subject to some time-constraints. In fact, since the information objects can be updated over time, each information request has a recency constraint that limits the delay in which an object can reach the destination.

We are interested in computing the minimum set of source-to-node (costly) transmissions that satisfies the information

demand. After the optimal set is found, the information objects could be delivered using standard techniques [13, 21], or a reasonable route can be detected efficiently by a post-processing phase. Since the node-to-node communication is not expensive, a suboptimal route would not penalize the quality of the solution.

Next we define the problem formally. We consider a single information object, since the case of multiple objects can be solved optimally by applying the solution one object at a time.

3.1 The demand cover problem

We consider a set of *trajectories* $T = \{T_1, T_2, \ldots, T_m\}$ associated with *moving nodes* $(n_1, n_2, \ldots, n_m$ respectively). A trajectory T_i is a function $T_i : [0, t_{MAX}] \rightarrow R^D$ that associates each time instant $t \in [0, t_{MAX}]$ with a point in the space (typically a plane) that the corresponding node occupies at time t. At a specific time, two nodes can communicate with each other if their Euclidean distance is within a fixed threshold d. When this happens, we say that the two nodes are within *radio range*. The communication between nodes does not involve any cost. Each node can also communicate at any time with the *information source* (e.g. a central server) with a certain cost.

The information is organized in a number of information objects, each of them representing a single indivisible part. Given an information object, we define its *information demand* I as a set of *information requests*, i.e. triples of the form (n_i, t, δ), where n_i, t and δ represent the node that needs the object, the time instant in which the information is required and the maximum delay allowed respectively.

In order to model the flow of information in the network we consider two kind of transmissions: *source-to-node object transmission*, represented by a pair (n_i, t) where n_i represents the node that receives the object and t is the time instant in which the transmission occurs, and *node-to-node object transmission*, represented by a triple (n_{i_1}, n_{i_2}, t) where n_{i_1}, n_{i_2}, t represent the node that transmits the object, the node that receives the object and the time instant at which the transmission occurs respectively. For simplicity, all the object transmissions are considered instantaneous. Although it may not be true in real cases, the movement between nodes is usually very slow compared to the speed of transmission. Therefore, in most cases all the necessary objects can always be transmitted before the two communicating nodes exit from the radio range. We say that a source-to-node object transmission (n_{i_s}, t_s) covers an information request (n_{i_d}, t_d, δ) if there exists a sequence of node-to-node object transmissions (n_{i_0}, n_{i_1}, t_1), (n_{i_1}, n_{i_2}, t_2), ..., $(n_{i_{k-1}}, n_{i_k}, t_k)$ with $n_{i_0} = n_{i_s}$ and $n_{i_k} = n_{i_d}$, such as $t_s \leq t_1 \leq t_2 \leq \ldots \leq t_k \leq t_d$ and $t_d - t_s \leq \delta$. The demand cover problem is defined as follows:

PROBLEM DEFINITION 1. *Given a set T of trajectories and an information object with demand I, detect the minimum set of source-to-node object transmissions that covers all the information requests in I.*

Figure 1 depicts an instance of the demand cover problem. At the starting time t_0, the nodes n_2 and n_3 are within the radio range and can communicate with each other, while n_1 and n_4 are far from other nodes. At time t_1, n_4 enters in the radio range of n_2. At time t_4, n_2 exits from the radio range of n_3. Three information requests are also represented with filled triangles. Note that to be able to communicate, two nodes must be close to each other in space and time, therefore in general the fact that two trajectories overlap in some point in the space does not imply that the corresponding nodes can communicate.

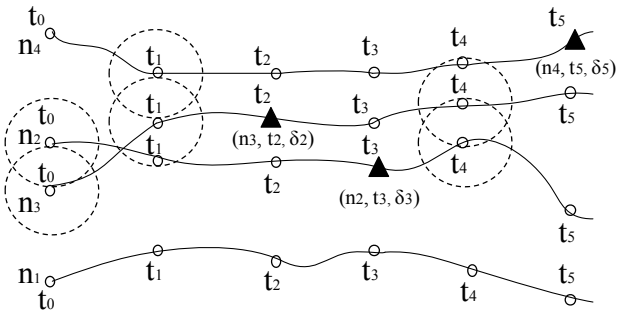


Figure 1: An instance of the demand cover problem. The four solid lines represents four trajectories. The big dashed circle represents the radio range of nodes. The information requests are represented by triangles.

The formulated problem can be shown to be NP-hard (even in the 2D plane) by reduction from the well known Set-cover problem. Given a family of sets $S = \{S_1, S_2, ..., S_m\}$ of elements taken from a set C, Set-cover calls for finding the minimum sub-family of S that covers all the elements of C. A proof is given in the Appendix.

THEOREM 1. *Any instance of Set-cover can be reduced in polynomial time to an instance of the demand cover problem.*

3.2 Graph representation

The demand cover problem can be formulated in ILP (Integer Linear Programming) and solved by a standard solver. However, this solution would be extremely inefficient, in particular when applied on large datasets. An alternative approach is to represent an instance of demand cover with a graph, whose nodes that represent points in the plane and with two kind of edges that represent motion between two points and connectivity (distance lower than d) between two nodes respectively. A path connecting two nodes s and d of the graph represents a possible route from s to d. There are several advantages in working with graphs. First, an instance of the problem is represented in a compact manner, by hiding details concerning the motion and maintaining only details that are relevant. Second, we can take advantage of a large amount of literature available on graph algorithms.

However, the above representation has an important drawback concerning the choice of the time discretization interval. Although we could employ the approach of Jain et. al. [9], it would generate a lot of redundancy, since between two consecutive samples most of the connectivity graph must be repeated. Instead we resort to a novel approach that allows us to remove unnecessary nodes and edges without compromising the quality of the result. We identify three types of events that have an effect on the information flow:

- *information request event* or *ir-event* which denotes the occurrence of an information request;

- *in radio range event* or *in-event* which denotes the moment at which two nodes enter in the radio range (at this moment they are close enough to communicate);
- *out radio range event* or *out-event* which denotes the moment at which two nodes exit from the radio range (after this moment the communication between them is not possible).

Ir-events and out-events are sufficient for solving the demand cover problem optimally. In fact, given an optimal solution for demand cover, it is always possible to modify this solution in such a way that each transmission between two nodes n_1 and n_2 is delayed until an ir-event or out-event that involves n_1 or n_2 occurs, without increasing the cost. Since communication is assumed to be instantaneous, the length of time in which two specific nodes are in the radio range is not important. In practice, the communication can be anticipated in order to guarantee that all the necessary objects are transmitted correctly. However, for simplicity of description we do not consider this practical detail in our algorithm. In-events will be considered in Section 4.

In-events and out-events can be extracted from the database of trajectories. Ir-events depend on the information demand. Figure 1 shows an in-event, involving the nodes n_4 and n_2 at time t_1, an out-event, involving n_4 and n_3 at time t_4, and three ir-events, represented by filled triangles, at time t_2, t_3 and t_5 respectively. Now we describe in detail our graph representation named Event-Driven Influence Graph (Edig).

3.2.1 Event-Driven Influence Graph (Edig)

The Edig representation is based on two main observations. First, in the *connectivity graph*, all the vertices in the same connected component have the same property of reachability, so one vertex is representative of all the others. Second, when an event occurs, connected components of the connectivity graph that do not contain any nodes involved in the event, are not influential.

A connectivity graph is a graph in which vertices represent moving nodes and two vertices are connected by an undirected edge if the corresponding nodes are in the radio range (distance lower than d). This graph evolves in time. For each event we consider a connectivity graph that represents the network connectivity in the time interval that spans from the previous event to the current event. We call this interval the *lifetime* of the connected component.

Given a set of events Ev, we first remove from Ev all the in-events and out-events that do not change the connected components of the connectivity graph. Then, for each event $e \in Ev$, the connected components of the connectivity graph that contain nodes involved in e are considered and collapsed into single vertices. For completeness, the connectivity graph at time t_{MAX} (the last time stamp) is considered and each of its connected components is collapsed into vertices. All the collapsed connected components are then connected by directed edges in the following way: given a component c_t corresponding to time t, for each moving node n in c_t, consider the component $c_{t'}$ corresponding to time $t' > t$, containing the node n and such as there is not any other component at time $t'' < t'$ that satisfies the same properties. Connect c_t to c'_t by an edge with weight $t' - t$.

One example of Edig is shown in Figure 2. It represents the Edig graph corresponding to the example in Figure 1. Small circles represents nodes. Undirected edges represents edges of the connectivity graphs. Filled circles represent nodes involved in in-events or out-events while filled triangles represents nodes involved in ir-events. The big ovals represent connected components.

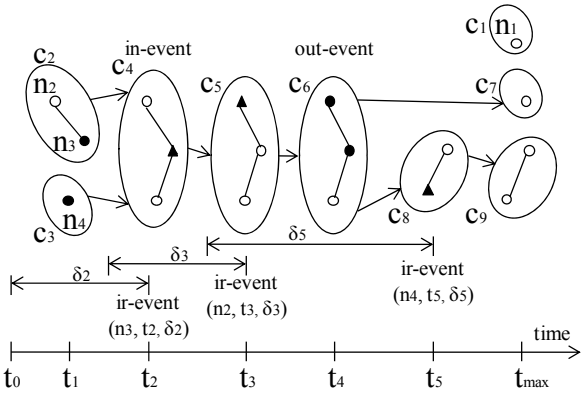


Figure 2: The Edig representation of the example in Fig. 1. All the in-events, out-events and ir-events are shown. Small circles or triangles represent nodes at specific time instants. Filled circles represent nodes that are involved in the current in-event or out-event, while triangles represent nodes that perform information requests. Big ovals represent connected components (vertices of the Edig graph).

3.2.2 Solving demand cover on Edig

The demand cover problem can be formulated as a graph problem. Given a directed weighted graph G_{edig} and a set of destinations (information requests) $I = \{(c_1, \delta_1), (c_2, \delta_2), \ldots, (c_k, \delta_k)\}$, where $c_1, c_2, \ldots, c_k$ are vertices of G_{edig} and $\delta_1, \delta_2, \ldots, \delta_k$ are distance constraints (time delay), find the minimum set of vertices V_S of G_{edig} so that each destination v_i can be reached by at least one vertex of V_S with a distance not larger than δ_i.

The demand cover problem on graphs is a general problem that has applications in other fields. For instance, suppose one wants to arrange a number of facilities that cover certain sensitive locations, where cover means that each sensitive location must not be too far from the closest facility. An interesting question is how many facilities are required to satisfy the constraints and where they must be located. Another possible application is on Named Data Networking [6]. In this model, data packets can be cached in some routers in order to optimize the demand. An important question here is which is the minimum set of caches that need to be stored to satisfy a certain demand with maximum delay constraints, and where to place them.

The demand cover problem on graphs can be reduced to Set-cover by building a family of sets for Set-cover in the following way: consider one set for each vertex of the graph that contains all the destinations that can be reached by it satisfying the distance constraints. This family can be built by executing a backward (following the edges in the opposite direction) BFS or DFS from each destination. The optimal subfamily for Set-cover corresponds to an optimal set of vertices for demand cover.

The following theorem states that the original demand

cover problem can be reduced to a demand cover problem on graphs.

THEOREM 2. *Let (T, I) be an instance of demand cover, where T is a set of trajectories and I is a set of information requests. Let G_{edig} be an Edig graph built on (T, I) by considering only out-events and ir-events. If $C = \{c_1, c_2, \ldots, c_q\}$ is the optimal solution of demand cover on G_{edig}, the set of vertices obtained by choosing arbitrarily one vertex from each connected component in C is an optimal solution for the original demand cover problem.*

4. AN INDEXING SYSTEM FOR THE DEMAND COVER PROBLEM

Solving the demand cover problem efficiently raises several challenges. First, for each vertex v of the Edig graph, and each information request vertex u, we need to check if u can be reached by v in acceptable time. This operation may be very expensive when the size of Edig is large. Second, Set-cover is NP-hard, therefore no polynomial-time solutions exist (unless P=NP) in the general case. For small instances, Set-cover can be solved optimally in acceptable time by pruning techniques as *branch-and-bound*. In our case, however, the number of sets generated by Edig is usually very high, since an information request can potentially be reached by many vertices. Many of these sets are redundant, i.e. they are fully contained in other sets. Removing redundant sets leads to a considerable reduction of the Set-cover instance. However, removing the redundancy by traditional methods is expensive, since it requires to find *maximal sets* [24]. Additionally, in a typical application, a large amount of information objects are requested and each information object has its own set of information requests. Solving the demand cover problem for each information object may be extremely expensive.

We propose a novel approach, *Compressed Edig based on time-Interval labeled Disjoint Paths* (CIDP, for short) that builds an index of the set of trajectories with the purpose of efficiently performing queries of the form: *given a set of information requests, return the the minimum set of source-to-node object transmissions that covers all the information requests.*

We use a preprocessing-filtering-optimization scheme to solve the demand cover problem. Given a database of trajectories, a *preprocessing* phase generates a compact data index. When a query (represented by a set of information requests) needs to be performed, we use the data index to generate a lightweight instance of Set-cover (*filtering* phase). Set-cover is then solved optimally (*optimization* phase) and the solution is returned.

The proposed indexing system gives several advantages. First, the index is much more compact than the Edig graph, and hence require less memory and is much more efficiently manageable. Second, it allows to fast identify the set of nodes in the Edig graph from which the information requests are reachable. Note that current reachability indexes cannot be efficiently applicable to our problem since many reachability tests need to be performed. Finally, we can efficiently prune nodes of the Edig graph that are not promising and generate a small instance for Set-cover. Next, we introduce the proposed index and describe the three phases of our indexing system: preprocessing, filtering and optimization.

4.1 Index structure

The key idea is that the set of requests covered by a node in an Edig path includes the set of requests covered by other subsequent nodes in the path. Therefore, a node can be taken as representative of a portion of the path. Moreover an Edig node can be univocally determined by a path and a time instant. Based on these considerations, we partition the Edig graph in a set of disjoint paths and build a compact graph, named CIDP, whose vertices represent disjoint paths and edges preserve the connectivity across paths. Each vertex of CIDP is labeled with a time interval (named *lifetime*) that is the union of the lifetimes of its composing vertices (note that a vertex of Edig represents a connected component of moving nodes and has a lifetime). Instead of exploring all the nodes of a path, we can determine a set of time instants that is representative of the whole path by exploring the compact CIDP graph. We describe this idea more in detail by highlighting the two key properties of CIDP nodes.

PROPERTY 1. *Let G_{edig} be an Edig graph and G_{cidp} its corresponding CIDP graph. A pair (p_i, t), where p_i is a node of G_{cidp} and t is a time instant, identify univocally a node of G_{edig}.*

This property imply that we can use the coverage of a pair (p_i, t) in place of the coverage of the corresponding Edig node. We denote the coverage of (p_i, t) as $C_{(p_i,t)}$. The following lemma states that the coverage of a time instant contains the coverage of following time instants until the minimum transmission time of some request is reached.

LEMMA 1. *Let G_{cidp} be a CIDP graph, I be a set of information requests, and (p_i, t_1), (p_i, t_2) be two pairs where p_i represents a node of G_{cidp} and $t_1 \leq t_2$ two time instants. If for all requests $(n, t, \delta) \in I$ it holds $(t - \delta) \notin [t_1, t_2]$, then $C_{(p_i,t_2)} \subseteq C_{(p_i,t_1)}$.*

The lemma follows by the fact that all the nodes that precede a node c in a directed path conserve the reachability properties of c. Therefore, all the requests $(n, t, \delta) \in I$ that are covered by (p_i, t_2) can also be covered by (p_i, t_1), provided that the time constraints are compatible with both t_1 and t_2 (controlled by the condition $t - \delta \notin [t_1, t_2]$). An interesting consequence of the lemma is that a time instant can be used as representative of an interval in a path.

Figure 3 shows an example. The set of requests covered by p_i at time t_a is $C_{(p_i,t_a)} = \{r_1, r_2\}$. Since no other requests $(n, t, \delta) \in I$ have $(t - \delta) \in [t_a, t_b]$, (p_i, t_a) is representative of the interval $[t_a, t_b]$. t_b coincides with the *minimum transmission time* of r_3 (i.e. $(t_3 - \delta_3)$). Therefore, its coverage ($C_{(p_i,t_b)} = \{r_3\}$) cannot be contained in $C_{(p_i,t_a)}$. The pair (p_i, t_b) is instead representative of the remaining part of the path. The path p_i produces only two sets ($C_{(p_i,t_a)}$ and $C_{(p_i,t_b)}$) for Set-cover, instead of four that would be produced without indexing.

Next we describe in details the three steps of our method: preprocessing, filtering and optimization.

4.2 Preprocessing

Given the set of trajectories, first an Edig graph (G_{edig}) based on in-events and out-events is generated. Ir-events are not considered since during the preprocessing phase the information demand is unknown. For each vertex of G_{edig}

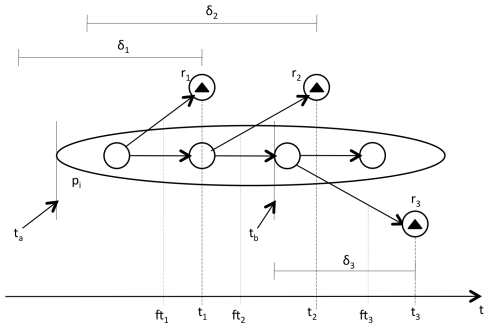


Figure 3: An example of path in the Edig graph. The directed graph composed by small circles represents the Edig graph. The big oval p_i represents a path. Solid triangles within circles represent information requests. The times ft_1, ft_2 and ft_3, associated with outgoing edges, represent the end-times of source vertices.

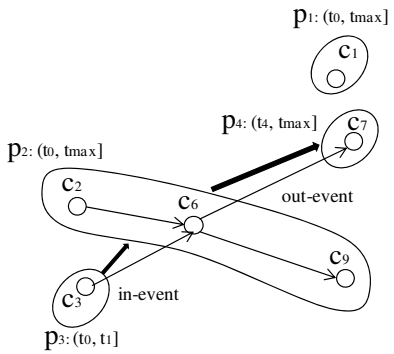


Figure 4: An example of CIDP built over the example of Figure 1. The small circles and thin arrows represents the Edig graph (compared to Figure 2 here the components corresponding to ir-events are removed). Each IDP is circumscribed by a solid line and links between IDPs are represented by thick arrows. For each IDP, its lifetime is reported.

we consider its lifetime, i.e. the interval $[s, e]$ in which the represented component is valid. We call s the start-time and e the end-time. The Edig graph is then decomposed into a disjoint set of paths. We associate each path to the time interval that represents its *lifetime* (the time associated with the two extremes) and refer to it as IDP (Interval-labeled Disjoined Path). We consider a graph of IDPs, G_{cidp}, where an edge is placed between two IDPs if there is an edge that connects two vertices across the two IDPs. Each edge (p_i, p_j) of G_{cidp} is associated with the set $FT(p_i, p_j)$ of end-times of all the source vertices in p_i (note that in general an edge in G_{cidp} can be associated to many edges in G_{edig}). $FT(p_i, p_j)$ represents the set of time instants in which an information object can flow through (p_i, p_j). Note that there is a large number of possible ways to partition the graph in disjoint paths. We use a simple strategy consisting on choosing one vertex at a time (proceeding in time order) and elongating it by random walk until a vertex without outgoing edges is reached. Figure 4 depicts an example of G_{cidp}. The small vertices and thin edges represent G_{edig}, while the big vertices and thick edges represent the compressed G_{cidp}.

4.3 Filtering

Our filtering algorithm finds a set of time instants TI_p for each IDP that are representative of the whole IDP, and the family $\mathcal{S}$ of corresponding sets. Our strategy guarantees that each vertex of the Edig graph that covers at least one information request, is fully contained in at least one set in $\mathcal{S}$. Since CIDP is much smaller than Edig, exploring the former is much more advantageous in terms of elaboration time and memory consumption.

The filtering procedure considers two steps: *backflow* and *prune*. *Backflow* propagates the requests in reverse order from the destination paths to all the possible source paths. For each path, we compute the *validity interval* of a request that define the time interval in which the information object must reach the path for the request to be covered. At the end, each path has a set of requests that it can cover with their validity intervals. The coverage of a pair (p, t) can be identified by the set of requests such as their validity interval in p include t. After the validity intervals are generated, *prune* computes the family of sets for Set-cover. It collects the set of coverages that correspond to minimum validity times (starting times of validity intervals) in all paths. Below we show that this procedure discards only redundant sets.

4.3.1 Backflow

We define the *validity interval* of a request $r = (n, t, \delta)$ in a path p ($valid\_int(r, p)$) recursively in the following way:

If p is the path that contains r, with lifetime $[b, e]$:

$$valid\_int(r, p) = [\max(b, t - \delta), t]$$

If p is an unexplored path with lifetime $[b, e]$ that links to a set of paths $p_1, p_2, \ldots, p_k$ with validity intervals $[b_1, e_1]$, $[b_2, e_2], \ldots, [b_k, e_k]$ respectively:

$$valid\_int(r, p) = \begin{cases} \Phi \quad \text{if } FT(p, p_i) \cap [b_i, e_i] = \Phi \ \forall i \mid 1 \leq i \leq k \\ [\max(b, t - \delta), \max_{t' \in \bigcup_{i=1}^{k} (FT(p,p_i) \cap [b_i, e_i])} t'] \quad \text{otherwise} \end{cases}$$

The latter equation means that the maximum validity time in a path is given by the maximum time in which the information object can flow in another path that have a compatible validity interval, while the minimum validity time is limited by $t - \delta$ and the starting time of the path.

The following lemma states that the coverage of a pair (p, t) can be identified by the set of requests whose validity intervals include t.

LEMMA 2. *Let (T, I) be an instance of demand cover and G_{CIDP} be the CIDP graph built from (T, I). Given a path p of G_{CIDP} and a time instant t, the coverage of p at time t is:*

$$C_{(p,t)} = \{r \in I \mid t \in valid\_int(r, p)\}$$

$valid\_int(r, p)$ can be computed for all paths in a BFS fashion, by starting from the path containing r and exploring the CIDP edges in reverse time order until the minimum starting time is reached. When a new vertex is visited, the validity interval of r in it is updated. The last step requires to explore the set $FT(p_i, p_j)$ of the incoming edge. This operation can be done in time $O(\log(|p_j|))$ by using a binary tree, where $|p_j|$ represents the number of Edig vertices that p_j contains. The overall complexity is then $O(|E_{cidp}| \cdot \log(|V_{edig}|))$ in the worst case, where E_{cidp} is the set of edges in the CIDP graph and V_{edig} is the set of vertices in the Edig graph. In practice, the complexity is almost linear in the number of edges of the compressed graph, since the IDPs are usually short.

[Conjecture: the complexity can be reduced to $O(|E_{cidp}|)$ by using a suitable strategy of graph partitioning. More in details, doing the topological order of the Edig graph and generating paths by DFS starting from the vertices that have lower order. The point is that with this strategy $|FT(p_i, p_j)| = 1$]

4.3.2 Prune

For each path p, we identify a set TI_p of time instants that are representative of the whole path, i.e. such that for all $t \in lifetime(p)$ we have $C_{(p,t)}$ contained in at least one set $C_{(p,t')}$ with $t' \in TI_p$. The following lemma states that this set is composed by the starting times of all validity intervals in the path.

LEMMA 3. *Let (T, I) be an instance of demand cover and G_{CIDP} be the CIDP graph built from (T, I). Given a path p of G_{CIDP}, consider the set $TI_p = \{b \mid [b, e] = valid\_int(p, r)$ with $r \in I\}$. For any time instant $t \in lifetime(p)$ we have:*

$$\exists t' \in TI_p \mid C_{(p,t)} \subseteq C_{(p,t')}$$

PROOF. Set t' as:

$$t' = \max_{\substack{b \in TI_p \\ b \leq t}} b$$

By lemma 2, $C_{(p,t)} = \{r \in I \mid t \in valid\_int(r, p)\}$. Let r be a request in $C_{(p,t)}$. By definition of t', the starting time of r cannot follow t'. Therefore $r \in C_{(p,t')}$. □

Figure 5 shows the path p_j of the example in Fig. 3 and the validity interval of each request in it. The validity intervals of r_1 and r_2 start at the begin of the path, since their minimum transmission times are antecedent to it. These intervals end at times ft_1 and ft_2 respectively, times associated to outgoing edges (see Fig. 3). For request r_1 (r_2 resp.), if the information object does not leave the path before ft_1 (ft_2 resp.), the destination cannot be reached in time. The validity interval of r_3 starts at time t_b, corresponding to the minimum transmission time of r_3, and ends ar time ft_3, time associated to the unique outgoing edge that can reach r_3. The representative time instants for this path are t_a and t_b, corresponding to minimum validity times of requests. Therefore, the minimal family of sets for this path is $\mathcal{S} = \{C_{(p_i,t_a)}, C_{(p_i,t_b)}\}$. Note that no other time instants have a coverage that is not included in at least one set of the family.

TI_p can be built in time $O(|I| \cdot \log(|I|))$ in the worst case. Given the set of requests that have a non empty validity interval in p, the set of time instants that correspond to minimum validity times and maximum validity times of all requests is first ordered in reverse order. Then a partial set C of requests is build by scanning the set of time instants (in reverse order). When the maximum validity time of a request r is processed, r is added to C. When the minimum validity time of a request r is processed, C is added to the family of candidate sets for Set-cover and r is removed from C.

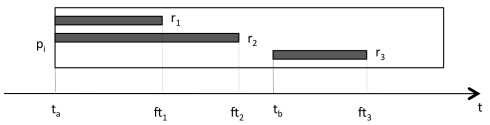


Figure 5: Validity intervals of a set of requests in a path, corresponding to the example depicted in Fig. 3. Bars represent the extent of validity intervals of requests. The minimal family of sets for this path is $\{C_{(p_i,t_a)}, C_{(p_i,t_b)}\}$.

4.4 Optimization

After the filtering process, a *post-pruning* (in short PP) phase can be applied in order to remove eventual sets that are fully contained in other sets. Note that, although the purpose of the filtering procedure is to remove these sets, this procedure is not guaranteed to be exhaustive. The post-pruning phase can be applied (expensively) independently from filtering (see Sect. 5).

We use an Integer Linear Program to solve Set-cover optimally. Finally, the optimal set of source-to-node object transmissions is extracted from the optimal subfamily returned by Set-cover.

5. EXPERIMENTAL ANALYSIS

5.1 Dataset

Cabs Mobility [16] (CAB, for short) contains mobility traces of taxi cabs in San Francisco, USA. It consists of GPS coordinates of 536 taxis collected over 23 days in the San Francisco Bay Area and the average time interval between two consecutive location updates is less than 10 seconds.

GeoLife GPS Trajectories [2] (GeoLife, for short) is a GPS trajectory dataset collected in (Microsoft Research Asia) GeoLife project by 165 users in a period of over two years.

Synthetic trajectories (SYN, for short) consists of $10K$ nodes that move on a 2-D plane with the size of 3600 km^2 over 10 days. The details on how we generate this dataset are specified in the Appendix.

For all datasets, the radio range allowing communication is set to 100 meters. The information demands (queries) are generated by a random process (detail are given in the Appendix).

5.2 Implementation

We implement the algorithm for the demand cover problem based on Edig (Sect. 3.2.2) as well as the CIDP indexing system (Sect. 4, details are discussed in the Appendix). We also tried a naive solution, developed as follows. We build a

composite connectivity graph by putting together a connectivity graph for each event. We use the composite graph to verify reachability between moving nodes and information requests and build an instance of Set-cover. We then execute an LP program for Set-cover on the computed instance. However, this solution was not feasible on the datasets employed, therefore we do not present it in the results. We do not compare against a naive LP solution, since clearly it would not be feasible, considering the huge number of variables that we should consider.

All the approaches are implemented in C++ (Dev C++ IDE ver. 4.9.9.2) and perform the experiments on a DELL Intel core I7 CPU with 2 Gb of memory. For the ILP solver we use *lp_solver* 5.5.2.0 [3], an open source tool based on branch-and-bound.

5.3 Results

On CAB, the execution time for demand cover queries is shown in Figure 6(a) and 6(b). Figure 6(a) shows the execution time for a number of datasets spanning from 1 to 10 days. The number of requests per cab per day is set to 2. Edig cannot answer queries in reasonable time for databases of more than 5 days. We extend the comparison to 10 days in order to demonstrate the strength of the CIDP approach. The reported time represents an average over 10 queries. We demonstrate the performance of both the Edig and the CIDP approach, with or without the post-pruning (PP) process. The post-pruning phase slightly improves the performance of both CIDP and Edig. However, CIDP performs more than 10 times faster than Edig in all cases and up to 100 times faster on a dataset of 5 days. CIDP performs a set of 10 queries on the whole dataset (23 days), with an average processing time of 40.2 seconds (not reported). In order to evaluate the scalability over the size of the query, we generate queries by varying the expected number of information requests per cab per day from 1 to 5. The results of the experiment over 1 day are reported in Figure 6(b). We do not report results for queries generated by setting the expected number of requests to more than 5, since in those cases Edig is unable to answer queries in an acceptable time.

On GeoLife, Figure 6(c) and 6(d) show the execution time for demand cover queries. In Figure 6(c) the expected number of requests per person per day is set to 10. The results refer to a set of datasets, each of them spanning a time interval ranging from 1 day to 30 days. As for CAB, the reported time represents an average over 10 queries. Here the post-pruning process barely improves the performance, due to the fact that the post-pruning itself is expensive. As before, we generate queries by varying the expected number of information requests per person per day from 1 to 10 aiming to evaluate the scalability over the size of the query. The results of the experiment over 1 day are presented in Figure 6(d). In this dataset, the CIDP approach scales better than Edig with the number of requests.

On SYN, the results are reported in Figure 6(e) and 6(f). In all cases the CIDP approach outperforms Edig. Figure 6(e) show that when we increase the time interval, the performances of CIDP on response time are at least 10 times better than for Edig in all cases. When we increase the expected number of requests per moving node per day, the comparison shown in Figure 6(f) indicates that the CIDP approach is more scalable than Edig.

Additional experimental results showing preprocessing time, index size and filtering performances are provided in the Appendix. Additionally, we report the communication cost achieved by our system in comparison to a naive solution.

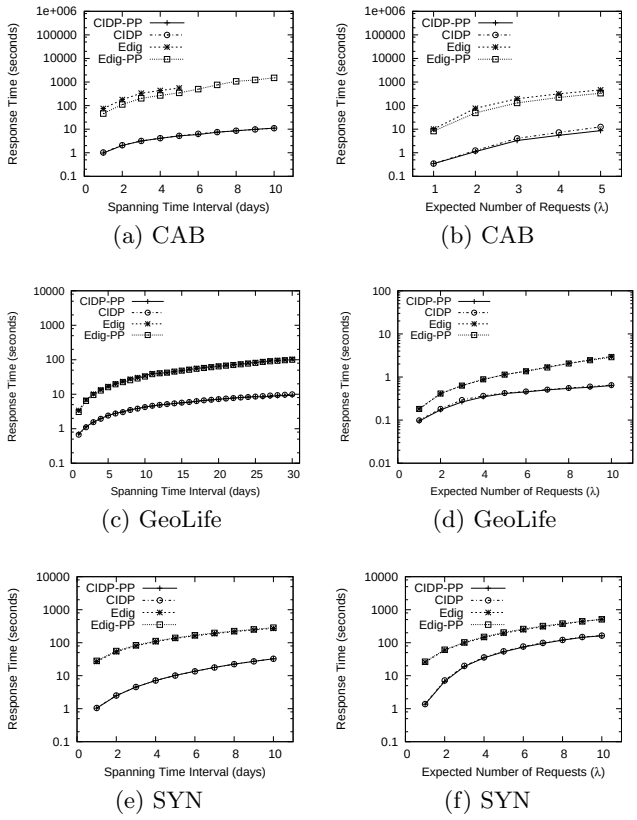


Figure 6: Evaluating the response time as a function of the size of the dataset (number of days) and the number of requests

6. CONCLUSION

We presented an a new approach that aim to optimize the communication cost for satisfying the information demand in a network of moving nodes with two kinds of communication. After formalizing the demand cover problem and showing that it is NP-hard, we provided a graph-based solution for it. Since solving the problem on large instances is expensive and may be unfeasible, we formulated the problem as a query problem and designed an indexing system that, after a preprocessing phase, is able to process queries in reasonable time. An extensive experimental analysis on real and synthetic datasets shows that our system effective and reasonably efficient on solving the demand cover problem. The proposed system can solve the demand cover problem optimally on large real instances (dataset with million od events and queries with thousands of nodes) in reasonable time (less than 10 seconds in most cases).

We plan to extend our work in two folds. First, we aim to take into account the uncertainty in mobility and information demand. This requires to fit stochastic mobility models in our framework with the purpose of optimizing the expected communication cost. Finally, we plan to consider the problem of scheduling new trajectories with the purpose

of guarantee the connectivity, in the case when the communication with a central information source is not always available.

7. REFERENCES

[1] Delay Tolerant Networking Research Group. http://www.dtnrg.org.

[2] GeoLife GPS Trajectories. http://research.microsoft.com/.

[3] M. Berkelaar et. al. Mixed Integer Linear Programming (MILP) solver. http://sourceforge.net/projects/lpsolve.

[4] S. Chen et. al. ST2B-tree: a self-tunable spatio-temporal B+-tree index for moving objects. In *SIGMOD*, pages 29–42, 2008.

[5] R. Di Natale et. al. SING: Subgraph search In Non-homogeneous Graphs. *BMC Bioinformatics*, 11(1):96, 2010.

[6] V. Jacobson et. al. Networking named content. In *CoNEXT*, pages 1–12, 2009.

[7] H. V. Jagadish. A compression technique to materialize transitive closure. *ACM Trans. Database Syst.*, 15:558–598, December 1990.

[8] S. Jain, M. Demmer, R. Patra, and K. Fall. Using redundancy to cope with failures in a delay tolerant network. *SIGCOMM Comput. Commun. Rev.*, 35:109–120, August 2005.

[9] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. *SIGCOMM Comput. Commun. Rev.*, 34:145–158, August 2004.

[10] C. S. Jensen, D. Lin, and B. C. Ooi. Query and update efficient B+-tree based indexing of moving objects. In *VLDB*, pages 768–779, 2004.

[11] H. Jeung et. al. A Hybrid Prediction Model for Moving Objects. In *ICDE*, pages 70–79, 2008.

[12] R. Jin et. al. Efficiently answering reachability queries on very large directed graphs. In *SIGMOD*, pages 595–608, 2008.

[13] J. Leguay, T. Friedman, and V. Conan. DTN routing in a mobility pattern space. In *2005 ACM SIGCOMM workshop on Delay-tolerant networking*, WDTN '05, pages 276–283, 2005.

[14] Z. Li et. al. MoveMine: mining moving object databases. In *SIGMOD*, pages 1203–1206, 2010.

[15] M. Mongiovi et. al. SIGMA: a set-cover-based inexact graph matching algorithm. *J Bioinform Comput Biol*, 8(2):199–218, 2010.

[16] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. CRAWDAD data set epfl/mobility (v. 2009-02-24). http://crawdad.cs.dartmouth.edu/epfl/mobility, Feb. 2009.

[17] T. Spyropoulos et. al. Efficient routing in intermittently connected mobile networks: the multiple-copy case. *IEEE/ACM Trans. Netw.*, 16:77–90, February 2008.

[18] T. Spyropoulos et. al. Efficient routing in intermittently connected mobile networks: The single-copy case. *IEEE/ACM Trans. Netw.*, 16(1):63–76, Feb 2008.

[19] Y. Tao, D. Papadias, and J. Sun. The TPR*-tree: an optimized spatio-temporal access method for predictive queries. In *VLDB*, pages 790–801, 2003.

[20] Y. Tao et. al. Prediction and indexing of moving objects with unknown motion patterns. In *SIGMOD*, pages 611–622, 2004.

[21] A. Vahdat and D. Becker. Epidemic Routing for Partially-Connected Ad Hoc Networks. Technical report, 2000.

[22] S. Šaltenis et. al. Indexing the positions of continuously moving objects. *SIGMOD Rec.*, 29:331–342, May 2000.

[23] X. Yan, P. S. Yu, and J. Han. Graph Indexing Based on Discriminative Frequent Structure Analysis. *ACM Trans. on Database Systems*, 30(4):960–993, 2005.

[24] D. M. Yellin. Algorithms for subset testing and finding maximal sets. In *SODA*, pages 386–392, 1992.

[25] S. Zhang, J. Yang, and W. Jin. SAPPER: subgraph indexing and approximate matching in large graphs. *PVLDB*, 3:1185–1194, September 2010.

[26] M. Zhang et. al. Effectively indexing uncertain moving objects for predictive queries. *PVLDB*, 2:1198–1209, August 2009.

APPENDIX

7.1 ACKNOWLEDGEMENTS

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

7.2 Proof of theorems

7.2.1 Proof of theorem 3.1

PROOF. Let the family S be an instance of Set-cover. Choose d, t_s and t_d arbitrarily and set $\delta = t_d - t_s$. For each element c of C, consider a fixed (non-moving) node n_c and an information request (n_c, t_d, δ). Set the trajectories of these nodes along a line so that any two of them are always at a distance greater than d from each other. Consider m points $p_1, p_2, \ldots, p_m$ separated by a distance greater than d in a parallel line at a considerable distance $dl >> d$ from the first line (how to choose the value of dl will be discussed later). Given a set S_i, for each element c of S_i consider a straight trajectory starting from p_i and ending to the location of m_c. The number of moving nodes will be $z = \sum_{i=1}^{n} |Si|$. The speed of each moving node is assigned in the following way. Divide the time interval $[t_s, t_d]$ in z slots of the same length, each of them associated to a moving node. Each node remains without moving until its time slot is reached. Then it moves with a constant speed that allows it to reach the destination at the end of the time slot. Then it stops again. The value dl is chosen so as to guarantee that the moving trajectories are always at a distance greater than d from any points in $\{p_1, p_2, \ldots, p_m\}$ and any fixed nodes, except the two endpoints. The full proof depends on geometrical considerations and will appear in the full paper.

It is easy to verify that an optimal set of source-to-node object transmissions corresponds to an optimal subfamily of S for Set-cover. □

7.3 Details of algorithms

7.3.1 Preprocessing

A pseudocode of the preprocessing phase is given in Algorithm 1. Given the set of trajectories, first the procedure *buildEvents* generates the initial connectivity graph (G) and a set of in-events and out-events that represents the changes in the connectivity graph over time. An Edig graph (G_{edig}) based on in-events and out-events is then generated (procedure *buildEdigTIL*). Ir-events are not considered since during the preprocessing phase the information demand is unknown. For each vertex of G_{edig} the lifetime of such a component i.e. the interval $(s, e]$ in which that component is valid, is stored in til_c. We call s the start-time and e the end-time.

The procedure *buildCIDP* decomposes the Edig graph in a disjoint set of paths. We generate a compressed graph G_{cidp} where each vertex is an IDP and two vertices are connected by an edge if and only if the two IDPs are connected by at least one edge in the original G_{edig} graph. Each IDP p is associated with its lifetime, i.e. the union of the lifetime of the vertices contained in p. The lifetime of paths is stored in til_p. Each edge of G_{cidp} is associated with the end-time of the source vertex in the G_{edig} graph (maximum time from which an information object can flow through this edge). We call this time the maximum edge time and store it in tel_e. During the process, the correspondence between original vertices, components of G_{edig} and paths of G_{cidp} is maintained in the data structures idx_{v2c}, idx_{c2v}, idx_{c2p} and idx_{p2c}.

Algorithm 1 preprocessing(T)

1: $(Ev, G) \leftarrow$buildEvents(T);
2: $(G_{edig}, idx_{v2c}, idx_{c2v}, til_c) \leftarrow$**buildEdigTIL**$(Ev, G)$;
3: $(G_{cidp}, idx_{c2p}, idx_{p2c}, til_p, tel_e) \leftarrow$**buildCIDP**$(G_{edig}, til_c)$;
4: **return** $(G_{cidp}, til_c, til_p, tel_e, idx_{v2c}, idx_{c2v}, idx_{c2p}, idx_{p2c})$;

Algorithm 2 buildEdigTIL(Ev, G)

1: $(G_{edig}, idx_{v2c}, idx_{c2v}, cnum) \leftarrow$cc_compress($G$);
2: **for** $i \leftarrow 1$ to $cnum$ **do**
3: $\quad til_c[i] \leftarrow (0, MXT]$;
4: **for** $k \leftarrow 1$ to $Ev.size()$ **do**
5: $\quad (n_i, n_j, type, time) \leftarrow Ev[k]$;
6: $\quad$ **if** $type$=IN **then**
7: $\quad\quad c_i \leftarrow idx_{v2c}[n_i, time]$, $c_j \leftarrow idx_{v2c}[n_j, time]$;
8: $\quad\quad$ **if** $c_i \neq c_j$ **then**
9: $\quad\quad\quad (s_i, e_i] \leftarrow til_c[c_i]$, $(s_j, e_j] \leftarrow til_c[c_j]$;
10: $\quad\quad\quad til_c[c_i] \leftarrow (s_i, time]$, $til_c[c_j] \leftarrow (s_j, time]$;
11: $\quad\quad\quad cnum \leftarrow cnum + 1$;
12: $\quad\quad\quad til_c[cnum] \leftarrow (time, MXT]$;
13: $\quad\quad\quad V(G_{edig}) \leftarrow V(G_{edig}) \cup \{cnum\}$;
14: $\quad\quad\quad E(G_{edig}) \leftarrow E(G_{edig}) \cup \{\langle c_i, cnum\rangle\}$;
15: $\quad\quad\quad E(G_{edig}) \leftarrow E(G_{edig}) \cup \{\langle c_j, cnum\rangle\}$;
16: $\quad\quad\quad vlist \leftarrow idx_{c2v}[c_i] \cup idx_{c2v}[c_j]$;
17: $\quad\quad\quad idx_{c2v}[cnum] \leftarrow vlist$;
18: $\quad\quad\quad$ **for** each $n_l \in vlist$ **do**
19: $\quad\quad\quad\quad idx_{v2c}[n_l, til_c[cnum]] \leftarrow cnum$;
20: $\quad\quad E(G) \leftarrow E(G) \cup \{(n_i, n_j)\}$;
21: $\quad$ **else**
22: $\quad\quad E(G) \leftarrow E(G) - \{(n_i, n_j)\}$;
23: $\quad\quad vlist \leftarrow$ccAt$(G, n_i)$;
24: $\quad\quad$ **if** $n_j \notin vlist$ **then**
25: $\quad\quad\quad c_i \leftarrow idx_{v2c}[n_i, time]$, $cnum \leftarrow cnum + 2$;
26: $\quad\quad\quad (s_i, e_i] \leftarrow til_c[c_i]$, $til_c[c_i] \leftarrow (s_i, time]$;
27: $\quad\quad\quad til_c[cnum - 1] \leftarrow (time, MXT]$;
28: $\quad\quad\quad til_c[cnum] \leftarrow (time, MXT]$;
29: $\quad\quad\quad V(G_{edig}) \leftarrow V(G_{edig}) \cup \{cnum - 1, cnum\}$;
30: $\quad\quad\quad E(G_{edig}) \leftarrow E(G_{edig}) \cup \{\langle c_i, cnum - 1\rangle\}$;
31: $\quad\quad\quad E(G_{edig}) \leftarrow E(G_{edig}) \cup \{\langle c_i, cnum\rangle\}$;
32: $\quad\quad\quad idx_{c2v}[cnum - 1] \leftarrow vlist$;
33: $\quad\quad\quad$ **for** each $n_l \in vlist$ **do**
34: $\quad\quad\quad\quad idx_{v2c}[n_l, til_c[cnum - 1]] \leftarrow cnum - 1$;
35: $\quad\quad\quad vlist \leftarrow$ccAt$(G, n_j)$;
36: $\quad\quad\quad idx_{c2v}[cnum] \leftarrow vlist$;
37: $\quad\quad\quad$ **for** each $n_l \in vlist$ **do**
38: $\quad\quad\quad\quad idx_{v2c}[n_l, til_c[cnum]] \leftarrow cnum$;
39: **return** $(G_{edig}, idx_{v2c}, idx_{c2v}, til_c)$;

7.3.2 Filtering

Algorithm 3 buildCIDP(G_{edig}, til_c)

```
1: pnum ← 0;
2: for each c_i ∈ V(G_edig) do
3:    visited[c_i] ← false;
4: for each c_i ∈ V(G_edig) do
5:    if visited[c_i] = false then
6:       pnum ← pnum + 1;
7:       V(G_cidp) ← V(G_cidp) ∪ {pnum};
8:       clist←simplepath_search(G_edig, visited, c_i);
9:       idx_p2c[pnum] ← clist;
10:      for each c_j ∈ clist do
11:         idx_c2p[c_j] ← pnum;
12:         (s_j, e_j] ← til_c[c_j];
13:         (s, e] ← til_p[pnum];
14:         til_p[pnum] ← (min(s_j, s), max(e_j, e));
15: for each ⟨c_i, c_j⟩ ∈ E(G_edig) do
16:    p_i ← idx_c2p[c_i], p_j ← idx_c2p[c_j];
17:    (s_i, e_i] ← til_c[c_i];
18:    if p_i ≠ p_j then
19:       E(G_cidp) ← E(G_cidp) ∪ {⟨p_i, p_j⟩};
20:       tel_e[⟨p_i, p_j⟩] ← e_i;
21: return (G_cidp, idx_c2p, idx_p2c, til_p, tel_e);
```

A pseudocode of the *backflow* procedure is given in Algorithm 4. A heap (*maxheap*) is used to maintain the vertices to visit and the associated times. $tel_e[(p_j, p_k)]$ represents the maximum edge time of the edge (p_j, p_k)

For each queue Q_p, the procedure *prune* (Alg. 5) proceeds in the following way. First, it initializes an empty set (*set*) and then examines the entries of Q_p in order (reverse time order). For each entry of kind +, the corresponding information request is added to *set*. When an entry of kind − is encountered, the current set is placed in the list of candidate sets and the information request corresponding to that entry is removed from *set*. At the end the final set is placed in the list of candidate sets.

Algorithm 4 backflow(r)

```
1: (n_i, t, δ) ← r;
2: c_i ← idx_v2c[n_i, t];
3: p_i ← idx_c2p[c_i];
4: for each p_k ∈ V(G_cipd) do
5:    visited[p_k] ← false;
6: t_start ← t − δ, t_end ← t;
7: maxheap.insert(t_end, p_i);
8: while maxheap ≠ φ do
9:    (t_k, p_k) ← maxheap.pop();
10:   if visited[p_k] = false then
11:      visited[p_k] ← true;
12:      Q[p_k].push((n_i, t, δ), t_k, '+');
13:      (s_k, e_k] ← til_p[p_k];
14:      if t_start > s_k then
15:         Q[p_k].push((n_i, t, δ), t_start, '−');
16:      for each ⟨p_j, p_k⟩ ∈ G_cidp do
17:         t_j ← tel_e[⟨p_j, p_k⟩];
18:         if visited[p_j] = false and t_start ≤ t_j < t_k then
19:            maxheap.insert(t_j, p_j);
20: return Q;
```

7.4 Additional Experiment Results

Algorithm 5 prune(Q)

```
1: candidates ← φ;
2: for each p ∈ V(G_cidp) do
3:    set ← φ, M ← '−';
4:    for i ← 1 to Q[p].size() do
5:       (r, t, symbol) ← Q[p][i];
6:       if symbol = '+' then
7:          set ← set ∪ {r};
8:       else
9:          if M = '+' then
10:            candidates ← candidates ∪ {(set, p, t)};
11:         set ← set\{r};
12:      M ← symbol;
13: return candidates;
```

7.4.1 Dataset

GeoLife GPS Trajectories (GeoLife) records a broad range of users' outdoor movements, including life routines like go home and go to work and some entertainments and sports activities, such as shopping, sightseeing, dining, hiking, and cycling.

Synthetic trajectories consists of $10K$ moving nodes, the starting coordinates of which are uniformly distributed on the plane. For each moving node we record its coordinates every 60 seconds. At some time points, the moving node updates its moving speed, by replacing it with another speed generated via normal distribution with $\mu = 1.2$ meters per second and $\sigma = 1$, and its moving direction, by adding a deviation generated via normal distribution with $\mu = 0$ and $\sigma = 1$ radiant. The time interval between two consecutive updates is generated by an exponential distribution with $\mu = 60$ seconds.

Queries are generated as follows: for each trajectory and for each information object, first a number of information requests is generated by a Poisson distribution (with an average of 1 request per moving node per day unless differently specified). Then, for each information request (n_i, t, δ), its request time t is generated by a uniform distribution in the whole interval and δ is generated by a normal distribution with mean $\mu = 15$ minutes and $\sigma = 1$ minute.

7.4.2 Implementation

The implementation of *Edig algorithm* goes as follows. Given an Edig representation G_{edig}, we run depth-first search on each candidate vertex v of G_{edig}, to generate a set of information needs that are covered by v. We then use an ILP solver to execute Set-cover optimally on the resulting family of sets. Before being fed into the ILP solver, the input family for Set-cover is divided into disjoint subfamilies. The Set-cover solver is then executed on each subfamily and eventually the results are merged. We set the timeout for each execution of the ILP solver to 300 seconds. If execution time exceeds the timeout, we consider it as a failure to answer a query.

The implementation of the CIDP approach is described in Section 3.2.2 and its Set-cover solver is implemented in the same way as for the Edig algorithm.

7.4.3 Results

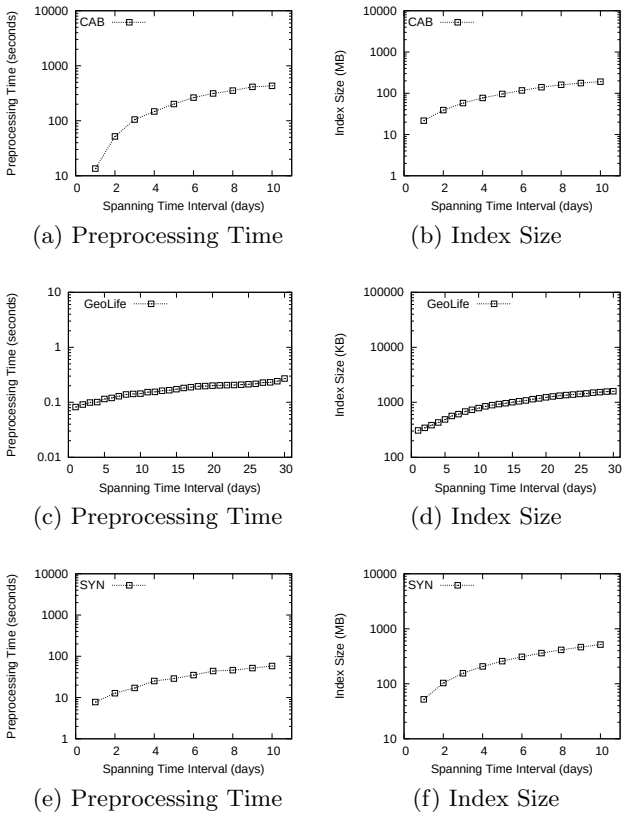
The preprocessing time and the index size of CIDP for all datasets are reported in Figure 7. Since the preprocessing phase does not apply to Edig we do not compare with it.

In order to verify scalability, we evaluate the algorithm on a number of datasets. For CAB and SYN, each dataset spans a number of days ranging from 1 to 10. For GeoLife, each spans a number of days from 1 to 30.

On CAB, the time for preprocessing and the size of index are presented in Figure 7(a) and 7(b) respectively.

On GeoLife, the preprocessing time and the index size of CIDP are demonstrated in Figure 7(c) and 7(d) respectively. We experiment on a number of datasets and each of them spans a number of days ranging from 1 to 30. In GeoLife, the preprocessing time is less than that for CAB and the index size is smaller. The main reason is that the number of events captured in Geolife is much smaller than the ones in CAB. In GeoLife, the average number of events per day is $1,401$, while in CAB it is $809,558$.

On SYN, the preprocessing time and the index size is shown in Figure 7(e) and Figure 7(f) respectively. This experiment performs on a number of datasets, each of them spanning a number of days ranging from 1 to 10. The number of events captured in SYN is $904,818$ per day, which is comparable to that in CAB. Since in SYN there are $10,000$ moving nodes, the average number of events per node per day is 90.5 while for CAB it is $1,510$. Therefore a moving node in CAB has more opportunities to connect to other nodes and it takes more time to identify connected components in preprocessing phase for CIDP.



(a) Preprocessing Time (b) Index Size

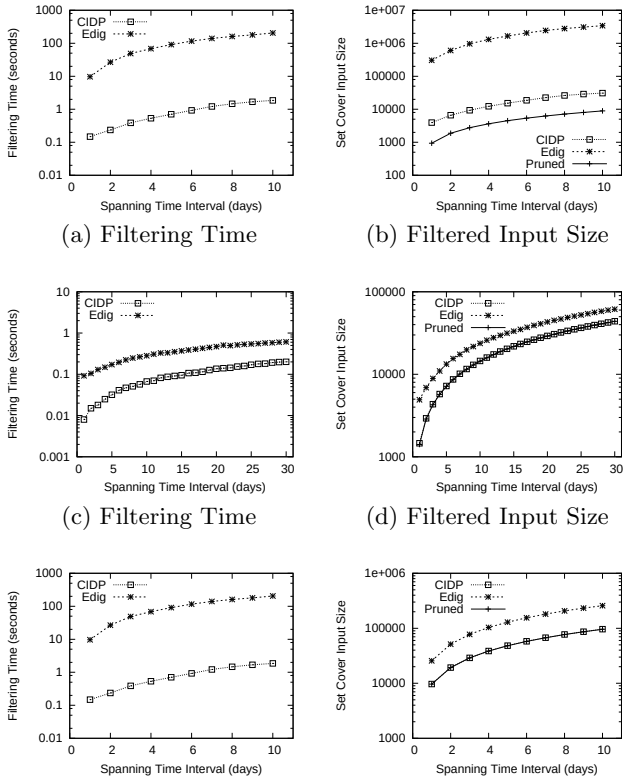
(c) Preprocessing Time (d) Index Size

(e) Preprocessing Time (f) Index Size

Figure 7: Preprocessing time and index size produced by CIDP on different datasets

7.4.4 Evaluation of Filtering Capability

In Figure 8, the filtering time and the size of the input family for Set-cover obtained by Edig and CIDP are reported respectively for all datasets. In the case of Edig, the filtering time refers to the time for generating the family of sets. CIDP strongly outperforms Edig on both filtering time and size of the input family generated for Set-cover in all cases. Figures 8(a), 8(c) and 8(e) show that CIDP spends much less time in generating the input family for Set-cover and especially in SYN, CIDP performs up to 100 times better than Edig. Figure 8(b), 8(d) and 8(f) show that the size of the input family filtered by CIDP is much smaller than the one generated by Edig. The size of the input family after post-pruning (referred as Pruned) is also reported and show that in SYN and GeoLife the input family generated by CIDP is almost the most compact one.



(a) Filtering Time (b) Filtered Input Size

(c) Filtering Time (d) Filtered Input Size

(e) Filtering Time (f) Filtered Input Size

Figure 8: Evaluation of filtering capability

7.4.5 *Communication cost*

In Figure 9 we show the communication cost obtained by our solution, compared to a naive solution that considers a source-to-node transmission for each information request. We do not distinguish between Edig and CIDP since they produce the same result. We perform this experiment on CAB. The queries are generated by varying the expected number of requests per cab per day λ from 1 to 20. With $\lambda = 20$ our solution reduces the number of transmissions by more than 50% and the gain increases with λ.

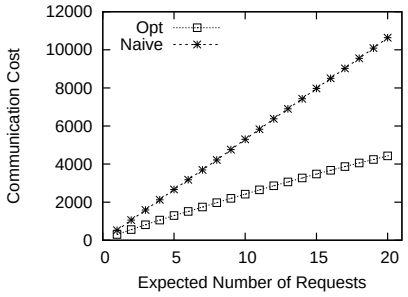


Figure 9: Communication cost with varying number of requests per cab per day