MANCHESTER 1824

The University of Manchester

The Right Way to Search Evolving Graphs

Chen, Jiahao and Zhang, Weijian

2016

MIMS EPrint: **2016.7**

ISSN 1749-9097

The Right Way to Search Evolving Graphs

Jiahao Chen
Computer Science and Artificial Intelligence Laboratory,
Massachusetts Institute of Technology,
Cambridge, Massachusetts, 02139-4307, USA
jiahao@mit.edu

Weijian Zhang
School of Mathematics,
University of Manchester,
Manchester, M13 9PL, England, UK
weijian.zhang@manchester.ac.uk

***Abstract*—Evolving graphs arise in many different contexts where the interrelations between data elements change over time. We present a breadth first search (BFS) algorithm for evolving graphs that can track (active) nodes correctly. Using simple examples, we show naïve matrix-matrix multiplication on time-dependent adjacency matrices miscounts the number of temporal paths. By mapping an evolving graph to an adjacency matrix of the equivalent static graph, we prove the properties of the BFS algorithm using the properties of the adjacency matrix. Finally, demonstrate how the BFS over evolving graphs can be applied to mining citation networks.**

***Keywords*- Evolving graph; complex network; breadth first search; data mining.**

I. Introduction

Let's imagine a game played by three people, numbered 1, 2, and 3, each of whom has a message, labeled a, b, and c respectively. At each turn, one particular player is allowed to talk to one other player, who must in turn convey all the messages in his or her possession. The goal of the game is to collect all the messages. Suppose 1 talks to 2 first, and 2 in turn talks to 3. Then, 3 can collect all the messages even without talking to 1 directly. However, if 2 talks to 3 before 1 talks to 2, then 3 can never get a.

We can analyze the spread of information between the players using graph theory. In this process, the time ordering of events matters, and hence its graph representation $G(t) = (V(t), E(t))$ must be time dependent. Such a graph is called an "evolving graph" [1], [2], "evolving network" [3] or "temporal graph" [4].

Treatments of evolving graphs vary in their generality and focus. Kivelä *et al.* [5] treat time dependence as a special case of families of graphs with multiple interrelationships. Others like Flajolet *et al.* [1] use time to index the family of related graphs, but are not concerned with explicit time-dependent processes. Yet others focus on incremental updates to large graphs [2]. Here, we describe evolving graphs as a time-ordered sequence of graphs, similar to the study of metrized graphs by Tang and coworkers [4], [6]–[8] and of the dynamics of communities by Grindrod, Higham and coworkers [9], [10].

The game described in the beginning can be encoded in an evolving graph. The spread of information to the winner can be described in terms of traversing this graph using discrete paths that step in both space and time. Traversals of an ordinary (static) graph may be computed using well known methods such as the breadth-first search (BFS). An informal description of BFS generalized to evolving graphs can be found in [4]. However, it turns out that naïve extensions can lead to incorrect descriptions of the resulting graph traversals. A proper treatment requires the notion of *node activeness* to describe the set of paths that can only traverse time or edges, which we call *temporal paths*. As a result, our treatment can be applied to any evolving graph, even those that are highly dynamic with arbitrary changes to the nodes and edges.

In Section II, we explain how the BFS algorithm can be applied correctly on evolving graphs. Section II-A provides an example showing that considering only products of the time-dependent adjacency matrices can lead to incorrect enumeration of temporal paths. We present and demonstrate the BFS algorithm over evolving graphs in Section II-B, showing that it is formally equivalent to BFS over a particular static graph generated by connecting together active notes of the evolving graph. This static graph generates an algebraic representation of the BFS as power iteration of its adjacency matrix to a starting search node, as shown in Section III-D. The algebraic formulation also demonstrates interesting connections between properties of the BFS algorithm and the adjacency matrix. Finally in Section V, we explain how BFS on evolving graphs may be applied to study dynamical processes over citation networks.

II. Breadth-First Search over Evolving Graphs

A. Temporal Paths over Active Nodes

The key new idea in generalizing BFS to evolving graphs is to be able to compute paths that evolve forward in time and can only traverse the node space along existing edges. We call these paths *temporal paths*.

Figure 1 shows a small example of an evolving directed graph, $G_3 = \langle G^{[1]}, G^{[2]}, G^{[3]} \rangle$, consisting of a sequence of three graphs $G^{[i]}$, each bearing a time stamp t_i. There are directed edges $1 \to 2$ at time t_1, $1 \to 3$ at time t_2, and $2 \to 3$ at time t_3. Each edge exists only at a particular discrete time

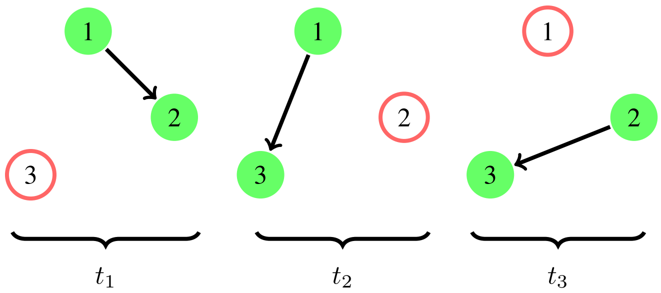


Figure 1. An evolving directed graph with 3 time stamps t_1, t_2 and t_3. At each time stamp, the evolving graph is represented as a graph. The green filled circles represent active nodes while the red circles represent inactive nodes. Directed edges are shown as black errors.

and the nodes connected by edges are considered *active* at that time.

Temporal paths connect only active nodes in ways that respect time ordering. Thus the sequences $\langle(1,t_1),(1,t_2),(3,t_2),(3,t_3)\rangle$ and $\langle(1,t_1),(2,t_1),(2,t_3),(3,t_3)\rangle$ are both examples of *temporal paths* from $(1,t_1)$ to $(3,t_3)$, which are drawn as dotted lines with arrowheads in Figure 2. However, $\langle(1,t_1),(1,t_2),(2,t_2),(3,t_2),(3,t_3)\rangle$ is not a temporal path because node 2 is inactive at time t_2.

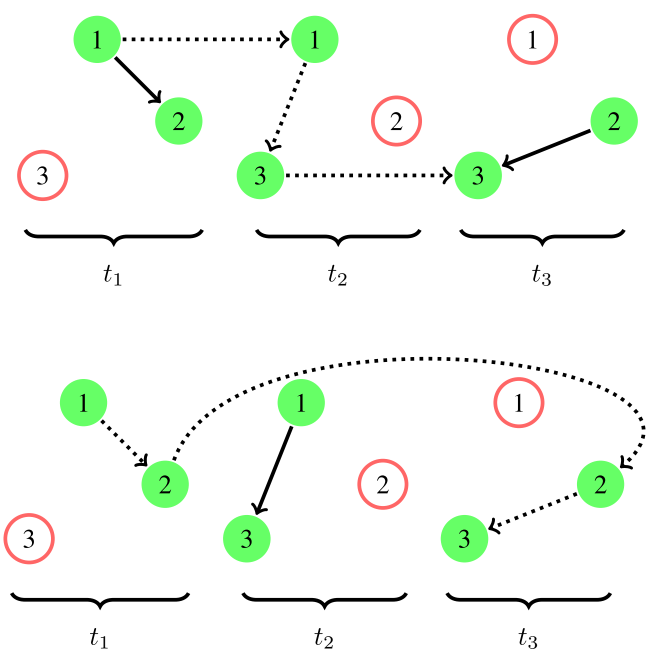


Figure 2. The two temporal paths of length 4 from $(1,t_1)$ to $(3,t_3)$ on the evolving graph shown in Figure 1, shown in black dashed lines. The paths traverse only active nodes along edges, and are allowed to advance between the same node if it is active at different times.

The restriction that temporal paths may only traverse active nodes reflects underlying structure in many real world applications, such as analyzing the influence of nodes over social networks. We will also show later in Section III-A that the resulting structure of allowable temporal paths leads to nontrivial subtleties in the generalization of algorithms and concepts from ordinary (static) graphs.

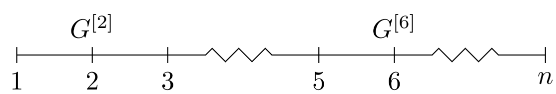
B. Breadth-First Traversal Over Temporal Paths

The example presented above in Section II-A demonstrates how active nodes restrict the set of temporal paths that need to be considered when traversing an evolving graph.

We now give a general description of the BFS algorithm over evolving graphs, both directed and undirected, which correctly takes into account the structure of temporal paths. Our notation generalizes that for static graphs presented in [11], [12].

Definition 1. *An* ***evolving graph*** G_n *is a sequence of (static) graphs* $G_n = \langle G^{[1]}, G^{[2]}, \ldots, G^{[n]}\rangle$ *with associated time labels* $t_1, t_2, \ldots, t_n$ *respectively. Each* $G^{[t]} = (V^{[t]}, E^{[t]})$ *represents a (static) graph labeled by a time* t.

Intuitively, an evolving graph is some discretization of the continuous-time family $G(t)$:



We assume no particular relation between the node and edge sets for each static graph $G^{[t]} = (V^{[t]}, E^{[t]})$. In particular, we allow the node sets to change over time, so that each $V^{[t]}$ may be different. Changing node sets happen naturally in citation networks, where nodes may appear or disappear from the citation network over time. The addition, removal, or relabeling of nodes can be expressed in terms of a map $\Pi^{[t,t']} : V^{[t]} \to V^{[t']}$ that expresses the appropriate permutations and/or projections.

Definition 2. *A* ***temporal node*** *is a pair* (v,t)*, where* $v \in V^{[t]}$ *is a node at a time* t.

Definition 3. *A temporal node* (v,t) *is an* ***active node*** *if there exists at least one edge* $e \in E^{[t]}$ *that connects* $v \in V^{[t]}$ *to another node* $w \in V^{[t]}$*,* $w \neq v$.

An ***inactive node*** *is a temporal node that is not an active node.*

In Figure 1, the temporal nodes $(1,t_1)$ and $(2,t_2)$ are active nodes, whereas the temporal node $(3,t_1)$ is an inactive node.

Definition 4. *A* ***temporal path*** *of length* m *on an evolving graph* G_n *from temporal node* (v_1,t_1) *to temporal node* (v_m,t_m) *is a time-ordered sequence of active nodes,* $\langle(v_1,t_1),(v_2,t_2),\ldots,(v_m,t_m)\rangle$*. Here, time ordering means that* $t_1 \le t_2 \le \cdots \le t_m$ *and* $v_i = v_j$ *iff* $t_i \neq t_j$.

This definition of a temporal path differs from that of the dynamic walk in [9], [10] in its explicit requirement of containing only active nodes. The definition implies that if either or both end points of a temporal path are inactive, then the entire temporal path must be the empty sequence $\langle\rangle$. Keeping track explicitly of the time labels of each temporal node allows greater generality to cases where the node sets change over time. Furthermore, we shall show later in Sec. III-A that the explicit bookkeeping of the time labels is essential for correctly generalizing the BFS to evolving graphs.

The following definition of **forward neighbors** generalizes the notion of neighbors and reachability in static graphs.

Definition 5. *The k-**forward neighbors** of a temporal node (v, t) are the temporal nodes that are the $(k+1)$st temporal node in every temporal path of length $k+1$ starting from (v, t). The **forward neighbors** of a temporal node (v, t) are its 1-forward neighbors.*

In Figure 1, the forward neighbors of $(1, t_1)$ are $(2, t_1)$ and $(1, t_2)$ and the only forward neighbor of $(2, t_1)$ is $(2, t_3)$. The 2-forward neighbors of $(1, t_1)$ are $(2, t_1), (1, t_2), (2, t_2)$ and $(3, t_2)$. By construction, time stamp of every forward neighbor of an active node (v, t) must be no earlier than t.

Definition 6. *The **distance** from a temporal node (v, t) to a temporal node (w, s) is the k for which (w, s) is a k-forward neighbor of (v, t).*

Note that this notion of distance is not a metric, since the distance from (v, t) to (w, s) will in general differ from the distance of (v, t) from (w, s) owing to time ordering.

Definition 7. *A temporal node (w, s) is **reachable** from a temporal node (v, t) if the distance to (w, s) from a temporal node (v, t) there exists some finite integer k for which (w, s) is a k-forward neighbor of (v, t).*

The BFS on evolving graphs is described in Algorithm 1. Given an evolving graph G_n and a root (v_1, t_1), Algorithm 1 returns all temporal notes reachable from the root and their distances from the root. $reached$ is a dictionary from temporal nodes to integers whose key set represents all visited temporal nodes and whose value set are the corresponding distances from the root.

The BFS constructs a tree inductively by discovering all k-forward neighbors of the root before proceeding to all $(k+1)$-forward neighbors of the root. Within the outermost loop, the algorithm iterates over $frontier$, a list of all temporal nodes of distance k from the root. The $nextfrontier$ list is populated with all temporal nodes that are forward neighbors of any temporal node in the $frontier$ list which have not yet been reached by the algorithm.

As a simple example, consider the BFS on the example graph in Figure 1 starting from the root $(1, t_2)$. The procedure is shown in Figure 3. The $frontier$ list is first initialized to $\{(1, t_2)\}$. Since the only forward neighbor of $(1, t_2)$ is $(3, t_2)$, iteration $k = 1$ produces $reached[(3, t_2)] = 1$ and $nextfrontier = \{(3, t_2)\}$. In the next iteration $k = 2$, the only forward neighbor of $(3, t_2)$ is $(3, t_3)$, so $reached[(3, t_3)] = 2$ and $nextfrontier = \{(3, t_3)\}$. The algorithm terminates after $k = 3$ after verifying that $(3, t_3)$ has no forward neighbors.

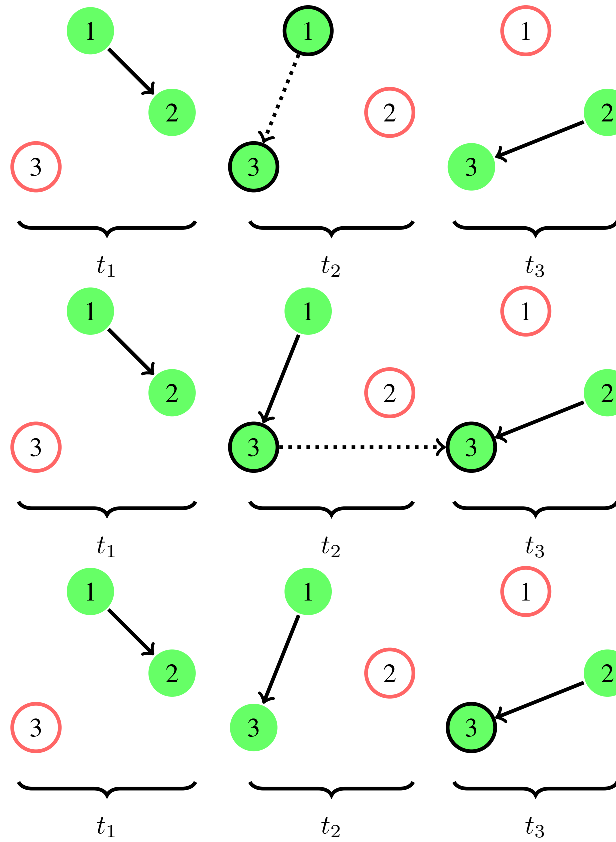


Figure 3. Breadth-first search (BFS) on the evolving graph shown in Figure 1 starting from the root $(1, t_2)$. Note that the time t_1 does not participate in the BFS. Black circles indicate the active nodes forming the $frontier$ and $nextfrontier$ sets in Algorithm 1, connected by the dotted black lines.

The preceding example illustrates the fact that $G^{[1]}$ plays no part in the BFS traversal of G_n starting from $(1, t_2)$. In general, all $G^{[t]}$ with time stamps $t < t'$ for a starting node (v, t') are irrelevant to the BFS traversal. Hence without loss of generality we may assume that BFS is always computed with a root at time t_1, the earliest time stamp in G_n.

Theorem 1 (Correctness of the evolving graph BFS)**.** *Let G_n be an evolving graph and (v_1, t_1) be an active node of G_n. Then Algorithm 1 discovers every active node that is reachable from the root (v_1, t_1), and $reached[(v, t)]$ is the distance from (v_1, t_1) to (v, t).*

Proof: Define the set of temporal nodes $\tilde{V}_L^{[t]} = \{(v_1, t) | (v_1, v_2) \in E^{[t]}\}$, which consists of the active nodes at time t which participate on the left side of an edge.

Algorithm 1: Breadth-first search (BFS) on an evolving graph G_n starting from a root (v_1, t_1). The return value, $reached$, is a dictionary mapping all reachable temporal notes from the root to their distances from the root. At the end of each iteration k, the $frontier$ set contains all temporal nodes of distance k from the root.

function *BFS*($G_n, (v_1, t_1)$)
  $reached[(v_1, t_1)] = 0$
  $frontier = \{(v_1, t_1)\}$
  $k = 1$
  while $frontier \neq \varnothing$
    $nextfrontier = \varnothing$
    for $(v, t) \in frontier$
      for $(v', t') \in$`forwardneighbors`$((v, t))$
        if $(v', t') \notin reached$
          $reached[(v', t')] = k$
          $nextfrontier =$
          $nextfrontier \cup \{(v', t')\}$
        end
      end
    end
    $frontier = nextfrontier$
    $k = k + 1$
  end
  return $reached$
end

Similarly, $\tilde{V}_R^{[t]} = \{(v_2, t) | (v_1, v_2) \in E^{[t]}\}$ contains the corresponding active nodes on the right side of an edge. Then $\tilde{V}^{[t]} = \tilde{V}_L^{[t]} \cup \tilde{V}_R^{[t]}$ is the set of active nodes at time t, and $V = \bigcup_t \tilde{V}^{[t]}$ is the set of all active nodes in G_n.

Similarly, define an edge set $E' = \{(u_s, v_t) | u_s = (u, s) \in V, v_t = (v, t) \in V, v = u, s < t\}$, which consists of temporal nodes that connect active nodes sharing the same node at different times. Each edge in E' is then in 1-1 correspondence with a temporal path of length 2, $\langle (v, s), (v, t) \rangle$. Define also $\tilde{E}^{[t]} = \{(e, t) | e \in E^{[t]}\}$, which are simply the edge sets in G_n with time labels. Then $E = \bigcup_t \tilde{E}^{[t]} \cup E'$ is the set of all edges representing all allowed temporal paths of length 2.

The node set V and edge set E now define a static graph $G = (V, E)$ that is in 1-1 correspondence with the evolving graph G_n. The node set V of G is in 1-1 correspondence with active nodes of G_n while the edge set E is in 1-1 correspondence with all temporal paths of length 2 on G_n.

We now establish a similar 1-1 correspondence of forward neighbors of an active node with a subset of G. By induction, we can show that at iteration k, all new nodes populated into the key set of $reached$ by definition are of distance k from the root. By definition, the forward neighbors of some active node $(v, t) \in G_n$ are active nodes of either the form (v, t') for some $t' > t$ or (u, t) for some $u \neq v$. Clearly, the former are elements of $E' \subseteq E$ while the latter are elements of $E^{[t]} \subseteq \bigcup_s \tilde{E}^{[s]} \subseteq E$. Thus each forward neighbor of an active node $(v, t) \in G_n$ is in 1-1 correspondence with a node in V that is a neighbor of $v_t \in V$.

The correctness of BFS on the evolving graph G_n now follows from the correctness of BFS on the static graph G, since we have also established a 1-1 correspondence for every intermediate quantity in Algorithm 1. ∎

As presented, the BFS over evolving graphs makes no assumptions about how the evolving graph G_n is represented. Suppose it is represented by a collection of adjacency lists, one for each active node in G_n. Then we have that the asymptotic complexity of BFS on G_n is the same as that for BFS on G, using the 1-1 construction of G from G_n.

Theorem 2 (Complexity of the evolving graph BFS)**.** *Let G_n be an evolving graph represented using adjacency lists, (v_1, t_1) be an active node of G_n, and $G = (V, E)$ be the static graph constructed from G_n using the 1-1 correspondences defined in the proof of Theorem 1. Then the asymptotic complexity of Algorithm 1 is $O(|E| + |V|)$.*

Proof: Any edge in any edge set of G_n can be accessed in constant time in random access memory. By construction, BFS on G_n is in 1-1 correspondence with BFS on the static graph G. The number of operations of BFS on G is $O(|E| + |V|)$, and so the result follows. ∎

III. Formulating the Evolving Graph BFS with Linear Algebra

A. Not all temporal paths are counted by products of adjacency matrices

For each static graph $G^{[t]} = (V^{[t]}, E^{[t]})$ that constitutes the evolving graph G_n, define its corresponding $\left|V^{[t]}\right| \times \left|V^{[t]}\right|$ adjacency matrix with elements

$$A_{ij}^{[t]} = \begin{cases} 1 & \text{if } (i, j) \in E^{[t]}, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

We can then represent G_n using the sequence of adjacency matrices $A_n = \langle A^{[1]}, A^{[2]}, \ldots, A^{n]} \rangle$. The example in Figure 1 can be represented as

$$\left\langle \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \right\rangle.$$

For a static graph G with adjacency matrix A, $(A^k)_{ij}$ counts the number of paths of length k between node i and node j. Naïvely, one might want to generalize this result to evolving graphs by postulating that the (i, j)th entry of the discrete path sum

$$S^{[t_n]} = A^{[t_1]} A^{[t_n]} + \sum_{t_1 \leq t \leq t_n} A^{[t_1]} A^{[t]} A^{[t_n]} + \cdots + \sum_{t_1 \leq t \leq t' \leq \cdots \leq t_n} A^{[t_1]} A^{[t]} A^{[t']} \cdots A^{[t_n]} \tag{2}$$

counts the number of temporal paths from (i, t_1) to (j, t_n). However, this postulate is incorrect. In the example of Figure 1,

$$(S^{[t_3]})_{13} = \left(A^{[t_1]}A^{[t_2]}A^{[t_3]} + A^{[t_1]}A^{[t_3]}\right)_{13} = 1$$

even though there are clearly two temporal paths from $(1, t_1)$ to $(3, t_3)$ as shown in Figure 2.

The first term in the sum $S^{[t_3]}$ vanishes since $A^{[t_1]}A^{[t_2]} = 0$. Furthermore, the vanishing of $S^{[t_2]} = A^{[t_1]}A^{[t_2]}$ itself reflects the absence of any temporal path from t_1 to t_2 that goes through at least one edge at t_1. However,

$$\langle (1, t_1), (1, t_2), (3, t_2) \rangle \tag{3}$$

is a clearly a valid temporal path as shown in Figure 2 which cannot be expressed by a product of adjacency matrices.

Sums $S^{[t]}$ of the form (2) produce an incorrect count of temporal paths because they do not capture temporal paths with subpaths of the form $\langle (v, s), (v, t) \rangle$, $s < t$. One might attempt to amend the sums $S^{[t]}$ in (2) by redefining the adjacency matrices to include ones along the diagonal, hence allowing paths containing the sequence $\langle (i, t_1), (i, t_2) \rangle$. However, the resulting sum is still incorrect, as it counts paths with subsequences $\langle (3, t_1), (3, t_2) \rangle$ and are hence not temporal paths. Instead, the temporal path (3) is counted by the matrix product $M^{[t_1,t_2]}A^{[t_2]}$, where

$$M^{[t_1,t_2]} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{4}$$

$M^{[t_1,t_2]}$ describes the forward time propagation of temporal nodes that are active at both times t_1 and t_2, i.e. it counts temporal paths that contain subsequences $\langle (i, t_1), (i, t_2) \rangle$, and both (i, t_1) and (i, t_2) are active nodes.

The simple example of Figure 1 provides several counterexamples which demonstrate why sums over products of adjacency matrices of the form (2) do not count temporal paths correctly. The interpretation from considering powers of A for static graphs therefore does not extend to evolving graphs, and sums like (2) cannot form the basis of correct analyses of evolving graphs.

B. Defining forward neighbors algebraically

The algebraic representation of evolving graphs presented in Section III-A allows us to exploit a graphical interpretation of matrix–vector products involving the adjacency matrix [12]. If A is the adjacency matrix of a (static) graph G and e_k is the kth elementary unit vector, then the nonzero entries of $A^T e_k$ have indices that are neighbors of k. The algebraic formulation of BFS on evolving graphs follows similarly, but requires a new kind of matrix–vector product, $\odot$, defined by

$$A^T \odot b = \begin{cases} b & \text{if } A^T b \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

The forward neighbors of a temporal node (k, t_1) in A_n can then be determined from the indices and time stamps of the nonzero elements in the sequence

$$\left\langle (A^{[1]})^T e_k,\ (A^{[2]})^T \odot e_k,\ \ldots\ (A^{[n]})^T \odot e_k \right\rangle. \tag{5}$$

The nonzero entries of the first vector represent forward neighbors that are on the same time stamp t_1, whereas nonzero entries of the other vectors represent forward neighbors that are advanced in time but remain on the same node k. The quantity (5) therefore encodes a BFS tree of depth 2, as its nonzero entries are labeled by all temporal nodes of distance 1 from (k, t_1).

Referring back to the example of Figure 1, the forward neighbors of node $(1, t_1)$ can be computed by

$$\left\langle \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right\rangle$$
$$= \left\langle \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right\rangle$$

From this computation, we can deduce that $(2, t_1)$ and $(1, t_2)$ are the forward neighbors of $(1, t_1)$.

C. Evolving graphs as a blocked adjacency matrix

The proof of Theorem 1 provides a construction for representing an evolving graph G_n by a static graph G with nodes corresponding to active nodes of G_n. It turns out that the block structure of G is useful for understanding the nature of the $\odot$ operation.

Consider the second iteration of BFS on G_n with root (k, t_1), which requires computing the sequences

$$\left\langle (A^{[1]})^T c_1, (A^{[2]})^T \odot c_1, \ldots, (A^{[n]})^T \odot c_1 \right\rangle \tag{6a}$$
$$\left\langle (A^{[2]})^T c_2, \ldots, (A^{[n]})^T \odot c_2 \right\rangle \tag{6b}$$
$$\ldots \tag{6c}$$
$$\left\langle (A^{[n]})^T c_n \right\rangle \tag{6d}$$

where $c_1 = (A^{[1]})^T e_k$ and $c_i = (A^{[i]})^T \odot e_k$ for $i > 1$. Summing resultant vectors that share the same time stamp, we obtain vectors whose nonzero elements have indexes labeled by the forward neighbors of the nodes computed at step 1.

Compare this with the matrix

$$\boldsymbol{M}_n = \begin{bmatrix} A^{[t_1]} & M^{[t_1,t_2]} & \ldots & M^{[t_1,t_n]} \\ 0 & A^{[t_2]} & \ldots & M^{[t_2,t_n]} \\ & \ldots & & \\ 0 & 0 & \ldots & A^{[t_n]} \end{bmatrix}$$

where $M^{[t_i,t_j]}$ is the matrix whose rows are labeled by $V^{[t_i]}$ and columns are labeled by $V^{[t_j]}$, and whose entries are

$$M^{[t_i,t_j]}_{uv} = \begin{cases} 1 & \text{if } (u, v) \in E', \\ 0 & \text{otherwise.} \end{cases}$$

The adjacency matrix blocks $A^{[t]}$ encode the edge set $\tilde{E}^{[t]}$, whereas the off-diagonal blocks $M^{[t_i,t_j]}$ together encode the edge set E' which capture temporal paths with subsequences of the form $\langle (v,t_i),(v,t_j)\rangle$. Then $\boldsymbol{M}_n$ is the adjacency matrix of the graph $(\cup_t V^{[t]}, E)$, which is the graph G together with all the inactive nodes. From the definition, $\boldsymbol{M}_n$ has nonzero entries only in rows and columns that correspond to active nodes V, and so retaining only these rows and columns corresponding to V produces the adjacency matrix $\boldsymbol{A}_n$ of $G = (V, E)$.

The off-diagonal blocks $M^{[t_i,t_j]}$ provide an explicit matrix representation for the $\odot$ product in that $(M^{[t_i,t_j]})^T b = (A^{[t_i]})^T \odot b$. An example of such an off-diagonal block was already provided in (4). These off-diagonal blocks represent traversal between active notes with the same node space labels but are still separated by time, and are essential for the correct enumeration of temporal paths. The upper triangular structure of $\boldsymbol{M}_n$ (and hence $\boldsymbol{A}_n$) reflects the causal nature of temporal paths in that they cannot go backward in time.

The BFS algorithm presented above can therefore be interpreted as computing the sequence of matrix–vector products $\boldsymbol{b}$, $\boldsymbol{A}_n^T\boldsymbol{b}$, $(\boldsymbol{A}_n^T)^2\boldsymbol{b}$, ..., formed by applying successive monomials of $\boldsymbol{A}_n^T$ to the block vector $\boldsymbol{b}^T = [b^T, 0, \cdots, 0]$ where b^T encodes the root in the space of active nodes $\tilde{V}^{[t_1]}$.

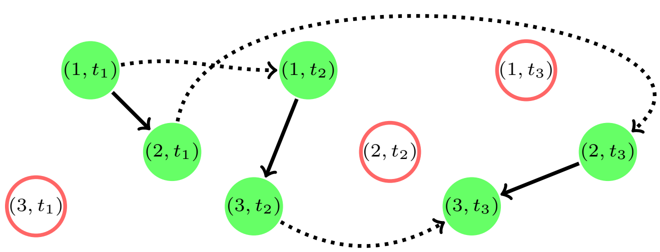


Figure 4. Static graphs corresponding to the evolving graph example of Figure 1. The green nodes are active nodes while the red nodes are inactive nodes. The black lines are edges in $\cup_t E^{[t]}$ and are encoded algebraically in the diagonal blocks $A^{[t]}$ of the adjacency matrix $\boldsymbol{A}_3$ or $\boldsymbol{M}_3$. The dotted lines are edges in E' and are encoded algebraically in the off-diagonal blocks $M^{[t_i,t_j]}$. The static graph G constructed in the proof of Theorem 1 is formed by retaining all the edges shown and only the active nodes, and has the adjacency matrix $\boldsymbol{A}_3$. The graph containing all the edges and temporal nodes shown has adjacency matrix $\boldsymbol{M}_3$.

For the example of Figure 1, we have

$$\begin{aligned} V &= \{(1,t_1),(2,t_1),(1,t_2),(3,t_2),(2,t_3),(3,t_3)\},\\ \cup_t E^{[t]} &= \{((1,t_1),(2,t_1)),((1,t_2),(3,t_2)),((2,t_3),(3,t_3))\},\\ E' &= \{((1,t_1),(1,t_2)),((2,t_2),(2,t_3)),((3,t_2),(3,t_3))\}. \end{aligned}$$

In the order specified for V, the adjacency matrix of G is then

$$\boldsymbol{A}_3 = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Starting from the vector $\boldsymbol{b} = e_1$, the sequence of iterates is then

$$\left\langle \begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\\0\\2\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix}, \dots \right\rangle.$$

We see that $(\boldsymbol{A}_n^T)^2$ correctly encodes the collection of products in (6a)-(6d). Furthermore, $(\boldsymbol{A}_3^T)^3\boldsymbol{b}$ correctly counts the two allowed temporal paths from $(1,t_1)$ to $(3,t_3)$, and that the off-diagonal structure encoded in E' and the $M^{[t,t']}$ blocks are critical to obtaining the correct count.

Finally, we prove a simple lemma that the block adjacency matrix $\boldsymbol{A}_n$ is nilpotent whenever all the subgraphs $G^{[t]}$ of G_n are acyclic.

Lemma 1 (Acyclicity implies nilpotence)**.** *Let $G_n = \langle G^{[t_i]}\rangle_{i=1}^n$ be an evolving directed graph and let all the directed graphs $G^{[t]}$ be acyclic. Then $\boldsymbol{A}_n$ is nilpotent.*

Proof: Since each directed graph $G^{[t]}$ is acyclic, its corresponding adjacency matrix $A^{[t]}$ is strictly upper triangular. As a result, $\boldsymbol{A}_n$ must be upper triangular.

Furthermore, none of the graphs $G^{[t]}$ can have any self-edges, i.e. edges of the form (u,u), and so all diagonal entries of $A^{[t]}$ must be zero. Therefore all the diagonal entries of $\boldsymbol{A}_n$ by construction must be zero also.

We have now proven that $\boldsymbol{A}_n$ is an upper triangular matrix whose diagonal entries are all zero. Therefore, $\boldsymbol{A}_n$ is nilpotent. ■

Lemma 1 also holds for acyclic undirected graphs so long as the corresponding adjacency matrix representation is encoded in an asymmetric fashion akin to (1).

The blocked matrix structure of the adjacency matrices presented here provide interesting relationships between their matrix properties and the algorithmic properties of BFS, made possible because of the reformulation of BFS as repeated power iterations of the adjacency matrix in Algorithm 2. Note, however, that these matrices never need to be instantiated for practical computations. Rather, since Algorithm 2 only requires the matrix–vector product involving the adjacency matrix, the formulation of Algorithm 1 provides an efficient way to exploit the block structure of $\boldsymbol{A}_n$. The $\odot$ operation provides an efficient way to compute the action of the off-diagonal products. Representing the diagonal blocks $A^{[t]}$ as sparse matrices product further reduces the cost of BFS by exploiting latent sparsity in graphs that show up in practical applications.

D. The algebraic formulation of BFS on evolving graphs

The blocked matrix–vector products introduced in the previous section allows us to write down an elegant algebraic formulation of BFS on evolving graphs, as presented in Algorithm 2. Note that $\boldsymbol{A}_n^T$ need never be formed explicitly, as only its matrix–vector product is required in the algorithm.

Algorithm 2: An algebraic formulation of BFS on evolving graphs. Given A_n, the adjacency matrix representation of G_n and (v_1, t_1), a node of G_n, returns *reached* as defined in Algorithm 1. The function `nonzeros(v)` returns the nonzero indices of the vector v, and the function `map(b)` maps a block vector's indices to their corresponding active nodes.

function *ABFS*($A_n, (v_1, t_1)$)
    Form $\boldsymbol{A}_n^T$ from A_n.
    $\boldsymbol{b}_{v_1} = 1$
    $k = 1$
    $reached[(v_1, t_1)] = 0$
    while `nonzeros(`$\boldsymbol{b}$`)` $\neq \varnothing$
        $\boldsymbol{b} = \boldsymbol{A}_n^T \boldsymbol{b}$
        for $k \in$ `nonzeros(`$\boldsymbol{b}$`)`
            if `activeNodes(`k`)` $\in reached$
                $\boldsymbol{b}_k = 0$
            end
        end
        for *node* $\in$`activeNodes(`$\boldsymbol{b}$`)`
            $reached[node] = k$
        end
        $k = k + 1$
    end
    return *reached*
end

Theorem 3. *Algorithm 2 terminates.*

Proof: Recall from Lemma 1 that $\boldsymbol{A}_n$ is nilpotent, i.e. there exists some positive integer k for which $\boldsymbol{A}_n^k = 0$. Hence, after iteration k, $\boldsymbol{b}$ is assigned the value $\boldsymbol{A}_n^k \boldsymbol{b} = 0$. Therefore, Algorithm 2 must terminate after iteration k. ∎

Theorem 4. *Algorithm 1 and Algorithm 2 are equivalent.*

Proof: The initialization steps are trivially equivalent. At the beginning of iteration k, the block vector $\boldsymbol{b}$ represents the frontier nodes encoding the $frontier$ set of Algorithm 1. The matrix–vector product $\boldsymbol{A}_n^T \boldsymbol{b}$ encodes the forward neighbors of all the frontier nodes. Subsequently, active nodes that have already been visited in previous iterations are zeroed out of the new $\boldsymbol{b}$. ∎

IV. Implementation in Julia

To study evolving graphs and experiment with various graph types, we have developed EvolvingGraphs.jl [13], a software package for the creation, manipulation, and study of evolving graphs written in Julia [14]. It is freely available online at

https://github.com/weijianzhang/EvolvingGraphs.jl

and available with the MIT "Expat" license. The package contains an implementation of the evolving graph BFS.

`MatrixList`, a data type in EvolvingGraphs.jl, represents an evolving graph as a sequence of sparse matrices stored in compressed sparse column (CSC) format. Many basic graph functions are also implemented for `MatrixList`.

V. Application to Citation Networks

In this section, we show that evolving graph formalism presented above can be used to capture the structure of citation networks. Consider the evolving graph $G_n = \langle G^{[t]} \rangle_t$ such that $G^{[t]}$ has node set corresponding to authors active at time t and directed edge set $E^{[t]} \ni (i, j)$ representing a citation of author j by author i in a publication at time t.

Then given an author a at time t_1, the evolving graph BFS described above can compute $T(a, t_1)$, the set of all the authors that have been influenced by a's work at time t_1. Define also a *community* to be a group of researchers that have been influenced by the same authors. For example, given a paper published by a at time t, we can determine a's community by searching backward in time to find $T^{-1}(a, t)$, the authors that influenced a at time t, and then searching forward to find $T(l_1, t_1) \cup T(l_1, t_2) \cup \cdots \cup T(l_k, t_k)$, where $(l_1, t_1), (l_2, t_2), \ldots, (l_k, t_k)$ are the leaves of $T^{-1}(a, t)$. The backward search in time follows straightforwardly from the forward time traversal presented above simply by reversing the time labels, e.g. by the transformation $t \to -t$.

VI. Conclusion

The correct generalization of BFS to evolving graphs necessitates a careful enumeration of temporal paths which is not always captured by products of successive adjacency matrices from the snapshots. BFS over any evolving graph G_n is formally equivalent to BFS on a related static graph G formed by connecting only active nodes across different times. The adjacency matrix $\boldsymbol{A}_n$ of G correctly encodes causality of the temporal paths. When specialized to acyclic G_n, the matrix $\boldsymbol{A}_n$ is also nilpotent, thus providing a simple proof that BFS terminates. Nevertheless, the BFS can be computed efficiently without ever explicitly constructing $\boldsymbol{A}_n$, as only its action on a vector is required for the power iterations. The new concepts of activeness, temporal paths, and the explicit construction of an equivalent static graph G that make possible a correct implementation of BFS to evolving graphs provide powerful new insignts into how similar graphical algorithms may be generalized correctly.

Acknowledgments

We thank N. J. Higham and V. Šego (Manchester) for helpful suggestions. W. Z. thanks the School of Mathematics at U. Manchester for research & travel funding and A. Edelman for arranging for a fruitful visit to MIT.

References

[1] P. Flajolet, D. E. Knuth, and B. Pittel, "The first cycles in an evolving graph," *Annals of Discrete Mathematics*, vol. 43, pp. 167–215, 1989. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167506008705752

[2] B. Bahmani, R. Kumar, M. Mahdian, and E. Upfal, "PageRank on an evolving graph," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 24–32. [Online]. Available: http://dl.acm.org/citation.cfm?id=2339539

[3] P. Borgnat, E. Fleury, J.-L. Guillaume, C. Magnien, C. Robardet, and A. Scherrer, "Evolving networks," *NATO ASI on Mining Massive Data Sets for Security, NATO Science for Peace and Security Series D: Information and Communication Security*, pp. 198–204, 2008.

[4] J. Tang, M. Musolesi, C. Mascolo, and V. Latora, "Temporal distance metrics for social network analysis," in *Proceedings of the 2nd ACM workshop on Online social networks*. ACM, 2009, pp. 31–36. [Online]. Available: http://dl.acm.org/citation.cfm?id=1592674

[5] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter, "Multilayer networks," *Journal of Complex Networks*, vol. 2, no. 3, pp. 203–271, 2014. [Online]. Available: http://comnet.oxfordjournals.org/content/2/3/203.short

[6] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora, "Graph metrics for temporal networks," in *Temporal Networks*. Springer, 2013, pp. 15–40. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-36461-7_2

[7] J. Tang, S. Scellato, M. Musolesi, C. Mascolo, and V. Latora, "Small-world behavior in time-varying graphs," *Physical Review E*, vol. 81, no. 5, p. 055101(R), 2010. [Online]. Available: http://journals.aps.org/pre/abstract/10.1103/PhysRevE.81.055101

[8] J. Tang, M. Musolesi, C. Mascolo, V. Latora, and V. Nicosia, "Analysing information flows and key mediators through temporal centrality metrics," in *Proceedings of the 3rd Workshop on Social Network Systems*. ACM, 2010, p. 3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1852661

[9] P. Grindrod, M. C. Parsons, D. J. Higham, and E. Estrada, "Communicability across evolving networks," *Physical Review E*, vol. 83, no. 4, p. 046120, 2011. [Online]. Available: http://journals.aps.org/pre/abstract/10.1103/PhysRevE.83.046120

[10] P. Grindrod and D. J. Higham, "A matrix iteration for dynamic network summaries," *SIAM Review*, vol. 55, no. 1, pp. 118–128, 2013. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/110855715

[11] S. Even and G. Even, *Graph algorithms*, 2nd ed. Cambridge, UK: Cambridge University Press, 2012.

[12] J. Kepner and J. Gilbert, Eds., *Graph Algorithms in the Language of Linear Algebra*, ser. Software, Environments, Tools. Philadelphia, PA: SIAM, 2011. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9780898719918

[13] W. Zhang, "Dynamic network analysis in Julia," Manchester Institute for Mathematical Sciences, The University of Manchester, UK, Tech. Rep. 2015.83, Sep. 2015. [Online]. Available: http://eprints.ma.man.ac.uk/2376/

[14] J. Bezanson, S. Karpinski, V. B. Shah, and A. Edelman, "Julia: A fast dynamic language for technical computing," ArXiv preprint 1209.5145, 2012. [Online]. Available: http://arxiv.org/abs/1209.5145