An algebraic approach to temporal network analysis

Vladimir Batagelj
University of Ljubljana, FMF, Department of Mathematics,
Jadranska 19, 1 000 Ljubljana, Slovenia
vladimir.batagelj@fmf.uni-lj.si

Selena Praprotnik
University of Ljubljana, FMF, Department of Mathematics,
Jadranska 19, 1 000 Ljubljana, Slovenia
selena.praprotnik@gmail.com

May 30, 2014/April 20, 2014

Abstract

To describe temporal networks we introduce the notion of temporal quantities. We define the addition and multiplication of temporal quantities in a way that can be used for the definition of addition and multiplication of temporal networks with zero latency. The corresponding algebraic structures are semirings. We developed fast algorithms for the proposed operations. They are available as a Python library TQ and a program Ianus. The proposed approach enables us to treat as temporal quantities also other network characteristics such as degrees, connectivity components, centrality measures, Pathfinder skeleton, etc. It is an alternative to the usual approach to temporal network analysis based on time slices. To illustrate the developed tools we present some results from the analysis of Franzosi's violence network and Corman's Reuters terror news network.

Keywords: temporal network, semiring, algorithm, network measures, Python library, violence.

Math.Subj.Class (2010): 91D30; 16Y60; 90B10; 68R10; 93C55

1 Introduction

In a *temporal network*, the presence and activity of nodes and links can change through time. The time dimension was added to networks in different disciplines. The earliest were the transportation network analysis (Bell and Iida [4], Correa et al. [6]), project scheduling (CPM, Pert) in Operations Research (Moder [19]) and constraints networks in Artificial Intelligence (Dechter [11]). There are also qualitative approaches to temporal networks. See for example Allen [1] and Vilain et al. [26]. For statistical approaches see Kolaczyk's book [17] and Snijders' Siena page [22]. In this paper we shall stick to the quantitative approach based on temporal quantities presented in section 3.

In the last two decades the interest for analysis of temporal networks increased partially motivated by travel-support services and analysis of sequences of events (e-mails, news, phone calls, etc.). The approaches and results are surveyed by Holme and Saramäki in the paper [14] and the book [15]. Another overview was produced by Casteigts et al. [10] (see also [9]) based on their formalization of temporal networks – time-varying graphs or TVGs. A step forward to data analysis of temporal networks was recently made by Kontoleon et al. [18].

In the paper we first present the basic notions about temporal networks. In section 3 we introduce the temporal quantities and propose an algebraic approach, based on semirings, to the analysis of temporal networks. In the following sections we show that most of the traditional network analysis concepts and algorithms such as degrees, clustering coefficient, closeness, betweenness, weak and strong connectivity, PathFinder skeleton, etc. can be straightforwardly extended to their temporal versions. The proposed approach is an alternative to the usual approach to temporal network analysis based on time slices.

2 Description of temporal networks

For the description of temporal networks we propose an elaborated version of the approach used in Pajek [21]. It is similar to TVGs. Pajek supports two types of descriptions of temporal networks based on *presence* and on *events* (Pajek 0.47, July 1999). Here, we describe only the approach to capturing the presence of nodes and links.

A *temporal network* $\mathcal{N}_T = (\mathcal{V}, \mathcal{L}, \mathcal{P}, \mathcal{W}, \mathcal{T})$ is obtained by attaching the *time*, $\mathcal{T}$, to an ordinary network, where $\mathcal{T}$ is a set of *time points*, $t \in \mathcal{T}$. $\mathcal{V}$ is the set of nodes, $\mathcal{L}$ is the set of links, $\mathcal{P}$ is the set of node properties, and $\mathcal{W}$ is the set of link weights [3]. The time $\mathcal{T}$ is usually either a subset of integers, $\mathcal{T} \subseteq \mathbb{Z}$, or a subset of reals, $\mathcal{T} \subseteq \mathbb{R}$. In Pajek $\mathcal{T} \subseteq \mathbb{N}$.

In a temporal network, nodes $v \in \mathcal{V}$ and links $l \in \mathcal{L}$ are not necessarily present or active in all time points. Let $T(v)$, $T \in \mathcal{P}$, be the activity set of time points for node v; and $T(l)$, $T \in \mathcal{W}$, the activity set of time points for link l. $T(v)$ and $T(l)$ are usually described as a sequence of intervals. The following *consistency* condition is imposed: If a link $l(u, v)$ is active in time point t then its end-nodes u and v should be active in time t. Formally we express this by

$$T(l(u, v)) \subseteq T(u) \cap T(v).$$

The activity set $T(e)$ of node/link e is usually described as a sequence of time intervals $([s_i, f_i])_{i=1}^{k}$, where s_i is the *start*ing time and f_i is the *finish*ing time.

We denote a network consisting of links and nodes active in time $t \in \mathcal{T}$ by $\mathcal{N}(t)$ and call it the (network) *time slice* or *footprint* of t. Let $\mathcal{T}' \subset \mathcal{T}$ (for example, a time interval). The notion of a time slice is extended to $\mathcal{T}'$ by

$$\mathcal{N}(\mathcal{T}') = \bigcup_{t \in \mathcal{T}'} \mathcal{N}(t).$$

2.1 Journeys – walks in temporal networks

When dealing with walks in temporal networks we usually consider two additional information – weights on links:

- the *transition time* or *latency* $\tau \in \mathcal{W}$; $\tau \colon \mathcal{L} \to \mathbb{R}_0^+$. $\tau(l)$ is equal to the time needed to traverse the link l. If the function τ is not given we can assume $\tau(l) = 0$ for all links l.
- the *value* (length, cost, etc.) $w \in \mathcal{W}$; $w \colon \mathcal{L} \to \mathbb{R}$. If the function w is not given we can assume $w(l) = 1$ for all links l.

In applications related to flows in networks we need an additional weight

- the *capacity* $c \in \mathcal{W}$; $c \colon \mathcal{L} \to \mathbb{R}_0^+$. $c(l)$ is equal to the maximum quantity of items that can be transfered in a time unit over the link l. If the function c is not given we can assume $c(l) = \infty$ (no limits) for all links l.

In real-life networks the values $\tau_l(t)$, $w_l(t)$ and $c_l(t)$ of functions τ, w and c for a given link $l \in \mathcal{L}$ can also vary through time. For example, $\tau_l(t)$ can depend on the overall traffic in the network. In the following, see Section 3, we shall assume that they are constant on each time interval.

A *temporal walk* or *journey* $\sigma(v_0, v_k; t_0)$ from the (source) node v_0 to the (destination) node v_k starting at a time $t_0 \in T(v_0)$ is a finite sequence

$$(t_0, v_0, (t_1, l_1), v_1, (t_2, l_2), v_2, \ldots, v_{k-2}, (t_{k-1}, l_{k-1}), v_{k-1}, (t_k, l_k), v_k)$$

where $l_i \in \mathcal{L}$, $t_i \in T$, $i = 1, 2, \ldots k$. The triples $v_{i-1}, (t_i, l_i)$ tell that in the node v_{i-1} at time t_i the link l_i was selected for the next transition. The sequence σ has to satisfy the conditions: the link l_i links node v_{i-1} to node v_i and is active during the transition:

(a) $l_i(v_{i-1}, v_i)$

(b) $t'_{i-1} \leq t_i$

(c) $[t_i, t'_i] \subseteq T(l_i)$

for $i = 1, 2, \ldots k$; where $t'_i = t_i + \tau(l_i)$ and $t'_0 = t_0$.

The number k is called a *length* of the walk σ. The *time used* by the walk σ is equal to

$$t(\sigma) = t'_k - t_0$$

and its *value* is

$$w(\sigma) = \sum_{i=1}^{k} w(l_i, t_i, t'_i)$$

where $w(l_i, t_i, t'_i)$ is the value of the link l_i in the time interval $[t_i, t'_i]$ – it is a function combining the values of $w_{l_i}(t)$ on the time interval $[t_i, t'_i]$. From the condition (c) it follows by the consistency that $t_i \in T(v_{i-1})$ and $t'_i \in T(v_i)$.

A temporal walk is *regular* if also

$$[t'_{i-1}, t_i] \subseteq T(v_{i-1}), \text{ for } i = 1, 2, \ldots k.$$

While waiting for the next step (transition) in node v_{i-1} this node should be all the time active.

3 Temporal quantities

Besides the presence/absence of nodes and links also their properties can change through time. To describe them we introduce a notion of a *temporal quantity*

$$a(T_a) = \begin{cases} a(t) & t \in T_a \\ \text{⌘} & t \in \mathcal{T} \setminus T_a \end{cases}$$

where $a(t)$ is the value of a in an instant t, and ⌘ denotes the value *undefined*.

We assume that the values of temporal quantities belong to a set A which is a semiring $(A, +, \cdot, 0, 1)$ for binary operations $+ : A \times A \to A$ and $\cdot : A \times A \to A$ [2]. This means that $(A, +, 0)$ is an Abelian monoid – the addition $+$ is associative and commutative, and has 0 as its neutral element; and $(A, \cdot, 1)$ is a monoid – the multiplication $\cdot$ is associative and has 1 as its neutral element. Also, multiplication distributes from both sides over addition. We can extend both operations to the set $A_{\text{⌘}} = A \cup \{\text{⌘}\}$ by requiring that for all $a \in A_{\text{⌘}}$ it holds

$$a + \text{⌘} = \text{⌘} + a = a \quad \text{and} \quad a \cdot \text{⌘} = \text{⌘} \cdot a = \text{⌘}.$$

The structure $(A_{\text{⌘}}, +, \cdot, \text{⌘}, 1)$ is also a semiring.

The "default" semiring is the *combinatorial* semiring $(\mathbb{R}_0^+, +, \cdot, 0, 1)$ where $+$ and $\cdot$ are the usual addition and multiplication of real numbers. In some applications other semirings are useful.

In applications of semirings in analysis of graphs and networks the addition $+$ describes the composition of values on parallel paths and the multiplication $\cdot$ describes the composition of values on sequential paths – see Figure 1. For a combinatorial semiring these two schemes correspond to basic principles of combinatorics *Rule of Sum* and *Rule of Product* [23].

The semiring $(\overline{\mathbb{R}}^+, \min, +, \infty, 0)$ is suitable to deal with the shortest paths problem in networks; and the semiring $(\{0, 1\}, \vee, \wedge, 0, 1)$ for reachability problems.

Let $A_{\text{⌘}}(\mathcal{T})$ denote the set of all temporal quantities over $A_{\text{⌘}}$ in time $\mathcal{T}$. To extend the operations to networks and their matrices we first define the *sum* (parallel links)

$$a(T_a) + b(T_b) = s(T_s)$$

as

$$s(t) = \begin{cases} a(t) + b(t) & t \in T_a \cap T_b \\ a(t) & t \in T_a \setminus T_b \\ b(t) & t \in T_b \setminus T_a \\ \text{⌘} & \text{otherwise} \end{cases}$$

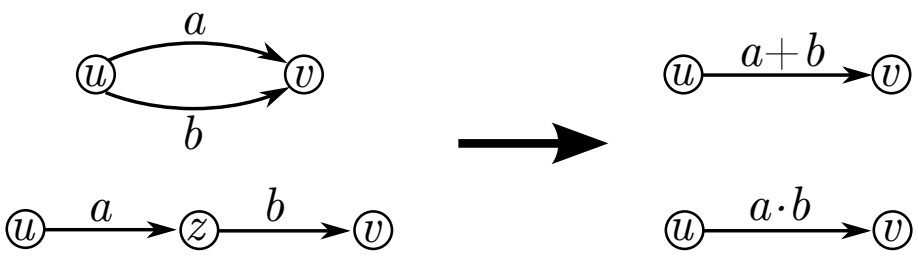

Figure 1: Semiring addition and multiplication in networks.

and $T_s = T_a \cup T_b$; and the *product* (sequential links)

$$a(T_a) \cdot b(T_b) = p(T_p)$$

as

$$p(t) = \begin{cases} a(t) \cdot b(t) & t \in T_a \cap T_b \\ \text{⌘} & \text{otherwise} \end{cases}$$

and $T_p = T_a \cap T_b$.

Let us define the temporal quantities $\mathbf{0}$ and $\mathbf{1}$ with requirements $\mathbf{0}(t) = \text{⌘}$ and $\mathbf{1}(t) = 1$ for all $t \in \mathcal{T}$. Again, the structure $(A_{\text{⌘}}(\mathcal{T}), +, \cdot, \mathbf{0}, \mathbf{1})$ is a semiring, and therefore so is the set of square matrices of order n over it for addition $\mathbf{A} \oplus \mathbf{B} = \mathbf{S}$

$$s_{ij} = a_{ij} + b_{ij}$$

and multiplication $\mathbf{A} \odot \mathbf{B} = \mathbf{P}$

$$p_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}.$$

The matrix multiplication is closely related to traveling on networks. For a value p_{ij} to be defined (different from ⌘) there should exist at least one node k such that both link (i, k) and link (k, j) exist – the transition from the node i to the node j through a node k is possible. Its contribution is $a_{ik} \cdot b_{kj}$.

In this paper we will, in the following, limit our attention to networks with *zero latency* ($\tau(l) = 0$, for all links l). This assumption implies that in journeys it holds $t'_i = t_i + \tau(l_i) = t_i$. Therefore the conditions from the definition of journeys simplify into:

(a) $l_i(v_{i-1}, v_i)$

(b) $t_{i-1} \leq t_i$

(c) $t_i \in T(l_i)$.

By consistency we also have $t_i \in T(v_{i-1})$ and $t_i \in T(v_i)$.

In networks with zero latency the transitions are immediate (they take no time) – in the product $\mathbf{A} \odot \mathbf{B}$ a link (i, j) exists in a time point t iff in the time point t there exist a link (i, k) in $\mathbf{A}$ and a link (k, j) in $\mathbf{B}$, for some node k.

In the following we shall limit our discussion to temporal quantities that can be described in the form of time-interval/value sequences

$$a(T_a) = ((I_i, v_i))_{i=1}^{k}$$

where I_i is a time-interval and v_i is a value of a on this interval. In general the intervals can be of different types: 1 – $[s_i, f_i]$; 2 – $[s_i, f_i)$; 3 – $(s_i, f_i]$; 4 – (s_i, f_i). Also the value v_i can be structured. For example $v_i = (w_i, c_i, \tau_i)$ – weight, capacity and transition time, or $v_i = (d_i, n_i)$ – length of geodesics and number of geodesics, etc.

Algorithm 1 Addition of temporal quantities.

```
@staticmethod
def get(S):
   try: return(next(S))
   except StopIteration: return((TQ.inf,TQ.inf,TQ.sZero))

@staticmethod
def sum(a,b):
   if len(a) == 0: return(b)
   if len(b) == 0: return(a)
   c = []; A = a.__iter__(); B = b.__iter__()
   (sa,fa,va) = TQ.get(A); (sb,fb,vb) = TQ.get(B)
   while (sa<TQ.inf) or (sb<TQ.inf):
      if sa < sb:
         sc = sa; vc = va
         if sb < fa: fc = sb; sa = sb
         else: fc = fa; (sa,fa,va) = TQ.get(A)
         c.append((sc,fc,vc))
      elif sa == sb:
         sc = sa; fc = min(fa,fb); vc = TQ.sAdd(va,vb)
         c.append((sc,fc,vc))
         sa = sb = fc; fA = fa
         if fA <= fb: (sa,fa,va) = TQ.get(A)
         if fb <= fA: (sb,fb,vb) = TQ.get(B)
      else:
         sc = sb; vc = vb
         if sa < fb: fc = sa; sb = sa
         else: fc = fb; (sb,fb,vb) = TQ.get(B)
         c.append((sc,fc,vc))
   return(TQ.standard(c))
```

To simplify the exposition we will assume in the following that all the intervals in our descriptions of temporal quantities are of type 2 – $[s_i, f_i)$. Therefore we can describe the temporal quantities with sequences

$$a(T_a) = ((s_i, f_i, v_i))_{i=1}^{k}$$

To provide a computational support for the proposed approach we are developing in Python a library TQ (Temporal Quantities). In the examples and the display of selected algorithms we will use the Python notation.

The following are two temporal quantities a and b represented in Python

```
a = [(1, 5, 2), (6, 8, 1), (11, 12, 3), (14, 16, 2),
     (17, 18, 5), (19, 20, 1)]
b = [(2, 3, 4), (4, 7, 3), (9, 10, 2), (13, 15, 5), (16, 21, 1)]
```

The temporal quantity a has on interval $[1, 5)$ value 2, on interval $[6, 8)$ value 1, on interval $[11, 12)$ value 3, etc. Outside the specified intervals its value is undefined, ⌘.

The temporal quantities can also be visualized as is shown for a and b at the top half of Figure 2.

For the simplified version of temporal quantities we wrote procedures `sum` (Algorithm 1) for addition and `prod` (Algorithm 2) for multiplication of temporal quantities over the selected semiring. The semiring operations are provided by functions `sAdd` and `sMul`. The procedure `standard` joins adjacent time intervals with the same value into a single interval.

Algorithm 2 Multiplication of temporal quantities.

```
@staticmethod
def prod(a,b):
    if len(a)*len(b) == 0: return([])
    c = []; A = a.__iter__(); B = b.__iter__()
    (sa,fa,va) = TQ.get(A); (sb,fb,vb) = TQ.get(B)
    while (sa<TQ.inf) or (sb<TQ.inf):
        if fa <= sb: (sa,fa,va) = TQ.get(A)
        elif fb <= sa: (sb,fb,vb) = TQ.get(B)
        else:
            sc = max(sa,sb); fc = min(fa,fb); vc = TQ.sMul(va,vb)
            c.append((sc,fc,vc))
            if fc == fa: (sa,fa,va) = TQ.get(A)
            if fc == fb: (sb,fb,vb) = TQ.get(B)
    return(TQ.standard(c))
```

The following are the sum s and the product p of temporal quantities a and b. They are visually displayed at the bottom half of Figure 2.

```
s = [(1, 2, 2), (2, 3, 6), (3, 4, 2), (4, 5, 5), (5, 6, 3),
     (6, 7, 4), (7, 8, 1), (9, 10, 2), (11, 12, 3), (13, 14, 5),
     (14, 15, 7), (15, 16, 2), (16, 17, 1), (17, 18, 6),
     (18, 19, 1), (19, 20, 2), (20, 21, 1)]
p = [(2, 3, 8), (4, 5, 6), (6, 7, 3), (14, 15, 10), (17, 18, 5),
     (19, 20, 1)]
```

In some applications we shall use the *aggregated value* of a temporal quantity $a = ((s_i, f_i, v_i))_{i=1}^{k}$. It is defined as

$$\Sigma a = \sum_{i=1}^{k} (f_i - s_i) \cdot v_i$$

and is computed using the procedure `total`. For example $\Sigma a = 23$ and $\Sigma b = 30$. Note that $\Sigma a + \Sigma b = \Sigma(a + b)$.

The description of temporal partitions has the same form as the description of temporal quantities $a = ((s_i, f_i, v_i))_{i=1}^{k}$. They differ only in the interpretation of values v_i. In the case of partitions $v_i = k$ means that the unit described with a belongs to a class k in the time interval $[s_i, f_i)$. We shall use temporal partitions to describe connectivity components in Section 8.

We obtain a more adequate description of temporal networks by using vectors of temporal quantities (temporal vectors and temporal partitions) for describing properties of nodes and

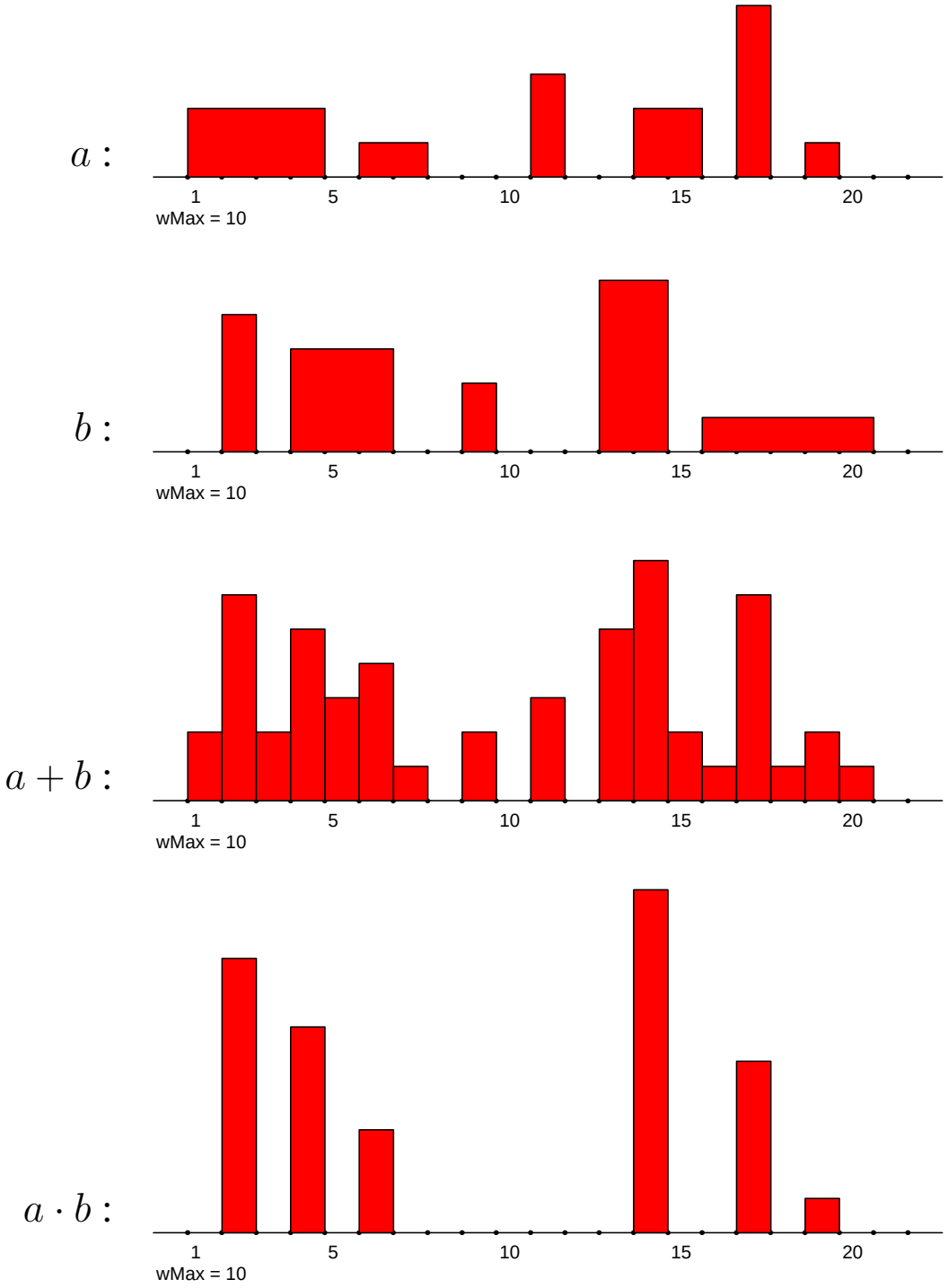

Figure 2: Addition and multiplication of temporal quantities.

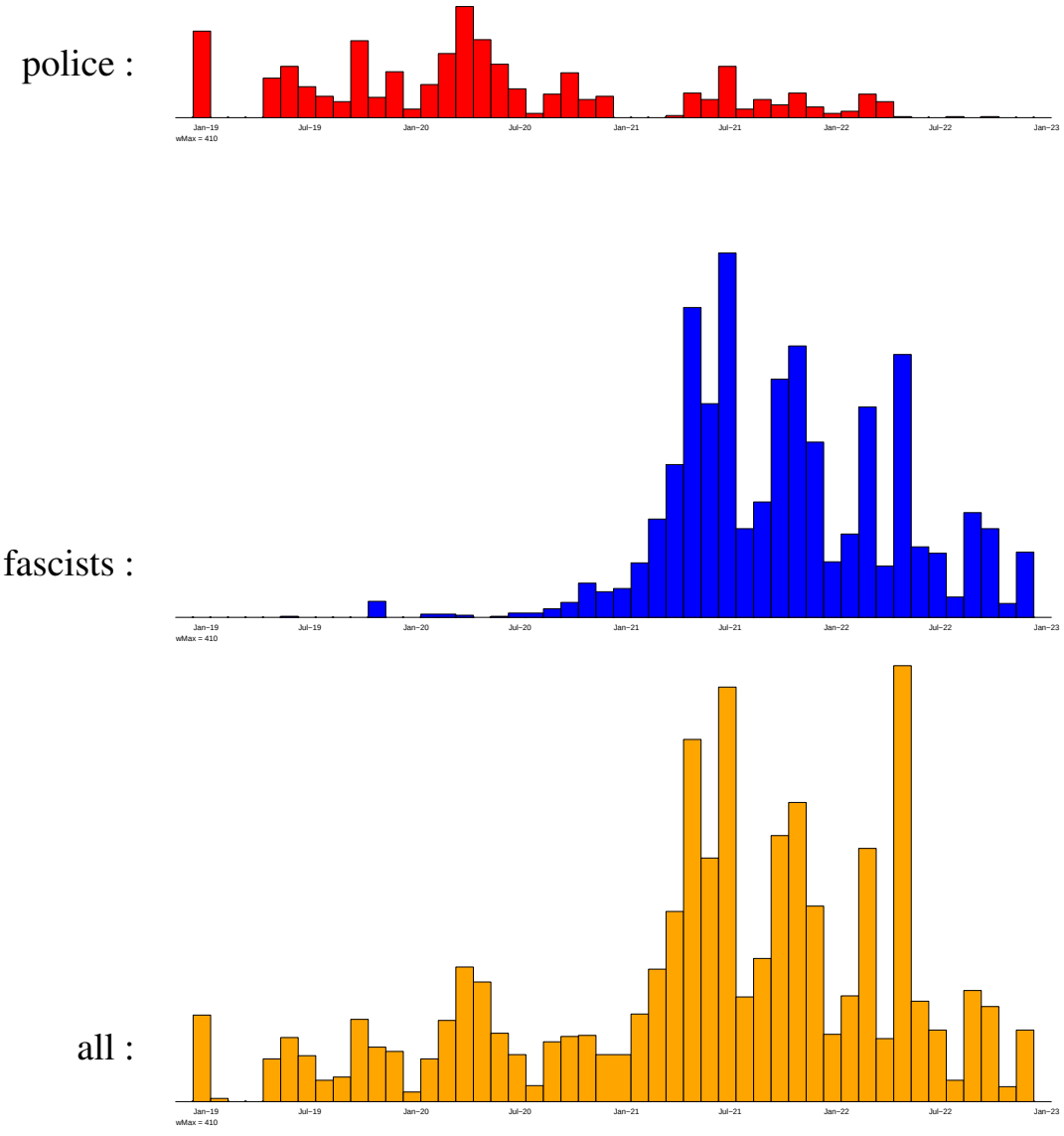

Figure 3: Intensity of violent activities of police, fascists and all.

making also link weights into temporal quantities. In the current version of library TQ we use a representation of a network $\mathcal{N}$ with its matrix $\mathbf{A} = [a_{uv}]$

$$a_{uv} = \begin{cases} w(u,v) & (u,v) \in \mathcal{L} \\ \text{⌘} & \text{otherwise} \end{cases}$$

where $w(u,v)$ is a temporal weight attached to a link (u,v).

3.1 Examples

Let us look at some examples of temporal networks.

Citation networks can be obtained from bibliographic data bases such as Web of Science (Knowledge) and Scopus. In a citation network $\mathcal{N} = (\mathcal{V}, \mathcal{L}, \mathcal{P}, \mathcal{W}, \mathcal{T})$ its set of nodes $\mathcal{V}$ consists of selected works (papers, books, reports, patents, etc.). There exists an arc $(u,v) \in \mathcal{L}$ iff the work u cites the work v. The time set $\mathcal{T}$ is usually an interval of years $[year_{first}, year_{last}]$ in which the works were published. The activity set of the work v, $T(v)$, is the interval $[year_{publication}(v), year_{last}]$; and the activity set of the arc $a = (u,v)$, $T(a)$, is the interval $[year_{publication}(u), year_{last}]$. An example of a property $p \in \mathcal{P}$ is the number of pages. Other properties, such as work's authors and keywords, are usually represented as two-mode networks.

Project collaboration networks are usually based on some project data base such as Cordis. The set of nodes $\mathcal{V}$ consists of participating institutions. There is an edge $(u : v) \in \mathcal{L}$ iff institutions u and v work on a joint project. The time set $\mathcal{T}$ is an interval of dates/days $[day_{first}, day_{last}]$. $T(v) = \mathcal{T}$ and $T(u,v) = \{[s,f] :$ exists a project P such that u and v are partners on P; s is the start and f is the finish date of $P\}$.

KEDS/WEIS networks are networks registering political events in critical regions in the world (Middle East, Balkans, and West Africa) on the basis of daily news. Originally they were collected by KEDS (Kansas Event Data System). Currently they are hosted by Parus Analytical Systems. The set of nodes $\mathcal{V}$ contains the involved actors (states, political groups, international organizations, etc.). The links are directed and are describing the events

$$(date, actor_1, actor_2, action)$$

on a given $date$ $actor_1$ made $action$ on $actor_2$. Different actions are determining different relations – we get a multirelational network $\mathcal{L} = \{\mathcal{L}_a : a \in \text{Actions}\}$. The time set is determined by the observed period $[day_{first}, day_{last}]$. Since most of the actors are existing during all the observed period their node activity time sets are $T(v) = \mathcal{T}$. Another option is to consider as their node activity time sets the period of their engagement in the region. The activity time set $T(l)$ of an arc $l(u,v) \in \mathcal{L}_a$ contains all dates – intervals $[day, day+1)$ – in which actor u made action a on actor v. Another possibility is to base the description on a single relation network and store the information about action a as a structured value in a triple $(day, day+1, value)$

$$value = [(action_1, count_1), (action_2, count_2), \ldots, (action_k, count_k)]$$

and introduce an appropriate semiring over such values.

There are many other examples of temporal networks such as: genealogies, contact networks, networks of phone calls, transportation time tables, etc.

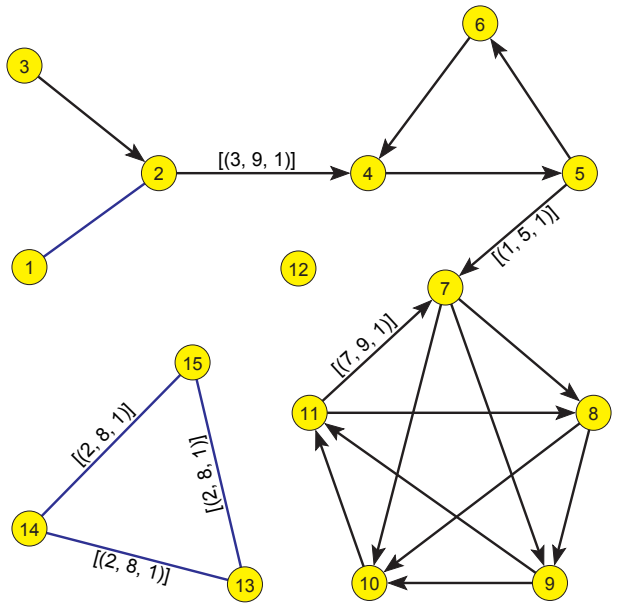

Figure 4: First example network. All unlabeled links have a value $[(1, 9, 1)]$.

4 Node activities and degrees

In this section we show how we can use the proposed operations with temporal quantities for a simple analysis of temporal networks.

We define the *activity* between groups of nodes $\mathcal{V}_1$ and $\mathcal{V}_2$ as

$$\text{act}(\mathcal{V}_1, \mathcal{V}_2) = \sum_{u \in \mathcal{V}_1} \sum_{v \in \mathcal{V}_2} a_{uv}.$$

To illustrate the notion of activity we applied it on Franzosi's violence temporal network [12]. Roberto Franzosi collected from the journal news in the period (January 1919 – December 1922) information about the different types of interactions between political parties and other groups of people in Italy. The violence network contains only the data about violent actions and counts the number of interactions per month.

We determined the temporal quantities $pol = \text{act}(\{\text{police}\}, \mathcal{V}) + \text{act}(\mathcal{V}, \{\text{police}\})$, $fas = \text{act}(\{\text{fascists}\}, \mathcal{V}) + \text{act}(\mathcal{V}, \{\text{fascists}\})$ and $all = \text{act}(\mathcal{V}, \mathcal{V})$. They are presented in Figure 3. Comparing the intensity charts of police and fascists activity with overall activity we see that most of the violent activity in the first two years 1919 and 1920 was related to the police. In the next two years (1921 and 1922) it was taken over by the fascists.

For an ordinary graph with a (binary) adjacency matrix $\mathbf{A}$ we can compute the corresponding indegre and outdegree vectors using the relations

$$\mathbf{i} = \mathbf{e}^T \odot \mathbf{A} \qquad \text{and} \qquad \mathbf{o} = \mathbf{A} \odot \mathbf{e}$$

where $\mathbf{e}$ is a column vector of size $n = |\mathcal{V}|$ with all its entries equal to 1. The same holds for temporal networks. In this case the vector $\mathbf{e}$ contains as values the temporal unit $\mathbf{1}$. The

Table 1: Temporal indegrees and outdegrees for the first example network.

```
Indegrees                          Outdegrees
 1 : [(1, 9, 1)]                    1 : [(1, 9, 1)]
 2 : [(1, 9, 2)]                    2 : [(1, 3, 1), (3, 9, 2)]
 3 : []                             3 : [(1, 9, 1)]
 4 : [(1, 3, 1), (3, 9, 2)]         4 : [(1, 9, 1)]
 5 : [(1, 9, 1)]                    5 : [(1, 5, 2), (5, 9, 1)]
 6 : [(1, 9, 1)]                    6 : [(1, 9, 1)]
 7 : [(1, 5, 1), (7, 9, 1)]         7 : [(1, 9, 3)]
 8 : [(1, 9, 2)]                    8 : [(1, 9, 2)]
 9 : [(1, 9, 2)]                    9 : [(1, 9, 2)]
10 : [(1, 9, 3)]                   10 : [(1, 9, 1)]
11 : [(1, 9, 2)]                   11 : [(1, 7, 1), (7, 9, 2)]
12 : []                            12 : []
13 : [(2, 8, 2)]                   13 : [(2, 8, 2)]
14 : [(2, 8, 2)]                   14 : [(2, 8, 2)]
15 : [(2, 8, 2)]                   15 : [(2, 8, 2)]
```

procedures for computing degrees are simple – see Algorithm 3. Function `MatBin`($\mathbf{A}$) transforms all values in the matrix $\mathbf{A}$ to 1; and function `VecConst`(n) constructs a vector of size n filled with values $[(1, \infty, 1)]$. For a temporal network presented in Figure 4 the corresponding temporal indegrees and outdegrees are given in Table 1. For example, node 5 has in the time interval $[1, 5)$ outdegree 2. Because the arc $(5, 7)$ disappears in time point 5 the outdegree of node 5 diminishes to 1 in the interval $[5, 9)$.

Algorithm 3 Computing temporal indegrees and outdegrees.

```
@staticmethod
def inDeg(A):
    return(TQ.MatVecLeft(TQ.MatBin(A),TQ.VecConst(len(A))))

@staticmethod
def outDeg(A):
    return(TQ.MatVecRight(TQ.MatBin(A),TQ.VecConst(len(A[0]))))
```

We will use the simple temporal network from Figure 4 also for the illustration of other algorithms because it allows the users to manually check the presented results.

5 Clustering coefficients

Let us assume that the network $\mathcal{N}$ is based on a simple directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ without loops. From a simple undirected graph we obtain the corresponding simple directed graph by replacing each edge with a pair of opposite arcs. In such a graph the *clustering coefficient*, $C(v)$, of the node v is defined as the proportion between the number of realized arcs among the node's neighbors and the number of all possible arcs among the node's neighbors $N(v)$, that is

$$C(v) = \frac{|\mathcal{A}(N(v))|}{k(k-1)}$$

where k is the number of neighbors of the node v. For a node v without neighbors or with a single neighbor we set $C(v) = 0$.

The clustering coefficient measures a local density of the node's neighborhood. A problem in its applications in network analysis is that the identified densest neighborhoods are mostly very small. For this reason we provided in Pajek the *corrected clustering coefficient*, $C'(v)$,

$$C'(v) = \frac{|\mathcal{A}(N(v))|}{\Delta(k-1)}$$

where Δ is the maximum number of neighbors in graph.

To count the number of realized arcs among the node's neighbors we use the observation that each arc forms a triangle with links from its end-nodes to the node v; and that the number of triangles in a simple undirected graph can be obtained as the diagonal value in the third power of the graph matrix.

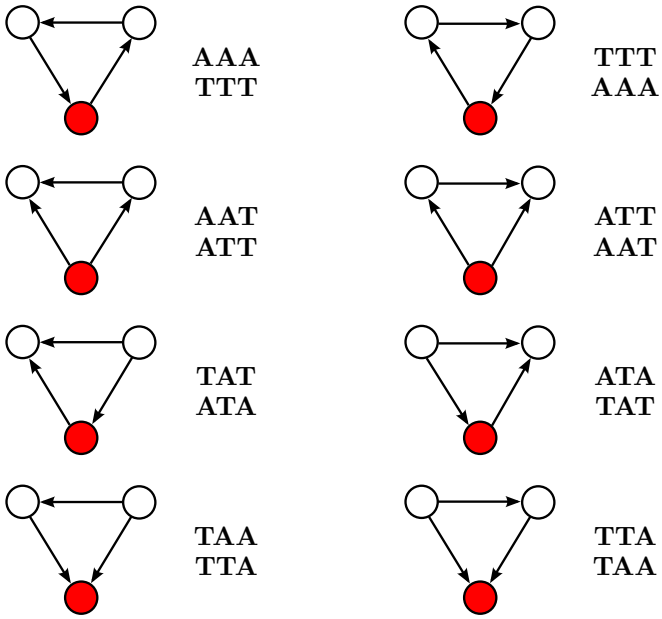

Figure 5: Counting triangles.

For simple directed graphs the counting of triangles is slightly more complicated. Let us denote $\mathbf{T} = \mathbf{A}^T$ and $\mathbf{S} = \mathbf{A} + \mathbf{T}$. From Figure 5 we see that each triangle (determined with a link opposite to the dark node) appears exactly once in

$$\mathbf{AAA} + \mathbf{AAT} + \mathbf{TAT} + \mathbf{TAA} = \mathbf{AAS} + \mathbf{TAS} = \mathbf{SAS}.$$

This gives us a simple way to count the triangles which is used in Algorithm 4. Since we need only the diagonal values of the matrix $\mathbf{SAS}$ we applied a special procedure `MathProdDiag`$(\mathbf{A}, \mathbf{B})$ that determines only the diagonal vector of the product $\mathbf{AB}$. Afterward, to get the clustering coefficient, we have to normalize the obtained counts. The number of neighbors of the node v is determined as its degree in the corresponding undirected temporal skeleton graph (in which an edge $e = (v : u)$ exists iff there is at least one arc between the nodes v and u). The maximum

number of neighbors Δ can be considered either for a selected time point or for the complete time window. Note that to determine the temporal Δ we used summing of temporal degrees over the *maxmin* semiring $(\mathbb{R}, \max, \min, -\infty, \infty)$.

In Table 2 and Table 3 the ordinary and the corrected clustering coefficients are presented for the example network from Figure 4 and its undirected skeleton.

Algorithm 4 Clustering coefficients.

```
@staticmethod
def clusCoef(A,type=1):
# type = 1 - standard clustering coefficient
# type = 2 - corrected clustering coefficient / temporal degMax
# type = 3 - corrected clustering coefficient / overall degMax
   nr = len(A); nc = len(A[0])
   if nr!=nc: raise TQ.TQerror("clusCoef: square matrix required")
   if (type<1)or(type>3):
      raise TQ.TQerror("clusCoef: unsuported type")
   Ab = TQ.MatSetDiag(TQ.MatBin(A),TQ.sN)
   S = TQ.MatBin(TQ.MatSym(Ab))
   deg = TQ.MatVecRight(S,TQ.VecConst(nr))
   if type == 1:
      fac = TQ.VecProd(deg,TQ.VecSum(deg,
                  TQ.VecConst(nr,const=[(1,TQ.inf,-1)])))
   else:
      TQ.maxmin(); delta = []
      for d in deg: delta = TQ.sum(delta,d)
      if type == 3:
         Delta = max([v for (s,f,v) in delta])
         delta = [(1,TQ.inf,Delta)]
      TQ.combinatorial(); fac = []
      degm = TQ.VecSum(deg,TQ.VecConst(nr,const=[(1,TQ.inf,-1)]))
      for d in degm: fac.append(TQ.prod(delta,d))
   tri = TQ.MatProdDiag(TQ.MatProd(S,Ab),S)
   cc = TQ.VecProd(TQ.VecInv(fac),tri)
   return(cc)
```

6 Closures in temporal networks

When the basic semiring $(A, +, \cdot, 0, 1)$ is *closed* – an unary *closure* operation $\star$ with the property

$$a^\star = 1 + a \cdot a^\star = 1 + a^\star \cdot a, \qquad \text{for all } a \in A$$

is defined in it – this property can be extended also to the corresponding matrix semiring. For computing the matrix closure we can apply the Fletcher's algorithm [7, 2]. In most of semirings, for which we are interested in determining the closures, also the *absorption law* holds

$$1 + a = 1, \qquad \text{for all } a \in A.$$

In these semirings $a^\star = 1$, for all $a \in A$, and therefore the Fletcher's algorithm can be simplified and performed in place as implemented in Algorithm 5.

Table 2: Clustering coefficients for the first example network.

```
Clustering coefficient
 1 : []
 2 : []
 3 : []
 4 : [(1, 3, 0.5), (3, 9, 0.1667)]
 5 : [(1, 5, 0.1667), (5, 9, 0.5)]
 6 : [(1, 9, 0.5)]
 7 : [(1, 5, 0.25), (5, 9, 0.5)]
 8 : [(1, 7, 0.4167), (7, 9, 0.5)]
 9 : [(1, 7, 0.4167), (7, 9, 0.5)]
10 : [(1, 7, 0.4167), (7, 9, 0.5)]
11 : [(1, 9, 0.5)]
12 : []
13 : [(2, 8, 1.0)]
14 : [(2, 8, 1.0)]
15 : [(2, 8, 1.0)]
```

```
Corrected clustering coefficient
 1 : []
 2 : []
 3 : []
 4 : [(1, 3, 0.25), (3, 9, 0.125)]
 5 : [(1, 5, 0.125), (5, 9, 0.25)]
 6 : [(1, 9, 0.25)]
 7 : [(1, 5, 0.25), (5, 7, 0.375),
      (7, 9, 0.5)]
 8 : [(1, 7, 0.4167), (7, 9, 0.5)]
 9 : [(1, 7, 0.4167), (7, 9, 0.5)]
10 : [(1, 7, 0.4167), (7, 9, 0.5)]
11 : [(1, 7, 0.375), (7, 9, 0.5)]
12 : []
13 : [(2, 8, 0.5)]
14 : [(2, 8, 0.5)]
15 : [(2, 8, 0.5)]
```

Table 3: Clustering coefficients for the skeleton of the first example network.

```
Clustering coefficient
 1 : []
 2 : []
 3 : []
 4 : [(1, 3, 1.0), (3, 9, 0.3333)]
 5 : [(1, 5, 0.3333), (5, 9, 1.0)]
 6 : [(1, 9, 1.0)]
 7 : [(1, 5, 0.5), (5, 9, 1.0)]
 8 : [(1, 7, 0.8333), (7, 9, 1.0)]
 9 : [(1, 7, 0.8333), (7, 9, 1.0)]
10 : [(1, 7, 0.8333), (7, 9, 1.0)]
11 : [(1, 9, 1.0)]
12 : []
13 : [(2, 8, 1.0)]
14 : [(2, 8, 1.0)]
15 : [(2, 8, 1.0)]
```

```
Corrected clustering coefficient
 1 : []
 2 : []
 3 : []
 4 : [(1, 3, 0.5), (3, 9, 0.25)]
 5 : [(1, 5, 0.25), (5, 9, 0.5)]
 6 : [(1, 9, 0.5)]
 7 : [(1, 5, 0.5), (5, 7, 0.75),
      (7, 9, 1.0)]
 8 : [(1, 7, 0.8333), (7, 9, 1.0)]
 9 : [(1, 7, 0.8333), (7, 9, 1.0)]
10 : [(1, 7, 0.8333), (7, 9, 1.0)]
11 : [(1, 7, 0.75), (7, 9, 1.0)]
12 : []
13 : [(2, 8, 0.5)]
14 : [(2, 8, 0.5)]
15 : [(2, 8, 0.5)]
```

Algorithm 5 Closure of a temporal matrix over an absorptive semiring.

```
@staticmethod
def MatClosure(R,strict=False):
   nr = len(R); nc = len(R[0])
   if nr!=nc: raise TQ.TQerror("MatClosure: square matrix required")
   C = deepcopy(R)
   for k in range(nr):
      for u in range(nr):
         for v in range(nr):
            C[u][v] = TQ.sum(C[u][v], TQ.prod(C[u][k],C[k][v]))
      if not strict: C[k][k] = TQ.sum(TQ.sE,C[k][k])
   return(C)
```

7 Temporal node partitions

In previous sections the nodes of temporal networks were considered as being present all the time. We can describe the presence of nodes through time using a temporal binary (single valued) node partition $T : \mathcal{V} \to A_{\text{⌘}}(\mathcal{T})$,

$$T(u) = ((s_i, f_i, 1))_{i=1}^{k}, \quad \text{for } u \in \mathcal{V}$$

specifying that a node u is present in time intervals $[s_i, f_i)$, $i = 1, \ldots, k$.

The node partition T determined from the temporal network links by

$$T(u) = \bigcup_{l \in \mathcal{L}: u \in \text{ext}(l)} \text{binary}(a_l),$$

for $u \in \mathcal{V}$, is the smallest temporal partition of nodes that satisfies the consistency condition from Section 2. The term ext(l) denotes the set of endnodes of the link l, a_l is the temporal quantity assigned to the link l, and the function `binary` sets all values in a given temporal quantity to 1. In library TQ the partition T can be computed using the procedure `minTime`.

Temporal node partition T can be used also to extract a corresponding subnetwork from a given temporal network described with a matrix $\mathbf{A}$. The subnetwork contains only the nodes active in the partition T and the active links satisfying the consistency condition with respect to T.

To formalize the described procedure we first define the procedure `extract`$(p, a) = b$, where p is a binary temporal quantity and a is a temporal quantity, as

$$b(t) = \begin{cases} a(t) & t \in T_p \cap T_a \\ \text{⌘} & \text{otherwise} \end{cases}.$$

Let $\mathbf{B}$ be a temporal matrix describing the links of the subnetwork. Its entries for $l(u, v) \in \mathcal{L}$ are determined by

$$b_l = \text{extract}(T(u) \cap T(v), a_l).$$

In TQ this operation is implemented as a procedure `MatExtract`$(\mathbf{T}, \mathbf{A})$.

Table 4: Temporal input and output reachability degrees for the first example network.

```
Input reachability
 1 : [(1, 9, 3)]
 2 : [(1, 9, 3)]
 3 : []
 4 : [(1, 3, 3), (3, 9, 6)]
 5 : [(1, 3, 3), (3, 9, 6)]
 6 : [(1, 3, 3), (3, 9, 6)]
 7 : [(1, 3, 3), (3, 5, 6), (7, 9, 5)]
 8 : [(1, 3, 8), (3, 5, 11), (5, 9, 5)]
 9 : [(1, 3, 8), (3, 5, 11), (5, 9, 5)]
10 : [(1, 3, 8), (3, 5, 11), (5, 9, 5)]
11 : [(1, 3, 8), (3, 5, 11), (5, 9, 5)]
12 : []
13 : [(2, 8, 3)]
14 : [(2, 8, 3)]
15 : [(2, 8, 3)]
```

```
Output reachability
 1 : [(1, 3, 2), (3, 5, 10), (5, 9, 5)]
 2 : [(1, 3, 2), (3, 5, 10), (5, 9, 5)]
 3 : [(1, 3, 2), (3, 5, 10), (5, 9, 5)]
 4 : [(1, 5, 8), (5, 9, 3)]
 5 : [(1, 5, 8), (5, 9, 3)]
 6 : [(1, 5, 8), (5, 9, 3)]
 7 : [(1, 7, 4), (7, 9, 5)]
 8 : [(1, 7, 4), (7, 9, 5)]
 9 : [(1, 7, 4), (7, 9, 5)]
10 : [(1, 7, 4), (7, 9, 5)]
11 : [(1, 7, 4), (7, 9, 5)]
12 : []
13 : [(2, 8, 3)]
14 : [(2, 8, 3)]
15 : [(2, 8, 3)]
```

8 Temporal reachability and weak and strong connectivity

For a temporal network represented with the corresponding matrix $\mathbf{A}$ its transitive closure $\mathbf{A}^\star$ (over the reachability semirings based on the semiring $(\{0,1\}, \vee, \wedge, 0, 1)$) determines its *reachability* relation matrix. We obtain its *weak connectivity* temporal matrix $\mathbf{W}$ as

$$\mathbf{W} = (\mathbf{A} + \mathbf{A}^T)^\star$$

and its *strong connectivity* temporal matrix $\mathbf{S}$ as

$$\mathbf{S} = \mathbf{A}^\star \cap (\mathbf{A}^\star)^T.$$

The use of the strict transitive closure instead of a transitive closure in these relations preserves the inactivity value `[]` on the diagonal for all isolated nodes.

8.1 Reachability degrees

Let $\mathbf{R} = \overline{\mathbf{A}} = \mathbf{A}\mathbf{A}^\star$ be the strict reachability relation of a given network. Then the temporal vectors `inReach` and `outReach` determined by

```
TQ.reach()
R = TQ.MatClosure(TQ.MatBin(A),strict=True)
TQ.combinatorial()
inReach  = TQ.inDeg(R)
outReach = TQ.outDeg(R)
```

contain temporal quantities counting the number of nodes: from which a given node v is reachable (`inReach`$[v]$) / which are reachable from the node v (`outReach`$[v]$). The results for our example network are presented in Table 4. For example, 8 nodes $\{4,5,6,7,8,9,10,11\}$ are reachable from node 6 in the time interval $[1,5)$, and 3 nodes $\{4,5,6\}$ are reachable in the time interval $[5,9)$.

8.2 Temporal weak connectivity

The procedure `weakConnMat`($\mathbf{A}$) that for a given temporal network matrix $\mathbf{A}$ determines the corresponding temporal weak connectivity matrix is presented in Algorithm 6. We first switch to the reachability semirings and afterward compute the strict transitive closure $\mathbf{W}$ of the binary version of the symmetrized matrix $\mathbf{A}$.

To transform the obtained temporal equivalence matrix $\mathbf{E}$ into the corresponding temporal partition $\mathbf{p}$ we use the fact that on a given time interval equivalent (in our case weakly connected) nodes get the same value on this interval in the product of the matrix $\mathbf{E}$ with a diagonal matrix computed over the combinatorial semiring $(\mathbb{N}, +, \cdot, 0, 1)$. We take for the diagonal values randomly shuffled integers from the interval $1 : n$. With a very high probability the values belonging to different equivalence classes are different. This is implemented as a procedure `eqMat2Part`($\mathbf{E}$). Maybe in the future implementations we shall add a loop with the check of the injectivity of this mapping. The classes of the obtained partition are finally renumbered with consecutive numbers using the procedure `renumPart`.

To get the weak connectivity partition we have to combine these two procedures

```
p = TQ.eqMat2Part(TQ.weakConnMat(A))
```

For our first example network we obtain the temporal weak partition presented in the left side of Table 5.

Algorithm 6 Weak connectivity.

```
@staticmethod
def renumPart(p):
   C = {}; q = []
   for a in p:
      r = []
      for (sa,fa,ca) in a:
         if not(ca in C): C[ca] = 1+len(C)
         r.append((sa,fa,C[ca]))
      q.append(r)
   return(q)

@staticmethod
def weakConnMat(A):
   old = TQ.semiring; TQ.reach()
   W = TQ.MatClosure(TQ.MatSym(TQ.MatBin(A)),strict=True)
   old()
   return(W)

@staticmethod
def eqMat2Part(E):
   old = TQ.semiring; TQ.combinatorial()
   v = [ [(1,TQ.inf,i+1)] for i in range(len(E))]
   random.shuffle(v)
   p = TQ.MatVecRight(E,v)
   old()
   return(TQ.renumPart(p))
```

Table 5: Temporal weak and strong connectivity partitions for the first example network.

```
Weak partition                                  Strong partition
 1 : [(1, 3, 1), (3, 5, 2), (5, 9, 3)]          1 : [(1, 9, 1)]
 2 : [(1, 3, 1), (3, 5, 2), (5, 9, 3)]          2 : [(1, 9, 1)]
 3 : [(1, 3, 1), (3, 5, 2), (5, 9, 3)]          3 : []
 4 : [(1, 3, 4), (3, 5, 2), (5, 9, 3)]          4 : [(1, 9, 2)]
 5 : [(1, 3, 4), (3, 5, 2), (5, 9, 3)]          5 : [(1, 9, 2)]
 6 : [(1, 3, 4), (3, 5, 2), (5, 9, 3)]          6 : [(1, 9, 2)]
 7 : [(1, 3, 4), (3, 5, 2), (5, 9, 5)]          7 : [(7, 9, 3)]
 8 : [(1, 3, 4), (3, 5, 2), (5, 9, 5)]          8 : [(1, 7, 4), (7, 9, 3)]
 9 : [(1, 3, 4), (3, 5, 2), (5, 9, 5)]          9 : [(1, 7, 4), (7, 9, 3)]
10 : [(1, 3, 4), (3, 5, 2), (5, 9, 5)]         10 : [(1, 7, 4), (7, 9, 3)]
11 : [(1, 3, 4), (3, 5, 2), (5, 9, 5)]         11 : [(1, 7, 4), (7, 9, 3)]
12 : []                                        12 : []
13 : [(2, 8, 6)]                               13 : [(2, 8, 5)]
14 : [(2, 8, 6)]                               14 : [(2, 8, 5)]
15 : [(2, 8, 6)]                               15 : [(2, 8, 5)]
```

8.3 Temporal strong connectivity

The procedure `strongConnMat`$(\mathbf{A})$ that for a given temporal network matrix $\mathbf{A}$ determines the corresponding temporal strong connectivity matrix is presented in Algorithm 7. The procedure `MatInter`$(\mathbf{A}, \mathbf{B})$ determines the intersection of temporal network binary matrices $\mathbf{A}$ and $\mathbf{B}$.

Algorithm 7 Strong connectivity.

```
@staticmethod
def strongConnMat(A):
   old = TQ.semiring; TQ.reach()
   R = TQ.MatClosure(TQ.MatBin(A),strict=True)
   S = TQ.MatInter(R,TQ.MatTrans(R))
   old()
   return(S)
```

Again, to get the strong connectivity partition we have to apply the procedure `eqMat2Part` to the strong connectivity matrix

```
s = TQ.eqMat2Part(TQ.strongConnMat(A))
```

For our first example network we obtain the temporal strong partition presented in the right hand side of Table 5.

9 Temporal closeness and betweenness

Closeness and betweenness are among the traditional social network analysis indices measuring the importance of nodes [8]. They are somehow problematic when applied to non (strongly) connected graphs. In this section we will not consider these questions. We will only show how to compute them for nonproblematic temporal graphs.

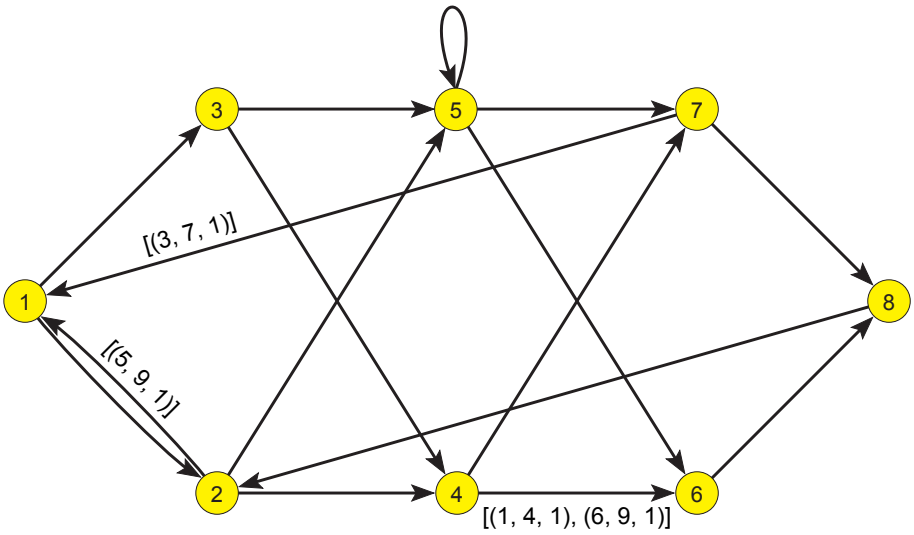

Figure 6: Second example network. All unlabeled arcs have value $[(1, 9, 1)]$.

9.1 Temporal closeness

The *output closeness* of the node v is defined as

$$ocl(v) = \frac{n-1}{\sum\limits_{u \in \mathcal{V} \setminus \{v\}} d_{vu}}.$$

To determine the closeness we need first to compute the the matrix $\mathbf{D} = [d_{uv}]$ of geodetic distances d_{uv} between nodes u and v. It can be obtained as a closure of the network matrix $\mathbf{A}$ over the *shortest paths* semiring $(\overline{\mathbb{R}}_0^+, \min, +, \infty, 0)$. Note that the values in the matrix $\mathbf{A}$ can be any nonnegative real numbers.

In Figure 6 we present our second example temporal network which is an extended version of the example given in Figure 3 from [2].

Because a complete strict closure matrix $\mathbf{D}$ is too large to be listed we present only some of its selected entries:

```
D[3,1] = [(3, 7, 3), (7, 9, 5)]
D[4,6] = [(1, 4, 1), (4, 6, 5), (6, 9, 1)]
D[6,3] = [(3, 5, 6), (5, 9, 4)]
D[7,6] = [(1, 9, 4)]
```

To compute the vector of closeness coefficients of nodes we have to sum the temporal distances to other nodes over the combinatorial semiring. While summing we replace gaps (inactivity intervals) with time intervals with value infinity, using the procedure `fillGaps`. See Algorithm 8.

The temporal closeness coefficients for our second example network are given in Table 6.

Algorithm 8 Temporal closeness.

```
   @staticmethod
   def closeness(A,type=2):
# type: 1 - output, 2 - all, 3 - input
      (s,f,x,y) = TQ.MatSummary(A)
      old = TQ.semiring; TQ.path(); n = len(A)
      D = TQ.MatClosure(A,strict=True)
      cl = [ [] for i in range(n) ]; TQ.combinatorial()
      fac = [(1,TQ.inf,(n-1)*(2 if type==2 else 1))]
      for v in range(n):
         d = []
         for u in range(n):
            if u!=v:
               if type<3: d = TQ.sum(d,TQ.fillGaps(D[v][u],s,f))
               if type>1: d = TQ.sum(d,TQ.fillGaps(D[u][v],s,f))
         cl[v] = TQ.prod(fac,TQ.invert(d))
      old()
      return(cl)
```

Table 6: Output closeness for the second example network.

```
1 : [(1, 9, 0.4375)]
2 : [(1, 3, 0.0000), (3, 5, 0.4375), (5, 9, 0.5833)]
3 : [(1, 3, 0.0000), (3, 7, 0.4375), (7, 9, 0.3889)]
4 : [(1, 3, 0.0000), (3, 4, 0.4375), (4, 6, 0.3500),
     (6, 7, 0.4375), (7, 9, 0.3500)]
5 : [(1, 3, 0.0000), (3, 7, 0.4375), (7, 9, 0.3500)]
6 : [(1, 3, 0.0000), (3, 5, 0.2917), (5, 9, 0.3500)]
7 : [(1, 3, 0.0000), (3, 7, 0.4375), (7, 9, 0.3500)]
8 : [(1, 3, 0.0000), (3, 5, 0.3500), (5, 9, 0.4375)]
```

9.2 Temporal betweenness

The *betweenness* of the node v is defined as

$$b(v) = \frac{1}{(n-1)(n-2)} \sum_{\substack{u,w \in \mathcal{V} \\ |\{v,u,w\}|=3}} \frac{n_{u,w}(v)}{n_{u,w}}$$

where $n_{u,w}$ is the number of u-w geodesics and $n_{u,w}(v)$ is the number of u-w geodesics passing through the node v.

Suppose that we know the matrix

$$\mathbf{C} = [(d_{u,v}, n_{u,v})]$$

where $d_{u,v}$ is the length of u-v geodesics. Then it is also easy to determine a temporal quantity $n_{u,w}(v)$:

$$n_{u,w}(v) = \begin{cases} n_{u,v} \cdot n_{v,w} & d_{u,v} + d_{v,w} = d_{u,w} \\ 0 & \text{otherwise} \end{cases}.$$

This gives the following procedure for computing the betweenness coefficients $\mathbf{b}$

compute $\mathbf{C}$;
for $v \in \mathcal{V}$ **do begin**
  $r := 0$;
  for $u \in \mathcal{V}$ **do for** $w \in \mathcal{V}$ **do**
    if $n[u,w] \neq 0 \wedge |\{v,u,w\}| = 3 \wedge d[u,w] = d[u,v] + d[v,w]$ **then**
      $r := r + n[u,v] * n[v,w]/n[u,w]$;
  $b[v] := r/((n-1)*(n-2))$
end;

In [2] it is shown that the matrix $\mathbf{C}$ can be obtained by computing a closure of the network matrix over the *geodetic semiring* $(\overline{\mathbb{N}}^2, \oplus, \odot, (\infty, 0), (0, 1))$, where $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$ and we define *addition* $\oplus$ with:

$$(a,i) \oplus (b,j) = (\min(a,b), \begin{cases} i & a < b \\ i+j & a = b \\ j & a > b \end{cases})$$

and *multiplication* $\odot$ with:

$$(a,i) \odot (b,j) = (a+b, i \cdot j).$$

The implementation of geodetic semiring in TQ library is presented as Algorithm 9.

To compute the geodetic closure we first transform the network temporal adjacency matrix $\mathbf{A}$ to a matrix $\mathbf{G} = [(d,n)_{u,v}]$ which has for entries pairs defined by

$$(d,n)_{u,v} = \begin{cases} (1,1) & (u,v) \in \mathcal{L} \\ (\infty,0) & \text{otherwise} \end{cases}$$

where d is the length of a shortest path and n is the number of shortest paths between u and v. In temporal networks the distance d and the counter n are temporal quantities.

Following the presented scheme of computing the betweenness vector and adapting it to temporal quantities (see Algorithm 10) in procedure `betweenness` we first transform the network matrix $\mathbf{A}$ into a matrix $\mathbf{G}$ with values $(1,1)$ on arcs and compute its strict geodetic closure $\mathbf{C}$ over the geodetic semiring.

Again we present only some selected entries of strict geodetic closure matrix $\mathbf{C}$ for our second example network:

```
C[1,7] = [(1, 9, (3, 4))]
C[2,2] = [(1, 3, (4, 4)), (3, 4, (4, 6)), (4, 5, (4, 5)), (5, 9, (2, 1))]
C[4,6] = [(1, 4, (1, 1)), (4, 6, (5, 3)), (6, 9, (1, 1))]
C[5,5] = [(1, 9, (1, 1))]
C[6,3] = [(3, 5, (6, 2)), (5, 9, (4, 1))]
C[7,6] = [(1, 3, (4, 2)), (3, 4, (4, 6)), (4, 6, (4, 3)), (6, 7, (4, 6)),
          (7, 9, (4, 2))]
```

For example, the value $\mathbf{C}[4,6]$ reflects the facts that an arc exists from node 4 to node 6 in time intervals $[1,4)$ and $[6,9)$; and in time interval $[4,6)$ they are connected with 3 geodesics of length 5: $(4,7,8,2,5,6)$, $(4,7,1,3,5,6)$, $(4,7,1,2,5,6)$.

We continue and using the combinatorial semiring we compute the temporal betweenness vector $\mathbf{bw}$. The specificity of temporal quantities $d[u,v]$ and $n[u,v]$ is considered in the procedure `between` that implements the temporal version of the statement

Algorithm 9 Geodetic semirings.

```
@staticmethod
def geoAdd(a,b):
   (av,ac) = a; (bv,bc) = b
   return((min(av,bv), ac if av<bv else ac+bc if av==bv else bc))

@staticmethod
def geoMul(a,b):
   (av,ac) = a; (bv,bc) = b
   return((av+bv,ac*bc))

@staticmethod
def geodetic():
   TQ.sAdd  = TQ.geoAdd;        TQ.sMul = TQ.geoMul
   TQ.sZero = (TQ.inf,0);       TQ.sOne = (0,1)
   TQ.sN    = [];               TQ.sE   = [(1,TQ.inf,TQ.sOne)]
   TQ.semiring = TQ.geodetic
```

Table 7: Betweenness for the second example network.

```
1 : [(3, 4, 0.2500), (4, 6, 0.2754), (6, 7, 0.2500), (7, 9, 0.1429)]
2 : [(1, 3, 0.3452), (3, 4, 0.4048), (4, 6, 0.4187), (6, 7, 0.4048), (7, 9, 0.6071)]
3 : [(1, 3, 0.0595), (3, 4, 0.0952), (4, 6, 0.1052), (6, 7, 0.0952), (7, 9, 0.0595)]
4 : [(1, 3, 0.1667), (3, 4, 0.2500), (4, 5, 0.1762), (5, 6, 0.1048), (6, 9, 0.1786)]
5 : [(1, 3, 0.1667), (3, 4, 0.2500), (4, 5, 0.3476), (5, 6, 0.2762), (6, 9, 0.1786)]
6 : [(1, 3, 0.1190), (3, 4, 0.0952), (4, 6, 0.0544), (6, 7, 0.0952), (7, 9, 0.1786)]
7 : [(1, 3, 0.1190), (3, 4, 0.4048), (4, 5, 0.4694), (5, 6, 0.3266), (6, 7, 0.2619),
     (7, 9, 0.1786)]
8 : [(1, 3, 0.3095), (3, 4, 0.2500), (4, 6, 0.2484), (6, 7, 0.2500), (7, 9, 0.5238)]
```

if $d[u,w] = d[u,v] + d[v,w]$ **then** $r := r + n[u,v] * n[v,w]/n[u,w]$
from the betweenness algorithm.

The temporal betweenness coefficients for our second example network are presented in Table 7.

10 Temporal PathFinder

The Pathfinder algorithm was proposed in the eighties (Schvaneveldt et al. 1988; Schvaneveldt 1990) [24, 25] for the simplification of weighted networks – it removes from the network all links that do not satisfy the triangle inequality – if for a weighted link there exists a shorter path connecting its endnodes then the link is removed. The basic idea of the Pathfinder algorithm is simple. It produces a network PFnet$(\mathbf{W}, r, q) = (\mathcal{V}, \mathcal{L}_{PF})$ determined by the following procedure

compute $\mathbf{W}^{(q)}$;
$\mathcal{L}_{PF} := \emptyset$;
for $e(u,v) \in \mathcal{L}$ **do begin**
  if $\mathbf{W}^{(q)}[u,v] = \mathbf{W}[u,v]$ **then** $\mathcal{L}_{PF} := \mathcal{L}_{PF} \cup \{e\}$

Algorithm 10 Temporal betweenness.

```
@staticmethod
def between(uv,vw,uw):
   if len(uv)==0: return([])
   if len(vw)==0: return([])
   if len(uw)==0: return([])
   r = []; A = uv.__iter__(); B = vw.__iter__(); C = uw.__iter__()
   (sa,fa,va) = TQ.get(A); (sb,fb,vb) = TQ.get(B); (sc,fc,vc) = TQ.get(C)
   if type(va) is tuple: (da,ca) = va
   if type(vb) is tuple: (db,cb) = vb
   if type(vc) is tuple: (dc,cc) = vc
   while (sa<TQ.inf) or (sb<TQ.inf) or (sc<TQ.inf):
      sr = max(sa,sb,sc); fr = min(fa,fb,fc)
      if fa <= sr:
         (sa,fa,va) = TQ.get(A)
         if type(va) is tuple: (da,ca) = va
      elif fb <= sr:
         (sb,fb,vb) = TQ.get(B)
         if type(vb) is tuple: (db,cb) = vb
      elif fc <= sr:
         (sc,fc,vc) = TQ.get(C)
         if type(vc) is tuple: (dc,cc) = vc
      else:
         if da+db == dc: r.append((sr,fr,ca*cb/cc))
         if fr == fa:
            (sa,fa,va) = TQ.get(A)
            if type(va) is tuple: (da,ca) = va
         if fr == fb:
            (sb,fb,vb) = TQ.get(B)
            if type(vb) is tuple: (db,cb) = vb
         if fr == fc:
            (sc,fc,vc) = TQ.get(C)
            if type(vc) is tuple: (dc,cc) = vc
   return(TQ.standard(r))

@staticmethod
def betweenness(A):
   G = TQ.MatSetVal(A,(1,1))
   old = TQ.semiring; TQ.geodetic(); n = len(G)
   C = TQ.MatClosure(G,strict=True)
   bw = [ [] for i in range(n) ]
   TQ.combinatorial()
   fac = [(1,TQ.inf,1/(n-1)/(n-2))]
   for v in range(n):
      b = []
      for u in range(n):
         for w in range(n):
            if (C[u][w]!=[]) and (u!=w) and (u!=v) and (v!=w):
               b = TQ.sum(b,TQ.between(C[u][v],C[v][w],C[u][w]))
      bw[v] = TQ.prod(b,fac)
   old()
   return(bw)
```

end;

where $\mathbf{W}$ is a network dissimilarity matrix and $\mathbf{W}^{(q)} = \bigoplus_{i=1}^{q} \mathbf{W}^i = (\mathbf{1} \oplus \mathbf{W})^q$ is the matrix of values of all walks of length at most q computed over the *Pathfinder* semiring $(\overline{\mathbb{R}}_0^+, \oplus, \boxed{r}, \infty, 0)$ with $a \boxed{r} b = \sqrt[r]{a^r + b^r}$ and $a \oplus b = \min(a, b)$.

The implementation of the Pathfinder semiring is presented in the initial part of Algorithm 11. The scheme of Pathfinder is expressed as the procedure `pathFinder`. The temporal version of the statement

if $\mathbf{W}^{(q)}[u, v] = \mathbf{W}[u, v]$ **then** $\mathcal{L}_{PF} := \mathcal{L}_{PF} \cup \{e\}$

is implemented in the procedure `PFcheck`.

Algorithm 11 Temporal PathFinder.

```
@staticmethod
def Minkowski():
   if TQ.rPF==TQ.inf: return(max)
   if TQ.rPF==1: return(operator.add)
   if TQ.rPF==2: return(lambda a,b: sqrt(a*a+b*b))
   else: return(lambda a,b: (a**TQ.rPF+b**TQ.rPF)**(1/TQ.rPF))

@staticmethod
def PFsemi():
   TQ.sAdd  = min;              TQ.sMul = TQ.Minkowski()
   TQ.sZero = TQ.inf;           TQ.sOne = 0
   TQ.sN    = [];               TQ.sE   = [(1,TQ.inf,0)]
   TQ.semiring = TQ.PFsemi

@staticmethod
def PFcheck(a,b):
   if len(a) == 0: return(a)
   if len(b) == 0: return(a)
   c = []; A = a.__iter__(); B = b.__iter__()
   (sa,fa,va) = TQ.get(A); (sb,fb,vb) = TQ.get(B)
   while (sa<TQ.inf) or (sb<TQ.inf):
      if fa <= sb: (sa,fa,va) = TQ.get(A)
      elif fb <= sa: (sb,fb,vb) = TQ.get(B)
      else:
         sc = max(sa,sb); fc = min(fa,fb)
         if vb == va: c.append((sc,fc,va))
         if fc == fa: (sa,fa,va) = TQ.get(A)
         if fc == fb: (sb,fb,vb) = TQ.get(B)
   return(TQ.standard(c))

@staticmethod
def pathFinder(W,r=1,q=inf):
   nr = len(W); nc = len(W[0])
   if nr!=nc: raise TQ.TQerror("pathFinder: square matrix required")
   PF = [[[] for v in range(nr)] for u in range(nr)]
   old = TQ.semiring; rold = TQ.rPF; TQ.rPF = r; TQ.PFsemi()
   Z = TQ.MatClosure(W) if q>nr else TQ.MatPower(TQ.MatSetDiag(W,TQ.sE),q)
   TQ.rPF = rold; old()
   for u in range(nr):
      for v in range(nr):
        PF[u][v] = TQ.PFcheck(W[u][v],Z[u][v])
   return(PF)
```

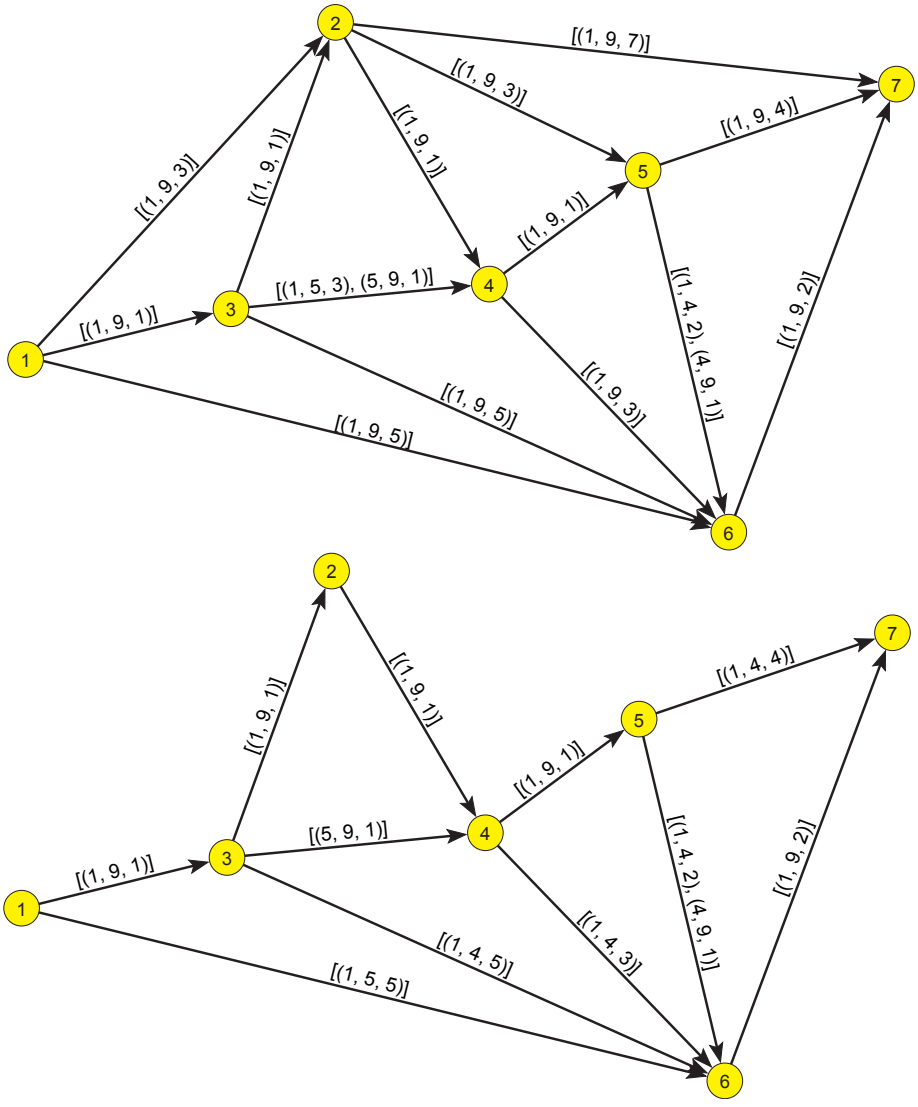

Figure 7: Pathfinder example.

The bottom network in Figure 7 presents the Pathfinder skeleton PFnet($\mathcal{N}, 1, \infty$) of a network $\mathcal{N}$ presented in the top part of the same figure. Because $r = 1$ a link e is removed if there exists a path, connecting its initial node to its terminal node, with the value (sum of link values) smaller than the value of the link e. The arc $(1, 2)$ is removed because $3 = v(1, 2) > v(1, 3) + v(3, 2) = 2$. The arc $(1, 6)$ is removed in the time interval $[5, 9)$ because in this interval $5 = v(1, 6) > v(1, 3) + v(3, 4) + v(4, 5) + v(5, 6) = 4$.

11 September 11th Reuters terror news

Reuters terror news network was obtained from the CRA networks produced by Steve Corman and Kevin Dooley at Arizona State University [5]. The network is based on all the stories released during 66 consecutive days by the news agency Reuters concerning the September 11 attack on the U.S., beginning at 9:00 AM EST 9/11/01. The nodes of this network are words (terms); there is an edge between two words iff they appear in the same text unit (sentence). The weight of an edge is its frequency. The network has $n = 13332$ nodes (different words in the news) and $m = 243447$ edges, 50859 with value larger than 1. There are no loops in the network.

The Reuters terror news network was used as a case network for the Viszards visualization session on the Sunbelt XXII International Sunbelt Social Network Conference, New Orleans, USA, 13-17. February 2002.

We transformed the Pajek version of the network into the Ianus format used in TQ. To identify important terms we computed their aggregated frequencies and extracted the subnetwork of 50 most active (during 66 days) nodes. They are listed in Table 8.

Trying to draw this subnetwork it turns out to be almost a complete graph. To obtain something readable we removed all temporal edges with a value smaller than 10. The corresponding underlying graph is presented in Figure 8. The isolated nodes were removed.

For each of the 50 nodes we determined its temporal activity and drew it. By visual inspection we identified 6 typical activity patterns – types of terms (see Figure 9). For all charts in the figure the displayed values are in the interval $[0, 200]$ – the largest activity value for term Wednesday is larger than 200.

The *primary* terms are the terms with a very high frequency of appearance in the first week after September 11th and smaller, slowly declining values in the following period. The representative of this group in Figure 9 is **hijack** and other members are: airport, american, attack, city, day, flight, nation, New York, official, Pentagon, people, plane, police, president Bush, security, tower, United States, Washington, world, World Trade center. These are the terms describing the event.

The *secondary* terms are a reaction to the event. There are no big changes in the values. We identified three subgroups: a) *slowly declining* represented with **bin Laden** (country, foreign, government, military, minister, new, Pakistan, tell, terrorism, terrorist, time, war, week); b) *stationary* represented with **taliban** (afghan, Afghanistan, force, group, leader); and c) *occasional* with several peaks, represented with **bomb** (air, building, office, strike, worker).

There are three special patterns – two *periodic* **Wednesday** and Tuesday; and one *episodic* **anthrax**.

To consider also the node's position in the network in a measure of importance of the node

Table 8: 50 most frequent terms in the Terror news network.

n	term	Σfreq	n	term	Σfreq
1	united_states	15000	26	terrorism	2212
2	attack	10348	27	day	2128
3	taliban	6266	28	week	2017
4	people	5286	29	worker	1983
5	afghanistan	5176	30	office	1967
6	bin_laden	4885	31	group	1966
7	new_york	4832	32	air	1962
8	pres_bush	4506	33	minister	1919
9	washington	4047	34	time	1898
10	official	3902	35	hijack	1884
11	anthrax	3563	36	strike	1818
12	military	3394	37	afghan	1775
13	plane	3078	38	flight	1775
14	world_trade_ctr	3006	39	tell	1746
15	security	2906	40	terrorist	1745
16	american	2825	41	airport	1741
17	country	2794	42	pakistan	1714
18	city	2689	43	tower	1685
19	war	2679	44	bomb	1674
20	tuesday	2635	45	new	1650
21	pentagon	2620	46	buildng	1634
22	force	2516	47	wednesday	1593
23	government	2380	48	nation	1589
24	leader	2375	49	police	1587
25	world	2213	50	foreign	1558

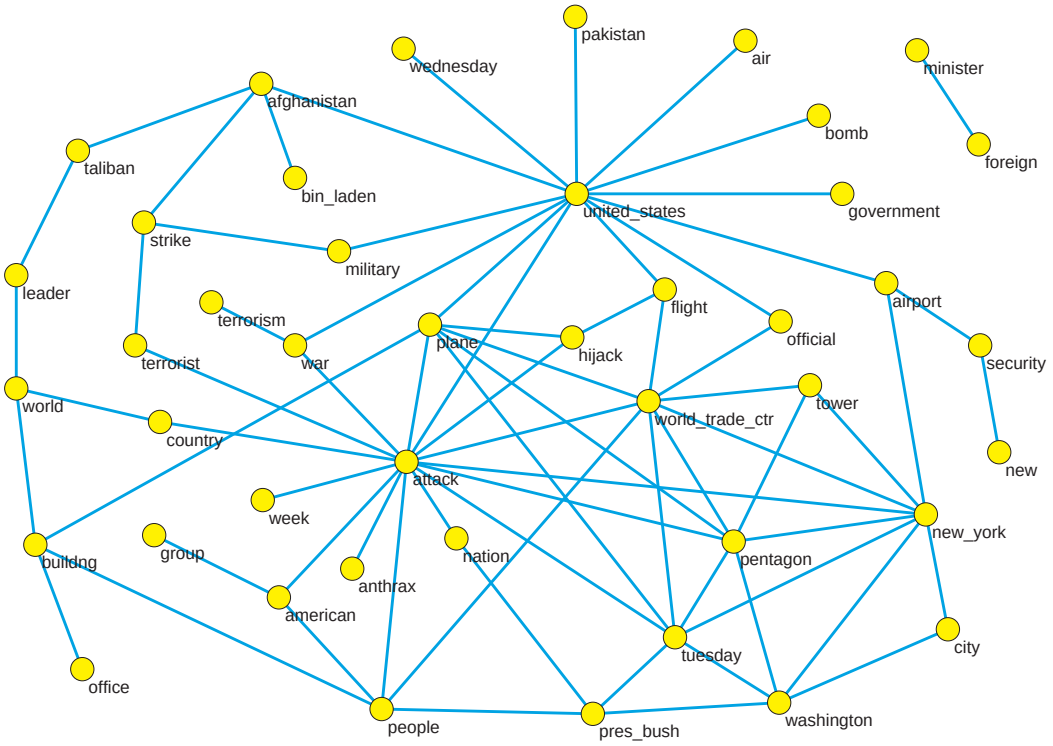

Figure 8: September 11th.

$u \in \mathcal{V}$ we constructed the attraction coefficient $\text{att}(u)$.

Let $\mathbf{W} = [w_{uv}]$ be a network matrix. We assume that the weights of links are positive $w_{uv} > 0$. We define the *node activity* $\text{act}(u)$ as (see Section 4)

$$\text{act}(u) = \text{act}(\{u\}, \mathcal{V} \setminus \{u\}) = \sum_{v \in \mathcal{V} \setminus \{u\}} w_{uv}.$$

Then the *attraction* of the node u is defined as

$$\text{att}(u) = \frac{1}{\Delta} \sum_{v \in \mathcal{V} \setminus \{u\}} \frac{w_{vu}}{\text{act}(v)}.$$

Note that the fraction $\frac{w_{vu}}{\text{act}(v)}$ is measuring the proportion of the activity of the node v that is shared with the node u.

From $0 \leq \frac{w_{vu}}{\text{act}(v)} \leq 1$ and $\deg(v) = 0 \Rightarrow w_{vu} = 0$ it follows that

$$\sum_{v \in \mathcal{V} \setminus \{u\}} \frac{w_{vu}}{\text{act}(v)} \leq \deg(u) \leq \Delta.$$

Therefore we have $0 \leq \text{att}(u) \leq 1$, for all $u \in \mathcal{V}$.

The maximum possible attraction value 1 is attained exactly for nodes: a) in undirected network: that are the root of a star; b) in directed network: that are the only out-neighbors of their in-neighbors – the root of a directed in-star.

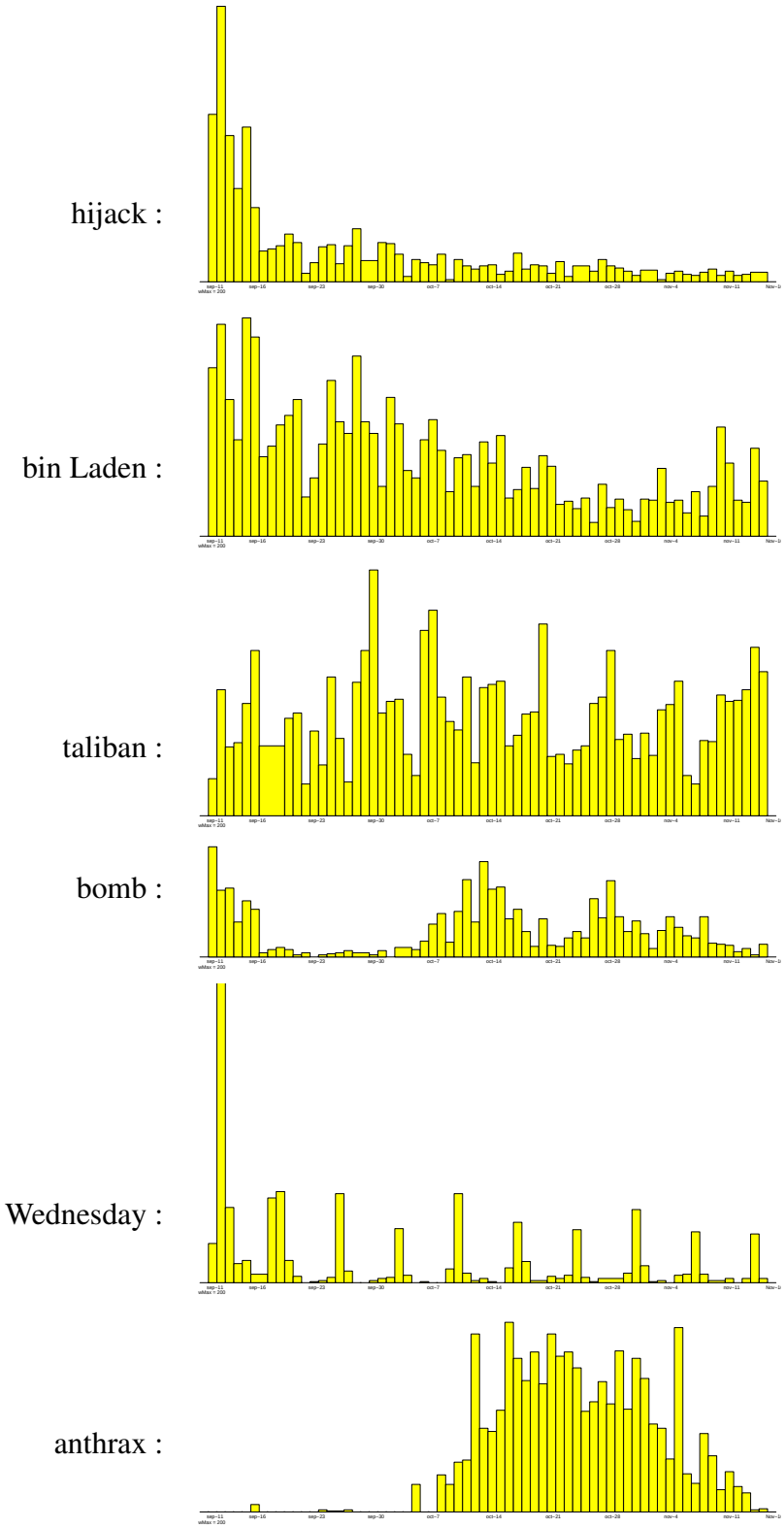

Figure 9: Types of activity.

Table 9: 30 most attractive terms in the Terror news network.

n	term	Σatt	n	term	Σatt
1	united_states	12.216	16	war	2.758
2	taliban	7.096	17	force	2.596
3	attack	7.070	18	new_york	2.590
4	afghanistan	5.142	19	government	2.496
5	people	5.023	20	day	2.338
6	bin_laden	4.660	21	leader	2.305
7	anthrax	4.601	22	terrorism	2.202
8	pres_bush	4.374	23	time	2.182
9	country	3.317	24	group	2.072
10	washington	3.067	25	afghan	2.040
11	security	2.939	26	world	1.995
12	american	2.922	27	week	1.961
13	official	2.831	28	pakistan	1.943
14	city	2.798	29	letter	1.866
15	military	2.793	30	new	1.851

We computed the temporal attraction and the corresponding aggregated attraction values for all the nodes in our network. We selected 30 nodes with the largest aggregated attraction values. They are listed in Table 9. Again we visually explored them. In Figure 10 we present temporal attraction coefficients for the 6 selected terms. For all charts in the figure the displayed attraction values are in the interval $[0, 0.2]$.

Comparing on the common terms (taliban, bomb, anthrax) the activity charts in Figure 9 with the corresponding attraction charts in Figure 10 we see that they are "correlated" (obviously act$(a)(t) = 0$ implies att$(a)(t) = 0$), but different in details.

For example, the terms taliban and bomb have small attraction values at the beginning of the time window – the terms were disguised by the primary terms. On the other hand, the terms taliban and Kabul get increased attraction towards the end of the time window.

12 Conclusions

In the paper we proposed an algebraic approach to the "deterministic" analysis of temporal networks with zero latency and presented algorithms for the temporal variants of basic network analysis measures and concepts. We expect that the support for many temporal variants of other network analysis notions can be developed in similar ways.

All the described algorithms (and some others) are implemented in a Python library TQ (temporal quantities). We started to develop a program Ianus that will provide an user-friendly (Pajek like) access to the capabilities of TQ library.

The results obtained from temporal procedures are relatively large. To identify interesting elements we used in the paper the aggregated values and the visualization of selected elements. Additional tools for browsing the results should be developed.

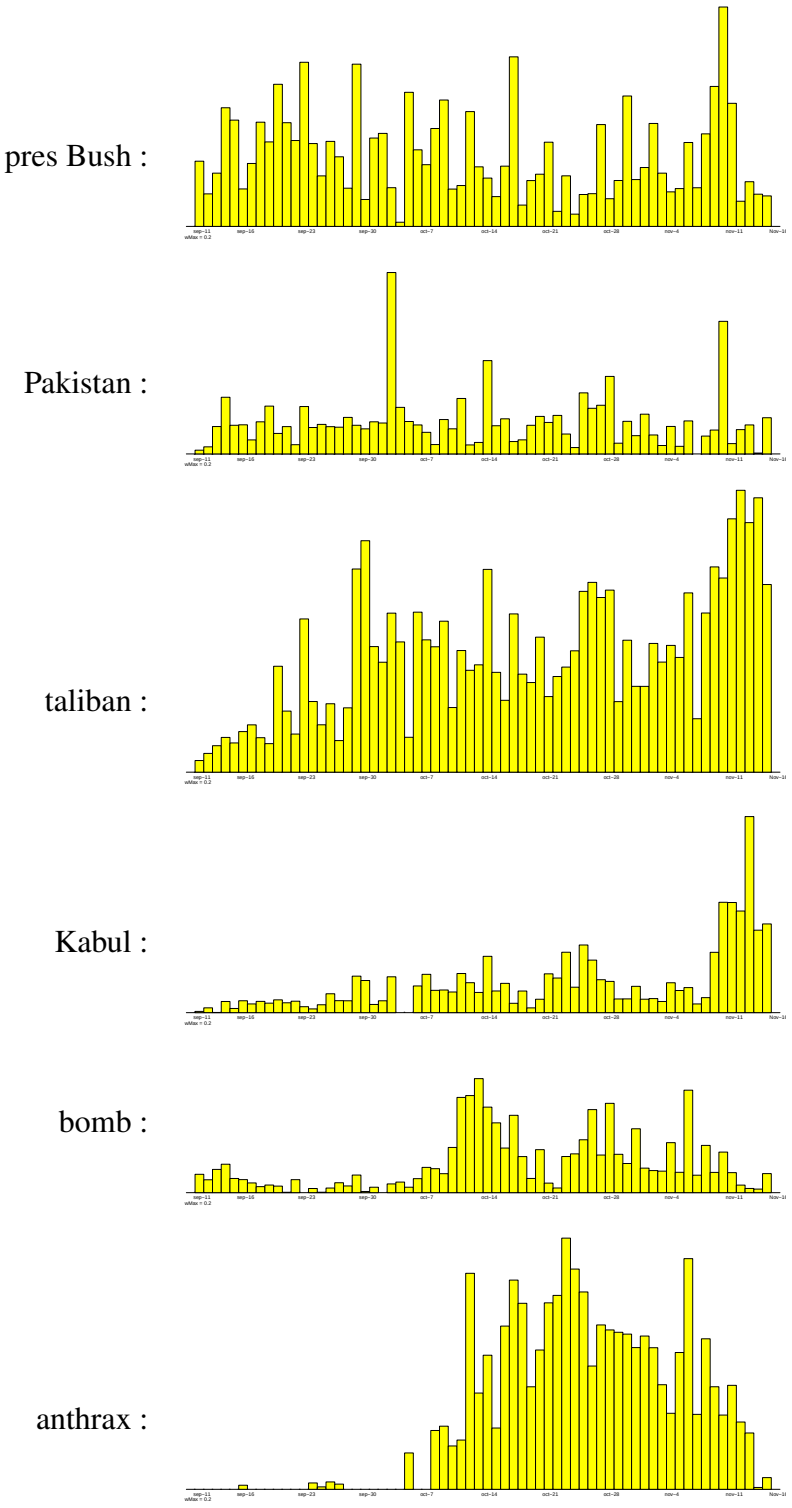

Figure 10: Attraction patterns.

Acknowledgements.

The work was supported in part by the ARRS, Slovenia, grant J5-5537, as well as by grant within the EUROCORES Programme EUROGIGA (project GReGAS) of the European Science Foundation.

References

[1] Allen, J.F.: Maintaining Knowledge about Temporal Intervals. Communications of the ACM 26, 11, 832-843, November 1983.

[2] Batagelj, V.: Semirings for social networks analysis. Journal of Mathematical Sociology, 19(1994)1, 53-68.

[3] Batagelj, V.: Social Network Analysis, Large-Scale. R.A. Meyers, ed., Encyclopedia of Complexity and Systems Science, Springer 2009: 8245-8265

[4] Bell, M.G.H., Iida, Y.: Transportation Network Analysis. Chichester: Wiley, 1997

[5] Corman, S.R., Kuhn, T., McPhee, R.D., Dooley, K.J.: Studying complex discursive systems: Centering resonance analysis of communication. Human Communication Research, 28(2002)2: 157-206.

[6] Correa, J.R., Stier-Moses, N.E.: Wardrop Equilibria. Wiley Encyclopedia of Operations Research and Management Science, 2011.

[7] Fletcher, J.G.: A more general algorithm for computing closed semiring costs between vertices of a directed graph. CACM 23 (1980), 350-351.

[8] Freeman, L.C.: Centrality in Social Networks; Conceptual Clarification. Social Networks 1 (1978), 215-239.

[9] Casteigts, A., Flocchini, P.: Deterministic Algorithms in Dynamic Networks: Formal Models and Metrics Commissioned by Defense Research and Development Canada (DRDC), 82p, 2013.

[10] Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-varying graphs and dynamic networks. International Journal of Parallel, Emergent and Distributed Systems, 27(2012)5, 387-408.

[11] Dechter, R. (ed.): Constraint Processing. Morgan Kaufmann, San Francisco, 2003.

[12] Franzosi, R.: Mobilization and Counter-Mobilization Processes: From the "Red Years" (1919-20) to the "Black Years" (1921-22) in Italy. A New Methodological Approach to the Study of Narrative Data. Theory and Society, 26(1997)2-3, 275-304.

[13] George, B., Kim, S., Shekhar, S.: Spatio-temporal Network Databases and Routing Algorithms: A Summary of Results. D. Papadias, D. Zhang, and G. Kollios (Eds.): SSTD 2007, LNCS 4605, Springer-Verlag, Berlin, Heidelberg, pp. 460-477, 2007.

[14] Holme, P., Saramäki, J.: Temporal networks. Physics Reports. Vol 519, Issue 3, 2012, p 97–125.

[15] Holme, P., Saramäki, J. (Eds.): Temporal Networks. Understanding Complex Systems. Springer, 2013.

[16] D. Kempe, J. Kleinberg, A. Kumar. Connectivity and inference problems for temporal networks. Proc. 32nd ACM Symposium on Theory of Computing, 2000.

[17] Kolaczyk, E.D.: Statistical Analysis of Network Data: Methods and Models. New York: Springer, 2009.

[18] Kontoleon, N., Falzon, L., Pattison, P.: Algebraic structures for dynamic networks. Journal of Mathematical Psychology, 57(2013)6, 310–319.

[19] Moder, J.J., Phillips, C.R: Project Management with CPM and Pert. Second Edition, Van Nostrand Reinhold, 1970.

[20] Nicosia, V., Tang, J., Mascolo, C., Musolesi, M., Russo, G., Latora, V.: Graph Metrics for Temporal Networks. Chapter in Petter Home and Jari Saramaki (Editors). Temporal Networks. Springer. 2013, 15-40.

[21] de Nooy, W., Mrvar, A., Batagelj, V.: *Exploratory Social Network Analysis with Pajek* (Structural Analysis in the Social Sciences), revised and expanded second edition. Cambridge University Press, Cambridge, 2012.

[22] Snijders, T.: Siena. http://www.stats.ox.ac.uk/ snijders/siena/

[23] Riordan, J.: Introduction to combinatorial analysis. New York: Wiley, 1958.

[24] Schvaneveldt, R. W., Dearholt, D. W., Durso, F. T.: Graph theoretic foundations of Pathfinder networks. Comput. Math. Applic. **15**(1988)4, 337-345.

[25] Schvaneveldt, R.W. (Ed.): Pathfinder Associative Networks: Studies in Knowledge Organization. Norwood, NJ: Ablex, 1990.

[26] Vilain, M., Kautz, H., Van Beek, P.: Constraint Propagation Algorithms for Temporal Reasoning; A revised Report. In D.S. Weld, J. de Kleer (eds.) Readings in Qualitative Reasoning about Physical Systems, p 373-381, Morgan Kaufmann, 1990.