Dynamic Shortest Path Algorithms for Hypergraphs

Jianhang Gao, Qing Zhao, Wei Ren, Ananthram Swami, Ram Ramanathan, Amotz Bar-Noy

Dynamic Shortest Path Algorithms for Hypergraphs

Jianhang Gao[†], Qing Zhao[†], Wei Ren[‡], Ananthram Swami[§], Ram Ramanathan[¶], Amotz Bar-Noy[♯]

[†] Department of Electrical and Computer Engineering , University of California, Davis, CA
[‡] Microsoft Corperation, Redmond, WA
[§] Army Research Laboratory, Adelphi, MD
[¶] Raytheon BBN Technologies, Cambridge, MA
[♯] Department of Computer Science, City University of New York, NY

***Abstract*—A hypergraph is a set V of vertices and a set of non-empty subsets of V, called hyperedges. Unlike graphs, hypergraphs can capture higher-order interactions in social and communication networks that go beyond a simple union of pairwise relationships. In this paper, we consider the shortest path problem in hypergraphs. We develop two algorithms for finding and maintaining the shortest hyperpaths in a dynamic network with both weight and topological changes. These two algorithms are the first to address the fully dynamic shortest path problem in a general hypergraph. They complement each other by partitioning the application space based on the nature of the change dynamics and the type of the hypergraph. We analyze the time complexity of the proposed algorithms and perform simulation experiments for both random geometric hypergraphs and the Enron email data set. The latter illustrates the application of the proposed algorithms in social networks for identifying the most important actor based on the closeness centrality metric.**

I. Introduction

A graph is a mathematical abstraction for modeling networks, in which nodes are represented by vertices and pairwise relationships by edges between vertices. A graph is thus given by a vertex set V and an edge set E consisting of cardinality-2 subsets of V. A hypergraph is a natural extension of a graph obtained by removing the constraint on the cardinality of an edge: any non-empty subset of V can be an element (a hyperedge) of the edge set E (see Fig 1). It thus captures group behaviors and higher-dimensional relationships in complex networks that are more than a simple union of pairwise relationships. Examples include communities and collaboration teams in social networks, document clusters in information networks, and cliques, neighborhoods, and multicast groups in communication networks.

[0]This work was supported by the Army Research Laboratory Network Science CTA under Cooperative Agreement W911NF-09-2-0053.

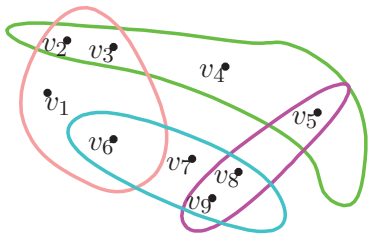

Fig. 1. An example hypergraph with 4 hyperedges: (v_1, v_2, v_3, v_6), (v_2, v_3, v_4, v_5), (v_6, v_7, v_8, v_9), and (v_5, v_8, v_9).

While the concept of hypergraph has been around since the 1920's (see, for example, [1]), many well-solved algorithmic problems in graph remain largely open under this more general model. Here, we address the shortest path problem.

A. The Shortest Path Problem in Graphs

The shortest path problem is perhaps one of the most basic problems in graph theory. It asks for the shortest path between two vertices or from a source vertex to all the other vertices (*i.e.,* the single-source version or the shortest path tree). Depending on whether the edge weights can be negative, the problem can be solved via Dijkstra's algorithm or the Bellman-Ford algorithm.

The dynamic version of the shortest path problem is to maintain the shortest path tree without recomputing from scratch during a sequence of changes to the graph. A typical change to a graph includes weight increase, weight decrease, edge insertion, and edge deletion. The last two types of changes model network topological changes, but they can be conceptually considered as special cases of weight changes by allowing infinite edge weight. Thus, if the sequence of changes contains

only weight increase and edge deletion, we call it a decremental problem; if it contains only weight decrease and edge insertion, we call it an incremental problem. Otherwise, we have a fully dynamic problem. If multiple edges change simultaneously, then it is called a batch problem. Several existing dynamic shortest path algorithms for graphs can be found in [2]–[5].

B. The Shortest Path Problem in Hypergraphs

Both the static and dynamic shortest path problems have a corresponding version in hypergraphs. The static shortest hyperpath problem was considered by Gallo *et al.* [6] in which Dijkstra's algorithm was extended to obtain the shortest hyperpaths. Ausiello *et al.* proposed a dynamic shortest hyperpath algorithm for directed hypergraphs, considering only the incremental problem with the weights of all hyperedges limited to a finite set of numbers [7]. A dynamic algorithm for the batch problem in a special class of hypergraphs was developed in [3].

With the exception of the above few studies, the shortest hyperpath problem remains largely unexplored. To the best of our knowledge, no algorithms exist for the fully dynamic problem in a general hypergraph.

In this paper, we develop two fully dynamic shortest path algorithms for general hypergraphs. These two algorithms complement each other, with each preferred in different types of hypergraphs and dynamics.

Referred to as the HyperEdge based Dynamic Shortest Path algorithm (HE-DSP), the first algorithm is an extension of the dynamic Dijkstra's algorithm for graphs to hypergraphs (parallel to Gallo's extension of the static Dijkstra's algorithm to hypergraphs in [6]). The extension of the dynamic Dijkstra's algorithm to hypergraphs is more involved than that of the static Dijkstra's algorithm. This is due to the loss of the tree structure (in the original graph sense) in the collection of the shortest hyperpaths from a source to all other vertices. Since the dynamic Dijkstra's algorithm relies on the tree structure to update the shortest paths after an incremental change, special care needs to be given when extending it to hypergraphs.

The second algorithm is rooted in the idea of Dimension Reduction and is referred to as DR-DSP. The basic idea is to reduce the problem to finding the shortest path in the underlying graph of the hypergraph. The underlying graph of a hypergraph has the same vertex set and has an edge between two vertices if and only if there is at least one hyperedge containing these two vertices in the original hypergraph. The weight of an edge in the underlying graph is defined as the minimum weight among all hyperedges containing the two vertices of this edge. The shortest hyperpath in the hypergraph can thus be obtained from the shortest path in the underlying graph by substituting each edge along the shortest path with the hyperedge that lent its weight to this edge. The correctness and advantage of this algorithm are readily seen: the definition of weight in the underlying graph captures the minimum cost offered by all hyperedges in choosing a path between two vertices, thus ensuring the correctness of the algorithm; the reduction of a hypergraph to its underlying graph removes many hyperedges from consideration when finding the shortest path, leading to efficiency and agility to dynamic changes.

HE-DSP is more efficient in hypergraphs that are densely connected through high-dimensional hyperedges and for network dynamics where changes often occur to hyperedges that are not on the current shortest hyperpaths. DR-DSP has lower complexity when hyperedge changes often lead to changes in the shortest hyperpaths. This is usually the case in networks where hyperedges in the shortest hyperpaths are more prone to changes due to attacks, frequent use, or higher priority in maintenance and upgrade. While we focus on undirected hypergraphs in the paper, the two algorithms apply to directed hypergraphs with minor modifications in their implementation details.

C. Applications

Shortest path computations on hypergraphs can be applied to communication as well as social networks. An example application in wireless communications, in particular, for multihop wireless networks, is in *opportunistic routing* schemes such as ExOR [8], GeRaF [9], and MORE [10]. In such schemes, any receiver of a packet is eligible to forward the packet. Receivers typically execute a protocol amongst themselves to decide who should forward it. This naturally leads to a hypergraph model where a node and its neighbors form a hyperedge. The cost of each hyperedge can be based on the cardinality of the hyperedge to capture the success rate of forwarding (lower the cardinality, lesser the chance that at least one of the nodes successfully receives the packet) and the associated overhead (higher the cardinality, higher the energy consumption and the overhead in choosing the

forwarding node). A shortest hyperpath from the source to the destination is thus a better route than merely the traditional shortest path. And as the network topology changes, a dynamic algorithm is required to maintain the shortest hyperpath.

In social networks, information (results, event reports, opinions, rumors, *etc.*) propagates through diverse communication means including direct links (e.g., gestures, optical, satcom, regular phone call, e-mail), social media (e.g., Facebook, Twitter, blogs), mailing lists, and newsgroups. Such a network may be modeled as a hypergraph with the weight of a hyperedge reflecting the cost, credibility, and/or delay for disseminating information among all vertices of this hyperedge. In particular, the weight of a hyperedge can capture the unique effect on the information after it passes through a group of people. For instance, a result can be discussed by overlapping blog collaboration networks as it spreads, and often the discussion yields a better result than if it only spreads through individuals. The minimum cost information-passing in social networks can thus be modelled as a shortest hyperpath problem.

Another potential application is that of finding the most important actor in a social network. Under a graph model of social networks, the relative importance of a vertex can be measured by its betweenness and closeness centrality indices. The former is defined based on the number of shortest paths that pass through this vertex, and the latter, the total weight of the shortest paths from this vertex to all the other vertices [11]. In a social network exhibiting hyper-relationships, betweenness and closeness centrality, based on the shortest hyperpaths, would be better indicators of the relative importance of each actor. In Sec. VI, we apply the proposed shortest hyperpath algorithms to the Enron email data set. We propose a weight function that leads to the successful identification of the CEO of Enron as the most important actor under the closeness centrality metric. The distance of each person in the data set to the CEO along the resulting shortest hyperpaths closely reflects the position of the person within the company.

II. Dynamic Shortest Hyperpath Problem

We introduce some basic concepts of hypergraph and define the static and the dynamic shortest hyperpath problems. Some basic properties of the shortest hyperpaths are established and will be used in developing the dynamic algorithms in subsequent sections.

A. Hypergraph and Hyperpath

Let V be a finite set and E a family of subsets of V. If for all elements $e_i \in E$, the following conditions are satisfied:

$$e_i \neq \emptyset, \qquad \cup_{e_i \in E}\ e_i = V,$$

then the couple $H = (V, E)$ is called a *(undirected) hypergraph*. Each element $v \in V$ is called a *vertex* and each element $e \in E$ a *hyperedge*.

A *weighted undirected hypergraph* is a triple $H = (V, E, w)$ with $w : E \rightarrow \{R^+ \cup \{0\}\}$ being a nonnegative weight function defined for each hyperedge in E.

In a hypergraph, a hyperpath is defined as follows.

Definition 1: A *hyperpath* between two vertices u and v is a sequence of hyperedges $\{e_0, e_1, \ldots, e_m\}$ such that $u \in e_0$, $v \in e_m$, and $e_i \cap e_{i+1} \neq \emptyset$ for $i = 0, ..., m-1$. A hyperpath is *simple* if non-adjacent hyperedges in the path are non-overlapping, *i.e.,* $e_i \cap e_j = \emptyset, \forall j \neq i, i \pm 1$.

Let $L_e = \{e_0, \ldots, e_m\}$ be a hyperpath in a weighted hypergraph H. We define the weight of L_e as:

$$w(L_e) = \sum_{i=0}^{m} w(e_i).$$

B. Shortest Hyperpath and Relationship Tree

Given two vertices u and v, a natural question is to find the shortest hyperpath (in terms of the path weight) from u to v. Since the weight function is nonnegative, it suffices to consider only simple hyperpaths. If the shortest hyperpath is not simple, we can always generate a simple hyperpath without increasing the weight by deleting all the hyperedges between two overlapping non-adjacent hyperedges.

The dynamic shortest hyperpath problem can be similarly defined for a sequence $C = \{\delta_1, \delta_2, \ldots, \delta_l\}$ of hyperedge changes. Hyperedge changes are of the same four types as edge changes in a graph: weight increase, weight decrease, hyperedge insertion, and hyperedge deletion. Similarly, weight increase and hyperedge deletion will be treated together, so will weight decrease and hyperedge insertion.

In this paper, we consider the single-source shortest hyperpath problem: find the shortest hyperpaths from a given source s to all other vertices. The presentation of the paper focuses on undirected hypergraphs. However, the two proposed dynamic algorithms apply to directed hypergraphs with minor modifications in their implementation details.

Below, we establish a basic property of shortest hyperpaths.

Lemma 1: Let $L = \{e_1, e_2, \ldots, e_l\}$ be a shortest hyperpath from $s \in e_1$ to $z \in e_l$. Then for any vertex $v \in e_i \cap e_{i+1}$, the hyperpath $L_v = \{e_1, e_2, \ldots, e_i\}$ is a shortest hyperpath from s to v. Furthermore, for any two vertices $u, v \in e_i \cap e_{i+1}$ (if there exist at least two vertices in $e_i \cap e_{i+1}$), $D[u] = D[v]$.

Proof: See [12]. ■

Next, we introduce the concept of relationship tree that is needed in the proposed dynamic shortest hyperpath algorithm HE-DSP. Since two adjacent hyperedges in a hyperpath may overlap at more than one vertex, the shortest hyperpaths from s to all other vertices do not generally form a tree in the original graph sense. For the development of the dynamic shortest hyperpath algorithms, we introduce the concept of *relationship tree* to indicate the parent-child relationship along shortest hyperpaths. The concept can be easily explained in the example given in Fig 2. Let $\{e_1, e_2\}$ be a shortest hyperpath from s to v_4. By Lemma 1, $\{e_1\}$ is a shortest hyperpath for both v_1 and v_2. As illustrated in Fig 2, there are 4 possible relationship trees to indicate the parent-child relationship in these shortest hyperpaths. We will show in Sec. III that the choice of the relationship tree does not affect the correctness or performance of the proposed algorithm HE-DSP.

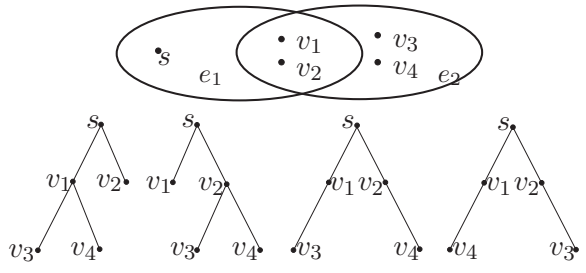

Fig. 2. Hyperpaths and the associated relationship trees.

In the following, $D[v]$ denotes the distance of a vertex v to the source s on the shortest hyperpath, $P[v]$ the parent of v in the chosen relationship tree associated with the shortest hyperpaths, and $E[v]$ the hyperedge containing v and $P[v]$ on the shortest hyperpath (*i.e.,* the hyperedge that leads to v from $P[v]$ on the shortest hyperpath). A vertex v is called an affected vertex if $D[v]$, $P[v]$, and/or $E[v]$ change in the new set of shortest hyperpaths. A hyperedge is called an affected edge if it contains an affected vertex. When it is necessary to distinguish the shortest distance before and after a weight change, $d[v]$ will denote the shortest distance before the change, $d'[v]$ the shortest distance after the change, and $D[v]$ the actual value stored in the data structure during the execution of the algorithm.

III. Hyperedge Based Dynamic Shortest Path (HE-DSP) Algorithm

In this section, we propose HE-DSP. It is an extension of the dynamic Dijkstra's algorithm to hypergraphs. The extension is more complex than Gallo's extension of the static Dijkstra's algorithm, since the dynamic Dijkstra's algorithm for graphs relies on the tree structure of the shortest paths, a structure no longer there for the shortest hyperpaths.

A. Hyperedge Weight Decrease

For weight decrease, consider that the weight of a hyperedge e decreases to w_{new}. It can be shown that the vertex $u \in e$ with $D[u] = \min_{v \in e}\{D[v]\}$ will not be affected. We then check whether the other vertices in e are affected by checking whether the following inequality holds:

$$D[u] + w_{new} < D[v]. \quad (1)$$

We then put all the affected vertices into a priority queue[1] Q. The rest of the procedure is similar to the dynamic Dijkstra's algorithm, except that when we update the distance of a vertex, we check all the hyperedges that contain this vertex. A detailed implementation of the algorithm can be found in [12].

Theorem 1: If before the weight decrease, $D[v] = d[v]$, $E[v]$ and $P[v]$ are correct for all $v \in V$, then after the weight decrease, $D[v] = d'[v]$, and $E[v]$ and $P[v]$ are correctly updated.

Proof: See [12]. ■

B. Hyperedge Weight Increase

To handle hyperedge weight increase, special care needs to be given when finding all the affected vertices. This process in the graph case relies on the tree structure of the shortest paths which no longer exists in shortest hyperpaths. Our solution is to use a *relationship tree* as introduced in Sec. II.

[1]A priority queue is an abstract data type with the following access protocol: only the highest-priority element can be accessed. Basic operations of a priority queue include Enqueue (add a new item to the queue), Dequeue (remove the item with the highest priority and return this item), Update (change the priority of one item in the queue), and Peek (obtain the value of the item with the highest priority). Standard implementations of a priority queue with different time complexities include array, link list, Binary heap, and Fibonacci heap [13].

Consider that the weight of a hyperedge e increases to w_{new}. If e is not an edge in any of the current shortest hyperpaths, then none of the vertices will be affected, all the shortest hyperpaths remain unchanged. Otherwise, the descendants, and only the descendants of this edge in the current relationship tree may be affected. For these vertices, some of them will have increased distances, some of them will go through an alternative path with the same distance (but changed parent or hyperedge), while the rest will not be affected. In order to classify the vertices into these three categories, we propose the following coloring process which is a modification of the coloring idea in Frigioni's algorithm for graphs [4] to take care of the non-unique choice of parent in hypergraph.

(1) v is colored *white* if $d'[v] = d[v]$ while keeping the current $P[v]$ and $E[v]$.
(2) v is colored *pink* if $d'[v] = d[v]$, but only possible through a new $P[v]$ or $E[v]$ or both.
(3) v is colored *red* if $d'[v] < d[v]$.

It can be shown that if a vertex x is white or pink, all its descendants in the relationship tree are white; if x is red, all its descendants are either red or pink. Therefore we have the following coloring procedure. We first determine the color of each vertex in e. Specifically, the vertex $u \in e$ with $D[u] = \min_{v \in e}\{D[v]\}$ will not be affected and will be colored white. Any other vertex (say v) in e will be either pink or red, which we can determine by checking whether there is an alternative shortest hyperpath with the same distance for v (note that v cannot be white due to the weight change of hyperedge e that is on its current shortest hyperpath); if such a path exists, then we color v pink, otherwise we color it red and put all its children in a priority queue M. The procedure then iterates for each vertex in M according to an increasing order of the vertex distances.

After the coloring process, we only need to deal with the red vertices. For each red vertex z, we initialize its distance with the distance of the shortest path through one of its non-red neighbors and put z in another priority queue Q (if no non-red neighbor exists, we initialize it with ∞). We then iterate by extracting the vertex at the top of Q and updating its neighbors and Q until Q is empty.

A detailed implementation of the algorithm can be found in [12]. The theorem below states the correctness of the algorithm.

Theorem 2: If before the weight increase, $D[v] = d[v]$, $E[v]$ and $P[v]$ are correct for all $v \in V$, then after the weight increase, $D[v] = d'[v]$, and also $E[v]$ and $P[v]$ are correctly updated regardless of the choice of the relationship tree.

Proof: See [12]. ■

IV. Dimension Reduction based Dynamic Shortest Path (DR-DSP) Algorithm

In this section, we propose DR-DSP. When the dynamic problem degenerates to the static problem, DR-DSP leads to an alternative algorithm for solving the static shortest hyperpath problem.

A. The Static Case: DR-SP

We first consider the static version of the algorithm (referred to as DR-SP), which captures the basic idea of dimension reduction.

The proposed DR-SP algorithm is based on the following theorem in which we show that for a general hypergraph H, the weight $\omega(L^*)$ of the shortest path L^* of H is equal to the shortest path L^*_G of a weighted graph G derived from H. Specifically, corresponding to every hyperedge e in H, G contains a clique defined on the vertices of e.

Theorem 3: Let $H = (V, E, w)$ be a hypergraph, and $G = (V, \tilde{E})$ the underlying graph of H where an edge $\tilde{e} \in \tilde{E}$ if and only if $\exists e \in E$ such that $\tilde{e} \subset e$. For each edge $\tilde{e}$ in G, its weight $w_G(\tilde{e})$ is defined as

$$w_G(\tilde{e}) = \min_{\{e \in E:\ e \supseteq \tilde{e}\}} w(e). \tag{2}$$

Let L^* and L^*_G be the shortest paths from $u \in V$ to $v \in V$ in H and G, respectively. Then we have that

$$w(L^*) = w_G(L^*_G).$$

Proof: See [12]. ■

It follows from Theorem 3 that the shortest path in a general hypergraph can be obtained by applying Dijkstra's algorithm to the underlying graph G with weights modified as stated in the theorem.

B. The Dynamic Case: DR-DSP

In the dynamic case, a sequence $C = \{\delta_1, \delta_2, \ldots, \delta_l\}$ of hyperedge changes in the hypergraph H results in a sequence of edge changes in the underlying graph G. For each hyperedge change δ_i, DR-DSP first updates the underlying graph G to locate all the changed edges caused by δ_i. In the next step, DR-DSP updates the shortest path tree in the underlying graph G.

Consider first the graph update. A change to a hyperedge e only affects those edges in G that are subsets of e, *i.e.,* a hyperedge change is localized in the underlying graph G. Furthermore, since the weight of an edge in G is the minimum weight of all hyperedges containing it, not all edges in G that are subsets of e will change weight. Based on these observations, we propose a special data structure and procedure for updating the underlying graph G without regenerating the graph from scratch. Specifically, at the initialization stage of the algorithm, a priority queue M_{uv} for each pair of vertices (u, v) in the hypergraph is established to store the weights of all hyperedges that contains both u and v. When a change occurs to hyperedge e, all the priority queues M_{uv} associated with the pair of vertices (u, v) that are contained in e are updated with the new weight of e. Thus, the top of these priority queues always maintain the weight for edge (u, v) in the underlying graph G for each (u, v). A detailed implementation of the proposed procedure is given in [12].

After the underlying graph G has been updated, we now face a dynamic shortest path problem in a graph. However, since a single hyperedge change can result in multiple edge changes in G, we need to handle a batch problem. While existing batch algorithms and iterative single-change algorithms for graphs can be directly applied here, we show that the batch problem we have at hand has two unique properties that can be exploited to improve the efficiency of the algorithm.

Property 1: The edge changes in G caused by a hyperedge change are either all weight decreases or all weight increases.

Property 2: All changed edges in G caused by a hyperedge change belong to a clique in G.

In the case of weight decrease, if the weight of hyperedge e decreases to w_{new} , by Theorem 3 and Property 1, there are (possibly) several edge-weight decreases in the underlying graph G. We know that there is at least one unaffected node $x = \text{argmin}_{v \in e}\{D[v]\}$. By Property 2, these affected edges are contained in a clique derived from the changed hyperedge; therefore it is sufficient to determine the distance of every node v (other than x) in the original changed hyperedge e by checking whether $D[x]+w_{new} < D[v]$. And we can initialize the priority queue with those nodes whose weights decrease. After that, the procedure is similar to that in the graph case.

In the case of weight incresae, if the weight of hyperedge e increases to w_{new}, by Theorem 3 and Property 1, there are (possibly) several edge-weight increases in the underlying graph G. Similar to the single-change case in graph, there is at least one unaffected node $x = \text{argmin}_{v \in e}\{D[v]\}$. Then another node $v \in e$ is affected only if $E[v] = e$, *i.e., e* is on its shortest hyperpath. We use all such nodes to initialize the priority queue M. The rest is similar to that in the graph case.

Detailed implementations of DR-DSP are given in [12].

V. Time Complexity Analysis

Given a hypergraph $H = (V, E, w)$ and a change to hyperedge $\check{e}$, let $|\delta|$ denote the number of affected vertices, $|\delta_\Phi| = \sum_{e \in E, e \text{ is affected}} |e|^2$, m the number of edges in the underlying graph, and $\|\tilde{\delta}\|$ the number of affected edges in the underlying graph plus $|\delta|$.

Theorem 4: The time complexities of HE-DSP and DR-DSP are $O(|\delta| \log |\delta| + |\delta_\Phi|)$ and $O(|\delta| \log |\delta| + \|\tilde{\delta}\| + |\check{e}|^2 \log m)$, respectively.

Proof: See [12]. ■

From Theorem 4 we see that if δ is small and $|\check{e}|$ is large, HE-DSP performs better, since in DR-DSP, the update of the underlying graph has to be done regardless of whether there are affected vertices. Thus in a sequence of hyperedge changes, if only a small fraction of them actually have affected nodes, then HE-DSP will outperform DR-DSP. On the other hind, if δ is large, DR-DSP will outperform HE-DSP since usually $|\delta_\Phi| \gg \|\tilde{\delta}\|$.

VI. Simulation Examples

We present simulation results on the running time of the proposed dynamic shortest hyperpath algorithms. We test the proposed algorithms on hypergraphs generated from a random geometric model as well as those generated by the Enron email data set. All simulation code is compiled and run on the same laptop equipped with a 3.0GHz i7-920XM Mobile Processor.

A. Random Geometric Hypergraph

We first consider a random geometric hypergraph model in which n nodes are uniformly distributed in an $a \times a$ square. All nodes within a circle with radius r form a hyperedge (circles are centered on a $h \times h$ grid). The weight of each hyperedge is given by the average distance between all pairs of vertices of this hyperedge.

A sequence of changes are then generated and the proposed dynamic algorithms are employed to maintain all the shortest hyperpaths from the

source s located at a corner of the $a \times a$ square. Each change can be a hyperedge insertion (with probability p_I), a hyperedge deletion (with probability p_D), or a weight change (with probability $1-p_I-p_D$) with new weight chosen uniformly in $[w_{min}, w_{max}]$. In the case of a hyperedge deletion or a weight change, the hyperedge to be deleted or to be assigned a new weight is chosen according to the two models detailed below. Hyperedge insertions are only *realized* when there are hyperedges that have been deleted, and a randomly chosen one is inserted back. This ensures that all hyperedges satisfy the geometric property determined by r at all time. It also models the practical scenario where a broken link is repaired.

In selecting a hyperedge for deletion or weight change, we consider two different models: the random change model and the targeted change model. In the former, the hyperedge is randomly and uniformly chosen among all hyperedges. In the latter, it is randomly and uniformly chosen from the current shortest hyperpaths. This models the scenarios where hyperedges in the shortest hyperpaths are more prone to changes due to attacks, frequent use, or higher priority in maintenance and upgrade.

In Fig. 3, we show the simulation results on the running time of the two proposed algorithms under a sequence of 10^4 changes. We see that HE-DSP has lower complexity in networks with random topological and weight changes (Fig. 3-Left), whereas DR-DSP should be preferred in networks with targeted changes (Fig. 3-Right). This partition of the application space can be explained from the structures of these two algorithms. Under the random change model, a large fraction of changes does not result in changes in the current shortest hyperpaths. Such changes lead to little computation in maintaining the shortest hyperpaths for both algorithms, but require about the same amount of computation in the Graph-Update step of DR-DSP for maintaining the underlying graph. On the other hand, under the targeted change model, all hyperedge deletions and weight changes affect the shortest hyperpaths. Updating the shortest hyperpaths can be done more efficiently in DR-DSP since it works on the underlying graph with a much smaller number of edges.

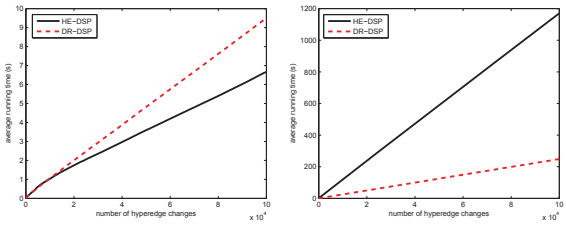

Fig. 3. Average running time. Left: the random change model; Right: the targeted change model ($n = 1000$, $a = 1000$, $r = \sqrt{1000}$, $h = 1$, $p_I = \frac{1}{4}$, $p_D = \frac{1}{4}$, $w_{\min} = 10$, $w_{\max} = 20$, the average is taken over 50 random hypergraphs).

B. Enron Email Data Set

In this example, we consider the application of the shortest hyperpath algorithms in finding the most important actor in a social network. We consider the Enron email data set and use the same hypergraph generation model as in [14]. Specifically, each person is a vertex of the hypergraph, and the sender and recipients of every email form a hyperedge. Our objective is to identify the most important person measured by the closeness centrality index (*i.e.,* the total weight of the shortest hyperpaths from this person to all the other persons). The first step is to assign weight to each hyperedge that reflects "distance". While there is no universally accepted way of measuring distance in a social network observed through email exchanges, certain general rules apply. First, a direct email exchange between two persons indicates a stronger tie than an email sent to a large group. Thus, the weight of an hyperedge should be an increasing function of the cardinality of this hyperedge. Second, more frequent email exchange among a given group of people shows stronger ties. Thus, the weight of an hyperedge should be decreasing with the number of times that this hyperedge appears in the email data set. Considering these two general rules, we adopt the following weight function:

$$w(e) = (\sqrt{|e|})^{\alpha^{(l-1)}} \tag{3}$$

where $|e|$ is the cardinality of the hyperedge e, α is the parameter measuring how fast the weight decreases with the number l of times that this hyperedge appears in the data set.

We can then apply DR-SP on the resulting (static) hypergraph to find the shortest hyperpaths rooted at each vertex and compute this vertex's closeness centrality index. With the weight function given in (3) using $\alpha = 0.6$, the identified most important actor is the CEO of Enron. The average distance (along the shortest hyperpath) from the CEO to other persons at various positions is shown in Fig. 4. We observe that in general, the higher the position, the shorter the distance. These results demonstrate that the adopted hypergraph model and weight function capture the essence of the problem.

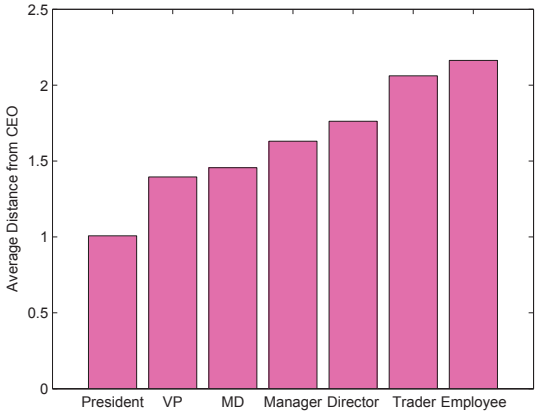

Fig. 4. The average distance from the CEO to others at different positions.

Next, we construct a dynamic hypergraph sequence based on the Enron data set. At the beginning, the hypergraph contains only individual vertices. We then consider each email chronologically. Each email either adds a new hyperedge or decreases the weight of an existing hyperedge (due to the increased number of appearances of this hyperedge). The two proposed algorithms are employed to maintain the shortest hyperpaths rooted at the CEO after each change. The running time is given in Fig. 5, which shows the lower complexity of DR-DSP. The reason is that a large fraction of hyperedge changes result in changes in the shortest hyperpaths.

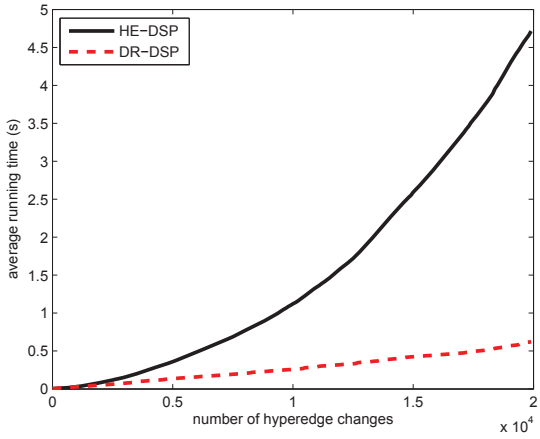

Fig. 5. The average running time for the Enron data set ($\alpha = 0.6$, the average is taken over 50 monte carlo runs).

VII. Conclusion

We have presented the first study of the fully dynamic shortest path problem in a general hypergraph. We have developed two dynamic algorithms for finding and maintaining the shortest hyperpaths. These two algorithms complement each other with each one preferred in different types of hypergraphs and network dynamics, as illustrated in the time complexity analysis and simulation experiments. We have discussed and studied via experiments over a real data set the potential applications of the dynamic shortest hyperpath problem in social and communication networks.

References

[1] C. Berge, *Graphs and hypergraphs*. North-Holland Pub. Co., 1976.

[2] G. Ramalingam and T. Reps, "On the computational complexity of dynamic graph problems," *Theoretical Computer Science*, vol. 158, no. 1-2, pp. 233–277, 1996.

[3] G. Ramalingam and T. Reps, "An incremental algorithm for a generalization of the shortest-path problem," *J. Algorithms*, vol. 21, no. 2, pp. 267–305, 1996.

[4] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni, "Fully dynamic algorithms for maintaining shortest paths trees," *Journal of Algorithms*, vol. 34, no. 2, pp. 251–281, 2000.

[5] P. Narvaez, K. Siu, and H. Tzeng, "New dynamic algorithms for shortest path tree computation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 6, pp. 734–746, 2000.

[6] G. Gallo, G. Longo, S. Nguyen, and S. Pallottino, "Directed hypergraphs and applications," *Discrete Applied Mathematics*, vol. 42, no. 2-3, pp. 177–201, April 1993.

[7] G. Ausiello, U. Nanni, and G.F. Italiano, "Dynamic maintenance of directed hypergraphs," *Theoretical Computer Science*, vol. 72, no. 2-3, pp. 97–117, 1990.

[8] S. Biwas, R. Morris, "EXOR: Opportunistic multi-hop routing for wireless networks," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 133–144, 2005.

[9] M. Zorzi and R. Rao, "Geographic random forwarding (geraf) for ad hoc and sensor networks: multihop performance," *IEEE Transactions on Mobile Computing*, pp. 337–348, 2003.

[10] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, Oct. 2007.

[11] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*. New York: Cambridge University Press, 1997.

[12] J. Gao, Q. Zhao, W. Ren, A. Swami, R. Ramanathan, A. Bar-Noy, "Dynamic Shortest Path Algorithms for Hypergraphs," available at http://arxiv.org/abs/1201.4906.

[13] N.B. Dale, *C++ Plus Data Structures,* Jones and Bartlett Publishers Inc., 2006.

[14] D.M.Y. Park, C.E. Priebe, D.J. Marchette, "Scan statistics on enron hypergraphs," in *Interface*, 2008.