Analysis of temporal networks

Vladimir Batagelj

IMFM Ljubljana and IAM UP Koper

XXXVII Sunbelt 2017 workshop

Beijing, China, May 30th – June 4th, 2017

Outline

Vladimir Batagelj: vladimir.batagelj@fmf.uni-lj.si

Current version of slides (May 31, 2017 at 04 : 55):
Sunbelt 2017 workshop slides PDF

Work in progress!!!

We present a longitudinal approach to analysis of temporal networks that was proposed in our paper:

> *Batagelj, V., Praprotnik, S.: An algebraic approach to temporal network analysis based on temporal quantities. Social Network Analysis and Mining, 6(2016)1, 1-22.*

To describe their temporal behavior we assign temporal quantities to nodes and links of a network. The proposed approach enables us to treat as temporal quantities also other network characteristics such as degrees, connectivity components, centrality measures, Pathfinder skeleton, cores, etc. It is an alternative to the usual approach to temporal network analysis based on time slices. We developed fast algorithms for the proposed operations.

. . . Introduction

They are available as an open source Python library TQ:

http://vladowiki.fmf.uni-lj.si/doku.php?id=tq

and

https://github.com/bavla/TQ/ .

The initial version of TQ library was based on matrices and on an extended Pajek input format. This limits the application of the proposed methods to some thousands of nodes (space and time complexity). The limits can be partially extended by switching to graph representation

https://github.com/bavla/graph/ .

Also for the input format we decided to base it on JSON – a netJSON format.

Networks and semirings

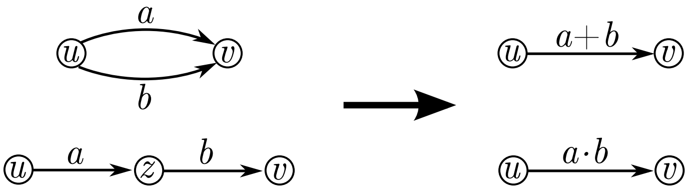
A *network* $\mathcal{N} = (\mathcal{V}, \mathcal{L}, \mathcal{P}, \mathcal{W})$: $\mathcal{V}$ is a set of nodes, $\mathcal{L}$ is a set of links (undirected – edges, directed – Arcs),$\mathcal{P}$ are properties of nodes, and $\mathcal{W}$ are properties of links, usually called weights.

We assume that the values of a property belong to a set A which is a *semiring* $(A, +, \cdot, 0, 1)$ for binary operations $+ : A \times A \to A$ and $\cdot : A \times A \to A$.

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.

Semiring addition and multiplication in networks.

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



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

Networks and semirings

We extend the weight w to a walk $\sigma = p_1 p_2 p_3 \dots p_k$

$$w(\sigma) = w(p_1) \cdot w(p_2) \cdot w(p_3) \cdots w(p_k)$$

and to a finite set of walks $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \dots, \sigma_m\}$

$$w(\Sigma) = w(\sigma_1) + w(\sigma_2) + w(\sigma_3) + \cdots + w(\sigma_m)$$

It holds

$$w(\Sigma_1) + w(\Sigma_2) = w(\Sigma_1 \cup \Sigma_2) + w(\Sigma_1 \cap \Sigma_2)$$

In some applications other semirings are useful.

Examples of semirings

The *reachability* semiring `reach` $(\{0,1\}, \vee, \wedge, 0, 1)$ for reachability problems.

The *maxmin* semiring `maxmin` $(\overline{\mathbb{R}}, \max, \min, -\infty, \infty)$.

The semiring `path` $(\overline{\mathbb{R}}_0^+, \min, +, \infty, 0)$ is suitable to deal with the *shortest paths* problem in networks.

The *geodetic* semiring `geodetic` $(\overline{\mathbb{R}}_0^+ \times \overline{\mathbb{N}}, \oplus, \odot, (\infty, 0), (0, 1))$, 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 *Pathfinder* semiring `PFsemi` $(\overline{\mathbb{R}}_0^+, \min, \boxed{r}, \infty, 0)$ with $a \boxed{r}\, b = \sqrt[r]{a^r + b^r}$.

Computing in semirings

Python library TQ

```
>>> gdir = 'c:/users/batagelj/work/python/graph/graph'
>>> import sys, os, datetime
>>> sys.path = [gdir]+sys.path
>>> from TQ import *
>>> TQ.report()
semiring =  combinatorial
add    =   add
mult   =   mul
sZero =   0
sOne   =   1
...
>>> TQ.sAdd(3,4)
7
>>> TQ.sMul(3,4)
12
>>> TQ.path()
>>> TQ.report()
semiring =  path
add    =   min
mult   =   add
sZero =   inf
sOne   =   0
...
>>> TQ.sAdd(3,4)
3
>>> TQ.sMul(3,4)
7
>>>
```

Temporal networks

A *temporal network* $\mathcal{N}_T = (\mathcal{V}, \mathcal{L}, \mathcal{T}, \mathcal{P}, \mathcal{W})$ is obtained by attaching the *time*, $\mathcal{T}$, to an ordinary network where $\mathcal{T}$ is a set of *time points*, $t \in \mathcal{T}$.

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.

Besides the presence/absence of nodes and links also their properties can change through time.

Temporal quantities

We introduce a notion of a *temporal quantity*

$$a(t) = \begin{cases} a'(t) & t \in T_a \\ \maltese & t \in \mathcal{T} \setminus T_a \end{cases}$$

where T_a is the *activity time set* of a and $a'(t)$ is the value of a in an instant $t \in T_a$, and $\maltese$ 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 $+$ and $\cdot$.

We can extend both operations to the set $A_\maltese = A \cup \{\maltese\}$ by requiring that for all $a \in A_\maltese$ it holds

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

The structure $(A_\maltese, +, \cdot, \maltese, 1)$ is also a semiring.

Operations with temporal quantities

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

$$(a + b)(t) = a(t) + b(t) \quad \text{and} \quad T_{a+b} = T_a \cup T_b.$$

The *product* (sequential links) $a \cdot b$ is defined as

$$(a \cdot b)(t) = a(t) \cdot b(t) \quad \text{and} \quad T_{a \cdot b} = T_a \cap T_b.$$

Let us define the temporal quantities $\mathbf{0}$ and $\mathbf{1}$ with requirements $\mathbf{0}(t) = ⌘$ and $\mathbf{1}(t) = 1$ for all $t \in \mathcal{T}$. Again, the structure $(A_{⌘}(\mathcal{T}), +, \cdot, \mathbf{0}, \mathbf{1})$ is a semiring.

Semiring for temporal quantities

A semiring is also the set of square matrices of order n over $A_{⌘}(\mathcal{T})$ it for addition $\mathbf{A} \oplus \mathbf{B} = \mathbf{S}$

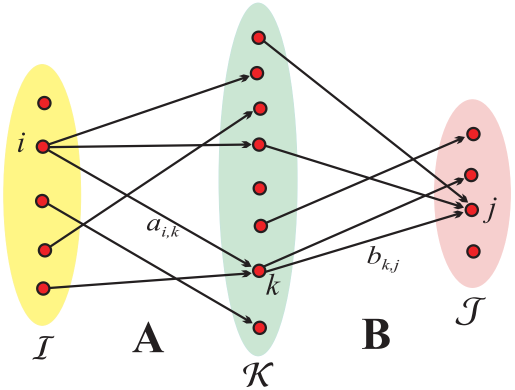
$$s_{ij} = a_{ij} + b_{ij}$$

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 (at the same time) – the transition from the node i to the node j through a node k is possible. Its contribution is $a_{ik} \cdot b_{kj}$.

Multiplication of networks



We will limit our attention to networks with *zero latency* in which the transitions on links 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 **A** and a link (k, j) in **B**, for some node k.

Temporal quantities on intervals

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

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

where I_i is a time-interval and v_i is a constant value of a on this interval. $T_a = \bigcup_{i=1}^{k} I_i$. To simplify the exposition we will assume in the following that all the intervals are of the form $[s_i, f_i)$, s_i is the starting time and f_i is the finishing time, $s_i < f_i$ and $f_i \leq s_{i+1}$. Therefore we can describe the temporal quantities with sequences

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

Library TQ

temporal quantities

To provide a computational support for the proposed approach we are developing in Python a library TQ (Temporal Quantities). Its initial version was based on matrix representation of networks.

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, ⌘.

Operations

For the simplified version of temporal quantities we wrote procedures `sum` for addition and `prod` for multiplication of temporal quantities over the selected semiring. Because the intervals in TQs are ordered we can base both operatins on "merging of seqeunces".

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.

In procedures for operations with temporal quantities we use

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

Addition of temporal quantities

```
@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))
```

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))
```

Sum and product of temporal quantities


```
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)]
```

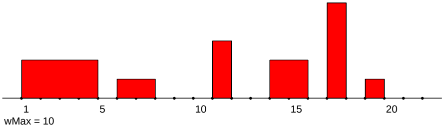
The following are the sum $s = a + b$ and the product $p = a \cdot b$ of temporal quantities a and b over combinatorial semiring.

```
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)]
```

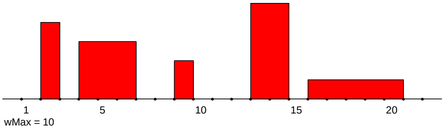
They are visually displayed at the bottom half of figures on the following slides.

Addition of temporal quantities

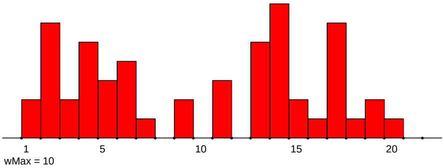
a :



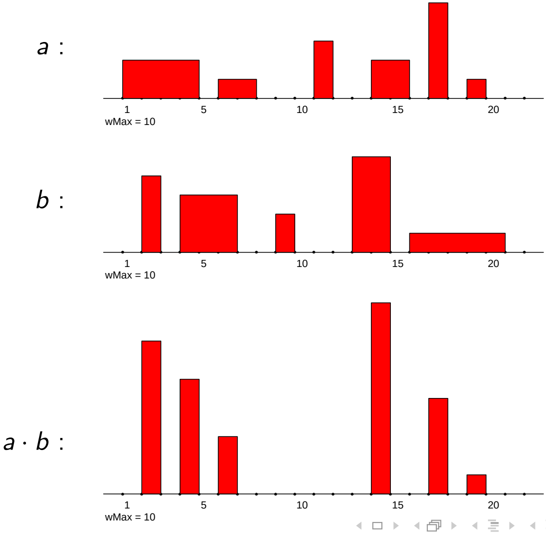
b :



$a + b$:



Multiplication of temporal quantities



Adding and multiplying TQs 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)]
>>> TQ.combinatorial()
>>> s = TQ.sum(a,b)
>>> 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,
 (16, 17, 1), (17, 18, 6), (18, 19, 1), (19, 20, 2), (20, 21, 1)]
>>> p = TQ.prod(a,b)
>>> p
[(2, 3, 8), (4, 5, 6), (6, 7, 3), (14, 15, 10), (17, 18, 5), (19, 20, 1)
>>> TQ.TqSummary(s)
(1, 21, 1, 7)
>>> TQ.TqSummary(p)
(2, 20, 1, 10)
>>> from GraphNew import Graph
>>> cdir = 'c:/users/batagelj/work/python/graph/chart'
>>> TQmax = 12; Tmin = 0; Tmax = 21; w = 600; h = 180
>>> tit = 'temporal sum'
>>> Graph.TQshow(s,cdir,TQmax,Tmin,Tmax,w,h,tit,fill='red')
>>>
```

Pictures of TQs

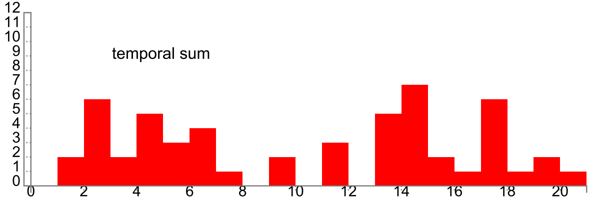
The procedure `Graph.TQshow` displays a given TQ using the Chrome web browser. The picture can be saved either as a bitmap or as an SVG file. To do the later activate in Chrome

`Tools/More tools/Developer tools`

Select the line with the `<svg>` tag and right click on it. Select the option

`Copy/Copy element`

and open a text editor. Paste the element and save as an SVG.file. Using Inkscape or some other vector graphics editor you can convert it to PDF.



The aggregated value

In some applications over the combinatorial semiring 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 $\Sigma a + \Sigma b = \Sigma(a + b)$.

```
>>> TQ.total(a)
23
>>> TQ.total(b)
30
>>> TQ.total(s)
53
```

netJSON format

For describing temporal networks we initially, extending Pajek format, defined and used a Ianus format.

Recently we started to develop a new format based on JSON – we named it netJSON.

netJSON has two formats: a *basic* and a *general* format. Current implementation of the TQ library supports only the basic format.

Informal description of the basic netJSON format


```
{
"netJSON": "basic",
"info": {
   "org":1, "nNodes":n, "nArcs":mA, "nEdges":mE,
   "simple":TF, "directed":TF, "multirel":TF, "mode":m,
   "network":fName, "title":title,
   "time": { "Tmin":tm, "Tmax":tM, "Tlabs": {labs} },
   "meta": [events], ...
   },
"nodes": [
   { "id":nodeId, "lab":label, "x":x, "y":y, ... },
   ***
   ]
"links": [
   { "type":arc/edge, "n1":nodeID1, "n2":nodeID2, "rel":r, ... }
   ***
   ]
}
```

where ... are user defined properties and *** is a sequence of such elements.

Basic netJSON formats

An event description can contain fields:

```
{  "date": date,
   "title": short description,
   "author": name,
   "desc": long description,
   "url": URL,
   "cite": reference,
   "copy": copyright
}
```

for describing temporal networks a node element and a link element has an additional required property `tq`

Example 1, Franzosi's violence network / UTF-8 no sig

Python library Graph

The Pythoni library **Graph** supports basic operations with networks based on graph representation. Each node/link has its *id*. If a link id is not specified by a user it is determined by **Graph**.

The library **Graph** is based on an object containing three dictionaries:

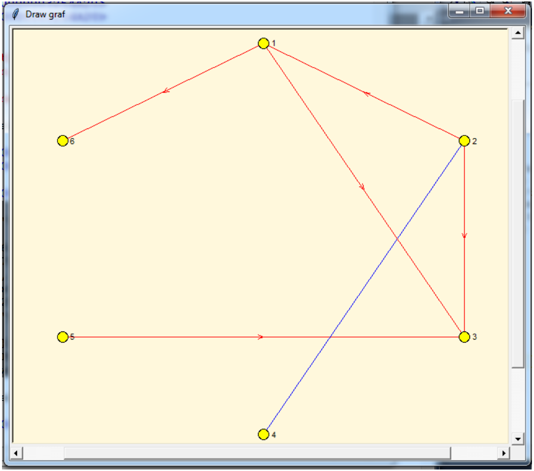
- `_graph` – keys are general properties of a network. System properties: `network`, `title`, `simple`, `directed`, `multirel`, `mode`, `temporal`, `meta`, `nNodes`, `nArcs`, `nEdges`, `time`, etc. User properties: `nWeak`, `planar`, etc.
- `_nodes` – keys are ids of nodes. A value is a list of four dictionaries:
  [edgeStar, inArcStar, outArcStar, nodeProperties]
  Each star is again a dictionary that has for keys ids of neighboring nodes and for values lists of link ids.
- `_links` – keys are ids of links. A value is a list
  [nodeId1, nodeId2, directed, relId, linkProperties]
  where linkProperties is a dictionary of weights.

Some basic functions for network construction

A new version of **Graph** labeled **GraphNew** is in development. It contains also support for TQs. Attention! It can happen that not all procedures were updated.

```
from GraphNew import Graph
def TestAdd():
    G = Graph()
    G.addNode(2); G.addNode(1); G.addNode(3); G.addNode(4)
    G.addEdge(2,4,{'w':3}); G.addArc(2,1,{'w':5});
    G.addArc(1,3,{'w':4}); G.addArc(2,3,{'w':6})
    G.addNode(5); G.addNode(6)
    i=G.addArc(5,3,{'w':5}); j=G.addEdge(2,4,{'w':7});
    G.addArc(1,6,{'w':8});G.addArc(1,3,{'w':5})
    G.onCircle()
    print(G)
    G.draw(800,800,"Cornsilk")
    G.savePajek('test.net')
    G.delLink(j); G.delLink(i)
    print(G)
    return G
```

Picture of network



Network representation in Python


```
>>> G._graph
{'mode': 1, 'multirel': False, 'temporal': False, 'simple': False}
>>> G._nodes
{
1: [{}, {2: [2]}, {3: [3, 8], 6: [7]}, {'x': 0.5, 'y': 0.95}],
2: [{4: [1]}, {}, {1: [2], 3: [4]}, {'x': 0.88971, 'y': 0.725}],
3: [{}, {1: [3, 8], 2: [4], 5: []}, {}, {'x': 0.88971, 'y': 0.275}],
4: [{2: [1]}, {}, {}, {'x': 0.5, 'y': 0.045}],
5: [{}, {}, {3: []}, {'x': 0.11029, 'y': 0.275}],
6: [{}, {1: [7]}, {}, {'x': 0.11029, 'y': 0.725}]
}
>>> G._links
{
1: [2, 4, False, None, {'w': 3}],
2: [2, 1, True, None, {'w': 5}],
3: [1, 3, True, None, {'w': 4}],
4: [2, 3, True, None, {'w': 6}],
7: [1, 6, True, None, {'w': 8}],
8: [1, 3, True, None, {'w': 5}]
}
```

Some new functions and programs

```
TQ.Ianus2netJSON(N,fileJSON="test.json",indent=None)
twoMode2netJSON(yFile,netFile,jsonFile,instant=True)

Graph.TQmultiply(A,B,oneMode=False)
Graph.TQtwo2oneRows(self)
Graph.TQtwo2oneCols(self)
Graph.TQnetDeg(self,u)
Graph.TQnetInDeg(self,u)
Graph.TQnetOutDeg(self,u)
Graph.TQnetSum(self,u)
Graph.TQnetInSum(self,u)
Graph.TQnetOutSum(self,u)
Graph.TQnetBin(self)
Graph.loadNetJSON(file, encoding='utf-8')
Graph.TQgraph2mat(self)
multiply.py
```

Node activities and degrees

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. 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: $all = \text{act}(\mathcal{V}, \mathcal{V})$,
$pol = \text{act}(\{\text{police}\}, \mathcal{V}) + \text{act}(\mathcal{V}, \{\text{police}\})$, and
$fas = \text{act}(\{\text{fascists}\}, \mathcal{V}) + \text{act}(\mathcal{V}, \{\text{fascists}\})$.

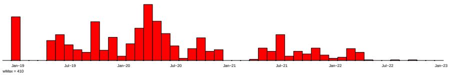
They are presented on the next slide. Comparing the intensity charts of police and fascists activity with overall activity we see that most of the violent activity in the first two years 1919 and 1920 was related to the police. In the next two years (1921 and 1922) it was taken over by the fascists.

Activities - Python

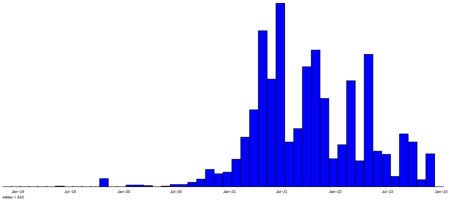

```
gdir = 'c:/users/batagelj/work/python/graph/graph'
cdir = 'c:/users/batagelj/work/python/graph/chart'
wdir = 'c:/users/batagelj/Documents/papers/2017/INSNA/ws/ex/violence'
import sys, os, datetime
sys.path = [gdir]+sys.path; os.chdir(wdir)
from TQ import *
from GraphNew import Graph
TQ.report()
file = 'violenceU.json'
G = Graph.loadNetJSON(file)
for v in G.nodes(): print(v,G.getNode(v,'lab'))
# All-All
n = len(G); All = range(1,n+1,1)
ca = G.TQactivity(All,All)
TQ.TqSummary(ca)
TQmax = 420; Tmin = 0; Tmax = 50; w = 800; h = 500
tit = 'violence'
Graph.TQshow(ca,cdir,TQmax,Tmin,Tmax,w,h,tit,fill='orange')
# police - 4
cb = TQ.sum(G.TQactivity([4],All),G.TQactivity(All,[4]))
tit = 'violence: all - police'
Graph.TQshow(cb,cdir,TQmax,Tmin,Tmax,w,h,tit,fill='red')
# fascists - 7
cc = TQ.sum(G.TQactivity([7],All),G.TQactivity(All,[7]))
tit = 'violence: all - fascists'
Graph.TQshow(cc,cdir,TQmax,Tmin,Tmax,w,h,tit,fill='blue')
```

Violent activities of police, fascists and all

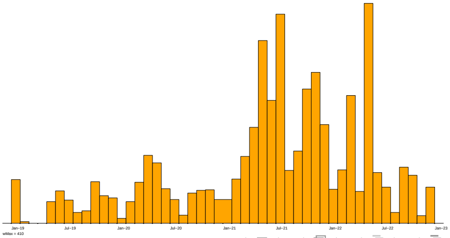
police :



fascists :



all :



Networks from bibliographic data

From special bibliographies (BibTEX) and bibliographic services (Web of Science, Scopus, SICRIS, CiteSeer, Zentralblatt MATH, Google Scholar, DBLP Bibliography, US patent office, and others) we can derive some two-mode networks on selected topics:
– works $\times$ authors (**WA**),
– works $\times$ keywords (**WK**);
and from some data also the network
– works $\times$ classification (**WC**), and the
– one-mode citation network works $\times$ works (**Ci**);
where works include papers, reports, books, patents etc.

Besides this we get also at least the partition of works by the journal or publisher and the partition of works by the publication year.

For converting WoS file into networks in `Pajek`'s format a program `WoS2Pajek` was developed (in Python).

Temporal co-occurrence networks

Let the binary matrix $\mathbf{A} = [a_{ep}]$ describe a two-mode network on the set of events E and the set of of participants P:

$$a_{ep} = \begin{cases} 1 & p \text{ participated in the event } e \\ 0 & \text{otherwise} \end{cases}$$

The function $d : E \to \mathcal{T}$ assigns to each event e the date $d(e)$ when it happened. $\mathcal{T} = [\mathit{first}, \mathit{last}] \subset \mathbb{N}$. Using these data we can construct two temporal affiliation matrices:

- **instantaneous Ai** $= [ai_{ep}]$, where

$$ai_{ep} = \begin{cases} [(d(e), d(e)+1, 1)] & a_{ep} = 1 \\ [\,] & \text{otherwise} \end{cases}$$

- **cumulative Ac** $= [ac_{ep}]$, where

$$ac_{ep} = \begin{cases} [(d(e), \mathit{last}+1, 1)] & a_{ep} = 1 \\ [\,] & \text{otherwise} \end{cases}$$

Multiplication of co-occurence networks

Instantaneous

Instantaneous **A** on $P \times A$ and **B** on $P \times B$. $\mathbf{C} = \mathbf{A}^T.\mathbf{B}$ on $A \times B$.

$$c_{ij}(t) = \sum_{p \in P} a_{pi}(t)^T \cdot b_{pj}(t)$$

$a_{pi} = [(d_{pi}, d_{pi}+1, v_{pi})]$ and $b_{pj} = [(d_{pj}, d_{pj}+1, v_{pj})]$
for $t = d$ we get

$$c_{ij} = [(d, d+1, \sum_{p \in P : d_{pi} = d_{pj} = d} v_{pi}.v_{pj})]_{d \in \mathcal{T}}$$

for $v_{pi} = v_{pj} = 1$ we finally get

$$v_{ij}(d) = |\{p \in P : d_{pi} = d_{pj} = d\}|$$

For binary temporal two-mode networks **A** and **B** the value $v_{ij}(d)$ of the product $\mathbf{A}^T.\mathbf{B}$ is equal to the number of different members of P with which both i and j have contact in the instant d.

Cumulative $\mathbf{A}$ on $P \times A$ and $\mathbf{B}$ on $P \times B$. $\mathbf{C} = \mathbf{A}^T.\mathbf{B}$ on $A \times B$.

$$c_{ij}(t) = \sum_{p \in P} a_{pi}(t)^T \cdot b_{pj}(t)$$

$a_{pi} = [(d_{pi}, last + 1, v_{pi})]$ and $b_{pj} = [(d_{pj}, last + 1, v_{pj})]$
for $t = d$ we get

$$c_{ij} = [(d, d+1, \sum_{p \in P : (d_{pi} \leq d) \wedge (d_{pj} \leq d)} v_{pi} . v_{pj})]_{d \in \mathcal{T}}$$

for $v_{pi} = v_{pj} = 1$ we finally get

$$v_{ij}(d) = |\{p \in P : (d_{pi} \leq d) \wedge (d_{pj} \leq d)\}|$$

For binary temporal two-mode networks $\mathbf{A}$ and $\mathbf{B}$ the value $v_{ij}(d)$ of the product $\mathbf{A}^T.\mathbf{B}$ is equal to the number of different members of P with which both i and j have contact in all instants up to including the instant d.

Temporal co-authorship networks

Using the multiplication of temporal matrices over the combinatorial semiring we get the corresponding instantaneous and cumulative co-occurrence matrices

$$\mathbf{Ci} = \mathbf{Ai}^T \cdot \mathbf{Ai} \qquad \text{and} \qquad \mathbf{Cc} = \mathbf{Ac}^T \cdot \mathbf{Ac}$$

A typical example of such a matrix is the papers authorship matrix **WA** where E is the set of papers W, P is the set of authors A and d is the publication year.

The triple (s, f, v) in a temporal quantity ci_{pq} tells that in the time interval $[s, f)$ there were v events in which both p and q took part.

The triple (s, f, v) in a temporal quantity cc_{pq} tells that in the time interval $[s, f)$ there were in total v accumulated events in which both p and q took part.

The diagonal matrix entries ci_{pp} and cc_{pp} contain the temporal quantities counting the number of events in the time intervals in which the participant p took part.

Temporal co-authorship networks

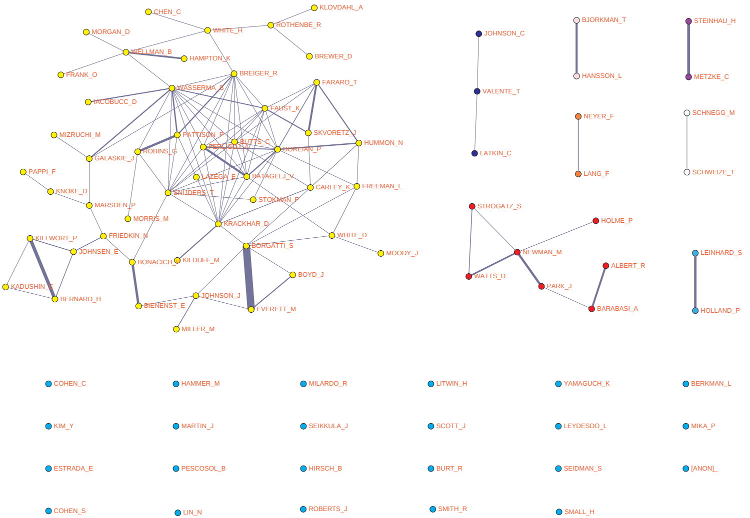
For an example, from a collection SN5 of network data about publications on social networks till 2008 we extracted data about 92 the most active researchers and transformed them to corresponding temporal networks: CiteInst, CiteCum, WAinst, WAcum, WKinst, WKcum and a partition W92 with outdegrees of works in the original WA network.

The matrices

$$\mathbf{Coi} = \mathbf{WAi}^T \cdot \mathbf{WAi} \qquad \text{and} \qquad \mathbf{Coc} = \mathbf{WAc}^T \cdot \mathbf{WAc}$$

describe the instantaneous co-autorship temporal network and the cumulative co-autorship temporal network.

92 authors


CHEN_C
KLOVDAHL_A
MORGAN_D
WHITE_H
ROTHENBE_R
WELLMAN_B
HAMPTON_K
BREWER_D
FRANK_O
BREIGER_R
FARARO_T
WASSERMA_S
IACOBUCC_D
FAUST_K
MIZRUCHI_M
PATTISON_P
SKVORETZ_J
HUMMON_N
BUTTS_C
GALASKIE_J
ROBINS_G
DOREIAN_P
PAPPI_F
LAZEGA_E
BATAGELJ_V
KNOKE_D
SNIJDERS_T
CARLEY_K
FREEMAN_L
MARSDEN_P
STOKMAN_F
MORRIS_M
KRACKHAR_D
KILLWORT_P
FRIEDKIN_N
WHITE_D
JOHNSEN_E
BORGATTI_S
MOODY_J
BONACICH_P
KILDUFF_M
KADUSHIN_C
BOYD_J
BERNARD_H
JOHNSON_J
BIENENST_E
EVERETT_M
MILLER_M
JOHNSON_C
BJORKMAN_T
STEINHAU_H
VALENTE_T
HANSSON_L
METZKE_C
NEYER_F
SCHNEGG_M
LATKIN_C
LANG_F
SCHWEIZE_T
STROGATZ_S
HOLME_P
NEWMAN_M
LEINHARD_S
ALBERT_R
WATTS_D
PARK_J
BARABASI_A
HOLLAND_P
COHEN_C
HAMMER_M
MILARDO_R
LITWIN_H
YAMAGUCH_K
BERKMAN_L
KIM_Y
MARTIN_J
SEIKKULA_J
SCOTT_J
LEYDESDO_L
MIKA_P
ESTRADA_E
PESCOSOL_B
HIRSCH_B
BURT_R
SEIDMAN_S
[ANON]_
COHEN_S
LIN_N
ROBERTS_J
SMITH_R
SMALL_H

Temporal co-authorship networks


```
>>> import os, sys, datetime
>>> os.chdir("C:/Users/batagelj/work/Python/WoS/SN5/ten")
>>> from TQ import *
>>> wai = TQ.Ianus2Mat("WAinst.ten")
>>> wac = TQ.Ianus2Mat("WAcum.ten")
>>> list(wai.keys())
['dim', 'met', 'typ', 'nam', 'mat', 'til', 'tin', 'tit']
>>> wai['dim']
(1346, 92, 1970, 2008)
>>> WAi = wai['mat']; WAc = wac['mat']
>>> AWi = TQ.MatTrans(WAi); AWc = TQ.MatTrans(WAc)
>>> Coi = TQ.MatProd(AWi,WAi); Coc = TQ.MatProd(AWc,WAc)
>>> auNames = wai['nam'][nr:]
>>> ia=dict(zip(auNames,range(92)))
>>> Coi[ia['BORGATTI_S']][ia['EVERETT_M']]
[(1988, 1989, 1), (1989, 1990, 2), (1990, 1991, 4),
 (1991, 1992, 1), (1992, 1995, 2), (1996, 1998, 1),
 (1999, 2000, 3), (2003, 2004, 1), (2005, 2007, 1)]
>>> Coc[ia['BORGATTI_S']][ia['EVERETT_M']]
[(1988, 1989, 1), (1989, 1990, 3), (1990, 1991, 7),
 (1991, 1992, 8), (1992, 1993, 10), (1993, 1994, 12),
 (1994, 1996, 14), (1996, 1997, 15), (1997, 1999, 16),
 (1999, 2003, 19), (2003, 2005, 20), (2005, 2006, 21),
 (2006, 2008, 22)]
```

Authors and keywords

Using the multiplication of temporal matrices over the combinatorial semiring on bibliographic matrices **WA** and **WK** we get the corresponding instantaneous and cumulative matrices

$$\mathbf{AKi} = \mathbf{WAi}^T \cdot \mathbf{WKi} \quad \text{and} \quad \mathbf{AKc} = \mathbf{WAc}^T \cdot \mathbf{WKc}$$

The triple (s, f, v) in a temporal quantity aki_{ak} tells that in the time interval $[s, f)$ the author a used the keyword k v times (in v works).

The triple (s, f, v) in a temporal quantity akc_{ak} tells that in an instant t in the time interval $[s, f)$ the author a used cumulatively (till time t) the keyword k v times (in v works).

Authors and keywords

```
>>> wki = TQ.Ianus2Mat("WKinst.ten")
>>> AKi = TQ.MatProd(AWi,wki['mat'])
>>> kwNames = wki['nam'][nr:]
>>> len(kwNames)
8571
>>> ik=dict(zip(kwNames,range(8571)))
>>> Bc = [ AKi[i][ik['centrality']] for i in range(92)]
>>> [auNames[i] for i in range(92) if Bc[i]!=[]]
['BORGATTI_S', 'CARLEY_K', 'GALASKIE_J', 'BURT_R', 'FREEMAN_L',
 'NEWMAN_M', 'BARABASI_A', 'WELLMAN_B', 'KNOKE_D', 'PAPPI_F',
 'HOLME_P', 'WATTS_D', 'JOHNSON_C', 'WHITE_D', 'BREWER_D',
 'MARSDEN_P', 'ROTHENBE_R', 'VALENTE_T', 'SNIJDERS_T',
 'KRACKHAR_D', 'WHITE_H', 'KILDUFF_M', 'LEYDESDO_L',
 'KLOVDAHL_A', 'MOODY_J', 'FRANK_O', 'BONACICH_P', 'BATAGELJ_V',
 'JOHNSON_J', 'FAUST_K', 'MIZRUCHI_M', 'YAMAGUCH_K',
 'FRIEDKIN_N', 'LAZEGA_E', 'CHEN_C', 'KILLWORT_P', 'ESTRADA_E',
 'BUTTS_C', 'EVERETT_M', 'FERLIGOJ_A', 'IACOBUCC_D']
```

Authors and keywords

```
>>> T = [ (i,TQ.total(Bc[i])) for i in range(92) ]
>>> I = sorted(T,key=lambda e:e[1],reverse=True)
>>> [[auNames[i],v,Bc[i]] for (i,v) in I[:5]]
[['BORGATTI_S', 11, [(1991, 1992, 1), (1994, 1995, 1),
    (1997, 1998, 1), (1999, 2000, 2), (2003, 2004, 1),
    (2005, 2007, 2), (2007, 2008, 1)]],
 ['NEWMAN_M', 9, [(2001, 2002, 2), (2002, 2003, 1),
    (2004, 2005, 2), (2005, 2006, 1), (2006, 2007, 2),
    (2007, 2008, 1)]],
 ['BONACICH_P', 7, [(1986, 1988, 1), (1991, 1992, 1),
    (1998, 1999, 1), (2001, 2002, 1), (2004, 2005, 2)]],
 ['EVERETT_M', 6, [(1997, 1998, 1), (1999, 2000, 2),
    (2004, 2007, 1)]],
 ['CARLEY_K', 5, [(1999, 2000, 1), (2003, 2004, 1),
    (2006, 2007, 3)]]]
```

Temporal citation networks

A citation matrix **Ci** describes the citation relation p cites q. Note that p cites $q \Rightarrow d(p) \geq d(q)$.
Then we can construct its instantaneous version **Cii**:

$$cii_{pq} = [(d(p), d(p) + 1, 1)] \quad \text{iff} \quad ci_{pq} = 1$$

and its cumulative version **Cic**:

$$cic_{pq} = [(d(p), last + 1, 1)] \quad \text{iff} \quad ci_{pq} = 1$$

Temporal versions of:
Bibliographic coupling $\mathbf{biCo} = \mathbf{Ci} \cdot \mathbf{Ci}^T$.
Co-citation $\mathbf{coCi} = \mathbf{Ci}^T \cdot \mathbf{Ci}$.
Citations between authors $\mathbf{Ca} = \mathbf{WA}^T \cdot \mathbf{Ci} \cdot \mathbf{WA}$.

$$\mathbf{ACA} = \mathbf{WAi}^T \cdot \mathbf{Cii} \cdot \mathbf{WAc}$$

Citations between authors


```
>>> cite = TQ.Ianus2Mat("CiteInst.ten")
>>> Cite = cite['mat']; WAc = wac['mat']
>>> ACA = TQ.MatProd(TQ.MatProd(AWi,Cite),WAc)
>>> ACA[ia['WASSERMA_S']][ia['HOLLAND_P']]
[(1977, 1978, 1), (1980, 1981, 5), (1981, 1982, 2),
 (1984, 1985, 2), (1985, 1986, 1), (1987, 1989, 2),
 (1990, 1991, 1), (1991, 1992, 2), (1992, 1994, 3),
 (1995, 1996, 2), (1996, 1997, 3), (1999, 2000, 5),
 (2000, 2001, 1), (2006, 2008, 1)]
>>> D = [(i,TQ.total(ACA[ia['DOREIAN_P']][i])) for i in range(92)
>>> J = sorted(D,key=lambda e:e[1],reverse=True)
>>> [[auNames[i],v,ACA[ia['DOREIAN_P']][i]] for (i,v) in J[:5]]
[['DOREIAN_P', 69, [(1980, 1983, 1), (1984, 1985, 2),
    (1985, 1986, 1), (1986, 1987, 3), (1987, 1988, 2),
    (1988, 1989, 7), (1989, 1990, 5), (1990, 1991, 2),
    (1992, 1993, 6), (1994, 1995, 8), (1995, 1996, 2),
    (1996, 1997, 4), (2000, 2001, 3), (2001, 2004, 4),
    (2004, 2005, 6), (2006, 2007, 3)]],
 ['BREIGER_R', 26, [(1980, 1981, 3), (1984, 1986, 1),
    (1986, 1987, 2), (1987, 1988, 1), (1988, 1989, 4),
    (1989, 1990, 1), (1992, 1993, 3), (1994, 1995, 2),
    (1995, 1996, 1), (1996, 1997, 2), (2000, 2001, 1),
    (2004, 2005, 2), (2007, 2008, 2)]],
```

Citations between authors


```
  ['BURT_R', 20, [(1985, 1986, 3), (1986, 1987, 1),
     (1987, 1988, 2), (1988, 1989, 5), (1989, 1990, 2),
     (1992, 1993, 4), (1994, 1995, 1), (2000, 2001, 1),
     (2004, 2005, 1)]],
  ['BATAGELJ_V', 17, [(1992, 1993, 2), (1994, 1995, 2),
     (1996, 1997, 4), (2000, 2001, 4), (2004, 2005, 5)]],
  ['FARARO_T', 15, [(1984, 1985, 1), (1985, 1986, 2),
     (1988, 1989, 2), (1989, 1990, 1), (1992, 1993, 1),
     (1995, 1996, 1), (2001, 2002, 3), (2002, 2003, 2),
     (2003, 2004, 1), (2006, 2007, 1)]]]
>>>
```

Temporal co-authorship network for SN5

SN5 (2008)

	W	A	K	J
raw	193376	75930	29267	14651
DC=1	7950	12458		

In `Pajek` we extract a subnetwork **WAc** and a corresponding partition **SN5yearC**. Using a program `twoMode2netJSON` we transform them into temporal network in the netJSON format.

Bibliographic networks are usually sparse. The network **WAcInst** has 19488 arcs. The co-authorship network $\mathbf{CoInst} = \mathbf{WAcInst}^T * \mathbf{WAcInst}$ has 64980 edges; the corresponding matrix in the package **TQ** would have $12458^2 = 155201764$ entries. Using a package **Graph** the co-authorship network is computed in a second and half – a big speed-up.

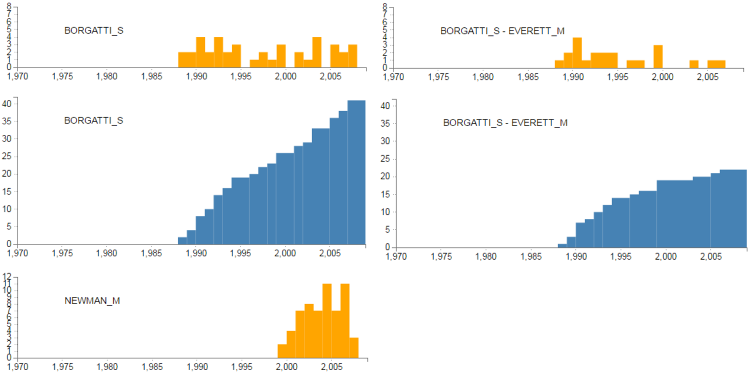
multiply.py


```
gdir = 'c:/users/batagelj/work/python/graph/graph'
wdir = 'c:/users/batagelj/work/python/graph/JSON/SN5'
cdir = 'c:/users/batagelj/work/python/graph/chart'
import sys, os, datetime, json
sys.path = [gdir]+sys.path; os.chdir(wdir)
import TQ
from GraphNew import Graph
# file = 'C:/Users/batagelj/work/Python/graph/JSON/WAtest.json'
file = 'C:/Users/batagelj/work/Python/graph/JSON/SN5/WAcInst.json'
# file = 'C:/Users/batagelj/work/Python/graph/JSON/SN5/WAcCum.json'
# file = 'C:/Users/batagelj/work/Python/graph/JSON/Gisela/papInst.json'
t1 = datetime.datetime.now()
print("started: ",t1.ctime(),"\n")
G = Graph.loadNetJSON(file)
t2 = datetime.datetime.now()
print("\nloaded: ",t2.ctime(),"\ntime used: ", t2-t1)
# T = G.transpose()
# Co = Graph.TQmultiply(T,G,True)
# CR = G.TQtwo2oneRows()
CC = G.TQtwo2oneCols()
t3 = datetime.datetime.now()
print("\ncomputed: ",t3.ctime(),"\ntime used: ", t3-t2)
ia = { v[3]['lab']: k for k,v in CC._nodes.items() }
# CC._links[(ia['BORGATTI_S'],ia['EVERETT_M'])][4]['tq']
# CC._links[(ia['IDI/B'],ia['HCL/B'])][4]['tq']
```

Temporal co-authorship network for SN5

```
======= RESTART: C:\Users\batagelj\work\Python\graph\graph\multiply.py =======
started:  Sun Nov 20 00:26:51 2016
loaded:  Sun Nov 20 00:26:51 2016
time used:  0:00:00.425024
computed:  Sun Nov 20 00:26:52 2016
time used:  0:00:01.165066
>>> BB = CC._links[(ia['BORGATTI_S'],ia['BORGATTI_S'])][4]['tq']
>>> BE = CC._links[(ia['BORGATTI_S'],ia['EVERETT_M'])][4]['tq']
>>> BB
[(1988, 1990, 2), (1990, 1991, 4), (1991, 1992, 2), (1992, 1993, 4),
 (1993, 1994, 2), (1994, 1995, 3), (1996, 1997, 1), (1997, 1998, 2),
 (1998, 1999, 1), (1999, 2000, 3), (2001, 2002, 2), (2002, 2003, 1),
 (2003, 2004, 4), (2005, 2006, 3), (2006, 2007, 2), (2007, 2008, 3)]
>>> BE
[(1988, 1989, 1), (1989, 1990, 2), (1990, 1991, 4), (1991, 1992, 1),
 (1992, 1995, 2), (1996, 1998, 1), (1999, 2000, 3), (2003, 2004, 1),
 (2005, 2007, 1)]
>>> TQmax = 8; Tmin = 1970; Tmax = 2009; w = 600; h = 120
>>> tit = 'BORGATTI_S'
>>> Graph.TQshow(BB,cdir,TQmax,Tmin,Tmax,w,h,tit,fill='orange')
>>> tit = 'BORGATTI_S - EVERETT_M'
>>> Graph.TQshow(BE,cdir,TQmax,Tmin,Tmax,w,h,tit,fill='orange')
>>> NN = CC._links[(ia['NEWMAN_M'],ia['NEWMAN_M'])][4]['tq']
>>> NN
[(1999, 2000, 2), (2000, 2001, 4), (2001, 2002, 7), (2002, 2003, 8),
 (2003, 2004, 7), (2004, 2005, 11), (2005, 2006, 7), (2006, 2007, 11),
 (2007, 2008, 3)]
>>> tit = 'NEWMAN_M'; TQmax = 12; h = 150
>>> Graph.TQshow(NN,cdir,TQmax,Tmin,Tmax,w,h,tit,fill='orange')
```

Visualization



Clustering coefficients

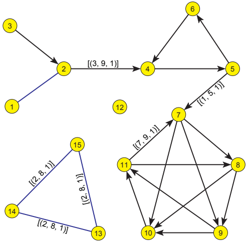
Let us assume that the network $\mathcal{N}$ is based on a simple directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ without loops. From a simple undirected graph we obtain the corresponding simple directed graph by replacing each edge with a pair of opposite arcs. In such a graph the *clustering 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

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

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

Let $\mathbf{A}$ be a temporal network matrix. The number of triangles in a node can be obtained as a diagonal entry of the matrix $\mathbf{SAS}$ where $\mathbf{T} = \mathbf{A}^T$ and $\mathbf{S} = \mathbf{A} + \mathbf{T}$.

Clustering coefficients




```
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)]
```

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.
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 the following procedure.

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)
```

Temporal weak and strong connectivity

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

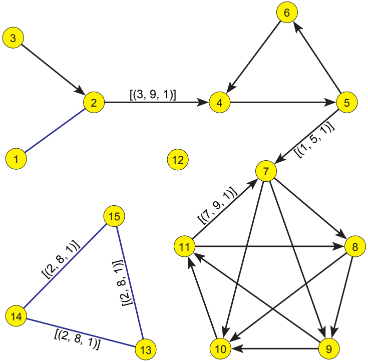
$$\mathbf{W} = (\mathbf{A} \cup \mathbf{A}^T)^\star$$

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.

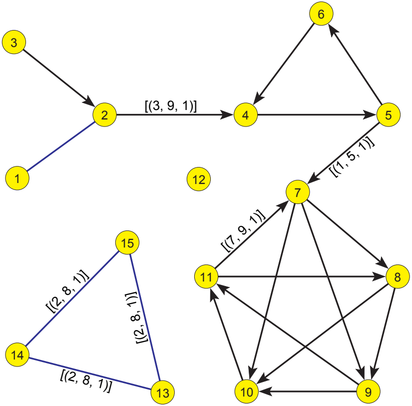
Temporal weak components in example network




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

All unlabeled links have a value $[(1, 9, 1)]$

Temporal strong components in example network




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

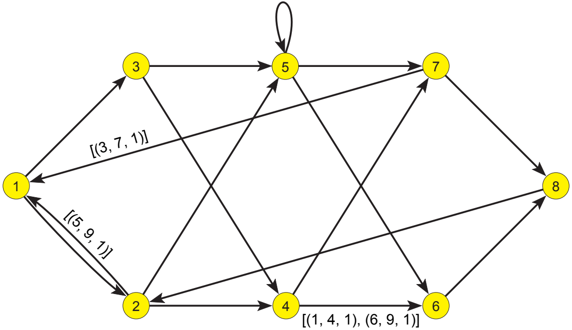
All unlabeled links have a value $[(1, 9, 1)]$

Other results

Using the closure over geodetic semiring we can determine the temporal closeness and betweenness vectors. For computing the "ingredients" for computing the betweenness centralities the geodetic semiring is used.

Using the Pathfinder semiring we can determine temporal Pathfinder skeletons of a given temporal network.

Betweenness centralities




```
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)]
```

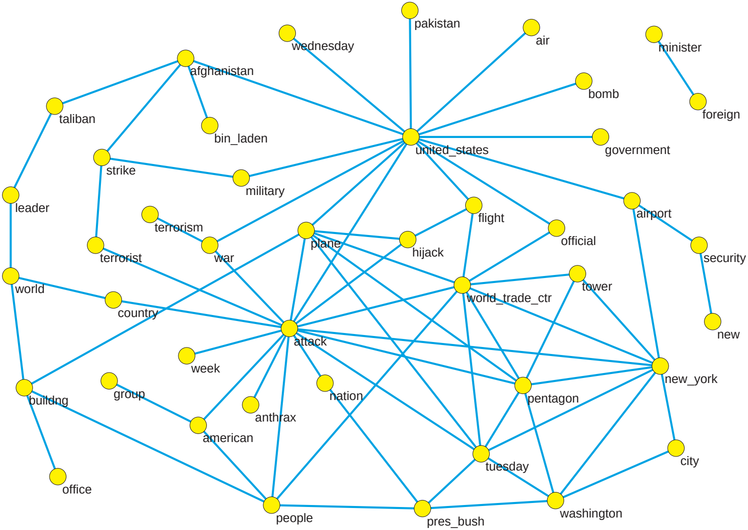
September 11th Reuters terror news

Reuters terror news network was obtained from the CRA networks produced by Steve Corman and Kevin Dooley at Arizona State University. 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.
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 in next slide.
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. The isolated nodes were removed.

50 most frequent terms in the Terror news network

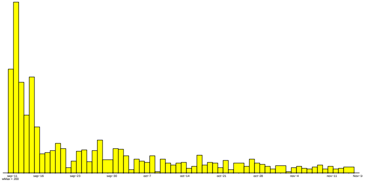
n	term	Σfreq	n	term	Σfreq
1	united_states	15000	26	terrorism	2212
2	attack	10348	27	day	2128
3	taliban	6266	28	week	2017
4	people	5286	29	worker	1983
5	afghanistan	5176	30	office	1967
6	bin_laden	4885	31	group	1966
7	new_york	4832	32	air	1962
8	pres_bush	4506	33	minister	1919
9	washington	4047	34	time	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

Most frequent terms in the Terror news network

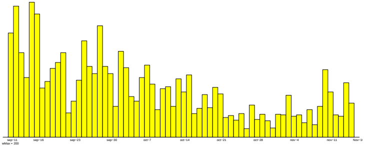


Types of activity

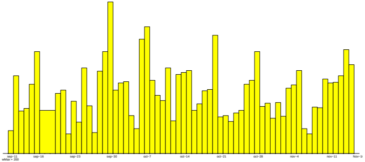
hijack :



bin Laden :

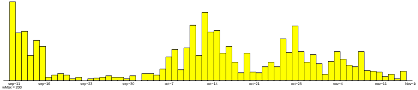


taliban :

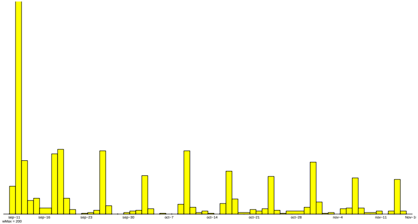


Types of activity

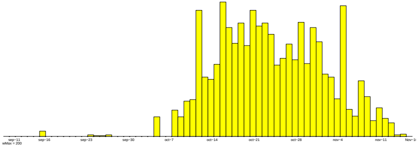
bomb :



Wednesday :



anthrax :



Attraction

To consider also the node's position in the network in a measure of importance of the node $u \in \mathcal{V}$ we constructed the attraction coefficient $\mathrm{att}(u)$.

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* $\mathrm{act}(u)$ as

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

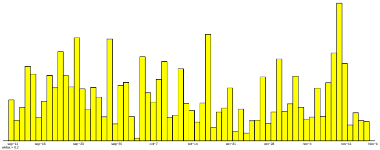
Then the *attraction* of the node u is defined as

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

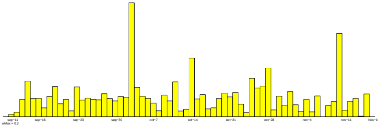
The fraction $\frac{w_{vu}}{\mathrm{act}(v)}$ is measuring the proportion of the activity of the node v that is shared with the node u. We have $0 \leq \mathrm{att}(u) \leq 1$, for all $u \in \mathcal{V}$.
The maximum possible attraction value 1 is attained exactly for nodes:
a) in undirected network: 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.

Attraction patterns

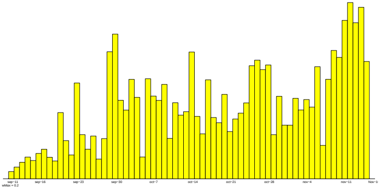
pres Bush :



Pakistan :

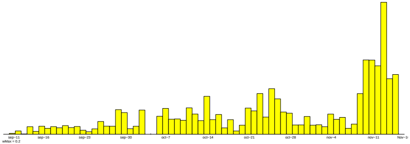


taliban :

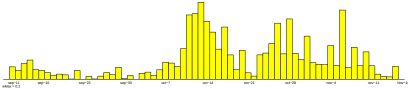


Attraction patterns

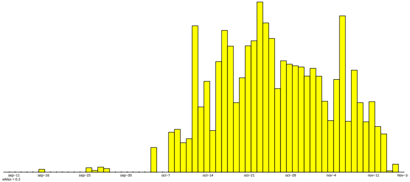
Kabul :



bomb :



anthrax :



Temporal generalized cores

In September 2016 we developed also an algorithm for computing temporal cores [4]. As a real data example we used the data set SN5 (2008). The temporal authorship network WA (restricted to works with DC=1) has two versions: the instant `WAcInst.json` and the cumulative `WAcCum.json`. Both networks have the same sizes $|W| = 7950$, $|A| = 12458$ and $|L| = 19488$. We computed the normalized cumulative temporal version **Ct** of co-authorship network [2]. It turned out that the p_S cores procedure is quite fast. It produced the cores in 20s.

```
>>>
====== RESTART: C:\Users\batagelj\work\Python\graph\graph\PsCoresTQ.py ======
Temporal Ps cores in:  CcCSN5.json
started:  Tue Oct 25 17:23:26 2016
finished:  Tue Oct 25 17:23:45 2016
time used:  0:00:19.644124
>>> C = TQ.TQ.TQdictCut(Core,3)
>>> for v in C:
   print("{0:3d} : {1:11s} ".format(v,G.getNode(v,'lab')),C[v])
```

p_S cores at level 3

in normalized co-authorship in SN5

```
  20 : BORGATTI_S   [(1991, 1992, 3.1667), (1992, 1993, 4.1667), (1993, 1994, 5.1667),
     (1994, 1996, 6.1667), (1996, 1997, 6.6667), (1997, 1999, 7.1667), (1999, 2003, 8.6667),
     (2003, 2005, 8.7917), (2005, 2006, 9.2917), (2006, 2009, 9.7917)]
3169 : EVERETT_M    [(1991, 1992, 3.1667), (1992, 1993, 4.1667), (1993, 1994, 5.1667),
     (1994, 1996, 6.1667), (1996, 1997, 6.6667), (1999, 2003, 8.6667), (2003, 2005, 8.7917),
     (2005, 2006, 9.2917), (2006, 2009, 9.7917)]
 317 : BERNARD_H    [(1990, 1991, 3.0244), (1991, 1995, 3.1494), (1995, 1997, 3.3094),
     (1997, 1998, 3.3894), (1998, 2001, 3.5494), (2001, 2003, 3.6294), (2003, 2006, 3.685),
     (2006, 2009, 4.0706)]
2232 : KILLWORT_P   [(1990, 1991, 3.0244), (1991, 1995, 3.1494), (1995, 1997, 3.3094),
     (2003, 2006, 3.685), (2006, 2009, 4.0706)]
4551 : STEINHAU_H   [(2003, 2005, 3.0), (2005, 2006, 3.2222), (2006, 2009, 3.6667)]
4860 : METZKE_C     [(2003, 2005, 3.0), (2005, 2006, 3.2222), (2006, 2009, 3.6667)]
3125 : SHELLEY_G    [(2006, 2009, 3.4767)]
1673 : MCCARTY_C    [(2006, 2009, 3.4767)]
1677 : JOHNSEN_E    [(2006, 2009, 3.4767)]
  75 : HOLLAND_P    [(1981, 1983, 3.0), (1983, 2009, 3.2222)]
  78 : LEINHARD_S   [(1981, 1983, 3.0), (1983, 2009, 3.2222)]
 925 : BONACICH_P   [(1997, 2009, 3.2222)]
3840 : BIENENST_E   [(1997, 2009, 3.2222)]
  69 : WASSERMA_S   [(2007, 2009, 3.0174)]
1164 : DOREIAN_P    [(2007, 2009, 3.0174)]
1166 : HUMMON_N     [(2007, 2009, 3.0174)]
1680 : PATTISON_P   [(2007, 2009, 3.0174)]
3225 : FARARO_T     [(2007, 2009, 3.0174)]
1056 : FAUST_K      [(2007, 2009, 3.0174)]
3170 : FERLIGOJ_A   [(2007, 2009, 3.0174)]
2083 : ROBINS_G     [(2007, 2009, 3.0174)]
2084 : SKVORETZ_J   [(2007, 2009, 3.0174)]
 949 : BATAGELJ_V   [(2007, 2009, 3.0174)]
  79 : NEWMAN_M     [(2005, 2009, 3.0)]
 796 : PARK_J       [(2005, 2009, 3.0)]
>>>
```

Conclusions

- New approach, an alternative to the traditional approach with slices. Many traditional concepts still need to be adapted and new ones developed.
- Which methods can be used also on large networks?
- It seems that extension of the approach to networks with nonzero latency is very difficult, if not impossible.
- Temporal networks methods produce large results. Special methods for identifying and presenting (visualizing) interesting parts need to be developed;
- We presented only some examples to show that it works. Many options have still to be elaborated;
- Temporal networks approach can give additional insight into bibliographic networks;

References I

- Batagelj, V.: Semirings for social networks analysis. Journal of Mathematical Sociology, 19(1994)1, 53-68.
- Batagelj, V, Cerinšek, M: On bibliographic networks. Scientometrics 96 (2013) 3, 845-864.
- Batagelj, V., Praprotnik, S.: An algebraic approach to temporal network analysis based on temporal quantities. Social Network Analysis and Mining, 6(2016)1, 1-22.
- Batagelj, V., Zaveršnik, M.: Fast algorithms for determining (generalized) core groups in social networks. Advances in Data Analysis and Classification, 2011. Volume 5, Number 2, 129-145.
- Casteigts, A., Flocchini, P.: Deterministic Algorithms in Dynamic Networks: Formal Models and Metrics. Commissioned by Defense Research and Development Canada (DRDC), 82p, 2013.

References II

- Fletcher, J.G.: A more general algorithm for computing closed semiring costs between vertices of a directed graph. CACM 23 (1980), 350-351.
- Holme, P., Saramäki, J. (Eds.): Temporal Networks. Understanding Complex Systems. Springer, 2013.
- Holme, P.: Modern temporal network theory: a colloquium. European Physical Journal B 88, 234 (2015).
- Praprotnik, S., Batagelj, V.: Spectral centrality measures in temporal networks. Ars Mathematica Contemporanea 11 (2016) 11–33.
- Praprotnik, S., Batagelj, V.: Semirings for temporal network analysis. March 2016. arXiv: 1603.08261