Applied Network Science

RESEARCH Open Access

TemporalRI: subgraph isomorphism in temporal networks with multiple contacts

Giovanni Micale[1*], Giorgio Locicero[2], Alfredo Pulvirenti[1] and Alfredo Ferro[1]

*Correspondence: giovanni.micale@unict.it
[1] Department of Clinical and Experimental Medicine, University of Catania, Catania, Italy
Full list of author information is available at the end of the article

Abstract

Temporal networks are graphs where each edge is associated with a timestamp denoting when two nodes interact. Temporal Subgraph Isomorphism (TSI) aims at retrieving all the subgraphs of a temporal network (called target) matching a smaller temporal network (called query), such that matched target edges appear in the same chronological order of corresponding query edges. Few algorithms have been proposed to solve the TSI problem (or variants of it) and most of them are applicable only to small or specific queries. In this paper we present *TemporalRI*, a new subgraph isomorphism algorithm for temporal networks with multiple contacts between nodes, which is inspired by RI algorithm. TemporalRI introduces the notion of temporal flows and uses them to filter the search space of candidate nodes for the matching. Our algorithm can handle queries of any size and any topology. Experiments on real networks of different sizes show that TemporalRI is very efficient compared to the state-of-the-art, especially for large queries and targets.

Keywords: Subgraph isomorphism, Temporal networks, Dynamic networks, Network analysis

Introduction and related works

Graphs (or networks) are mathematical objects that are suitable to represent complex systems formed by a set of entities that interact each other. Entities are called nodes while their interactions are called edges. In many applications graphs are considered as static objects, without taking into account when or how long two nodes interact. However, complex systems are inherently dynamic and evolve during time. For example, in a remote communication system, users may enter the network anytime to start communications with other users. In a protein-protein interaction network a protein can establish temporary interactions with one or more proteins to perform a biological process or transmit a signal to a cell. Therefore, time is crucial to understand the formation and the evolution of such systems. By associating a time information to each edge, a network becomes temporal.

There are several definitions of temporal networks in literature (Holme and Saramaki 2012; Masuda and Lambiotte 2020), which are commonly referred to as dynamic (Carley et al. 2007), evolutionary (Aggarwal and Subbian 2014) or time-varying (Casteigts et al. 2011). In this paper, we define temporal network as a multigraph (i.e a graph with one or more edges between any two nodes) where each edge is associated with an integer, called timestamp, denoting when two nodes interact.

Several frameworks have been proposed to study properties of temporal networks. Analysis of temporal networks includes network centrality (Lv et al. 2019; Tsalouchidou et al. 2020), network clustering (Crawford and Milenkovic 2018), community detection (Rossetti and Cazabet 2018), link prediction (Divakaran and Mohan 2020), graph mining (Sun et al. 2019a), graph embedding (Torricelli et al. 2020), network sampling (Rocha et al. 2017), random models (Singh and Cherifi 2020; Petit et al. 2018; Hiraoka et al. 2020) and epidemic spreading (Tizzani et al. 2018; Masuda and Holme 2020; Williams et al. 2019). Comprehensive reviews of temporal networks and their main properties can be found in Holme and Saramaki (2012, 2019) and Masuda and Lambiotte (2020).

Here, we focus on Temporal Subgraph Isomorphism (TSI) problem. Given two temporal graphs Q and T, called query and target, respectively, and a time interval Δ, TSI problem aims at finding a subgraph S of T (called occurrence of Q in T) such that: (1) Q and S are isomorphic, i.e. structurally equivalent, (2) edges in S follow the same chronological order imposed by corresponding matched edges in Q, (3) all interactions in T are observed in a time interval less than or equal to Δ. The problem can have more than one solution, i.e. there can exist two or more occurrences of Q in T. Figure 1 shows a toy example of TSI.

Subgraph isomorphism problem in static graphs have been widely investigated and several methods have been presented (Cordella et al. 2004; Carletti et al. 2017; Bonnici et al. 2013; Bonnici and Giugno 2017; Han et al. 2019; Sun and Luo 2020; Han et al. 2013; Bi et al. 2016). However, as far as we know, few algorithms have been proposed for TSI or similar definitions of the problem (Redmond and Cunningham 2013b, 2016; Mackey et al. 2018; Sun et al. 2019b; Kim et al. 2018).

Redmond and Cunningham (2013b, 2016) introduce for the first time the TSI problem, where: (1) queries are static graphs with no timestamps, (2) all paths in the matched target subgraph must be time-respecting, i.e. any outcoming event from a node has to follow any incoming event from the same node, and (3) consecutive events are d-adjacent, i.e. the difference between their timestamps must not exceed a threshold d. The authors also presented a modified version of the subgraph isomorphism algorithm VF2 (Cordella et al. 2004) capable to check the time constraints of the query during the search. The same definition of the TSI problem we introduced above is given in Mackey et al. (2018) and Sun et al. (2019b). Mackey et al. (2018) describe a general algorithm which orders both query and target edges before matching. By doing so, whenever a new match between a target edge t and a query edge q is found, the algorithm can continue the search from the target edge following t in the ordering. Sun et al. (2019b) illustrate a method for counting the number of occurrences of a temporal query in a temporal target, based on partitioning the graph into subgraphs and then counting the pattern in each subgraph while matching motif topology.

A problem closely related to TSI is temporal motifs search, which consists in finding all small and recurrent temporal subgraphs of interactions, called motifs, having k nodes and/or m edges (Kovanen et al. 2011; Paranjape et al. 2017; Liu et al. 2019; Hulovatyy et al. 2015). The main difference between TSI and motif search is that TSI enumerates all the subgraphs matching a specific query, while motif search counts the occurrences of all possible subgraphs with a specified number of nodes and edges. Due to the computational complexity of the problem, motif search algorithms usually focus on very small motifs or on specific topologies. Temporal motifs were introduced for the first time by Kovanen et al. (2011) and are defined as ordered sets of events such that: (1) the time difference between two consecutive events is within a threshold Δt and (2) adjacent events, i.e. events sharing a node, are consecutive, so the node cannot participate in any other event in the meanwhile. Hulovatyy et al. (2015) relax the latter constraint and introduce the concept of dynamic graphlets to capture how the neighborhood of a node changes over time. This is done to reduce the computational complexity while obtaining approximate results. As in Mackey et al. (2018) and Sun et al. (2019b), Paranjape et al. (2017) define a temporal motif as a subgraph with a temporal order of edges. The authors present an algorithm to efficiently calculate the frequencies of 2-node, 2-stars and 3-node triangle temporal motifs. For bigger motifs they use a naive algorithm that first computes static matches, then filters out occurrences which do not match the temporal constraints. To tackle with the NP-completeness of temporal motif search problem, Liu et al. (2019) propose a general sampling framework to estimate motif counts. It consists in partitioning time into intervals, finding exact counts of motifs in each interval and weighting counts to get the final estimate, using importance sampling.

Here, we present a new algorithm for the TSI problem, called TemporalRI, inspired by RI algorithm for subgraph matching in static networks (Bonnici et al. 2013; Bonnici and Giugno 2017). TemporalRI can be applied to queries of any size, in terms of number of nodes and edges. This paper extends in several directions the preliminary work presented in Locicero et al. (2021) where a first implementation of TemporalRI was presented. From a theoretical point of view, the definition of TSI has been refined by introducing a parameter Δ to set a maximum temporal interval in which interactions should be observed. The concept of temporal flow introduced in Locicero et al. (2021) has been better formalised to include more types of flows, improve filtering and speedup the matching process. From an algorithmic point of view, TemporalRI has been redesigned in order to support multiple edges between two nodes. More specifically, the new version of the algorithm performs matching edge-by-edge, while the preliminary version presented in Locicero et al. (2021) executes matching node-by-node as the original RI algorithm (Bonnici et al. 2013; Bonnici and Giugno 2017). We compare TemporalRI with Mackey's algorithm (Mackey et al. 2018) on a dataset of real temporal networks, showing that our method is faster, especially for large targets and queries with many nodes and edges, independently of Δ.

Preliminary definitions

In this section we formally define the Temporal Subgraph Isomorphism (TSI) problem and the concept of temporal flows, which are used by TemporalRI to filter candidate pairs of query and target nodes for the matching. Through the paper we will use the terms "graph" and "network" interchangeably.

Temporal subgraph isomorphism

A *temporal graph* (or network) is a pair $G = (V, E)$, where V is the set of nodes and $E \subseteq V \times V \times \mathbb{R}$ is the set of edges. Each edge is a triplet (s, d, t), where s is the source of

e, d is the destination of e and t is the timestamp of e. Triplets in E are distinct, therefore multiple edges between two nodes having the same timestamp are not allowed.

If $\forall(s, d, t) \in E$ also $(d, s, t) \in E$, then G is *undirected*, otherwise G is *directed*. With the notation *e.source*, *e.dest* and *e.time* we represent the source, the destination and the timestamp of an edge e, respectively. If $(s, d, t) \in E$ or $(d, s, t) \in E$ we say that s and d are neighbors. With $Neigh(u)$ we denote the set of all neighbors of node u. The out-degree of a node u, $outDeg(u)$, is the number of edges having u as source. Likewise, the in-degree of u, $inDeg(u)$, is the number of edges having u as destination. The total degree of u is the sum of its in- and out-degree.

Given two temporal graphs $Q = (V_Q, E_Q)$ and $T = (V_T, E_T)$, called *query* and *target*, respectively, and an integer Δ, the *Temporal Subgraph Isomorphism* (TSI) problem consists in finding an injective function $f : V_Q \rightarrow V_T$, called *node mapping* and an injective function $g : E_Q \rightarrow E_T$, called *edge mapping*, such that the following conditions hold:

1 $\forall\, e_Q = (u, v, t_Q) \in E_Q,\, g(e_Q) = (f(u), f(v), t_T)$;
2 $\forall\, e_Q,\, e'_Q \in E_Q$ *s.t.* $e_Q.time \leq e'_Q.time,\, g(e_Q).time \leq g(e'_Q).time$;
3 $\forall\, e_Q,\, e'_Q \in E_Q, |g(e_Q).time - g(e'_Q).time| \leq \Delta$;

Condition 1 ensures that edge mapping g is consistent with node mapping f. Condition 2 means that the chronological order of query edges based on their timestamps must be respected in the target too. So, timestamps of query edges are more like indexes denoting in which order target interactions should happen. Condition 3 implies that all matched target edges must be observed within a fixed time window. This is a reasonable constraint in many real contexts, because interactions that occur far apart in time are likely to be unrelated each other. For example, in a communication network, node B receives a message from node A and then after a finite amount of time it may reply back to A or act like a broker and send another message to a third node C. Notice that Condition 3 only applies to the target, indeed timestamps in the query can assume any value.

The TSI problem can have one or more solutions. Given an edge mapping g, a *match* of Q in T is the set of pairs of query and target matched edges $\mathcal{M} = \{(q_1, g(q_1)), (q_2, g(q_2)), \ldots, (q_k, g(q_k)\}$, where $k = |E_Q|$.

An *occurrence* of Q in T is a graph O formed by edges $g(q_1), g(q_2), \ldots, g(q_k)$ and all nodes that are sources or destinations of at least one of these edges.

Figure 1 illustrates an example of application of the TSI problem with $\Delta = 5$. There is exactly one match of query Q in target T, which is $\mathcal{M} = \{((a, b, 1), (C, A, 11)),$ $((b, c, 2), (A, B, 13)),$ $((c, b, 3), (B, A, 15)),$ $((c, d, 3), (B, D, 15))\}$, and nodes and edges of the corresponding occurrence are drawn in red. The two subgraphs on the right, S_1 and S_2, are not occurrences of Q in T because S_1 violates the Δ constraint, while S_2 does not satisfy the chronological order imposed by query edges.

Temporal flows

Two edges e_1 and e_2 of a temporal graph G sharing at least one node u form a *temporal flow*, denoted as $F = \{e_1, e_2\}$. u is the *center* of the flow. Depending on the number of nodes shared by e_1 and e_2, the timestamps of the two edges and whether G is directed or not, we can distinguish among different types of flows. Figure 2a depicts the 3 possible types of flows for undirected networks, while Fig. 2b, c illustrates the 12 different classes of flows in directed networks.

For simplicity, each flow is uniquely identified by a code of the form "$x - y$" for undirected and "$x - y - z$" for directed networks, where:

- x is the number of nodes involved in the flow (2 or 3);
- y represent the temporal direction of the flow:
    - *Asynchronous (AS)* e_1 and e_2 have the same direction but different timestamps;
    - *Synchronous (S)* e_1 and e_2 have the same direction and equal timestamps;
    - *Forward (F)* a flow where $e_1.dest = e_2.source$ and $e_1.time < e_2.time$ (only in directed networks);
    - *Backward (B)* a flow where $e_1.dest = e_2.source$ and $e_1.time > e_2.time$ (only in directed networks).
- z denotes the combination of directions of e_1 and e_2:
    - *Input–Input (II)* a flow where $e_1.dest = e2.dest$;
    - *Input–Output (IO)* a flow where $e_1.dest = e2.source$;
    - *Output–Output (OO)* a flow where $e_1.source = e2.source$.

Note that the undirected flow "2-S" is not allowed, because by definition of temporal graph we cannot have multiple edges between two nodes with the same timestamp.

Given a node u, we can build, for each flow type C, the set of all flows of class C centered in u. The result is a vector of temporal features $\mathcal{S}(u)$, called *temporal signature*, where each feature is a set of flows of a certain type. With $\mathcal{S}(u)[i]$ we denote the ith component of the vector and with $|\mathcal{S}(u)[i]|$ the cardinality of the corresponding set. Figure 3 shows the temporal signatures of each query and target node of Fig. 1.

By comparing temporal signatures of different nodes, we can derive a partial order binary relation $\preceq$, called *temporal inclusion*. Given two nodes u and v with temporal signatures $\mathcal{S}(u)$ and $\mathcal{S}(v)$, respectively, of length l, we say that u is temporally included in v ($u \preceq v$) iff $\forall\, 1 \leq i \leq l\; |\mathcal{S}(u)[i]| \leq |\mathcal{S}(v)[i]|$. In other words, $u \preceq v$ iff u contains all the types of flows centered in v and, for each of such types, at least the same number of flows of that type centered in v.

By looking at the temporal signatures of nodes in Fig. 3, we observe that node A contains at least one occurrence of all types of flows centered in node b, so $b \preceq A$. Instead, $b \npreceq B$ because there is no "2-B-IO" flow centered in node B.

Description of RI algorithm

TemporalRI is inspired by the RI-DS algorithm for subgraph isomorphism in static graphs (Bonnici et al. 2013; Bonnici and Giugno 2017), which introduces in RI the concept of compatibility domains to filter candidate pairs of query and target nodes before starting the matching. The three main steps of RI are: (1) computation of compatibility domains, (2) computation of the ordering of query nodes for the matching, (3) matching process. In the following, we briefly describe each step.

Computation of compatibility domains

In the first step, RI computes, for each query node q, the compatibility domain $Dom(q)$ which consists of the set of nodes in the target graph that could match q based on node in- and out-degrees. Formally, a target node t is compatible to a node q iff: (1) $inDeg(q) \leq inDeg(t)$, (2) $outDeg(q) \leq outDeg(t)$. This step speeds up the matching process, because only target nodes in $Dom(q)$ are considered as possible candidates for a match to q during the search.

Ordering of query nodes

Before starting the matching, RI computes the order in which query nodes have to be processed during the search. The processing order is computed without considering the target graph. The key idea is that query nodes which both have high degree and are highly connected to nodes already present in the partial ordering come earlier in the final ordering. The first node of the ordering is randomly chosen among the nodes with highest total degree. The next node in the ordering is chosen among the remaining nodes as the one with the maximum number of neighbors already present in the ordering. In case of tie, the algorithm chooses the node with the maximum number of neighbors, which in turns are also neighbors of nodes already present in the ordering. In case of further tie, the node with highest total degree is chosen. The process is iterated until all query nodes are in the ordering.

Matching process

Following the previously defined ordering of query nodes, RI performs matching to find occurrences of the query within the target. The matching process starts with the first node of the ordering and an initially empty match $\mathcal{M}$. If a new match between a query node q and a target node t is found, the pair (q, t) is added to $\mathcal{M}$ and RI continues the search with the next node in the ordering. If all query nodes have been matched, $\mathcal{M}$ constitutes a new match of Q in T, so it can be added to the list of matches found. Whenever all query nodes have been matched or no match has been found for a query node, the algorithm performs backtracking and continues the search from the last matched node. Target nodes evaluated for a match with a query node q are chosen from a list of candidates. For the first node of the ordering the set of candidate target nodes for matching is its compatibility domain, while for any other query node q candidates are both: (1) neighbors of the target node t' that has been already matched to the previous query node q' in the ordering, (2) compatible to q. The choice to add a candidate pair (q, t) to $\mathcal{M}$ is made based on the following feasibility rules: (1) t has not been already matched, (2) for every already mapped node q' neighbor of q, there must be an edge between t and $f(q')$ in T (recall that f is the node mapping function). The latter rule ensures the consistency of the partial mapping $\mathcal{M}$ in case the new pair (q, t) is added to $\mathcal{M}$.

Description of TemporalRI

As in the original RI-DS algorithm, the three main steps of TemporalRI are: (1) computation of compatibility domains, (2) ordering of processing query edges for the matching, (3) matching process. In the following subsections we will detail each step.

For the description of the algorithm and its time complexity analysis we refer to a temporal query $Q = (V_Q, E_Q)$ with k nodes and a temporal target $T = (V_T, E_T)$ with n nodes. To help the reader understand the functioning of TemporalRI, a toy example is presented in Fig. 4.

Compatibility domains

In order to build compatibility domains of query nodes, TemporalRI exploits not only node degrees but also the temporal signature of query and target nodes. Computation of compatibility domains is outlined in Algorithm 1.

```
Algorithm 1: COMPUTEDOMAINS($Q, T$)
   Input: $Q = (V_Q, E_Q)$: query, $T = (V_T, E_T)$: target
   Output: $Dom$: compatibility domains of query nodes
   Init: $Dom(q) := \emptyset$ for all $q \in V_Q$
 1 foreach $q \in V_Q$ do
 2     $\mathcal{S}(q)$ := computeTemporalSignature($Q, q$);
 3 foreach $t \in V_T$ do
 4     $\mathcal{S}(t)$ := computeTemporalSignature($T, t$);
 5     foreach $q \in V_Q$ do
 6         $included := true$;
 7         for $i := 1$ to $|\mathcal{S}(q)|$ do
 8             if $|\mathcal{S}(q)| > |\mathcal{S}(t)|$ then
 9                 $included := false$;
10         if $outDeg(q) \leq outDeg(t) \wedge inDeg(q) \leq inDeg(t) \wedge included = true$ then
11             $Dom(q) := Dom(q) \cup \{t\}$;
12 return $Dom$
```

First, temporal signatures for query nodes are built (lines 1–2). Then, for each target node t, we compute the relative signature (line 4) and check if t is mappable to some query node (lines 7–11). t is added to the domain of a query node q iff:

1 $outDeg(q) \leq outDeg(t)$;
2 $inDeg(q) \leq inDeg(t)$;
3 $q \preceq t$.

The first two conditions apply the standard degree rules that are valid also for subgraph isomorphism in static networks. The third condition is based on the temporal signatures of both nodes.

To efficiently compute the temporal signature $\mathcal{S}(u)$ of a node u (lines 2 and 4), we scan all pairs of edges incident in u. Each pair of edges defines a flow of a certain class, that can be added to the corresponding entry of $\mathcal{S}(u)$.

In Fig. 4 the compatibility domain of query node b contains only target node A. In fact, b has both in-degree and out-degree equal to 1, while target nodes C and D have no incoming edges. Target node B is not compatible to b because there is a 2-B-IO flow centered in b that is missing for B.

Ordering of processing query edges

TemporalRI follows an iterative approach to search for a match between Q and T. Starting from an empty match, the algorithm looks for a mapping between a first query edge and a target edge. Once the first edge has been mapped, TemporalRI tries to match a second edge, and so on, until all query edges have been mapped.

To this aim, it is crucial to find an optimal ordering of processing query edges. A simple greedy approach consists in choosing at each step the query edge with the minimum number of candidates. Ideally, this choice should be done dynamically (i.e. during the search). However, this would be computationally expensive because the number of candidates also depends on the target edges and nodes we have already mapped. Instead, TemporalRI uses a static approach that consists in defining the ordering before starting the matching process. Algorithm 2 details the computation of the ordering.

```
Algorithm 2: OrderEdges($Q, Dom$)
     Input: $Q = (V_Q, E_Q)$: query, $Dom$: compatibility domains of query nodes
     Output: $\mathcal{O}$: ordering of query edges
 1   $\mu := \emptyset$; // ordering of query nodes
 2   $S_1 := \arg\max_{u \in V_Q} |Neigh(u)|$;
 3   $S_2 := \arg\min_{u \in S_1} |Dom(u)|$;
 4   $u_{max}$ := sample($S_2$); // break ties randomly
 5   $\mu$.append($u_{max}$);
 6   while $|\mu| < |V_Q|$ do
 7       $S_1 := \arg\max_{u \in V_Q} |\{v \in \mathcal{O}_N : v \in Neigh(u)\}|$;
 8       $S_2 := \arg\min_{u \in S_1} |Dom(u)|$;
 9       $u_{max}$ := sample($S_2$); // break ties randomly
10       $\mu$.append($u_{max}$);
11   $\mathcal{O} := \emptyset$;
12   for $i := 2$ to $|\mu|$ do
13       for $j := 1$ to $i - 1$ do
14           if $\mu[j] \in Neigh(\mu[i])$ then
15               $E_{ij} := \{(\mu[i], \mu[j], t) \in E_Q\} \cup \{(\mu[j], \mu[i], t) \in E_Q\}$;
16               foreach $e \in E_{ij}$ do
17                   $\mathcal{O}$.append($e$);
18   return $\mathcal{O}$
```

First, TemporalRI computes an ordering of nodes μ mainly based on their total degree (lines 1–10). This is very similar to RI's ordering of query nodes, described in the previous Section.

The first node in the ordering is the one with highest degree (line 2). In case of tie, the node with smallest compatibility domain is chosen (line 3). In case of further tie, one of the candidate nodes is chosen randomly (line 4). A similar approach is followed for choosing the next nodes (lines 6–10), selecting, at each step, the node with the highest number of connections with nodes that are already present in the current ordering (line 7). Ties are handled as previously described (lines 8–9).

In the toy example of Fig. 4, the node ordering is $\mu = [b, c, a]$. In fact, b is the node with highest degree. Nodes c and a are both linked to b, but c precedes a because c has a smaller compatibility domain.

Starting from the resulting ordering of query nodes μ, TemporalRI derives an ordering of query edges $\mathcal{O}$ (lines 11–18). For increasing values of i, we consider the ith node u in μ and all nodes that precede u in μ and are neighbors of u (lines 12–14). Following the ordering of such nodes in μ, we add all edges between them and u to $\mathcal{O}$ (lines 15–17).

Therefore, in the computation of edge ordering each query edge is considered only once and immediately added to the partial ordering.

In Fig. 4 edges e_{q_2} and e_{q_3} link nodes b and c that come before a in μ. Hence, edge e_{q_1} which connects nodes a and b, must follow both e_{q_2} and e_{q_3}, yielding the final ordering $\mathcal{O} = [e_{q_2}, e_{q_3}, e_{q_1}]$.

Matching process

Following the previously defined ordering of query edges, TemporalRI performs matching to find occurrences of the query within the target. The matching process is outlined in Algorithm 3.

```
Algorithm 3: SubgraphMatching($Q, T, Dom, \mathcal{O}$)
Input: $Q = (V_Q, E_Q)$: query, $T = (V_T, E_T)$: target, $Dom$: compatibility domains of
    query nodes, $\mathcal{O}$: ordering of query edges
Output: $Matches$: list of matches
Init: $f(u) := undefined$ for all $u \in V_Q$   //Current node mapping
Init: $g(e) := undefined$ for all $e \in E_Q$   //Current edge mapping
Init: $\mathcal{M} := \emptyset$   //Current matching
Init: $tTimes := \emptyset$   //Sorted list of timestamps of target edges
Init: $minTime := undefined$   //Minimum timestamp of target edges in the current
    mapping
Init: $maxTime := undefined$   //Maximum timestamp of target edges in the
    current mapping
Init: $Cand(e) := undefined$ for all $e \in E_Q$   //Lists of candidate target edges
Init: $CandIndex(e) := 1$ for all $e \in E_Q$   //Iterators for candidate lists
$qTimes :=$ sortedListTimestamps($E_Q$);
$e := \mathcal{O}[1]$;
$Cand(e) :=$ FindCandidates($e, T, Dom, f, \Delta, minTime, maxTime$);
$i := 1$;
while $i > 0$ do
    if $CandIndex(e) > |Cand(e)|$ then
        RestoreInfo($f, g, \mathcal{M}, tTimes, minTime, maxTime$);
        $i := i - 1$;
        $e := \mathcal{O}[i]$;
    else
        $c := Cand(e)[candIndex(e)]$;
        if $\nexists(e, c) \in \mathcal{M} \wedge rank(c.time, tTimes) = rank(e.time, qTimes)$ then
            $\mathcal{M} := \mathcal{M} \cup \{(e, c)\}$;
            if $i = |E_Q|$ then
                $Matches := Matches \cup \{\mathcal{M}\}$;
                $CandIndex(e) := CandIndex(e) + 1$;
            else
                if $f(e.source) = undefined$ then
                    $f(e.source) := c.source$;
                if $f(e.dest) = undefined$ then
                    $f(e.dest) := c.dest$;
                $g(e) = c$;
                $tTimes$.add($c.time$);
                $minTime := \min(tTimes)$;
                $maxTime := \max(tTimes)$;
                $i := i + 1$;
                $e := \mathcal{O}[i]$;
                $Cand(e) :=$
                  FindCandidates($e, T, Dom, f, \Delta, minTime, maxTime$);
                $CandIndex(e) := 1$;
        else
            $CandIndex(e) := CandIndex(e) + 1$;
return $Matches$
```

Matching is done by building a node mapping function $f : V_Q \rightarrow V_T$, an edge mapping function $g : E_Q \rightarrow E_T$ and the corresponding match $\mathcal{M}$. The list of candidates to scan during the search is stored in variable *Cand*. To ensure that the chronological order imposed by query edges is satisfied each time the partial match is extended, a sorted list *tTimes* with timestamps of mapped target edges is also stored. A similar list *qTimes* is calculated for the query before starting the match (line 1). Finally, variables *minTime* and *maxTime* contain the lower and upper bounds of timestamps of candidate target edges during the search. All such information must be kept updated during the process to guarantee the correctness of the results.

Figure 4b shows a computational graph describing all the steps performed during matching for the toy example of Fig. 4a. Each node of the computational graph is annotated with the state of the computation, i.e. the current values of all the auxiliary variables previously described. Edges correspond to transitions from one state of the computation to another one, which happen whenever a new match between a query edge and a candidate target edge is examined. Numbers attached to edges indicate the order in which transitions are performed.

The matching process starts with the first edge e in the ordering (line 2). An initial list of candidates $Cand(e)$ is calculated (line 3) and the list is scanned starting from the first element. A given candidate c can be matched with e iff: (1) c has not been mapped yet, (2) by adding c, the resulting match $\mathcal{M}$ satisfy the chronological order imposed by query edges (line 12). The latter condition can be easily verified by comparing the rank of e's timestamp in *qTimes* with the rank of c's timestamp in *tTimes*. Whenever a new match is found, the current matching $\mathcal{M}$ is updated (line 13). If e was the last query edge to match, $\mathcal{M}$ is added to the list of matches found (line 15) and the search goes on with the next candidate (line 16). Otherwise, we first update the mapping and the target's time information (lines 18–25), then we continue the search with the next query edge to match (lines 26–27) and find the set of candidates for such edge (line 28). In the computational graph of Fig. 4, transitions 1 and 2 imply an update of mapping and time information, while transition 4 updates only current matching. If candidate c does not match e, the algorithm just skips to the next candidate (line 31). In Fig. 4b, this is represented by transition 3. When all candidates for e have been examined (line 6), TemporalRI performs backtracking, i.e. restores mappings and target's time information to the previous values (line 7) and goes back to the previous query edge (lines 8–9). Backtracking implies removing: (1) the mapping for the last matched query edge e_q, (2) the last match from $\mathcal{M}$ and (3) optionally, the mapping for one or both nodes of e_q (e.g. in transition 7). Moreover, we need to remove the timestamp of the last target matched edge from *tTimes* and, if needed, update *minTime* and *maxTime*. In the computational graph of Fig. 4, backtracking corresponds to transitions 6 and 7 which restore the computation to the previous state. To guarantee that every time we do backtracking the search continues from last examined candidate for the previous query edge, TemporalRI uses a set of list iterators *CandIndex*, one for each query edge. $CandIndex(e)$ contains the position of the last examined candidate in $Cand(e)$. Every time we pass to the next candidate, the corresponding iterator is incremented (lines 16 and 31). The search ends when no more candidates are available for the first query edge. At the end of the process, TemporalRI returns the list of all matches found (line 32).

The procedure used to find the set of candidates $Cand(e)$ for a query edge $e = (u, v)$ is detailed in Algorithm 4.

Algorithm 4: FindCandidates$(e, T, Dom, f, \Delta, minTime, maxTime)$

```
Input: $e$: query edge, $T = (V_T, E_T)$: target, $Dom$: compatibility domains of query
       nodes, $f$: current node mapping, $\Delta$: time window, $minTime$: minimum
       target timestamp, $maxTime$: maximum target timestamp
Output: $Cand$: list of candidates
1  if $f(e.source) = undefined \wedge f(e.dest) = undefined$ then
2      $Cand(e) = \{(u, v, t) \in E_T : u \in Dom(e.source) \wedge v \in Dom(e.dest)\}$;
3  else
4      $\eta := \Delta - (maxTime - minTime)$;
5      $\mathcal{T} := \{minTime - \eta, ..., maxTime + \eta\}$;
6      if $f(e.source) \neq undefined \wedge f(e.dest) \neq undefined$ then
7          $Cand(e) = \{(f(e.source), f(e.dest), t) \in E_T : t \in \mathcal{T}\}$;
8      else if $f(e.source) \neq undefined$ then
9          $Cand(e) = \{(f(e.source), v, t) \in E_T : v \in Dom(e.dest) \wedge t \in \mathcal{T}\}$;
10     else
11         $Cand(e) = \{(u, f(e.dest), t) \in E_T : u \in Dom(e.source) \wedge t \in \mathcal{T}\}$;
12 return $Cand$
```

The content of *Cand*(*e*) depends on whether *u* and/or *v* have already been mapped or not. Moreover, except for the first edge in the ordering, we can also use temporal information about already mapped target edges to constraint the search for candidates. In fact, *minTime* and *maxTime* define the time interval of the current match and Δ is the maximum desired time interval between any two edges. This implies that we can look for edges whose timestamp is between *minTime* and *maxTime*, as well as below *minTime* and beyond *maxTime* of at most a quantity $\eta = \Delta - (maxTime - minTime)$ (line 4). So, the temporal interval where to search candidates is given by $\mathcal{T} = [minTime - \eta, maxTime + \eta]$ (line 5).

We distinguish four cases:

1 If both *u* and *v* are unmapped (line 1), then *Cand*(*e*) is the set of all edges between any target node compatible to *u* and any target node compatible to *v* (line 2);
2 If both *u* and *v* have been mapped (line 6), then *Cand* is the set of all possible target edges between *f*(*u*) and *f*(*v*) with timestamp $t \in \mathcal{T}$ (line 7);
3 If only *u* has been mapped (line 8), then *Cand* is the set of all possible target edges between *f*(*u*) and any node compatible to *v* with timestamp $t \in \mathcal{T}$ (line 9);
4 If only *v* has been mapped, then *Cand* is the set of all possible target edges between any node compatible to *u* and *f*(*v*) with timestamp $t \in \mathcal{T}$ (line 11).

Note that the case where both *u* and *v* are unmapped is possible iff *e* is the first edge in the ordering. In fact, by construction of the ordering $\mathcal{O}$ (Algorithm 2) each following edge must have one or two nodes in common with at least one of its predecessors in $\mathcal{O}$.

Complexity analysis

In this subsection we analyze the time complexity of TemporalRI.

Suppose, for simplicity, that $\Delta = \infty$ and $m << n$ is the average number of edges (with different timestamps) connecting two nodes both in the query and in the target. Let $\overline{d_Q}$ and $\overline{d_T}$ the average number of ingoing and outgoing edges in query and target nodes, respectively. In the worst case both Q and T are complete graphs, so $\overline{d_Q} = O(mk)$ and $\overline{d_T} = O(mn) \simeq O(n)$. Moreover, $|E_Q| = O(k^2)$ and $|E_T| = O(n^2)$.

The first step of TemporalRI is the computation of compatibility domains (Algorithm 1). The complexity of this step mainly depends on the efficiency of the computation of temporal signatures, especially in the target (line 4). To calculate the signature of a node n we have to consider all possible pairs of distinct edges incident in n. Then, for each such pair of edges we identify the flow type they form and increment the corresponding count. In the query, there are on average $\overline{d_Q} * (\overline{d_Q} - 1)/2$ possible pairs of edges to examine for each node. Therefore building the temporal signature for all k query nodes (lines 1–2) requires $O(m^2k^2k) = O(m^2k^3)$. Likewise, computing the temporal signature of a target node t (line 4) takes $O(n^2)$. However, in practice Δ is usually finite, so computing signatures is much faster because we can just consider pairs of edges for which the difference between timestamps is within Δ. Since temporal signatures have small and finite lengths, checking if a query node q is temporally included in t (lines 6–9) can be done in constant time and the check for all query nodes only requires $O(k)$. Therefore, the overall computation of domains takes $n * O(n^2) * O(k) = O(kn^3)$ time.

Then, TemporalRI computes the ordering of query edges for the matching process (Algorithm 2). The preliminary ordering of query nodes (lines 1–10) is an iterative process. The first node of the ordering can be identified in $O(k)$ time by simply looking at the total degree of all query nodes and eventually the cardinality of their domains (lines 2–3). For the next steps, we need to count, for each node u not yet in the ordering, how many neighbors of u are already in the ordering and this requires scanning the list of u's neighbors (line 7). Therefore, ordering nodes takes $O(mk * k) = O(mk^2)$ time. Ordering of query edges (lines 12–17) requires scanning the list of neighbors of each node, so it takes $O(mk^2)$ time.

The core of TemporalRI is the matching process (Algorithm 3). The computational complexity of this step mainly depends on the number of examined candidate edges for the matching, which can be retrieved in constant time from the adjacency lists of target nodes using Algorithm 4. Assuming $\Delta = \infty$, all $|E_T|$ target edges are candidates for the first query edge in the ordering. Candidates for next query edges are chosen among the target edges sharing at least one node with one or more previously matched edges and they are at most $\overline{d_T} = O(n)$. Again, in practice Δ is finite, so much less candidates are examined. Any combination of $|E_Q|$ candidates, one for each query edge, is a possible match between Q and T.

In the worst case (i.e. assuming no early backtracking), all possible combinations of candidates are examined and the number of combinations is given by $|E_T| * \overline{d_T}^{|E_Q|-1} = O(n^2) * O(m^k n^k) = O(m^k n^{2k})$. Checking if a candidate edge c matches a query edge e (Algorithm 3, line 12) requires a comparison between two ranks in sorted lists of length at most $|E_Q| = O(k^2)$ which takes $O(k^2 logk^2)$. All remaining operations consist in updating mappings and time information and require constant time. Therefore, the complexity of the matching process is $O(m^k n^{2k})$, which is also the time complexity of TemporalRI.

Experiments

In this section we assess the performance of TemporalRI considering a dataset of 7 real medium and large networks and queries of different sizes randomly extracted from such networks. As far as we know, this is the first comprehensive evaluation of temporal

Table 1 Dataset of real networks used for the experiments

Network	Nodes	Edges	Static edges	Timestamps	Resolution (s)
SFHH-CONF	403	70,261	9,889	3,509	20
AS-TOPOLOGY	34,761	155,507	114,496	32,824	1
CONTACTS-DUBLIN	10,972	415,912	52,761	76,944	20
ENRON-EMAIL	86,978	1,134,990	320,154	213,218	1
DIGG-FRIENDS	279,374	1,729,983	1,729,983	1,644,369	1
YAHOO-MESSAGES	100,001	3,157,315	898,174	1,492,800	1
PROSPER-LOANS	89,269	3,343,284	3,330,225	1,259	86,400

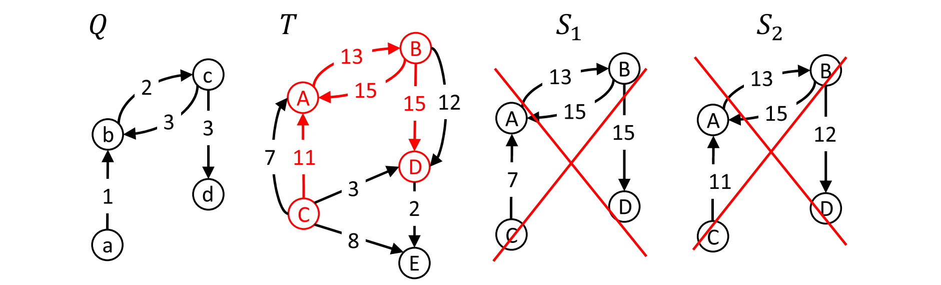


Fig. 1 Example of temporal subgraph isomorphism with $\Delta = 5$. Target T contains exactly one occurrence of query Q, which is the subgraph formed by nodes and edges colored in red. Subgraph S_1 is not a solution because contacts do not all occur within time interval Δ. Subgraph S_2 is not a solution because contacts do not follow the chronological order imposed by query edges, though all contacts occur within time interval Δ

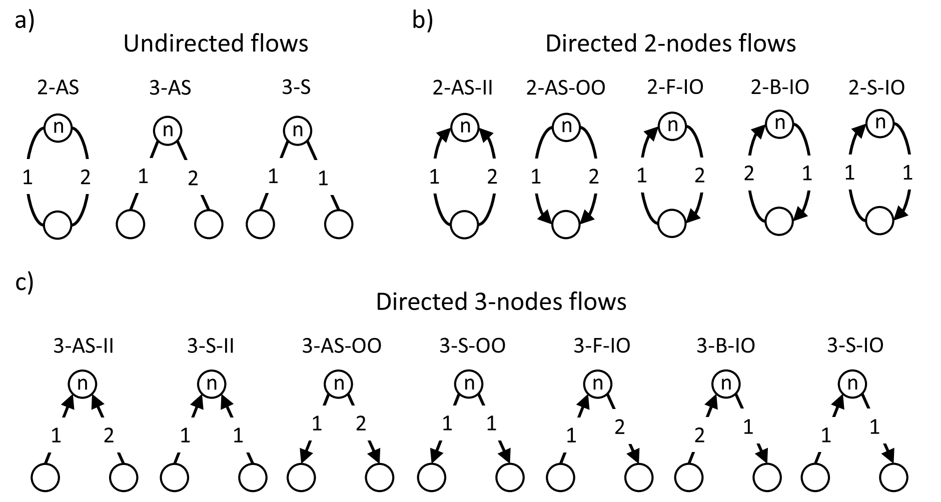
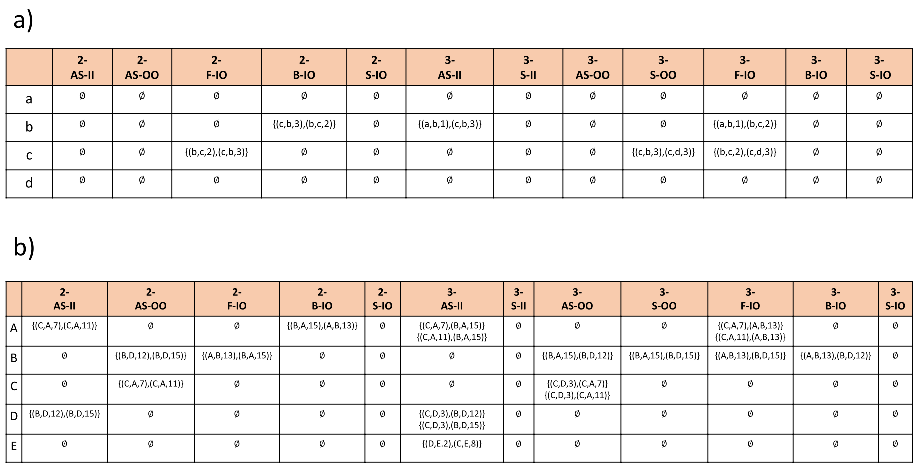


Fig. 2 Types of temporal flows. **a** Temporal flows in undirected networks; **b** temporal flows with 2 nodes in directed networks; **c** temporal flows with 3 nodes in directed networks

subgraph isomorphism algorithms, since all previous works only consider few small queries (Mackey et al. 2018; Redmond and Cunningham 2016; Sun et al. 2019b).

For the comparison with other tools, we only focused on exact motifs counting or subgraph isomorphism algorithms having the same or very similar definition of temporal queries (Mackey et al. 2018; Sun et al. 2019b; Paranjape et al. 2017). Among these,



a)

	2-AS-II	2-AS-OO	2-F-IO	2-B-IO	2-S-IO	3-AS-II	3-S-II	3-AS-OO	3-S-OO	3-F-IO	3-B-IO	3-S-IO
a	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
b	∅	∅	∅	{(c,b,3),(b,c,2)}	∅	{(a,b,1),(c,b,3)}	∅	∅	∅	{(a,b,1),(b,c,2)}	∅	∅
c	∅	∅	{(b,c,2),(c,b,3)}	∅	∅	∅	∅	∅	{(c,b,3),(c,d,3)}	{(b,c,2),(c,d,3)}	∅	∅
d	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅

b)

	2-AS-II	2-AS-OO	2-F-IO	2-B-IO	2-S-IO	3-AS-II	3-S-II	3-AS-OO	3-S-OO	3-F-IO	3-B-IO	3-S-IO
A	{(C,A,7),(C,A,11)}	∅	∅	{(B,A,15),(A,B,13)}	∅	{(C,A,7),(B,A,15)} {(C,A,11),(B,A,15)}	∅	∅	∅	{(C,A,7),(A,B,13)} {(C,A,11),(A,B,13)}	∅	∅
B	∅	{(B,D,12),(B,D,15)}	{(A,B,13),(B,A,15)}	∅	∅	∅	∅	{(B,A,15),(B,D,12)}	{(B,A,15),(B,D,15)}	{(A,B,13),(B,D,15)}	{(A,B,13),(B,D,12)}	∅
C	∅	{(C,A,7),(C,A,11)}	∅	∅	∅	∅	∅	{(C,D,3),(C,A,7)} {(C,D,3),(C,A,11)}	∅	∅	∅	∅
D	{(B,D,12),(B,D,15)}	∅	∅	∅	∅	{(C,D,3),(B,D,12)} {(C,D,3),(B,D,15)}	∅	∅	∅	∅	∅	∅
E	∅	∅	∅	∅	∅	{(D,E.2),(C,E,8)}	∅	∅	∅	∅	∅	∅

Fig. 3 Temporal signatures for: **a** nodes of query *Q* of Fig. 1, **b** nodes of target *T* of Fig. 1. Based on this, only target node *A* temporally includes query node *b*, because *A* contains, for each flow type *C*, at least the same number of flows of type *C* centered in *b*

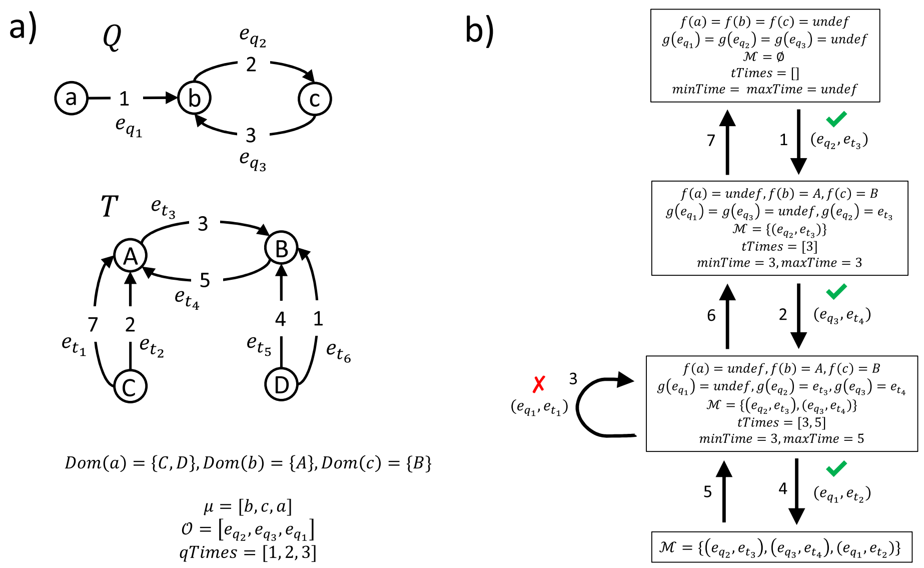

a)
Q
b)
T
$Dom(a) = \{C, D\}, Dom(b) = \{A\}, Dom(c) = \{B\}$
$\mu = [b, c, a]$
$\mathcal{O} = [e_{q_2}, e_{q_3}, e_{q_1}]$
$qTimes = [1, 2, 3]$
$f(a) = f(b) = f(c) = undef$
$g(e_{q_1}) = g(e_{q_2}) = g(e_{q_3}) = undef$
$\mathcal{M} = \emptyset$
$tTimes = []$
$minTime = maxTime = undef$
$f(a) = undef, f(b) = A, f(c) = B$
$g(e_{q_1}) = g(e_{q_3}) = undef, g(e_{q_2}) = e_{t_3}$
$\mathcal{M} = \{(e_{q_2}, e_{t_3})\}$
$tTimes = [3]$
$minTime = 3, maxTime = 3$
$f(a) = undef, f(b) = A, f(c) = B$
$g(e_{q_1}) = undef, g(e_{q_2}) = e_{t_3}, g(e_{q_3}) = e_{t_4}$
$\mathcal{M} = \{(e_{q_2}, e_{t_3}), (e_{q_3}, e_{t_4})\}$
$tTimes = [3, 5]$
$minTime = 3, maxTime = 5$
$\mathcal{M} = \{(e_{q_2}, e_{t_3}), (e_{q_3}, e_{t_4}), (e_{q_1}, e_{t_2})\}$
(e_{q_2}, e_{t_3})
(e_{q_3}, e_{t_4})
(e_{q_1}, e_{t_2})
(e_{q_1}, e_{t_1})

Fig. 4 Toy example describing the matching process performed by TemporalRI. **a** A query *Q*, a target *T* and the values of all variables computed before starting the matching process, including compatibility domains *Dom*, ordering μ and $\mathcal{O}$ of processing query nodes and edges, respectively, and sorted list *qTimes* of query timestamps. **b** Computational graph describing all steps performed by the algorithm during matching. Each node represent a state of the process and is annotated with the values of all the auxiliary variables described in Algorithm 3. Edges correspond to transitions from one state of the computation to another one. Numbers indicate the order in which transitions from one state to another one in the SST are done

Mackey's algorithm (Mackey et al. 2018) was the only one we managed to compare with TemporalRI. However, we had to slightly modify the original implementation of Mackey's algorithm, because the latter does not work correctly with queries and/or networks having simultaneous contacts. For all other tools, it was impossible to make any comparison. Sun et al. (2019b) did not provide any implementation of their algorithm. SNAP temporal algorithm (Paranjape et al. 2017) works only for 2-node and specific 3-node queries, namely 3-edge stars and 3-edge cliques. The general algorithm described in their paper is not actually temporal, because it performs subgraph matching in a static graph and then, in a post-processing step, it removes the occurrences that do not match the temporal constraints.

TemporalRI has been implemented in Java. All experiments have been performed on an Intel Core i5-8259U with 16 GB of RAM. The source code of TemporalRI is available on Github at https://github.com/GMicale/TemporalRI. For reproducibility purpose, in the same Github repository, we also make available the source code of the modified version of Mackey's algorithm together with the temporal networks and queries used in our experiments.

Dataset and setup

Temporal networks for the experiments were downloaded from Network Repository (Rossi and Ahmed 2015; http://networkrepository.com). Table 1 reports, for each network, the number of nodes, the number of edges, the number of static edges (i.e. ignoring timestamps), the number of timestamps and the resolution, i.e. the minimum difference between consecutive timestamps.

SFHH-CONF is a proximity network describing the face-to-face interactions of 405 participants to the 2009 SFHH conference in Nice, France (Génois and Barrat 2018). AS-TOPOLOGY is a peer-to-peer communication network of Autonomous Systems (AS) with data collected between February and March 2010. CONTACT-DUBLIN is a dynamic contact network of people participating to the Infectious SocioPatterns event that took place at the Science Gallery in Dublin, Ireland (Isella et al. 2011). EMAIL-ENRON describes e-mail exchanges between employees of the Enron corporation between 1999 and 2003 (Cohen 2009). DIGG-FRIENDS is a network of friendship links between users of the American news aggregator web service Digg collected over a period of one month in 2009 (Hogg and Lerman 2012). YAHOO-MESSAGES contains email exchanges between users of Yahoo Mail in a one-month interval in 2010. PROSPER-LOANS is a directed network of transactions occurring from November 2005 to September 2011 between members of Prosper.com, a website where people can either invest in personal loans or request to borrow money (Redmond and Cunningham 2013a). Edges link lenders to borrowers and are timestamped with the origination date of the loan.

Queries with $k = 3, 6, 9$ edges were randomly extracted from each network. The query extraction procedure is iterative and works as follows. First, an edge is randomly sampled from the target. At each iteration, an edge having at least one node in common with (i.e. incident to) at least one of the already sampled edges is added to the partial query. To avoid any bias, this is done by picking uniformly at random one of such incident edges. If no incident edge exists, the extraction process is repeated from scratch starting from a new edge. The procedure ends when a query with k edges is obtained. Finally, edge timestamps of the extracted query are randomly sampled with replacement from the set $\{1, 2, \ldots, k\}$. We extracted 100 queries for each value of k from each network, for a total number of 2100 queries.

For each network and each query, we ran TemporalRI and Mackey's algorithm with different values of time interval Δ. The choice of Δ is not easy, because Δ depends both on the size of the query and on the rate at which interactions have been measured to build the target network. We considered $\Delta = kr, 2kr, 3kr$, where r is the resolution of the network. In each experiment, we set a timeout of 30 min for both algorithms.

Experimental results

To compare the performance of TemporalRI and Mackey's, we first focused on the experiments completed by both algorithms before the timeout. In Figs. 5, 6 and 7 we show boxplots of the running times obtained by TemporalRI and Mackey's on the set of queries completed by both algorithms with $k = 3$, $k = 6$ and $k = 9$ edges, respectively, on varying Δ. In all these experiments the two algorithms returned the same number of

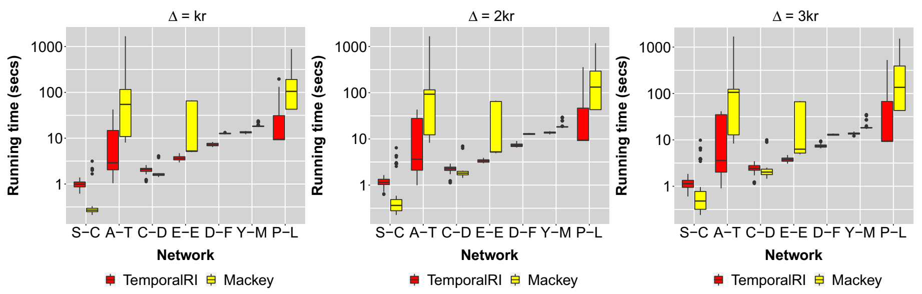


Fig. 5 Comparison between TemporalRI and Mackey's algorithm on the set of queries with $k = 3$ contacts on varying Δ. Boxplots of running times are built on the set of experiments in which both algorithms ended before the timeout. Network names are abbreviated

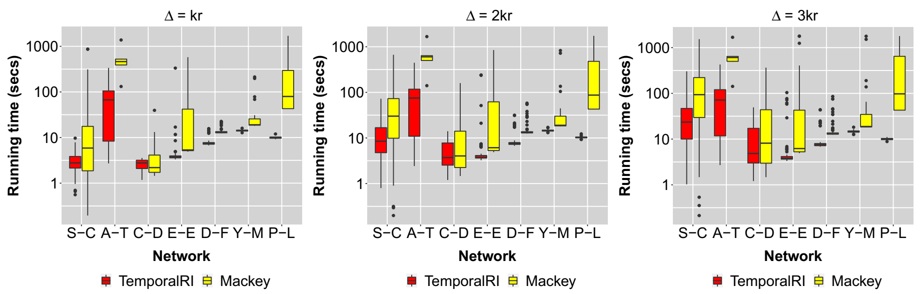


Fig. 6 Comparison between TemporalRI and Mackey's algorithm on the set of queries with $k = 6$ contacts on varying Δ. Boxplots of running times are built on the set of experiments in which both algorithms ended before the timeout. Network names are abbreviated

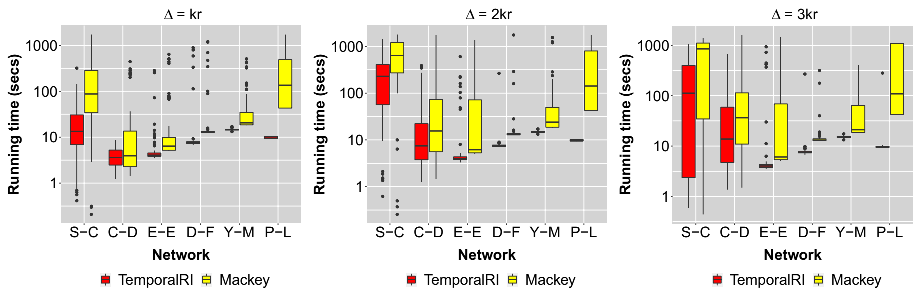


Fig. 7 Comparison between TemporalRI and Mackey's algorithm on the set of queries with $k = 9$ contacts on varying Δ. Boxplots of running times are built on the set of experiments in which both algorithms ended before the timeout. Network names are abbreviated

occurrences. For $k = 9$, Mackey's didn't manage to complete any query before the timeout in the AS-TOPOLOGY network (see also Table 4), therefore in this case boxplots for the two algorithms are not drawn.

Results show that TemporalRI is almost always faster on average than Mackey's algorithm, independently of Δ and the size of query. The speedup is higher for larger queries and denser networks. We believe that this is due to two aspects that distinguish TemporalRI from Mackey's algorithm: (1) the effective filtering of candidates done using temporal flows, (2) the ordering of query edges based on the degrees of query nodes. Indeed, in Mackey's algorithm there is no a-priori filtering of candidate edges. This may affect a lot the running time, especially in the case of large queries, where there are more combinations of target edges to explore. In this scenario, temporal flows prove to be features that are both fast to compute and effective to speedup matching. Concerning the ordering of processing query edges, Mackey's algorithm just order them based on their timestamps. This strategy does not take into account the structure of the query. However, nodes with high degree and edges incident in such nodes tend to have less candidates. Though filtering seems to have the greatest impact on performance, there are several works in the context of subgraph matching in static graphs showing that ordering can be important too (Carletti et al. 2017; Bonnici et al. 2013; Bonnici and Giugno 2017; Han et al. 2019; Bi et al. 2016). In general, for both algorithms Δ does not seem to impact a lot on the running times.

In order to analyze the behaviour of TemporalRI and Mackey's on the whole set of experiments, we calculated, for every combination of values of query size k and Δ, in each network, the percentage of queries completed before the timeout by: (1) both algorithms; (2) only one of them; (3) none of them. Results are reporeted on Tables 2, 3 and 4.

As expected, when the query size increases the number of completed tests drops down for both algorithms, but TemporalRI always manages to complete more experiments than Mackey's. Mackey's algorithm seems to suffer in AS-TOPOLOGY and PROSPER-LOANS. This could be related to the low number of distinct timestamps in these two networks, yielding to an increased average number of occurrences of tested queries, even for small values of Δ. Once again, except for ENRON-EMAIL, Δ does not seem to impact on the percentage of experiments completed before the timeout. Overall, out of 6300 experiments, there were just 11 cases (0.17%) where only Mackey's algorithm completed before the timeout, while in 909 queries (14.43%) only TemporalRI ended before the timeout. In 4675 (74.2%) queries and in 705 (11.19%) queries both algorithms and none of the two completed before the timeout, respectively.

Finally, we compared the performance of TemporalRI and Mackey's based on the topology of the query. For each query size k, results were grouped according to the topology class and the number of nodes of the query. We specified five distinct topology classes: paths, stars, trees, cliques and cyclic graphs. The notation "x-class" indicate a cluster of queries with x nodes and having one of the above topology classes (e.g. 3-path, 4-tree, etc.). Grouping was done ignoring edge direction, timestamps and possible multiple edges between two nodes. For each cluster, we computed the percentage of tests in which TemporalRI was the fastest algorithm and we did the same for Mackey's. We excluded from this analysis all the

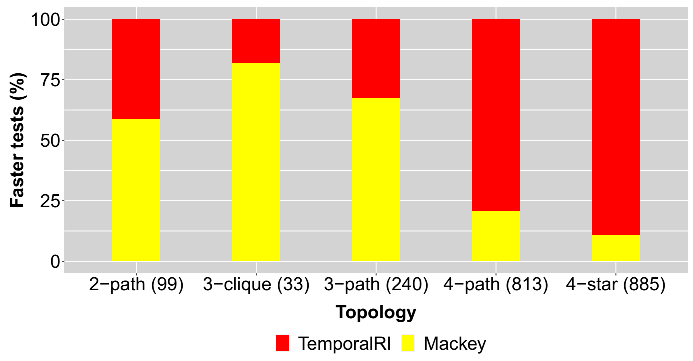


Fig. 8 Percentage of tests on queries with $k = 3$ edges in which TemporalRI or Mackey's algorithm is the fastest. Results are clustered by subgraph topology. Number of tests in each cluster are indicated in parentheses. Experiments completed neither by TemporalRI nor by Mackey's algorithm are excluded from the analysis

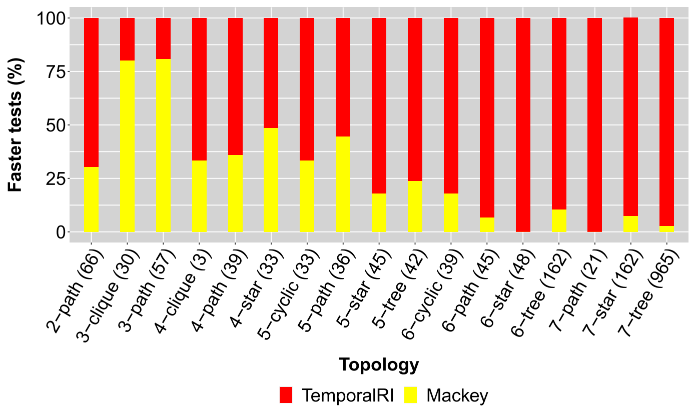


Fig. 9 Percentage of tests on queries with $k = 6$ edges in which TemporalRI or Mackey's algorithm is the fastest. Results are clustered by subgraph topology. Number of tests in each cluster are indicated in parentheses. Experiments completed neither by TemporalRI nor by Mackey's algorithm are excluded from the analysis

experiments in which none of the two algorithms ended before the timeout. Percentages for each cluster of queries are reported in Figs. 8, 9 and 10.

Interestingly, in all three cases ($k = 3, 6, 9$), Mackey's algorithm seems to be generally faster than TemporalRI in queries with very few (2 or 3) nodes, while TemporalRI is almost always faster than Mackey's in queries with many (from 4-5 on) nodes.

To sum up, TemporalRI is generally faster than Mackey's algorithm, especially in large queries and targets that are either large or have a limited number of timestamps. In queries with very few nodes and many multiple contacts between nodes, Mackey's algorithm behaves generally better than TemporalRI. In all other cases, the latter is faster, independently of the topology of the query.

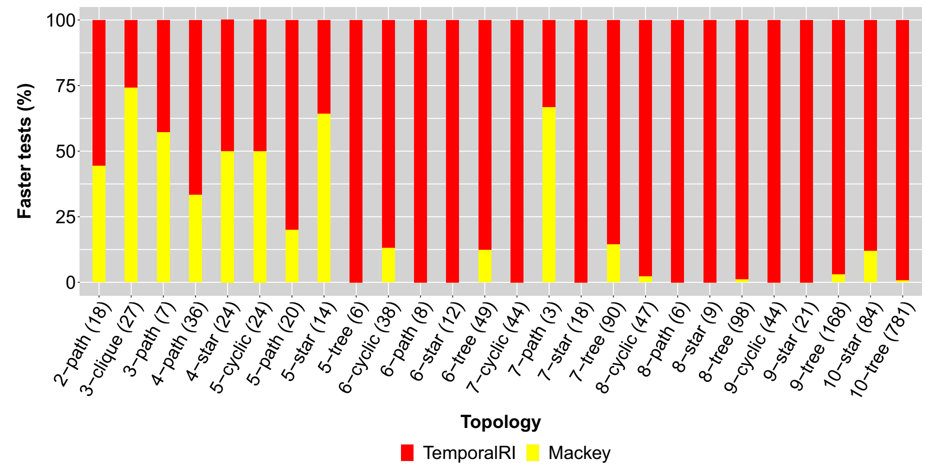


Fig. 10 Percentage of tests on queries with $k = 9$ edges in which TemporalRI or Mackey's algorithm is the fastest. Results are clustered by subgraph topology. Number of tests in each cluster are indicated in parentheses. Experiments completed neither by TemporalRI nor by Mackey's algorithm are excluded from the analysis

Table 2 Number of experiments with $k = 3$ completed by both algorithms, TemporalRI only, Mackey's algorithm only and none of them

	S-C (%)	A-T (%)	C-D (%)	E-E (%)	D-F (%)	Y-M (%)	P-L (%)	Total (%)
$\Delta = kr$								
Both	100	64	100	98	100	100	62	89.14
TemporalRI	0	27	0	2	0	0	37	9.43
Mackey's	0	0	0	0	0	0	0	0
None	0	9	0	0	0	0	1	1.43
$\Delta = 2kr$								
Both	100	63	100	98	100	100	62	89.0
TemporalRI	0	28	0	2	0	0	37	9.57
Mackey's	0	0	0	0	0	0	0	0
None	0	9	0	0	0	0	1	1.43
$\Delta = 3kr$								
Both	100	64	100	98	100	100	62	89.14
TemporalRI	0	27	0	2	0	0	37	9.43
Mackey's	0	0	0	0	0	0	0	0
None	0	9	0	0	0	0	1	1.43

Conclusions

In this paper we focused on Temporal Subgraph Isomorphism (TSI) problem, which consists in finding all the occurrences of a small temporal graph (the query) in a larger temporal graph (the target), such that timestamps of target edges follows the same chronological order of corresponding matched query edges and all interactions occur within a user-defined time interval Δ. TSI can be considered a baseline framework to study related problems, such as motif search, anomaly detection and node centralities. We illustrated a novel algorithm for the TSI problem, called TemporalRI, which introduces the concept of temporal flows to filter the set of candidates before starting the matching. Experiments performed on a dataset of medium and large networks

Table 3 Number of experiments with $k = 6$ completed by both algorithms, TemporalRI only, Mackey's algorithm only and none of them

	S-C (%)	A-T (%)	C-D (%)	E-E (%)	D-F (%)	Y-M (%)	P-L (%)	Total (%)
$\Delta = kr$								
Both	100	5	100	98	100	100	18	74.43
TemporalRI	0	26	0	2	0	0	71	14.14
Mackey's	0	0	0	0	0	0	0	0
None	0	69	0	0	0	0	11	11.43
$\Delta = 2kr$								
Both	98	5	100	92	100	100	18	73.29
TemporalRI	2	20	0	5	0	0	70	13.86
Mackey's	0	0	0	1	0	0	0	0.14
None	0	75	0	2	0	0	12	12.71
$\Delta = 3kr$								
Both	98	5	100	77	100	100	18	71.14
TemporalRI	2	16	0	7	0	0	69	13.43
Mackey's	0	0	0	3	0	0	0	0.43
None	0	79	0	13	0	0	13	15

Table 4 Number of experiments with $k = 9$ completed by both algorithms, TemporalRI only, Mackey's algorithm only and none of them

	S-C (%)	A-T (%)	C-D (%)	E-E (%)	D-F (%)	Y-M (%)	P-L (%)	Total (%)
$\Delta = kr$								
Both	81	0	100	83	100	99	13	68
TemporalRI	19	5	0	12	0	1	85	17.43
Mackey's	0	0	0	1	0	0	0	0.14
None	0	95	0	4	0	0	2	14.43
$\Delta = 2kr$								
Both	43	0	100	74	96	97	13	60.43
TemporalRI	44	5	0	17	0	3	85	22
Mackey's	0	0	0	3	0	0	0	0.43
None	13	95	0	6	4	0	2	17.14
$\Delta = 3kr$								
Both	18	0	88	69	95	91	12	53.29
TemporalRI	18	5	6	21	1	9	84	20.57
Mackey's	0	0	2	1	0	0	0	0.43
None	64	95	4	9	4	0	4	25.71

show that TemporalRI is overall faster than the state-of-the-art Mackey's algorithm independently of Δ and the number of query edges. The speedup is higher for large queries and large networks.

It is worth noting that the concept of temporal flow and the techniques introduced to prune the search space are general and therefore can be, in principle, plugged into any subgraph matching algorithm to handle the TSI problem.

In this paper we decided not to consider labeled temporal graphs, i.e. graphs containing node and edge labels, even though both TemporalRI and Mackey's algorithm are designed to handle such graphs. This is due to the fact that adding labels just requires few additional checks without changing the structure of both algorithms and affecting results.

For the future, we plan to optimize TemporalRI and implement it on top of a SPARK framework to deal with very large networks. We also aim to adapt TemporalRI as a general algorithm for counting temporal motifs. This implies the need of a clever strategy to identify and count occurrences of different temporal subgraphs without scanning the target several times.

Abbreviations
TSI: Temporal subgraph isomorphism; AS: Asynchronous; S: Synchronous; F: Forward; B: Backward; II: Input–input; IO: Input–output; OO: Output–output.

Acknowledgements
Not applicable.

Author's contributions
All authors conceived the project. GL and GM developed the algorithm. GM performed the experiments and was a major contributor in writing the manuscript. All authors read and approved the final manuscript.

Funding
AP and AF have been partially supported by the following research Project: PO-FESR Sicilia 2014-2020 "DiOncoGen: Innovative diagnostics" (CUP: G89J18000700007).

Availability of data and materials
Source codes of TemporalRI and the modified version of Mackey's algorithm, networks and queries used in our experiments are available on Github at https://github.com/GMicale/TemporalRI.

Declaration

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Department of Clinical and Experimental Medicine, University of Catania, Catania, Italy. [2]Department of Maths and Computer Science, University of Catania, Catania, Italy.

Received: 3 March 2021 Accepted: 4 July 2021
Published online: 17 July 2021

References
Aggarwal C, Subbian K (2014) Evolutionary network analysis: a survey. ACM Comput Surv (CSUR) 47(1):10
Bi F, Chang L, Lin X, Qin L, Zhang W (2016) Efficient subgraph matching by postponing cartesian products. SIGMOD '16, pp 1199–1214
Bonnici V, Giugno R (2017) On the variable ordering in subgraph isomorphism algorithms. IEEE/ACM Trans Comput Biol Bioinform 14(1):193–203
Bonnici V, Giugno R, Pulvirenti A, Shasha D, Ferro A (2013) A subgraph isomorphism algorithm and its application to biochemical data. BMC Bioinform 14(S13):1–13
Carletti V, Foggia P, Saggese A, Vento M (2017) Introducing vf3: a new algorithm for subgraph isomorphism. In: Graph-based representations in pattern recognition, pp 128–139
Carley KM, Diesner J, Reminga J, Tsvetovat M (2007) Toward an interoperable dynamic network analysis toolkit. Decis Support Syst 43(4):1324–1347
Casteigts A, Flocchini P, Quattrociocchi W, Santoro N (2011) Time-varying graphs and dynamic networks. In: Ad-hoc, mobile, and wireless networks, pp 346–359
Cohen WW (2009) Enron email dataset (2005). http://www.cs.cmu.edu/enron
Cordella LP, Foggia P, Sansone C, Vento M (2004) A (sub)graph isomorphism algorithm for matching large graphs. IEEE Trans Pattern Anal Mach Intell 26(10):1367–1372
Crawford J, Milenkovic T (2018) Cluenet: clustering a temporal network based on topological similarity rather than denseness. PLoS ONE 13(5):1–25
Divakaran A, Mohan A (2020) Temporal link prediction: a survey. New Gener Comput 38:213–258
Génois M, Barrat A (2018) Can co-location be used as a proxy for face-to-face contacts? EPJ Data Sci 7(11):1–18
Han W, Lee J, Lee J-H (2013) Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data. SIGMOD '13, pp 337–348 (2013)
Han M, Kim H, Gu G, Park K, Han W (2019) Efficient subgraph matching: harmonizing dynamic programming, adaptive matching order, and failing set together. In: Proceedings of the 2019 international conference on management of data. SIGMOD '19, pp 1429–1446

Hiraoka T, Masuda N, Li A, Jo H (2020) Modeling temporal networks with bursty activity patterns of nodes and links. Phys Rev Res 2(2):023073

Hogg T, Lerman K (2012) Social dynamics of digg. EPJ Data Sci 1(1):1–26

Holme P, Saramaki J (2012) Temporal networks. Phys Rep 519(3):97–125

Holme P, Saramaki J (2019) Temporal network theory. Springer, Cham

Hulovatyy Y, Chen H, Milenkovic T (2015) Exploring the structure and function of temporal networks with dynamic graphlets. Bioinformatics 31(12):171–180

Isella L, Stehlé J, Barrat A, Cattuto C, Pinton JF, Van den Broeck W (2011) What's in a crowd? Analysis of face-to-face behavioral networks. J Theor Biol 271(1):166–180

Kim K, Seo I, Han W, Lee J, Hong S, Chafi H, Shin H, Jeong G (2018) Turboflux: a fast continuous subgraph matching system for streaming graph data. In: Proceedings of the 2018 international conference on management of data. SIGMOD '18, pp 411–426

Kovanen L, Karsai M, Kaski K, Kertész J, Saramaki J (2011) Temporal motifs in time-dependent networks. J Stat Mech Theory Exp 2011(11):11005

Liu P, Benson AR, Charikar M (2019) Sampling methods for counting temporal motifs. In: Proceedings of the twelfth ACM international conference on web search and data mining. WSDM '19, pp 294–302

Locicero G, Micale G, Pulvirenti A, Ferro A (2021) TemporalRI: a subgraph isomorphism algorithm for temporal networks. In: Complex networks and their applications IX, pp 675–687

Lv L, Zhang K, Zhang T, Bardou D, Zhang J, Cai Y (2019) Pagerank centrality for temporal networks. Phys Lett A 383(12):1215–1222

Mackey P, Porterfield K, Fitzhenry E, Choudhury S, Chin G (2018) A chronological edge-driven approach to temporal subgraph isomorphism. In: 2018 IEEE international conference on big data (big data), pp 3972–3979

Masuda N, Holme P (2020) Small inter-event times govern epidemic spreading on networks. Phys Rev Res 2(2):023163

Masuda N, Lambiotte R (2020) A guide to temporal networks, 2nd edn. World Scientific, Singapore

Network Repository: an interactive scientific network data repository (2021). http://networkrepository.com. Accessed 4 Jan 2021

Paranjape A, Benson AR, Leskovec J (2017) Motifs in temporal networks. In: Proceedings of the tenth ACM international conference on web search and data mining. WSDM '17, pp 601–610

Petit J, Gueuning M, Carletti T, Lauwens B, Lambiotte R (2018) Random walk on temporal networks with lasting edges. Phys Rev E 98(5):052307

Redmond U, Cunningham P (2013a) A temporal network analysis reveals the unprofitability of arbitrage in the prosper marketplace. Expert Syst Appl 40(9):3715–3721

Redmond U, Cunningham P (2013b) Temporal subgraph isomorphism. In: Proceedings of the 2013 IEEE/ACM international conference on advances in social networks analysis and mining. ASONAM '13, pp 1451–1452

Redmond U, Cunningham P (2016) Subgraph isomorphism in temporal networks. arXiv:1605.02174

Rocha LEC, Masuda N, Holme P (2017) Sampling of temporal networks: methods and biases. Phys Rev E 96(5):052302

Rossetti G, Cazabet R (2018) Community discovery in dynamic networks: a survey. ACM Comput Surv 51(2):1–37

Rossi RA, Ahmed NK (2015) The network data repository with interactive graph analytics and visualization. In: Proceedings of the twenty-ninth AAAI conference on artificial intelligence. AAAI'15, pp 4292–4293

Singh EA, Cherifi H (2020) Centrality-based opinion modeling on temporal networks. IEEE Access 8:1945–1961

Sun S, Luo Q (2020) Subgraph matching with effective matching order and indexing. IEEE Trans Knowl Data Eng 1:1–14

Sun X, Tan Y, Wu Q, Chen B, Shen C (2019) Tm-miner: Tfs-based algorithm for mining temporal motifs in large temporal network. IEEE Access 7:49778–49789

Sun X, Tan Y, Wu Q, Wang J, Shen C (2019) New algorithms for counting temporal graph pattern. Symmetry 11(10):1188

Tizzani M, Lenti S, Ubaldi E, Vezzani A, Castellano C, Burioni R (2018) Epidemic spreading and aging in temporal networks with memory. Phys Rev E 98(6):062315

Torricelli M, Karsai M, Gauvin L (2020) weg2vec: event embedding for temporal networks. Sci Rep 10:7164

Tsalouchidou I, Baeza-Yates R, Bonchi F, Liao K, Sellis T (2020) Temporal betweenness centrality in dynamic graphs. Int J Data Sci Anal 9:257–272

Williams OE, Lillo F, Latora V (2019) Effects of memory on spreading processes in non-Markovian temporal networks. New J Phys 21(4):043028

Publisher's Note