Efficient multicasting for delay tolerant networks using graph indexing

Misael Mongiovì, Ambuj K. Singh,
Xifeng Yan and Bo Zong
Department of Computer Science
University of California
Santa Barbara, CA 93106, US
Email: {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
Email: kpsounis@usc.edu

***Abstract*—In Delay Tolerant Networks (DTNs), end-to-end connectivity between nodes does not always occur due to limited radio coverage, node mobility and other factors. Remote communication may assist in guaranteeing delivery. However, it has a considerable cost, and consequently, minimizing it is an important task. For multicast routing, the problem is NP-hard, and naive approaches are infeasible on large problem instances.**

In this paper we define the problem of minimizing the remote communication cost for multicast in DTNs. Our formulation handles the realistic scenario in which a data source is continuously updated and nodes need to receive recent versions of data. We analyze the problem in the case of scheduled trajectories and known traffic demands, and propose a solution based on a novel graph indexing system. We also present an adaptive extension that can work with limited knowledge of node mobility. Our method reduces the search space significantly and finds an optimal solution in reasonable time. Extensive experimental analysis on large real and synthetic datasets shows that the proposed method completes in less than 10 seconds on datasets with millions of encounters, with an improvement of up to 100 times compared to a naive approach.

I. Introduction

Delay Tolerant Networks (DTNs) are communication networks that lack continuous connectivity due to node mobility, failures or other factors. They experience frequent partitioning, and end-to-end paths between two nodes may never exist. Routing in DTNs uses a *store-carry-forward* approach [1], where intermediate nodes delay the transmission of messages until new links are available and messages are "eventually" delivered with some delay. When the lack of connectivity is due to node mobility, the movement of nodes can be exploited to carry the messages.

In recent years, routing protocols for multicast in DTNs have received considerable attention [2], [3], [4]. Multicast protocols optimize the transmission cost by sharing routing paths among multiple destinations. Recent advances allow us to achieve a good tradeoff between minimization of the transmission cost and maximization of the delivery rate. However, due to the nature of DTNs, proper delivery cannot always be guaranteed.

To guarantee the connectivity in DTNs, nodes can be equipped with *remote communication* (or long range communication) devices [5], to be used when end-to-end communication cannot be otherwise achieved. Since remote communication is expensive, its utilization should be limited as much as possible. Providing an extra remote communication to guarantee delivery introduces the new challenging problem of minimizing its cost.

We formulate the problem of optimizing the remote communication cost in a network of moving nodes, and call it the *demand cover problem*. Our model considers a set of *moving nodes* (e.g. people or vehicles) that are equipped with devices providing two kinds of communication. A *short-range* communication (e.g. radio), considered non-costly, can occur between two nodes when they are close to each other. A remote communication (e.g. cellular or satellite), which involves a fixed cost, can occur at any time independently of node positions. In our model, a set of continuously-updated *data objects* needs to be shared among nodes. Each data object needs to be received by a subset of destinations. For each destination, its *deadline* (time instant at which the object is needed) is specified. To avoid receiving non-recent copies of objects, a *latency* is also specified. We aim to find the set of remote transmissions that minimizes the communication cost subject to the aforementioned delay constraints.

The described problem has several practical applications. For instance, consider a network of city buses, in which a transportation agency wants to provide passengers with personalized news that depends on their position, traveling plan etc. Each bus can obtain the news updates by the cellular network (costly). However, it is more convenient to share the information among various buses via radio communication (non-costly). Another example considers soldiers or military vehicles 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 (costly) and node-to-node communication (zero cost). The proposed approach also has applications in data ferrying [5], [6]. A set of moving nodes (ferries) is charged for gathering data. Depending on time constraints on data delivery, the ferries may decide whether to use short-range or remote communication. Note that in these examples, the node trajectories and the traffic demands are known in advance.

Solving the demand cover problem introduces new chal-

978-1-4673-0775-8/12/$31.00 ©2012 IEEE

lenges due to temporal constraints. A data object may need to be transmitted remotely more than once, due to either lack of connectivity or a latency constraint. For instance, consider the four nodes in Fig. 1(a) that move following certain trajectories. Initially, nodes 2 and 3 are in contact. At time t_1, nodes 2 and 4 enter in each other's radio range and a new contact begins. At time t_2, the contact between nodes 2 and 3 ends since they move away from each other. Three of these nodes (shown with triangles) need to receive the same data object. Each of them has a given deadline (t_a, t_b and t_c, resp.) and a latency (δ_a, δ_b and δ_c, resp.). A remote transmission to node 3 at time t_1 covers the data needs r_a and r_b. Although r_c can be reached by transmitting the object to node 4 at any time after t_1, the latency δ_c cannot be satisfied. Therefore, an updated copy of the object needs to be transmitted during the interval $[t_c - \delta_c, t_c]$.

In this paper, we prove that the demand cover problem is NP-hard and present a baseline graph-based approach for it. In order to make this problem feasible on large datasets, we formulate it as a query processing problem and develop a novel graph-indexing-based solution. Due to the index, we are able to handle thousands of destinations on a network with millions of encounters in less than 10 seconds, with an improvement of up to 100 times compared to a naive approach.

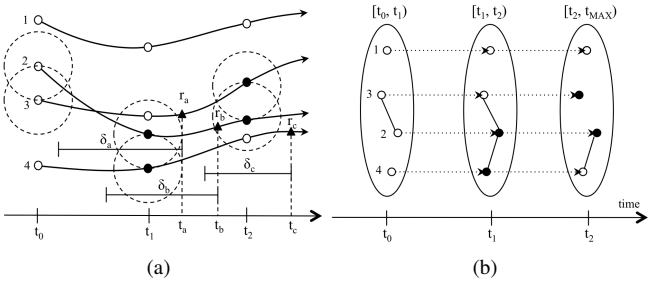


Fig. 1. An example of a DTN among moving nodes. (a) The four solid lines represent four trajectories. Time is denoted on the x-axis. The big dashed circles represent radio range of nodes. Nodes that are involved in a transition (contact beginning or contact end) are filled. Three data needs (r_a, r_b and r_c) are represented with filled triangles. Their deadlines are t_a, t_b and t_c, respectively while their latencies are δ_a, δ_b and δ_c, respectively. (b) The corresponding space-time graph. Snapshots of the connectivity graph at three different times are depicted within big ovals. Temporal links joining contiguous snapshots are represented with dotted lines.

Our contribution can be summarized as follows: (i) We define a novel problem (namely *demand cover*) that formalizes the problem of optimizing the remote communication cost in a network of moving nodes. (ii) We prove that demand cover is NP-hard. (iii) We develop a compact graph-based representation of a *demand cover* instance and present a baseline algorithm. (iv) We propose a novel indexing system for quickly solving demand cover on graphs optimally. Our system further compresses the compact graph and uses an efficient filtering approach to retrieve a small portion of vertices that are relevant for achieving an optimal solution. (v) 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 encounters.

The paper is structured as follows: Sect. II describes the related work. Sect. III introduces some basic concepts in DTNs. Sect. IV defines the problem, provides a graph representation and presents a simple solution. Sect. V describes our indexing system for demand cover. Sect. VI presents extensive experimental analysis on real and synthetic datasets. Finally, Sect. VII concludes the paper with some future directions.

II. Related Work

Previous works on DTNs have focused on three types of contacts: scheduled, predicted and opportunistic. Scheduled contacts result from applications of known trajectories, such as deep-space communication and data service in developing regions [1], [7], [6], [8]. Predicted contacts are considered in applications where mobility patterns exist [9], [10], [11], [12], [13], [14], [15]. Opportunistic contacts deal with completely uncertain circumstances where mobile nodes meet each other by random chance. Our work falls into the category of scheduled contacts [6], [1].

Graph representations are widely applied in studying routing strategies. In [16], [17], [6], evolving graphs are employed to model topological mutations in DTNs. In [18], the authors consider the shortest path problem in evolving graphs and its generalizations. In [19], the authors focus on the problem of finding the multicast tree with minimum overall transmission time in evolving graphs. In our work, we use compression and indexing techniques to efficiently explore evolving graphs with the purpose of minimizing the remote communication cost.

Multicast for DTNs has recently drawn considerable attention. In [2], semantic models are proposed to unambiguously describe multicast in the context of DTNs. The throughput of multicast is discussed in [3] and mobility-assisted routing is used to improve the throughput bound of wireless multicast. In [4], multicast problems in DTNs are considered in a social network setting where centrality and community in DTNs are employed to help determine the appropriate selection of relays, with the objective of minimizing the delay of multicast message transmissions. In this paper, we study a novel optimization problem which is similar to traditional multicast problems. However, instead of minimizing the delay of message transmissions, we are interested in minimizing the communication cost subject to time constraints. To this end, the question of whether a node is reachable from another node is more important than the question of how a message flows in the network.

Graph indexing systems have been widely studied by the database community. The most common approaches aim to efficiently solve problems as graph matching [20], [21] or reachability test [22], [23]. The closest to our work are systems for reachability tests, which aim to efficiently check if two vertices are reachable from each other (a path that connects them exists) in a directed graph. Some systems use chains [24] (generalization of paths) decomposition or path-tree [23] 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 also

reachable from v. We use a similar idea, but our system is designed to quickly identify the regions of the graph that can reach a given destination instead of verifying the reachability between pairs. Moreover, we propose a method for finding a small subset of representative vertices that allows us to solve the demand cover problem optimally and with reasonable efficiency on large datasets.

III. Preliminaries

In this section, we describe some basic concepts concerning DTNs for introducing our approach. We consider a network of moving nodes whose *trajectories* are known in advance or can be predicted. When two nodes enter each other's radio coverage area, a link between them is formed and a *contact* (or encounter) begins. A contact between two nodes terminates when they lose radio connectivity as they move away from each other. Contact beginnings and contact ends are also called *transitions*. The status of the network at a certain time instant can be described by a *connectivity graph*, whose vertices represent moving nodes and a link is placed between two nodes if their distance is within a given threshold d, called the *radio range*. The network dynamics can be described by a series of snapshots of the connectivity graph over time [17], [1]. All the snapshot graphs are aggregated in a unique composite graph (called the *space-time graph*) where vertices corresponding to the same moving node in two consecutive connectivity graphs are joined by a *temporal link*. In contrast to *spatial links*, temporal links are directed. A message can travel across a so called *space-time path*. If some spatial links toward the destination are available, the message is forwarded, otherwise the message is *carried* by the moving node (a temporal link is traversed) and forwarded when another suitable node is encountered. In the following, we use the term *route* to indicate the space-time path that a message traverses.

Fig. 1(b) shows the space-time graph corresponding to Fig. 1(a). Three snapshot graphs are represented, each of them describing the connectivity of the network at the time intervals $[t_0, t_1)$, $[t_1, t_2)$ and $[t_2, t_{MAX})$, respectively. Contiguous snapshots are joined by temporal links (in dotted line). Each snapshot is associated with its *lifetime*, i.e. the extent in time to which it refers. Message traversals (routes) can include both spatial and temporal links.

IV. Problem Description

In this section, we formally define the demand cover problem and develop some baseline approaches. We consider a set of n *nodes* (numbered $1, 2, \ldots, n$) that move following certain *trajectories* ($T_1, T_2, \ldots, T_n$, respectively). A trajectory associates a node with a position in space (typically a plane) at each time instant in the range $[t_0, t_{MAX})$. At a specific time, two nodes can communicate with each other through a so called *contact transmission* (short-range, typically radio) if their Euclidean distance is within a fixed threshold d (we also say that they are *in contact*). This contact transmission (between nodes) does not incur any cost. Each node can also communicate at any time with a *data source* (Internet or a central server) through a costly *remote transmission* (cellular or satellite), with a fixed cost (our approach can also be extended to a decentralized scenario where a central server is not available and data are distributed among nodes).

We consider the problem of delivering *data objects* to multiple *destinations*. In contrast to other multicast approaches in which static messages are sent to multiple destinations, we consider the problem of sharing data objects that are continuously updated over time. The dynamic character of data objects introduces new constraints: each destination needs to receive the object before a given *deadline* and with a delay that is limited by a given *latency*. We define the *data demand* I of a data object as its set of *data needs*, i.e. triples of the form (i, t, δ), where i, t and δ represent the destination, the deadline and the latency, respectively. We call the instant $t - \delta$ the *release time*. It represents the earliest time instant at which an object can be obtained from the data source.

The data flow in our setting is modeled by two kinds of transmissions: a *remote transmission* is denoted by a pair (i, t), where i represents the node that receives the object and t is the time instant in which the transmission occurs; a *contact transmission*, is represented by a triple (i_1, i_2, t) where i_1, i_2, t represent the node that transmits the object, the node that receives it and the time instant at which the transmission occurs, respectively. For simplicity, all the transmissions are considered instantaneous (the movement between nodes is usually very slow compared to the speed of transmission; therefore, all the necessary objects can be transmitted before the contact terminates). We say that a remote transmission (i_s, t_s) *covers* a data need (i_d, t_d, δ) if there exists a sequence of contact transmissions (i_0, i_1, t_1), (i_1, i_2, t_2), ..., (i_{k-1}, i_k, t_k) with $i_0 = i_s$ and $i_k = i_d$, 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 set of data needs covered by a remote transmission is also called the *coverage* of the remote transmission. The demand cover problem is defined as follows:

Problem definition: Given a set of trajectories and a data object with demand I, find the minimum set of remote transmissions that covers I.

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 $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ of elements taken from a set C, Set-cover calls for finding the minimum sub-family of $\mathcal{S}$ that covers all the elements of C. A proof is given in our extended technical report [25].

A. ILP formulation

The demand cover problem can be formulated in ILP (Integer Linear Programming) and solved by a standard solver. Here, we give an ILP formulation and show that solving it on large datasets is infeasible.

We consider a set of n moving nodes numbered 1 through n and a special node that represents the central server, numbered 0. We write $i \rightarrow_t j$ if node i can communicate with node j at time t (i.e. they are within distance d or $i = 0$). We also consider a discrete set $\mathcal{T}$ of time instants that correspond to

transitions or deadlines of data needs. This restriction does not compromise the result. 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 is delayed until the contact between them ends (right before the link breaks) or a data need that involves one of the nodes expires, without increasing the cost. Since communication is assumed to be instantaneous, the contact length is not important.

We employ two classes of boolean variables. The first class contains variables of the kind $x_{i,j,t,r}$, where i and j represent nodes, t represents a time instant and $r = (i_r, t_r, \delta_r) \in I$ represents a data need. This class of variables models the flow of objects. The variable $x_{i,j,t,r}$ has value 1 if i sends a message to j at time t to satisfy the data need r. Variables of this kind are considered for $i \rightarrow_t j$ and $t_r - \delta_r \leq t \leq t_r$. The second class of variables, of the kind $y_{i,t}$, counts the number of remote transmissions. Each variable says whether a remote transmission between the central server and a particular node occurs at a certain time or not. The complete ILP formulation is as follows.

$$min \quad \sum_{t \in \mathcal{T}} \sum_{i=1}^{n} y_{i,t}$$

$$s.t. \quad \sum_{\substack{t \in \mathcal{T} \\ t \geq t_r - \delta_r \\ t \leq t_r}} \sum_{\substack{i=0 \ldots n \\ i \rightarrow_t i_r}} x_{i,i_r,t,r} \geq 1 \quad \forall r = (i_r, t_r, \delta_r) \in I \quad (1)$$

$$\sum_{\substack{t' \in \mathcal{T} \\ t_r - \delta_r \leq t' \leq t}} \sum_{\substack{i=0 \ldots n \\ i \rightarrow_{t'} j_1}} x_{i,j_1,t',r} - x_{j_1,j_2,t,r} \geq 0 \quad (2)$$

$$\forall r = (i_r, t_r, \delta_r) \in I, \forall j_1, j_2, t \mid j_1 \rightarrow_t j_2$$

$$y_{i,t} \geq x_{0,i,t,r} \quad \forall r \in I, i = 1 \ldots n, t \in \mathcal{T} \quad (3)$$

$$x_{i,j,t,r}, y_{i,t} \in \{0, 1\}$$

Constraint 1 models the fact that for each data need, the data object must be sent to the destination at a time instant between the release time and the deadline. Constraint 2 models the propagation of data objects. It says that if a data object is transmitted from a source j_1 to j_2 at time t for satisfying a data need r, then j_1 must receive the object before t and after the release time. Finally, constraint 3 assigns value 1 to each variable of the kind $y_{i,t}$ if a message is transmitted from the central server to node i at time t in order to satisfy some data need.

Solving this formulation with standard solvers is infeasible on large instances. The main problem is the number of variables and constraints. Even on a sparse network with 100 nodes, 100 encounters per node and 100 data needs, we have hundreds of millions of variables and constraints.

B. A naive approach for the demand cover problem

An improvement on executing the ILP program can be obtained by reducing the problem to Set-cover. Each candidate remote transmission can cover a set of data needs. The minimum set of remote transmissions that covers all the data needs corresponds to the minimum Set-cover in this family of sets. Since remote transmissions can occur at any time, the number of sets for the Set-cover family is huge. However, not all time instants need to be considered. To guarantee that all the data needs are covered, one can consider only time instants that correspond to the release time (the earliest time instant at which the data object needs to be sent for a data need to be satisfied) of a data need. Given a candidate remote transmission, the set of covered data needs can be computed by exploring the space-time graph (e.g., by depth-first search or breadth-first search).

C. A compact graph representation

The naive approach requires exploring a space-time graph, whose size can be huge. However, this graph can be compacted, thanks to two observations. First, in each snapshot graph, all the vertices that are in the same connected component have the same reachability properties, so one vertex can be taken as a representative of all the others. Second, when a transition occurs, connected components of the snapshot graph that do not contain any nodes involved in the transition are not influential.

To generate the *compressed graph*, we focus on two kinds of transitions. A *split transition* causes a connected component to be split into two connected components. A *merge transition* causes two components to merge into a single connected component. We generate a space-time graph considering only these two kinds of transitions. Then, for all snapshot graphs, each connected component is collapsed into one single vertex. At this point, all the edges of the graph are directed and can be classified as follows: (i) *split edges*, which connect components that split with their respective partitions; (ii) *merge edges*, which connect components that merge with the resulting components and (iii) *non-influential edges*, which connect components that do not change. Finally, each non-influential edge is removed by collapsing its endpoints and each vertex is labeled with its lifetime (note that a vertex can span several snapshots), which we call the *component lifetime*.

One example of a compressed graph is shown in Fig. 2, which refers to the example in Fig. 1. Boxes represent vertices of the compressed graph. Edges of the connected graph are represented by solid lines. The extent of a box in time represents the component lifetime. For instance, the extent of the component c_1 is $[t_0, t_{MAX})$ (the whole time horizon) since this component is never involved in any split or merge. The naive approach can be executed on the compressed graph in place of the cumbersome space-time graph. The family of sets for Set-cover is obtained by building a set for each vertex of the compressed graph whose lifetime contains the release time of some data needs. Each set can be computed by exploring the compressed graph.

V. An indexing system for the demand cover problem

Solving the demand cover problem efficiently raises several challenges. First, for each vertex v of the compressed graph, the set of data needs that can be covered by v needs to be retrieved. This operation can be very expensive when the

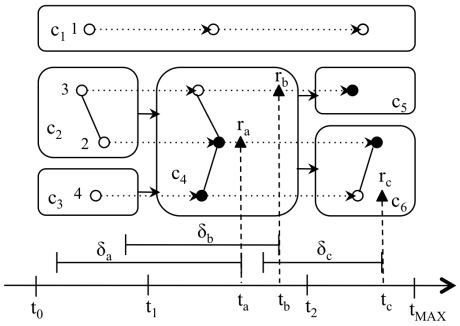


Fig. 2. The compressed graph representation of the example in Fig. 1. A compressed graph is depicted over the space-time graph. Boxes and solid arrows represent vertices and edges of the compressed graph, respectively. The extent of a box in time represents the component lifetime. Three data needs are represented (by filled triangles) with their extent in time. From left to right: $r_a = (2, t_a, \delta_a)$, $r_b = (3, t_b, \delta_b)$, $r_c = (4, t_c, \delta_c)$.

size of the graph 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 is usually very high, since a data need can potentially be covered by many vertices. Many of these sets are redundant, i.e. they are fully contained in other sets. For example, in Fig. 2, the set of data needs covered by c_6 contains only r_c. The vertex c_4 covers the set $\{r_a, r_b, r_c\}$, which contains the data need covered by c_6. Therefore, c_6 can be excluded by the computation since all the data needs that can be covered by it can also be covered by c_4. 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 one to find *maximal sets* [26]. Additionally, in a typical application, a large number of data objects are requested and each data object has its own set of data needs. Solving the demand cover problem for each data object can be extremely expensive.

We propose a novel approach, called *Path-wise indexing* (PIE, for short), which builds an index of the set of trajectories with the purpose of efficiently performing queries of the form: *given a set of data needs, return the minimum set of remote transmissions that covers all the data needs*. 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 data needs) has 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 has several advantages. First, the index is much more compact than the compressed graph, and hence requires less memory and is much more efficiently manageable. Second, the set of vertices in the compressed graph from which the data needs are reachable can be identified fast. Note that current reachability indexes cannot be efficiently applied to our problem since many reachability tests need to be performed. Finally, we can efficiently prune nodes of the compressed graph that are not promising and generate a small instance for Set-cover.

A. Index structure

The key idea behind the index structure is that the set of data needs covered by a node in a path p of the compressed graph includes the set of data needs covered by other subsequent nodes p. Therefore, a node can be taken as a representative of a portion of the path. Moreover, a node of the compressed graph can be uniquely determined by a path and a time instant. This implies that we can use the coverage of a pair (p, t) in place of the coverage of the corresponding compressed node. We denote the coverage of (p, t) as $C(p, t)$. Based on these considerations, we partition the compressed graph into a set of disjoint paths and build a compact graph, named a *PIE graph*, whose vertices represent disjoint paths and edges preserve the connectivity across paths. Each vertex of the PIE graph is labeled with a time interval (namely its *lifetime*) that is the union of the lifetimes of its composing vertices. Edges are labeled with the end of the lifetime of the source nodes in the compressed graph. 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 PIE graph.

Figure 3(a) shows an example (not related to previous figures). The small circles and thin edges form the compressed graph, while the big ovals represent disjoint paths. Consider the path p_3. $\{t_a, t_b\}$ is the set of time instants that are representative for p_3. Indeed, the set of data needs covered by p_3 at time t_a is $C(p_3, t_a) = \{r_1, r_2\}$ (r_3 does not satisfy δ_3). Since no other data needs (i, t, δ) have $(t - \delta) \in [t_a, t_b)$, (p_3, t_a) is representative of the interval $[t_a, t_b)$. t_b coincides with the *release time* of r_3 (i.e. $(t_{r3} - \delta_3)$). Therefore, its coverage ($C(p_3, t_b) = \{r_3\}$) cannot be contained in $C(p_3, t_a)$. The pair (p_3, t_b) is instead representative of the remaining part of the path. The path p_3 produces only two sets ($C(p_3, t_a)$ and $C(p_3, t_b)$) for Set-cover. In general, up to 4 sets would be produced without indexing, since we may have many other non-reachable data needs whose release times fall within all vertices of p_3. Next we describe in details the three steps of our method: preprocessing, filtering and optimization.

B. Preprocessing

Given the set of trajectories, first a compressed graph (G_C) is generated. The graph is then decomposed into a disjoint set of paths. There is a large number of possible ways to partition the graph into disjoint paths. A suitable partition strategy should satisfy the following properties: (i) the number of disjoint paths should be small and (ii) the number of edges across two paths should be small as well. In general, finding the minimum set of disjoint paths that covers a graph is a non-trivial problem [27]. However, since the compressed graph is a DAG and is generated by a simple split-merge model, we can use the following simple and optimal [25] strategy: pick one vertex a time (proceeding in time order) and elongate it by random walk until a vertex without outgoing edges.

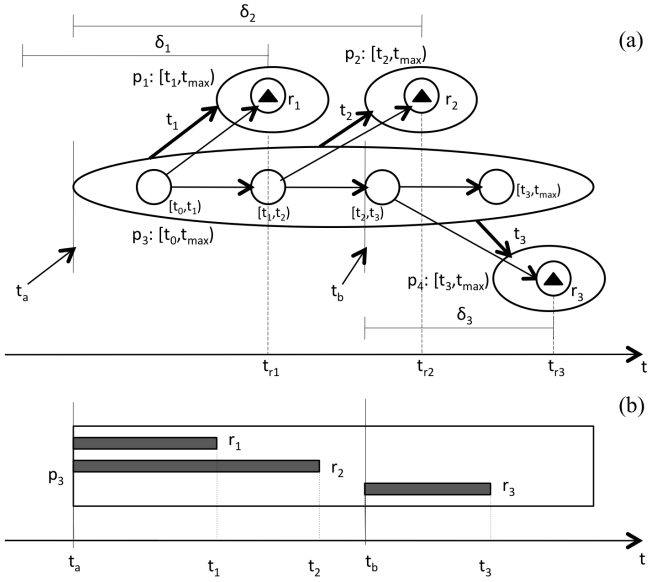


Fig. 3. (a) An example of a PIE graph. The small circles and thin arrows form the compressed graph. Each path is circumscribed by an oval and its lifetime is reported. Links between paths are represented by thick arrows. They are labeled by the ends of the lifetimes of their source vertices. Solid triangles within circles represent data needs. (b) Validity intervals of a set of data needs in a path p_3. Bars represent the extent of validity intervals of data needs. The minimal family of sets for this path is $\{C(p_3, t_a), C(p_3, t_b)\}$.

We can prove that across two paths no more than one edge exists in each direction. Indeed, each edge of the compressed graph comes from a merge or a split between two components. In the case of a merge, the source vertex cannot have other outgoing edges, while in the case of split, the target vertex cannot have other incoming edges. This implies that no edges can exist between internal nodes of two different paths, and hence each edge connecting two paths can be either outgoing from the last vertex of the source path or incoming to the first vertex of the target path. Since the compressed graph is a DAG, and each path is elongated as much as possible, at most two edges can connect two paths, one in each direction.

We associate each edge (p_i, p_j) of the PIE graph with the end of the lifetime of the source vertex in p_i. We denote this time instant as $ft(p_i, p_j)$. It represents the time in which a data object can traverse the edge (p_i, p_j). Fig. 3 depicts an example of PIE graph. The small vertices and thin edges form the compressed graph, while the big vertices and thick edges represent the PIE graph.

C. Filtering

For each vertex p of the PIE graph, our filtering algorithm finds a set of time instants TI_p that are representative of the whole path p, and the family $\mathcal{S}$ of corresponding sets. Our strategy guarantees that the coverage of each vertex of the compressed graph is fully contained in at least one set in $\mathcal{S}$. Since the PIE graph is much smaller than the compressed graph, exploring the former is much more advantageous in terms of running time and memory consumption.

The filtering procedure consists of two steps: *backflow* and *prune*. *Backflow* propagates the data needs in reverse order from the destination paths to all the possible source paths. For each path, we compute the *validity interval* of a data need, which defines the time interval in which the data object must reach the path for the data need to be covered. At the end, each path is associated with a set of data needs that it can cover with their validity intervals. The coverage of a pair (p, t) can be identified by the set of data needs such as their validity intervals in p include t. After the validity intervals are generated, the *prune* procedure computes the family of sets for Set-cover. It collects the family of coverages of representative time instants from each path.

Before describing these two procedures in detail, we first present an example. Fig. 3(b) shows the path p_3 of the example in Fig. 3(a) and the validity interval of each data need in it. The validity intervals of r_1 and r_2 start at the beginning of the path, since their release times precede it. These intervals end at times t_1 and t_2, respectively, times associated with outgoing edges (see Fig. 3(a)). Each of them represents the last time instant at which the data object must leave the path to be able to reach the respective data need. For the data need r_1 (r_2 resp.), if the data object leaves the path after t_1 (t_2 resp.), the destination cannot be reached. The validity interval of r_3 starts at time $t_b = t_{r3} - \delta_3$, corresponding to the release time of r_3, and ends at time t_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 maximal sets of data needs. Therefore, the minimal family of sets for this path is $\mathcal{S} = \{C(p_3, t_a), C(p_3, t_b)\}$. Note that no other time instants have a coverage that is not included in at least one set of the family.

1) Backflow: We define the *validity interval* of a data need $r = (i, t, \delta)$ in a path p (named $valid\_int(r, p)$) recursively in the following way:

If p has lifetime $[b, e)$ and is the destination path of r (i.e. $t \in [b, e)$), we have: $valid\_int(r, p) = [\max(b, t - \delta), t)$.

If p is a non-destination 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:

$$valid\_int(r, p) = \begin{cases} \Phi & \text{if } ft(p, p_i) \notin [b_i, e_i)\ \forall i = 1 \ldots k \\ [t_1, t_2) & \text{otherwise} \end{cases}$$

where $t_1 = \max(b, t - \delta)$ and t_2 is the maximum t' such as $t' = ft(p, p_i)$ for some $i = 1 \ldots k$ and $t' \in [b_i, e_i)$.

Intuitively, the end of a validity interval in a path is given by the last time instant in which the data object can flow in another path that has a compatible validity interval, while the beginning of a validity interval is limited by $t - \delta$ and the starting time of the path.

The coverage of a pair (p, t) can be identified by the set of data needs whose validity intervals include t. Intuitively, if validity intervals are represented by horizontal bars (as in Fig. 3(b)), the coverage of a pair (p, t) can be easily identified by drawing a vertical line and taking all the data needs whose validity intervals are intersected. For instance, in Fig. 3(b) a vertical line drawn at time t_a intersects the validity intervals

of r_1 and r_2. Therefore, the coverage of (p, t_a) is $\{r_1, r_2\}$. This property is formally stated by the following lemma:

Lemma 1: Let (T, I) be an instance of demand cover, where T is the set of trajectories and I is the set of data needs, and G_P be the corresponding PIE graph. Given a vertex p of G_P and a time instant t, the coverage of p at time t is:

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

The interval $valid\_int(r, p)$ can be computed for all paths in a breadth-first search fashion, by starting from the path containing r and exploring the PIE edges in reverse time order until the release time is reached. When a new vertex is visited, the validity interval of r in it is updated. The resulting time complexity is $O(|E_P|)$, where E_P is the set of edges in the PIE graph.

2) Prune: For each path p, we identify the minimum-size set TI_p of time instants that is 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 $t' \in TI_p$. This problem corresponds to the problem of finding the maximal sets in the family of all possible coverage sets of p (i.e. $\{C(p,t)|t \in lifetime(p)\}$).

Figure 4 shows an example path with the validity intervals of five data needs. The coverage of a time instant can be easily identified by drawing a vertical line and taking all the validity intervals that it intersects. The representative time instants for this path are t_1, t_2 and t_3, corresponding to the maximal sets of data needs. Note that no other time instants have a coverage that is not included in the coverage of at least one of the time instants t_1, t_2 or t_3.

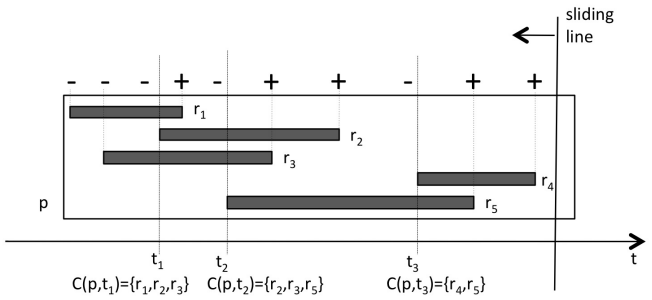


Fig. 4. An example of maximal coverage sets in a path. Bars represent the extent of validity intervals of data needs. The coverage of the time instants t_1, t_2 and t_3 are maximal sets among all the coverage sets in the path. The family of maximal sets can be found by sliding a vertical line in reverse time order and taking each time instant that corresponds to the beginning of a validity interval (indicated by the symbol "-" at the top) that occurs right after the end of the same or another validity interval (indicated by the symbol "+"). This family has minimum size.

In general, the maximal sets can be found in time $O(mn)$, where m is the number of maximal sets and n is the size of the input [26]. In our case, since each element corresponds to a contiguous interval, we can find the maximal sets in linear time. Our procedure slides a vertical line across the path in reverse time order, and takes all the time instants that correspond to maximal sets. Each position t of the line corresponds to a coverage $C(p, t)$. As the line is slid, the coverage is modified, by either adding or deleting data needs. Whenever a deletion follows an addition, the current coverage is taken as a maximal set. Note that additions correspond to the end of validity intervals, while deletions correspond to the beginning of validity intervals. In Fig. 4, the coverage associated with the sliding line is initially empty. When the line intersects the validity interval of r_4, r_4 is added to the coverage (additions are indicated by the symbol "+" at the top). The interval of r_5 is then encountered and r_5 is also added to the coverage. When the beginning of the validity interval of r_4 is encountered (at time t_3), the current coverage is taken as maximal set and r_4 is deleted (indicated by the symbol "-"). Other two additions are then encountered (r_2 and r_3) followed by a deletion (r_5). The coverage at time t_2 (before deleting r_5) is then taken as another maximal set. The last maximal set is taken at time t_1, after another addition and another deletion are encountered. The following lemma states that this procedure finds all and only the maximal sets in the family of coverage sets.

Lemma 2: Let (T, I) be an instance of demand cover and G_P be the PIE graph built from (T, I). Given a path p of G_P, consider the sequence of time instants $t_1, t_2, \ldots, t_k$ corresponding to extremes (beginnings or ends) of validity intervals in reverse time order and the set $TI_p = \{t_i \mid t_i$ is a beginning time and t_{i-1} is an ending time$\}$.

1) For each time instant $t \in lifetime(p)$ we have: $\exists t' \in TI_p \mid C(p,t) \subseteq C(p,t')$;
2) For each time instant $t' \in TI_p$ we have: $\nexists t \in lifetime(p) \mid C(p,t') \subset C(p,t)$;
3) For each pair of distinct time instants $t', t'' \in TI_p$ we have $C(p,t') \neq C(p,t'')$.

A clear consequence of this lemma is that the family of maximal sets generated by our procedure has minimum size. TI_p can be built in time $O(|I| \cdot \log(|I|))$.

D. Optimization

After the filtering process, a *post-pruning* (in short PP) phase is applied in order to remove 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, since redundant sets can occur across different paths. The post-pruning phase can be applied to the naive approach as well.

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

E. Adaptive extension

In real world, it is difficult for many applications to guarantee that moving objects travel with known trajectories over a long time interval. However, it is reasonable to assume that moving objects stick to known traveling plans in near future. In this case, the time dimension is partitioned into discrete time slots, where trajectories of moving objects are updated after each time slot. PIE can adapt to this variation without

much modification. In the following, we briefly introduce two possibilities: *null-initial-state* and *adaptive* extensions.

The most straightforward way is to build an independent index for each time slot. We call it null-initial-state extension because this method simply ignores previous knowledge and treats each time slot as a new start. One weakness of this method is that it neglects information objects transmitted during the prior time slots, producing more remote transmissions. An alternative is to apply the adaptive extension. To reuse data objects transmitted before, we keep track of the distribution of the objects over the nodes, together with the remote transmission time of each object, and use them as initial state for the new time slot. As a result, some data needs can be satisfied without any additional remote transmissions.

In case of deviations from expected trajectories within a time slot, information requests that cannot be served based on the existing schedule can be served by additional remote transmissions.

VI. Experimental Analysis

Here, we describe the datasets, the implementation of our methods, and the experimental results obtained.

A. Dataset

Cabs Mobility [28] (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. The average time interval between two consecutive location updates is less than 10 seconds.

GeoLife GPS Trajectories [29] (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 randomly on a 2-D plane with size 3600 km^2 over 10 days. Starting from a uniformly random position, the speed of each node is updated periodically with normal distribution ($\mu = 1.2$ m/s and $\sigma = 1$) as well as its direction ($\mu =$ current direction and $\sigma = 1$ radiant). The update rate is generated with exponential distribution ($\mu = 60$ sec).

For all datasets, the radio range is set to 100 meters. The data needs are generated by the following process. First, for each moving node, the number of data needs is generated with a Poisson distribution. Then, each data need is generated with a deadline that is uniformly distributed and a latency that is normally distributed ($\mu = 15$ min and $\sigma = 1$).

B. Implementation

We implemented the naive approach described in Sect. IV-B on the space-time graph. We also implemented the naive approach on the compressed graph (called *naive-c* for short) and the PIE indexing system (Sect. V). All methods include the post-pruning phase described in Sect. V-D. We experimented with a version without the post-pruning phase, obtaining a slight degradation of performances in each method. We also implemented the ILP program described in Sect. IV-A, but it did not terminate due to the huge number of variables and constraints (hundreds of billions). All methods were implemented in C++ (Dev C++ IDE ver. 4.9.9.2). The experiments were performed on a DELL Intel core I7 CPU with 2 GB of memory. For the ILP solver, we used *lp_solver* 5.5.2.0 [30], an open source tool based on branch-and-bound.

C. Results

Each dataset is first preprocessed and its PIE index is generated. Fig. 5(a) reports the preprocessing time on CAB, on a number of datasets spanning from 1 to 13 days. Depending on the dataset size, the preprocessing phase takes tens through thousands of seconds. Although the preprocessing phase is sometimes expensive, it is executed only once. The rate of compression of the PIE graph and the compressed graph with respect to the space-time graph is shown in Fig. 5(b). The compressed graph is about four orders of magnitude smaller than the space-time graph and PIE further reduces the size of about three times.

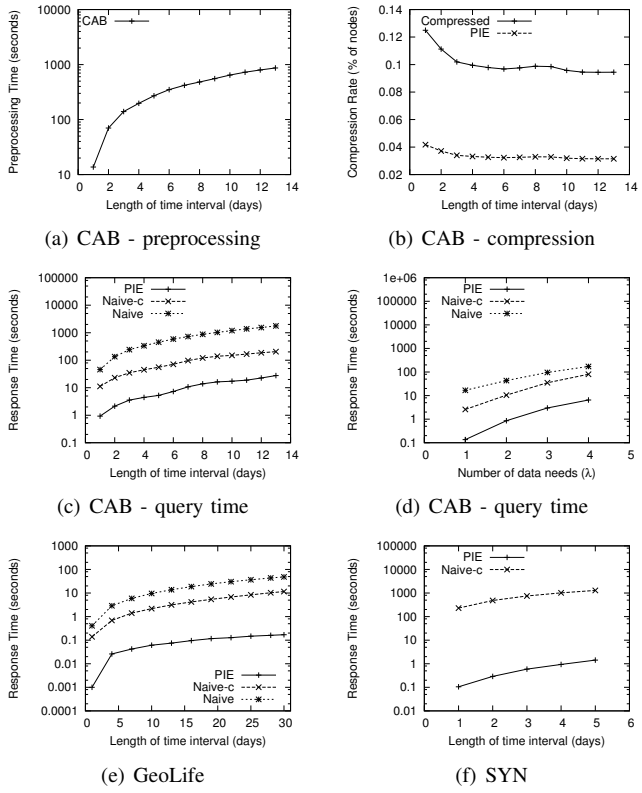
On CAB, the running time for demand cover queries is shown in Fig. 5(c) and 5(d). Fig. 5(c) shows the running time for a number of datasets spanning from 1 to 13 days. The average number of data needs per cab per day is set to 2 (resulting in 1072 data needs per day). The reported times represent an average over 10 queries. PIE performs about one order of magnitude faster than naive-c and two orders of magnitude faster than naive in almost all cases. In order to evaluate the scalability over the size of the query, we generate queries by varying the average number of data needs per cab per day λ from 1 to 4. The results over 1 day are reported in Fig. 5(d). For more than 4 data needs, the naive method is unable to answer queries in acceptable time.

We also execute the adaptive extension (Sect.V-E) on CAB, for one day with time slot 15 minutes. Over a total number of 1089 data needs, null-initial-states method returns 675 remote transmissions, while the adaptive method returns 617 ones, with approximately a 10% improvement. For reference, the number of transmissions suggested by using full knowledge is 480. All the results of the adaptive extension refer to an average over 10 executions.

Fig. 5(e) show the execution time for demand cover queries on GeoLife. The average number of data needs 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 to 30 days. As for CAB, the reported times represent an average over 10 queries. In this dataset, PIE scales better than naive and naive-c with length of the spanning interval. For SYN, the results are reported in Fig. 5(f). They refer to one data need per node per day. The naive approach here is not able to terminate in acceptable time even for one day, therefore we report only PIE and naive-c. PIE performs about three orders of magnitude faster then naive-c. Additional results are provided in our extended technical report [25].

VII. Conclusion

We examined the problem of optimizing the remote communication cost for multicast in DTNs. After formalizing the demand cover problem and showing that it is NP-hard, we



(a) CAB - preprocessing (b) CAB - compression

(c) CAB - query time (d) CAB - query time

(e) GeoLife (f) SYN

Fig. 5. Performances of our method in comparison with naive and naive-c. We report (a) prepocessing time and (b) compression rate for CAB and running time for (c) CAB, (e) GeoLife and (f) SYN. We show the dependence on the number of data needs in (d).

provided a graph-indexing-based solution for it. Our system can solve the demand cover problem optimally on large real instances (dataset with million of events and queries with thousands of nodes) in less than 10 seconds in most cases. We plan to extend our work in two ways. First, we aim to take into account the uncertainty in mobility and data needs. For this, we need to fit stochastic mobility models in our framework and optimize the expected communication cost. Finally, we plan to consider the problem of scheduling new trajectories with the purpose of guaranteeing the connectivity, in the case when the communication with a central data source is not always available.

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.

REFERENCES

[1] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant network," vol. 34, August 2004, pp. 145–158.

[2] W. Zhao, M. Ammar, and E. Zegura, "Multicasting in delay tolerant networks: semantic models and routing algorithms," in *Proc. of WDTN*, 2005, pp. 268–275.

[3] U. Lee, S. Y. Oh, K.-W. Lee, and M. Gerla, "Relaycast: Scalable multicast routing in delay tolerant networks," in *IEEE ICNP*, 2008, pp. 218–227.

[4] W. Gao, Q. Li, B. Zhao, and G. Cao, "Multicasting in delay tolerant networks: a social network perspective," in *Proc. of MobiHoc*, 2009, pp. 299–308.

[5] W. Zhao, M. Ammar, and E. Zegura, "A message ferrying approach for data delivery in sparse mobile ad hoc networks," in *Proc. of MobiHoc*. New York, NY, USA: ACM, 2004, pp. 187–198.

[6] B. K. Polat, P. Sachdeva, M. H. Ammar, and E. W. Zegura, "Message ferries as generalized dominating sets in intermittently connected mobile networks," in *Proc. of the MobiOpp*, 2010, pp. 22–31.

[7] I. F. Akyildiz et.al., "Interplanetary internet: state-of-the-art and research challenges," *Comput. Netw.*, vol. 43, pp. 75–112, October 2003.

[8] Q. Li and D. Rus, "Sending messages to mobile users in disconnected ad-hoc wireless networks," in *Proc. of MobiCom*, 2000, pp. 44–55.

[9] X. Zhang, J. Kurose, B. N. Levine, D. Towsley, and H. Zhang, "Study of a bus-based disruption-tolerant network: mobility modeling and impact on routing," in *Proc. of MobiCom*, 2007, pp. 195–206.

[10] J. Leguay, T. Friedman, and V. Conan, "DTN routing in a mobility pattern space," in *Proc. of WDTN*, 2005, pp. 276–283.

[11] C. Liu and J. Wu, "An optimal probabilistic forwarding protocol in delay tolerant networks," in *Proc. of MobiHoc*, 2009, pp. 105–114.

[12] E. Altman et.al., "Decentralized stochastic control of delay tolerant networks," in *Proc. of INFOCOM*, 2009, pp. 1134–1142.

[13] R. Groenevelt, P. Nain, and G. Koole, "Message delay in manet," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, pp. 412–413, June 2005.

[14] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Efficient routing in intermittently connected mobile networks: the single-copy case," *IEEE/ACM Trans. Netw.*, vol. 16, pp. 63–76, February 2008.

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

[16] S. Merugu, M. Ammar, and E. Zegura, "Routing in space and time in networks with predictable mobility," Georgia Institute of Technology, Tech. Rep., 2004.

[17] V. Borrel, M. H. Ammar, and E. W. Zegura, "Understanding the wireless and mobile network space: a routing-centered classification," in *Proc. of ACM CHANTS*, 2007, pp. 11–18.

[18] A. Faragó and V. R. Syrotiuk, "Merit: A unified framework for routing protocol assessment in mobile ad hoc networks," in *Proc. of MobiCom*. New York, NY, USA: ACM, 2001, pp. 53–60.

[19] S. Bhadra and A. Ferreira, "Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks," *Ad Hoc Networks and Wireless*, pp. 259–270, 2003.

[20] M. Mongiovi et. al., "SIGMA: a set-cover-based inexact graph matching algorithm," *J Bioinform Comput Biol*, vol. 8, no. 2, pp. 199–218, 2010.

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

[22] Y. Chen and Y. Chen, "An Efficient Algorithm for Answering Graph Reachability Queries," in *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 893–902.

[23] R. Jin et. al., "Efficiently answering reachability queries on very large directed graphs," in *SIGMOD*, 2008, pp. 595–608.

[24] H. V. Jagadish, "A compression technique to materialize transitive closure," *ACM Trans. Database Syst.*, vol. 15, pp. 558–598, 1990.

[25] M. Mongiovi, A. K. Singh, X. Yan, B. Zong, and K. Psounis, "Efficient multicasting for delay tolerant networks using graph indexing," http://www.cs.ucsb.edu/research/tech_reports/reports/2011-07.pdf, UC Santa Barbara, Tech. Rep., 2011.

[26] D. M. Yellin, "Algorithms for subset testing and finding maximal sets," in *SODA*, 1992, pp. 386–392.

[27] R. Diestel, *Graph theory*, ser. Graduate texts in mathematics. Springer, 2006.

[28] M. Piorkowski et.al., "CRAWDAD data set (v. 2009-02-24)," http://crawdad.cs.dartmouth.edu/epfl/mobility, Feb. 2009.

[29] "GeoLife GPS Trajectories," http://research.microsoft.com/.

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