

Temporal Pattern Matching

Campbell, Wu

June 6, 2016

1 Problem Definition

In this section, we provide a set of formal definitions of temporal graph and temporal graph query. Then, we will discuss different semantics for interpreting the graph pattern matching problem on temporal graphs.

A temporal graph is a graph that is annotated with time (of domain \mathbb{Z}). Formally,

Definition 1.1. A *temporal graph* is a node and edge labeled $G = (V_G, E_G)$ where V_G is the set of vertices, and $E_G \subset V^2 \times \mathbb{Z}^2$ is the set of edges. A label function L_G maps each node and each edge to its label.

Associated with each edge e is a time interval (ts_e, tf_e) , which we call the **active period** of the edge.

We provide a few helper functions to obtain the end nodes of each edge and the set the outgoing and incoming edges of a node:

- given a node $u \in V_G$, $out(u) = \{e \in E_G \mid e = (u, v) \text{ for some } v \in V_G\}$;
 $in(u) = \{e \in E_G \mid e = (v, u) \text{ for some } v \in V_G\}$.
- given an edge $e = (u, v) \in E_G$, $\pi_1(e) = u$ and $\pi_2(e) = v$.

$T : E \rightarrow \mathbb{Z}^2$ is a function that for any $e \in E_G$, $T(e) = (ts_e, tf_e)$.

We provide two helper functions T_s and T_f that return the start and finish time of the active period of an edge: $T_s(e) = ts_e$ and $T_f(e) = tf_e$.

Given two time intervals $T_1 = (ts_1, tf_1)$ and $T_2 = (ts_2, tf_2)$, we define the following predicates and computation:

- $T_1 = T_2 \Leftrightarrow ts_1 = ts_2 \wedge tf_1 = tf_2$;
- $T_1 \subseteq T_2 \Leftrightarrow ts_1 \geq ts_2 \wedge tf_1 \leq tf_2$;
- $T_1 \cap T_2 = (max(ts_1, ts_2), min(tf_1, tf_2))$;

A time interval $(ts, tf) = \emptyset$, if $tf < ts$.

Remarks:

1. G is not necessarily connected.

2. There maybe more than one edge between a pair of nodes, bearing different active period (t_s, t_f) .
3. For each pair of edges e_1 and e_2 between the same pair of nodes (u, v) that have the same edge label, we can assume that the active period of the edges do not overlap, e.g., $T(e_1) \cap T(e_2) = \emptyset$, since if they do overlap, we can combine the two edges to form one whose active period is the union of the two.
4. There can be many simplified versions of the node and edge labeling. For instance, only nodes are labeled, but edges are not, hinting that all edges are labeled the same.
5. In this definition, we only associate edges with timestamps. We can assume that nodes are always active.
6. In this definition, G is a directed graph. To make it undirected, we can
 - define $E \subseteq P^2(V) \times \mathbb{Z}^2$, where $P^2(V)$ is the powerset of size two over the naturals.
 - require that $(u, v, t_s, t_f) \in E_G \rightarrow (v, u, t_s, t_f) \in E_G$

We define graph patterns in a way similar to how graph patterns are defined in SPARQL-like queries, but allowing users to provide additional constraints on time.

Definition 1.2. A temporal graph query $q = \langle G_q, T_q \rangle$ consists of a graph pattern G_q and a time interval T_q .

Both the start and finish time of T_q can be a constant or ?.

The graph pattern is a connected graph $G_q = (V_q, E_q)$, where V_q is the set of nodes and E_q is the set of edges. A label function L_q maps each node/edge to its label, which can also be ?. Associated with each edge in E_q , user can also provide a temporal constraint in the form of a time interval, again, both the start and finish time can be a constant or ?.

We call T_q the **global temporal constraint** of q and $T(e_q)$ for each $e_q \in E_q$ the **local temporal constraints**.

We overload the helper functions introduced earlier to apply to graph pattern and time intervals that serve as temporal constraints.

Remark: the definition above can be incorporated easily into SPARQL. We can investigate the details when we settle on the definition.

Definition 1.3. A **match** of a graph pattern G_q in G is a total mapping $h : \{e_q : e_q \in E_q\} \rightarrow \{e_G : e_G \in E_G\}$ such that:

- for each edge $e_q \in E_q$, the edge label predicate associated with e_q is satisfied by $h(e_q) \in E_G$.

- for each node $v_q \in V_q$, the mapping of the outgoing and incoming edges of v_q share the same end node $v_G \in V_G$ and the node label predicate associated with v_q is satisfied by v_G . Formally, for any two edges $e_1, e_2 \in E_q$, if $\pi_i(e_1) = \pi_j(e_2)$, where i, j can be 0 or 1, the following must hold: $\pi_i(h(e_1)) = \pi_j(h(e_2))$.

Please note that the definition of matching is the same as pattern matching defined for SPARQL-like graph queries. The new problem is how we can take the temporal constraints into consideration, which will be defined next.

Note that we allow users to provide temporal constraints in the form of a time window for the whole pattern and for each edge, but we also provide the flexibility for them not to provide any specific temporal constraints via the “?” option. Hence, users’ temporal specification can be very strict, or very relaxed, or anywhere in between. Here are some scenarios:

- most strict: user specifies explicit global and local temporal constraints, and for each constraint specified, the start time is the same as the end time.
- most relaxed: user specifies temporal constraints with all ?’s, which means infinity.
- anywhere in between: including the cases in which some temporal constraints contains ?.
- conflicted: the intersection of the global temporal constraint and at least one of the local temporal constraint is empty. **remark:** we can easily identify conflict cases and return empty results without query evaluation.

We first define a few semantics that explicitly address user-specified temporal constraints:

Definition 1.4. Given a graph G , a temporal graph query $q = \langle G_q, T_q \rangle$, we say that a graph pattern matching h explicitly satisfies the temporal constraint of q if

- under the **exact** semantics, h satisfies that:
 - for all $e_q \in E_q$, $T(h(e_q)) = T(e_q)$ and $T(h(e_q)) \subseteq T_q$.
- under the **contain** semantics, h satisfies that:
 - for all $e_q \in E_q$, $T(h(e_q)) \subseteq T(e_q)$ and $T(h(e_q)) \subseteq T_q$.
- under the **contained** semantics, h must satisfy that:
 - for all $e_q \in E_q$, $T(h(e_q)) \supseteq T(e_q)$ and $T(h(e_q)) \supseteq T_q$.
- under the **intersection** semantics, h must satisfy that:
 - for all $e_q \in E_q$, $T(h(e_q)) \cap T(e_q) \neq \emptyset$ and $T(h(e_q)) \cap T_q \neq \emptyset$.

Remark: We need to think more carefully about how global temporal constraint T_q is interpreted in these semantics. Best way is to come up with some example queries.

The *explicit temporal semantics* defined above address only the issue of interpreting temporal constraints specified by users. Matching returned under all these semantics will include subgraphs that are not temporally traversable.

We next define a few *implicit temporal semantics* as remedy.

Definition 1.5. Given a temporal graph G , we say that the graph is **concurrent** if $\bigcap_{e \in E_G} T(e) \neq \emptyset$.

Definition 1.6. Given a temporal graph G , we say that the graph is **consecutive** if for any $e_1 = (w, u), e_2 = (u, v) \in E_G$, $T(e_1) \cap T(e_2) \neq \emptyset$.

Hence, we can define *implicit temporal semantics* to demand that resultant matching sub-graph be **concurrent** or **consecutive**.

Conjectures:

1. If the graph G is of star shape, e.g., there exist a center node u such that for any other nodes v there exists an edge between u and v , then, G is concurrent iff G is consecutive.
2. If the graph G is a clique, then, G is concurrent iff G is consecutive.
3. If there exists a circle that traverse through all nodes in G , then, there exist an edge on this circle whose active period dominates those of all other edges on the circle.

Remarks:

- Whether these conjectures are true or not may depend on whether the graph is directed or undirected.
- Either we need to find some work that has proved them, or to prove them, which should not be hard.
- These conjectures, if proved true, can be used to speed up query evaluation process. It should be easy to identify star shape, clique and circle in query patterns, which are usually very small. This will allow us to strength the temporal constraints based on the properties described in the conjectures to enhance filtering.

2 Edge Isomorphism

In this section we want to enumerate the basics of the interval graph method by expanding on the matches h we defined above in Definition 1.3. Here we introduce the concept of the Coincidence Interval Graph, \mathcal{I}_G^c of a temporal

graph G . This interval graph is a lossless encoding (shown to be bijective) of the full temporal graph G , which allows us to run existing algorithms to find appropriate patterns.

Definition 2.1. *The Coincidence Interval Graph, \mathcal{I}_G^c of a graph G under temporal condition c , is a tuple $\mathcal{I}_G^c = (E(G), \mathcal{E})$, where $E(G)$ is the edge set of the graph G (and the vertex set of \mathcal{I}_G^c , and $\mathcal{E} \subseteq E(G)^2$ is the set of “meta-edges” between the edges of G (vertices of \mathcal{I}_G^c). There is an edge between $e, f \in E(G)$ exactly when*

- e and f share an endpoint, and
- $c(e, f)$ returns True

Where $c : E^2 \rightarrow \{0, 1\}$ is a function that allows the user to control the conditions under which two edges are “contemporary”. Let the function that computes this graph under the condition c be I_c .

The generic nature of this definition of the Coincidence Interval Graph allows for the user to define textithow the contemporaneity of the query can be defined. Simply, this can be extended to define the **implicit temporal semantics** of the resultant query. In practice we will use the functions CONSEC and CONCUR (enforcing definitions ?? and ?? respectively) most frequently. But one could also imagine infinitely nuanced definitions. Another that is adapted from one commonly used for time-respecting paths (Kempe et al) is T-RESP, wherein the graph must be weakly temporally connected (citation).

Lemma 2.1. *The function I_c that constructs the Coincidence Interval Graph given some contemporaneity condition c is a bijection.*

Proof. injective. Want to show that for any c , $I_c(G) = I_c(H)$ implies $G = H$ for any two graphs G and H .

prove injectivity

surjective. Want to show that for every interval graph \mathcal{I}_G^c , there exists a graph G , such that $I_c(G) = \mathcal{I}_G^c$.

prove surjectivity

□

The construction of this graph (as defined in Algorithm ??) is a fairly straightforward algorithm (and in fact is $O(|E|d_{\max}(G))$ in the edge-relational representation of the graph). A quick corollary of Lemma 2.1 is that given a condition c , if there exists some isomorphism $f_c : \mathcal{I}_G^c \rightarrow \mathcal{I}_H^c$, then $I_c^{-1} \circ f_c \circ I_c : G \rightarrow H$ is also an isomorphism.

write this
alg

3 Neighborhood Encoding

This section focuses on encoding methods that we can use to prune the search space. In a similar vein to the way that GADDI, QuickSI, and GraphQL have developed subgraph signatures that allow for the search space to be pruned, we will

extend, specifically, the notion of neighborhood subgraph profiles (GraphQL) to include temporal information. First, let's present the notion of Neighborhood Subgraphs.

Definition 3.1. (*Neighborhood Subgraph*) Given a static graph G_s , vertex v and radius r , the r -neighborhood subgraph of vertex v , denoted $N(G, v, r)$, consists of all vertices within distance r from v and all edges between the vertices. Note that $N(G, v, 0) = (v, \emptyset)$. (Taken from GraphQL)

This is a fairly basic notion that is easily extensible to consider temporal information. The first requirement that will help us to prune the search space, is to force these subgraphs to be contemporary under some condition c (e.g. CONSEC or CONCUR). So it is possible that a given neighborhood subgraph is contemporary under the given c , but that there are certain subgraphs of the static neighborhood subgraph that are. So our extension will return a set of subgraphs, not just a single subgraph.

Definition 3.2. (*Temporal Neighborhood Subgraphs*) Given a temporal graph G , a vertex v , a radius r , and a condition c , the Temporal Neighborhood Subgraph Set $N_c(G, v, r)$ of a vertex v consists of those static neighborhood subgraphs that are concurrent under c . Note that $\bigcup N_c(G, v, r) = N(G, v, r)$.

However, calculating these temporal r -neighborhood subgraphs are expensive, so we will condense these into a much more lightweight *profile*. For static r -neighborhood subgraphs, we simply lexicographically order the labels on the vertices within the set. Then given a similar ordering of the vertices for the query graph Q , we can prune the search space when the generated r -neighborhood subgraph of the query graph is not a subsequence of the data graph profile.

For temporal r -neighborhood subgraphs, we have more information that we can use to prune the search space. The *temporal profile* of an r -neighborhood subgraph will be a tuple $p = (p_s, p_t)$ of the static profile and some temporal information. This temporal information will be an interval constructed from the intervals on the edges of graph. We will henceforth refer to p_s as the semantic profile, and p_t as the profile interval.

How exactly this profile interval will be constructed is closely related to the contemporaneity condition c , and will be user defined. For now we will notate this as $c.p$, to signify that it is the folding protocol defined for the contemporaneity condition c . For example in the CONSEC case we will take the union of the intervals on the edges, in the T-RESP case we will take interval equal to (t_{smin}, ∞) , and in the CONCUR case we will take the intersection of the intervals on the edges.

Once we have this information, we will compare the temporal r -neighborhood profiles of the query graph and the data graph. The semantic condition will be that the query semantic profile must be a subsequence of the data semantic profile, where the interval condition will just be that the intersection must be nonempty. If there is no query interval specified, then we will consider the interval to be (∞, ∞) . This is defined formally below

Definition 3.3. *The temporal profile p of a graph $G = (V, E)$ given some contemporaneity condition c is defined to be a tuple (p_s, p_t) where*

- $p_s := \text{toSortedList } \{L(v) | v \in V\}$, and
- $p_t := \text{foldr1 } c.p \ \{T(e) | e \in E\}$, where *foldr1* accumulates $c.p$ accross the given set, assuming the set is non-empty. If the set is empty then let p_t be (∞, ∞) .