Temporal Graphs Research at Pomona College
Working Definitions and Vocabulary

Campbell, Wollman, Wu

May 23, 2016

1 About

As we continue researching temporal graphs and their related properties there is an increased need to establish fundamental definitions. Although many papers have established definitions of the terms and concepts contained within this paper, there are subtle details of each of these definitions that vary from paper to paper making it difficult to efficiently communicate exact ideas.

This document shall serve as an ongoing record of the definitions we will use in our research. The definitions contained within should serve as the defaults in conversations; if you intend to use an alternate definition to a concept defined in this document, you should state and/or cite the alternate definition. (In the future, we will look to incorporate alternate definitions in this document as well.) It should be noted that these definitions may change as we continue our research.

2 Preliminary Definitions

Definition 2.0.1. A **temporal graph** or **temporal network**[1] is defined as a tuple $G = (V, E, \mathrm{T}, \mathcal{W})$ where V is the set of vertices, and $E \subset V^2$ is the set of edges. A graph is **directed** if $(u, v) \in E$ is an ordered pair, and is **undirected** if unordered.

T is a function $\mathrm{T} : E \to \mathcal{T}^2$ with the co-domain being the start and end times of the vertices, in number of time units (you can always pick a smaller time unit so that the bounds of this interval are integral). Note that forall $e \in E$ with $\mathrm{T}(e) = (t_s, t_f)$, $t_s \leq t_f$. In a **static** graph, where $\mathrm{T}(e) = (\infty, \infty)$ for all $e \in E$, leave T out of the definition of the graph. $\mathcal{W}$ is a function

$\mathcal{W} : E \to \mathbb{Z}$ representing the integer weight of each edge. In an **unweighted** graph if every edge has the same weight, then we can leave $\mathcal{W}$ out of the definition.

[1] We will treat the terms *graph* and *network* synonymously throughout this paper.

All edges $e \in E$ are to be treated as directed edges with some weight $\mathcal{W}(e)$ and time-window $\mathrm{T}(e)$ (possibly instantaneous) in which they are *active*. Additionally, we will maintain the standard definitions of *size* and *order* defined respectively: $|E| = m$, $|V| = n$.

With this generalized definition of a temporal network, we will begin to explain some common, specialized descriptors of these networks:

Note that there is a direct transformation between undirected temporal graphs and directed temporal networks. Simply remove the direction of the edges, i.e. make the ordered pairs unordered. This is called the **underlying graph**.

Definition 2.0.2. We will describe a temporal graph as **unweighted** if every edge in E has equal weight w.

Definition 2.0.3. Edge persistence is the amount of time for which an edge is present. More formally, in a graph $(V, E, \mathrm{T}, \mathcal{W})$ the persistence of an edge e is defined, for $T(e) = (t_s, t_f)$, as $|T(e)| = t_f - t_s$.

Definition 2.0.4. An edge $e \in E$ is said to be **infinitely persistent** if $T(e) = (t_s, \infty)$.

Definition 2.0.5. A **instantaneous edge** or **contact edge** is any edge $e \in E$ where $|T(e)| = 0$.

Definition 2.0.6. Windowed networks shall be defined as any temporal network where there is an edge e with $|\mathrm{T}(e)| \neq 0$.

Definition 2.0.7. Contact networks shall be defined as a temporal network where every edge in the network is an *instantaneous edge* or *contact edge*. It is simply a special case of an interval network.

Definition 2.0.8. An **infinitely-edge-persisting network** shall be defined as a temporal network where every edge is **infinitely persistent**.

Definition 2.0.9. A **co-authorship network** is an undirected interval network in which each node represents an author, and the existence of an edge between two nodes represents a collaboration over the a period of time. Unless stated otherwise, we shall assume a *collaboration* to mean two authors a_1 and a_2 worked on (at least) one publication together in $\mathrm{T}(a_1, a_2)$.

Definition 2.0.10. A **citation network** is a directed interval network $(V, E, \mathrm{T}, \mathcal{W})$ with infinite edge persistence. For simplicity, we can write edges as $u \to_{t_1} v$ or $(u, v)_{t_1}$, with $(t_1, \infty) = \mathrm{T}(u, v)$. In this network, each node represents an author, and the existence of an edge $u \to_{t_1} v$ means that author v cited author u at time t. This way the direction of the arrow represents the flow of information.

Now we will move on to consider the different analysis methodologies that will result from different temporal definitions of 'shortest path'.

Definition 2.0.11. A graph is a **path** if it is a simple graph whose vertices can be linearly ordered such that there is an edge uv if and only if u and v are adjacent in the ordering. A digraph is a **path** if it is a simple directed graph whose vertices can be linearly ordered such that there is an edge $u \to v$ if and only if v immediately follows u in the ordering.

Definition 2.0.12. A static **shortest path** between u, v in a static (di)graph $G = (V, E, \mathcal{W})$ is $P = e_1 = (u, u'), e_2, \cdots, e_n = (v', v)$, $P \in E$ such that for all $e_i = (v_1, v_2), e_{i+1} = (v_3, v_4) \in E$, $v_2 = v_3$, and for every u, v-path $P' = e'_1, e'_2, \cdots, e'_m$

$$\sum_{e'_i \in P'} \mathcal{W}(e'_i) < \sum_{e_i \in P} \mathcal{W}(e_i)$$

Note that this definition does not enforce uniqueness of shortest paths.

3 Consecutive Contemporaneity

Here, we consider the addition of paths to include a temporal component, where any two consecutive edges must share some contemporary period. This is a sensible definition as two edges should not be able to form a path in a temporal network if they did not happen at the same time. This is formally defined below.

Definition 3.0.13. A **consecutive temporal path** between u and v at time t is a t, v_1, v_n-path $P = e_1, e_2, \cdots, e_n$ such that for consecutive edges $e_i, e_{i+1} \in P$, $T(e_i) \cap T(e_j) \neq \emptyset$, and $t \in T(e_1)$.

We will consider the consequences of this definition in the context of different edge-behaviors, infinite persistence, and windowed persistence.

3.1 Infinitely Persistent Edges

The first model we will consider is the simplest of the three, where we disallow edge-deletion. We will define the persistent coauthorship network to have this property. [note about semantic equivalence to railway network?]

Definition 3.1.1. The **persistent co-authorship network** a co-authorship network $G = (V, E)$, where for all $(u, v, t_1, t_2) \in E$, $t_2 = \infty$. For simplicity, we can denote an edge by $(u, v)_{t_1}$ or $u -_{t_1} v$. Since this network is undirected, $(u, v)_{t_1} = (v, u)_{t_1}$.

Then, we can consider what a reasonable definition of 'shortest path' might be. In this model, once an edge exists, it is always traversible, so if author a_1 wrote a paper with author a_2 in 1932, and author a_2 wrote a paper with a_3 in 1990, we can find a path between a_1 and a_3. It is also feasible that a_1 wrote a paper with a_4 in 1932 as well, and then a_4 and a_5 wrote a paper in 1933. These two paths $P_1 = a_1 -_{1932} a_2 -_{1990} a_3$, and $P_2 = a_1 -_{1932} a_4 -_{1933} a_5$ should have some manner of differentation, since the difference in start times of the edges is 58 in P_1 and only 1 in P_1. This can be seen in Figure 1, to motivates a difference in 'fastest' vs. 'shortest' path.

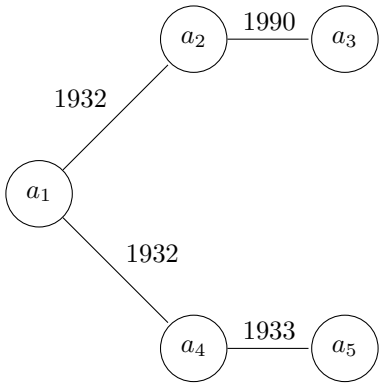

Figure 1: Example motivating the difference in shortest vs. fastest path

Definition 3.1.2. The **temporal shortest path** between u and v at time t is a consecutive temporal t, u, v-path $P = e_1, \cdots, e_n$ such that that there is no other path u, v-path $P = e'_1, \cdots, e_m$ where

$$\sum_{e'_i \in P'} \mathcal{W}(e'_i) < \sum_{e_i \in P} \mathcal{W}(e_i)$$

The **temporal fastest path** from v_1 to v_n at time t is a consecutive temporal path $v_1, v_2, \cdots, v_n$, with first edge $(v_1, v_2)_{t_1}$ and last edge $(v_{n-1}, v_n)_{t_{n-1}}$, such that there exists no other $u_1, u_2, \cdots, u_m$ with first edge $(u_1, u_2)_{s_1}$, last edge $(u_{m-1}, u_m)_{s_{m-1}}$ and $s_{m-1} - s_1 < t_{n-1} - t_1$.

Corollary 3.1.3. *Shortest path in persistent coauthorship network is the same as the shortest path in the aggregated static graph.*

Proof. (Idea). Since the edges have infinite persistence, can just wait at a vertex until the desired edge in the aggregated graph shows up. □

3.2 Windowed edges

Now consider that we in fact limit the persistence of the edges with an endpoint specific to each edge (as is specified in the definition of an interval network temporal graph). We call this graph a **windowed co-authorship network** or simply a **co-authorship network**. If we specify a universal edge-persistence Δt such that for all edges e in the network, $\Delta t = |T(e)|$, then we call this co-authorship network Δt**-windowed**.

The definitions for temporal paths will be the same as for infinitely persistent edges, 3.0.13. As well as those for fastest and shortest path will be the same as in definition 3.1.2.

4 Complete Contemporaneity

Here we can consider many of the same definitions, but under are different lens of contemporaneity for paths. Here we want all edges to have some overlap in their time interval.

Definition 4.0.1. A **complete temporal path** between u and v at time t in a graph $(V, E, \mathrm{T}, \mathcal{W})$ is a u, v-path $P \in E$ such that $t \in \bigcap_{e \in P} T(e)$.

As might be expected, complete temporal paths behave the same way that consecutive temporal paths do in the persistent co-authorship network. Since the edges have infinite persistence, all edges are contemporary 'at infinity.'

In the case of the windowed co-authorship network, the definitions remain the same for shortest and fastest paths.

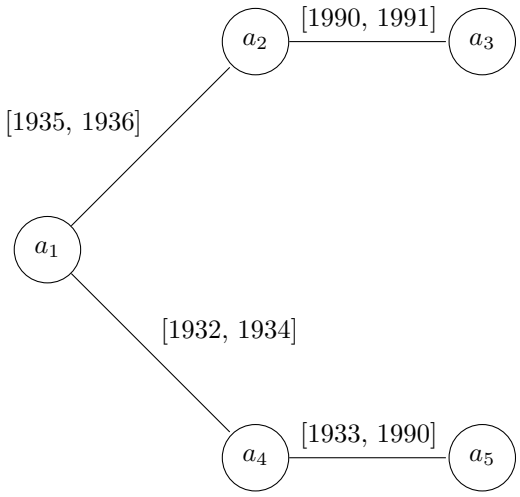

Figure 2: Example showing difference in windowed fastest and shortest paths

In Figure 2, there exists a complete contemporary path P_1 such that $P_1 = a_5a_4, a_4a_1$, but there does not exist a complete contemporary path P_2 such that $P_2 = a_5a_4, a_4a_1, a_1a_2$.

5 Shortest Path Algorithms

In this section we present naive shortest path algorithms along with their proofs based on Dijkstra for consecutive and complete temporal paths. We will analyze these in the case of the unweighted coauthorship network (V, E, T). They are each heavily based on the standard Breadth First Search, but performed on edges - not on vertices.

For both algorithms we use the matrix `dist`, which represents a complete mapping between pairs of vertices and initial time stamp and the distance be-

tween them, with the default value set to ∞. Similarly `interval` is an array representing the union of all edges on the shortest path to each edge, with the default value being `null`. R is a min-heap of edges `e`, sorted on `dist[src][snd e]`.

The last piece of this is the helper-function `edge_neighborhood`, which for an edge $e = (u, v)$ corresponds to the standard mathematical set $N_e(v) - \{e\}$. This computation will take $O(logm)$ using a B+-Tree index on the edge set. Its also possible to include an adjacency list in the definition of an edge, and have `edge_neighborhood` pull it out, giving $O(1)$, but requiring precomputation with cost $O(m \log m)$. Each algorithm iterates through every vertex, and then in the worst case, every edge will enter R at some point, giving $O(nm \log m)$. Then we do this accross all time slices, so we have $O(tnm \log m)$.

5.1 Consecutive Shortest Path Distance Algorithm

Algorithm 1: Consecutive Shortest Path Distance Algorithm

Input: A temporal graph $G = (V, E, T)$ and *dist* which encapsulates a complete mapping between pairs of vertices along with an initial timestamp to the distance between them, with the default values set to ∞.

Output: The algorithm outputs *dist* such that $dist[t][v_{src}][v_{dest}] = d$ where d is the (shortest consecutive path) distance from node v_{src} to node v_{dest} such that time t intersects the time interval of the first edge in the path.

```
1  foreach time, start node pair (t, v_src) in T × V do //loop over nodes in time slices
2      Let e_src = (v_src, v_src);
3      Let shortest interval be T(e_src) = (t, t);
4      Let the reached edges min-heap R = {e_src};
5      Let the searched edges S = ∅;
6      Let dist[t][v_src][v_src] = 0;
7      while R ≠ ∅ do
8          Let e_pred = pop(R);
9          Insert e_pred into S;
10         foreach e_succ = (u, v) ∈ N_{e_pred}(v) − S ∪ R do
11             if T(e_pred) ∩ T(e_succ) ≠ ∅ then
12                 Let dist[t][v_src][v] = min(dist[t][v_src][u] + 1, dist[t][v_src][v]);
13                 Insert e_succ into R;
14             end
15         end
16     end
17 end
18 return dist;
```

Proof. The inductive proof follows by the proof of BFS and the definition of

consecutive shortest path. □

5.1.1 Improvements

Now we can consider several improvements on the existing algorithm. The first is to define $V_t \subseteq V$, the active vertices at time $t \in \mathrm{T}$ (recall that a vertex v is active at a time t if and only if there is an edge e incident to v such that $t \in T(e)$). This will cut down significantly on the number of neighborhood queries that will return empty, instead choosing to focus on those that will return results.

We can also consider a slightly different model for this algorithm, considering the map *dist* to be a database table with B+ indices (which will be represented more simply by matrix and then introducing a temporal adjacency matrix (DB table).

Definition 5.1.1. For a temporal graph $G = (V, E, \mathrm{T})$, where V is the vertex set, E is the edge set, and $\mathrm{T} : E \to \mathcal{T}^2$ is the temporal activity function. Let $A_{\mathcal{T}} = (a_{uv}) \in M_n(\mathbb{P}(\mathcal{T}^2))$ be the **temporal adjacency matrix** of G, where a_{uv} is the set of all pairs of start and end times such that there is an edge bewteen u and v.

It's possible that we really want $A_t \in M_n(\mathcal{T}^2)$ for $t \in \mathcal{T}$, but I'm not certain whether we want this matrix to exist only at time $t \in \mathcal{T}$, or if we want this to be a time-agnostic matrix

For example, in a graph $G = (V, E, \mathrm{T})$, say that for vertices $u_0, v_0 \in V$, there are edges e_1, e_2, e_3 with u_0 and v_0 as endpoints, such that $\mathrm{T}(e_1) = (s_1, t_1)$, $\mathrm{T}(e_2) = (s_2, t_2)$, $\mathrm{T}(e_3) = (s_3, t_3)$, and $\mathrm{T}(e_1) \cap T(e_3) \neq \emptyset$, then the entry a_{u_0,v_0} of $A_{\mathcal{T}}$ is $\{(\mathrm{T}(e_1) \cup \mathrm{T}(e_3)), \mathrm{T}(e_2)\}$.

We can use this temporal adjacency matrix along with a tentative version of the *dist* matrix, to calculate the paths of length l or starting from any vertex.

Definition 5.1.2. The tentative distance matrix $\tau_t \in M_N(\mathbb{Z} \times \mathbb{P}(\mathcal{T}^2))$ is used in the iteration of the algorithm to hold the tentative distance between two vertices and the set of broadest active windows for the path.

We can use these structures to define a more efficient algorithm (which may in fact be the same algorithm)

Algorithm 2: Consecutive Shortest Path Distance Algorithm

Input: A temporal graph $G = (V, E, T)$ and τ which encapsulates a complete mapping between pairs of vertices along with an initial timestamp to the distance between them, with the default values set to $\emptyset$, and a length l which specifies the max distance to search.

Output: The algorithm outputs τ such that $dist[t][v_{src}][v_{dest}] = d$ where d is the (shortest consecutive path) distance from node v_{src} to node v_{dest} such that time t intersects the time interval of the first edge in the path.

```
1  foreach time, start node pair (t, v_src) in T × V_t do
2      Let R := {v_src};
3      Let S := ∅;
4      foreach u ∈ R do
5          foreach (d, I_{v_src,u}) ∈ τ_t[v_src][u] do
6              Let S_I := A_T[u];
7              foreach I ∈ S_I at vertex u do
8                  foreach i ∈ I do
9                      if i ∩ I_{v_src,u} then
10                         τ_t[v_src][u'] += (d, i) ;
11                         if d + 1 < l then
12                             R += u' ;
13                         end
14                     end
15                 end
16             end
17         end
18         S += u;
19     end
20 end
21 dist := τ ;
22 return dist;
```

The efficiency of this is $O(ltn^{\log_{\Delta G} n} \log^3 m) = O(ltn \log^3 m)$, so it seems that this algorithm, in fact, fares similarly to the previous one, Solely dependent on whether

Unfortunately, this is still doing a bunch of double counting, and we want to find a way, so that when we get to a node v from time t for which we have calculated a distance tree in $t-1$, and t intersects with the active interval of the subtree, we can skip that subtree. One way to do this is by running dijkstra on the co-incidence graph.

Definition 5.1.3. For a temporal graph $G = (V, E, T)$, its **coincidence graph**, $C_G = (E, \mathcal{E})$, with $\mathcal{E} \subseteq E^2$ is a simple graph where the edges of

G become nodes of C_G, and there is an edge between two nodes exactly when they share an incident vertex in C_G. Symbolically, for two edges $e = uv, f = wz$ in the graph G, there is an edge $ef \in \mathcal{E}$ in the graph C_G if and only if $\{u, v\} \cap \{w, z\} \neq \emptyset$.

The great thing about this coincidence matrix is that it is simply another way of looking at existing data, and doesn't require addtional computation to create. This gives rise to another version of this algorithm. It actually requires two steps. The first is to run dijkstra on $C_G \cap I_G$, so that the edges are co-incident and contemporary (we won't actually pre-compute this graph, but we'll do it as we go, to allow for one(ish) pass).

Algorithm 3: Shortest Edge-Distance Algorithm

Input: A temporal graph $G = (V, E, T)$ and *e_dist* which encapsulates a complete mapping from pairs of edges to the distance between them, with the default value set to ∞

Output: The algorithm outputs *e_dist* such that $e\_dist[e_{src}][e_{dest}]$ is the distance between e_{src} and e_{dest} in C_G.

```
1  foreach e_src ∈ E do
2  |  Let R := {e_src};
3  |  Let S := ∅;
4  |  Let e_dist[e_src][e_src] := 0;
5  |  while R ≠ ∅ do
6  |  |  Let e := pop R;
7  |  |  S += e;
8  |  |  foreach f ∈ N(e) − (S ∪ R) do
9  |  |  |  if T(e) ∩ T(f) ≠ ∅ then
10 |  |  |  |  e_dist[e_src][f] := e_dist[e_src][e] + 1;
11 |  |  |  |  R += f;
12 |  |  |  end
13 |  |  end
14 |  end
15 end
16 return e_dist;
```

The above algorithm is simply a BFS so it has worst-case time complexity described by $O(|E| + |\mathcal{E}|)$.

Once this step has completed, we must convert this intermediate result, the shortest paths on C_G, to the one we are searching for, the shortest paths on G. To do this, for every pair of vertices, find the smallest distance in C_G between the incident edges, and add one, we see this in Algorithm 4

Algorithm 4: Edge-Distance to Distance

Input: A graph $G = (V, E, T)$, where $T : E \to \mathcal{T}$ and the precomputed map *e_dist* describing the distances between all vertices in $C_G = (E, \mathcal{E})$.

Output: The map *dist* describing the distances between all pairs of vertices at every time slice.

```
foreach (t, u, v) ∈ 𝒯 × V_t × V do
    d_e := min{e_dist[e][f] | e ∈ N_E(u), f ∈ N_E(v), t ∈ T(e)} ;
    dist[t][u][v] := d_e + 1 ;
end
return dist;
```

This has the higher complexity, and is $O(tn^2\Delta^2)$, where Δ is the maximum degree of G.

So the overall time complexity of algorithms 3 and 4 composed is $O(tn^2\Delta^2 + m + |\mathcal{E}|) = O(tn^2\Delta^2 + |\mathcal{E}|)$, and for a sparse graph $O(tn^2)$.

5.2 Complete Shortest Path Distance Algorithm

Complete SP Distance

6 Fastest Path Algorithms

6.1 Consecutive Fastest Path Distance Algorithm

Consec FP Distance

6.2 Complete Fastest Path Algorithm

Complete FP Distance