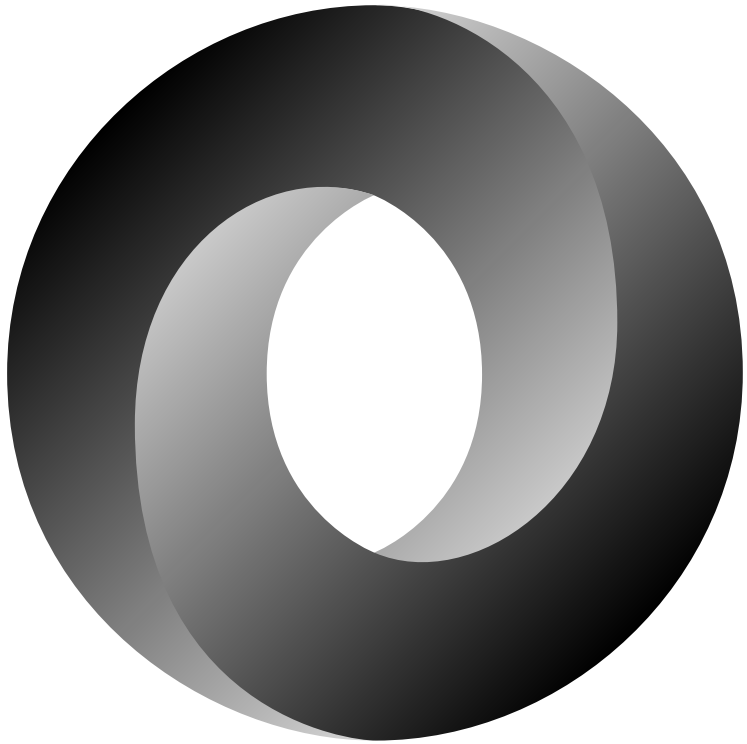
Network visualization based on JSON and D3.js

Vladimir Batagelj

IMFM Ljubljana and IAM UP Koper

Second European Conference on Social Networks
June 14-17, 2016, Paris

Outline



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

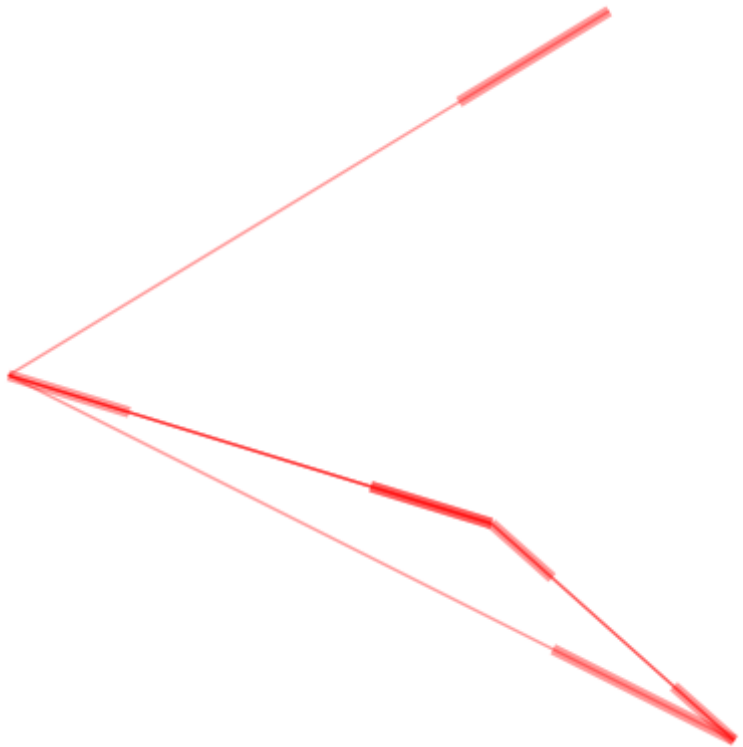
Current version of slides (14. junij 2016 ob 14 : 40):
EUSN'16 slides PDF

Motivation

Work in progress!!!

A year and half ago I wanted to visualize some networks in Python. I was not satisfied with existing options. The library matplotlib (also used by NetworkX) is developed for visualization of statistical data and results.

On the other side there exists an excellent library D3.js for interactive visualization on the web (and locally) in SVG format. Most of the network data for D3.js are prepared in the JSON format. There exist many nice D3.js based network visualizations.



Networks in D3.js

- Force: Force-Directed Graph, Force Layout & Matrix Market Format, 3D Force Layout; An A to Z of extra features for the d3 force layout
- Directed: Directed Graph Editor, Directed Edges (Curves and Arrow Markers), Mobile Patent Suits
- Matrix: Co-occurrence Matrix
- Hive: Hive Plots
- Chord: Chord Diagram, Hierarchical Edge Bundling
- Applications: Linked JAZZ, Ontology Visualization, Visualizing Package Dependencies, Connectome explorer for the "brain" of C. elegans, Gene functional interaction networks
- More: D3 gallery, The Big List of D3.js Examples - Christophe Viau, Over 2000 D3.js Examples and Demos

Project

- **netJSON**: develop a JSON based format for description of networks. It should be "complete" – it can be used also to describe multi-relational, temporal, two-mode networks, and collections of networks. netJSON network description can be extended with a layout information. (jsongraph)
- **netD3.js**: collect and adapt for netJSON selected existing network visualization solutions based on D3.js, and develop new ones.

netJSON could serve as a data exchange format among network analysis programs (conversion program from/to netJSON).

Programers may export their results in netJSON and use net3D.js for their visualization.

netJSON and netD3.js for SNA analysts

network $\rightarrow$ netJSON $\rightarrow$ SVG $\rightarrow$ {PDF, PNG, EPS}

- Prepare your network data in netJSON format (in a text editor, from Excel tables using R, export from SNA programs and convert to netJSON). Add the layout information.
- Use selected netD3.js templates to visualize the network.
- Optionally, save the SVG picture, enhance it in some vector graphics editor (AI, Inkscape) and export it in selected format (PDF, ESP, PNG, ...).

Networks

A *network* is based on two sets – set of *nodes* (vertices), that represent the selected *units*, and set of *links* (lines), that represent *ties* between units. They determine a *graph*. A line can be *directed* – an *arc*, or *undirected* – an *edge*.
Additional data about nodes or links may be known – their *properties* (attributes). For example: name/label, type, value, ...

Network = Graph + Data

A *network* $\mathcal{N} = (\mathcal{V}, \mathcal{L}, \mathcal{P}, \mathcal{W})$ consists of:

- a *graph* $\mathcal{G} = (\mathcal{V}, \mathcal{L})$, where $\mathcal{V}$ is the set of nodes, $\mathcal{A}$ is the set of arcs, $\mathcal{E}$ is the set of edges, and $\mathcal{L} = \mathcal{E} \cup \mathcal{A}$ is the set of links.
  $n = |\mathcal{V}|$, $m = |\mathcal{L}|$
- $\mathcal{P}$ *vertex value functions* / properties: $p : \mathcal{V} \rightarrow A$
- $\mathcal{W}$ *line value functions* / weights: $w : \mathcal{L} \rightarrow B$

Description of networks

How to describe a network $\mathcal{N}$? In principle the answer is simple – we list its components $\mathcal{V}$,$\mathcal{L}$,$\mathcal{P}$, and $\mathcal{W}$.

The simplest way is to describe a network $\mathcal{N}$ by providing $(\mathcal{V}, \mathcal{P})$ and $(\mathcal{L}, \mathcal{W})$ in a form of two tables.

As an example, let us describe a part of network determined by the following works:
Generalized blockmodeling, Clustering with relational constraint, Partitioning signed social networks, The Strength of Weak Ties

There are nodes of different types (modes): persons, papers, books, series, journals, publishers; and different relations among them: author_of, editor_of, contained_in, cites, published_by.

Both tables are often maintained in Excel. They can be exported as text in CSV (Comma Separated Values) format.

bibNodes.csv

```
name;mode;country;sex;year;vol;num;fPage;lPage;x;y
"Batagelj, Vladimir";person;SI;m;;;;;;809.1;653.7
"Doreian, Patrick";person;US;m;;;;;;358.5;679.1
"Ferligoj, Anuška";person;SI;f;;;;;;619.5;680.7
"Granovetter, Mark";person;US;m;;;;;;145.6;660.5
"Moustaki, Irini";person;UK;f;;;;;;783.0;228.0
"Mrvar, Andrej";person;SI;m;;;;;;478.0;630.1
"Clustering with relational constraint";paper;;;1982;47;;413;426;684.1;3
"The Strength of Weak Ties";paper;;;1973;78;6;1360;1380;111.3;329.4
"Partitioning signed social networks";paper;;;2009;31;1;1;11;408.0;337.8
"Generalized Blockmodeling";book;;;2005;24;;1;385;533.0;445.9
"Psychometrika";journal;;;;;;;;741.8;086.1
"Social Networks";journal;;;;;;;;321.4;236.5
"The American Journal of Sociology";journal;;;;;;;;111.3;168.9
"Structural Analysis in the Social Sciences";series;;;;;;;;310.4;082.8
"Cambridge University Press";publisher;UK;;;;;;;534.3;238.2
"Springer";publisher;US;;;;;;;884.6;174.0
```

bibNodes.csv

bibLinks.csv

```
from;relation;to
"Batagelj, Vladimir";authorOf;"Generalized Blockmodeling"
"Doreian, Patrick";authorOf;"Generalized Blockmodeling"
"Ferligoj, Anuška";authorOf;"Generalized Blockmodeling"
"Batagelj, Vladimir";authorOf;"Clustering with relational constraint"
"Ferligoj, Anuška";authorOf;"Clustering with relational constraint"
"Granovetter, Mark";authorOf;"The Strength of Weak Ties"
"Granovetter, Mark";editorOf;"Structural Analysis in the Social Sciences
"Doreian, Patrick";authorOf;"Partitioning signed social networks"
"Mrvar, Andrej";authorOf;"Partitioning signed social networks"
"Moustaki, Irini";editorOf;"Psychometrika"
"Doreian, Patrick";editorOf;"Social Networks"
"Generalized Blockmodeling";containedIn;"Structural Analysis in the Soci
"Clustering with relational constraint";containedIn;"Psychometrika"
"The Strength of Weak Ties";containedIn;"The American Journal of Sociolo
"Partitioning signed social networks";containedIn;"Social Networks"
"Partitioning signed social networks";cites;"Generalized Blockmodeling"
"Generalized Blockmodeling";cites;"Clustering with relational constraint
"Structural Analysis in the Social Sciences";publishedBy;"Cambridge Univ
"Psychometrika";publishedBy;"Springer"
```

bibLinks.csv

CSV2Pajek.R

```
# transforming CSV file to Pajek files
# by Vladimir Batagelj, June 2016
setwd("C:/Users/batagelj/work/Python/graph/SVG/EUSN")
colC <- c(rep("character",4),rep("integer",7)); nas <- c("","NA","NaN")
nodes <- read.csv2("bibNodes.csv",encoding='UTF-8',colClasses=colC,na.strings=nas)
n <- nrow(nodes); M <- factor(nodes$mode); S <- factor(nodes$sex)
mod <- levels(M); sx <- levels(S); S <- as.numeric(S); S[is.na(S)] <- 0
links <- read.csv2("bibLinks.csv",encoding='UTF-8',colClasses="character")
F <- factor(links$from,levels=nodes$name,ordered=TRUE)
T <- factor(links$to,levels=nodes$name,ordered=TRUE)
R <- factor(links$relation); rel <- levels(R)
net <- file("bib.net","w"); cat('*vertices ',n,'\n',file=net)
clu <- file("bibMode.clu","w"); sex <- file("bibSex.clu","w")
cat('%',file=clu); cat('%',file=sex)
for(i in 1:length(mod)) cat(' ',i,mod[i],file=clu)
cat('\n*vertices ',n,'\n',file=clu)
for(i in 1:length(sx)) cat(' ',i,sx[i],file=sex)
cat('\n*vertices ',n,'\n',file=sex)
for(v in 1:n) {
  cat(v,' "',nodes$name[v],'"\n',sep='',file=net);
  cat(M[v],'\n',file=clu); cat(S[v],'\n',file=sex)
}
for(r in 1:length(rel)) cat('*arcs :',r,' "',rel[r],'"\n',sep='',file=net)
cat('*arcs\n',file=net)
for(a in 1:nrow(links))
  cat(R[a],': ',F[a],' ',T[a],' 1 1 "',rel[R[a]],'"\n',sep='',file=net)
close(net); close(clu); close(sex)
```

CSV2Pajek.R

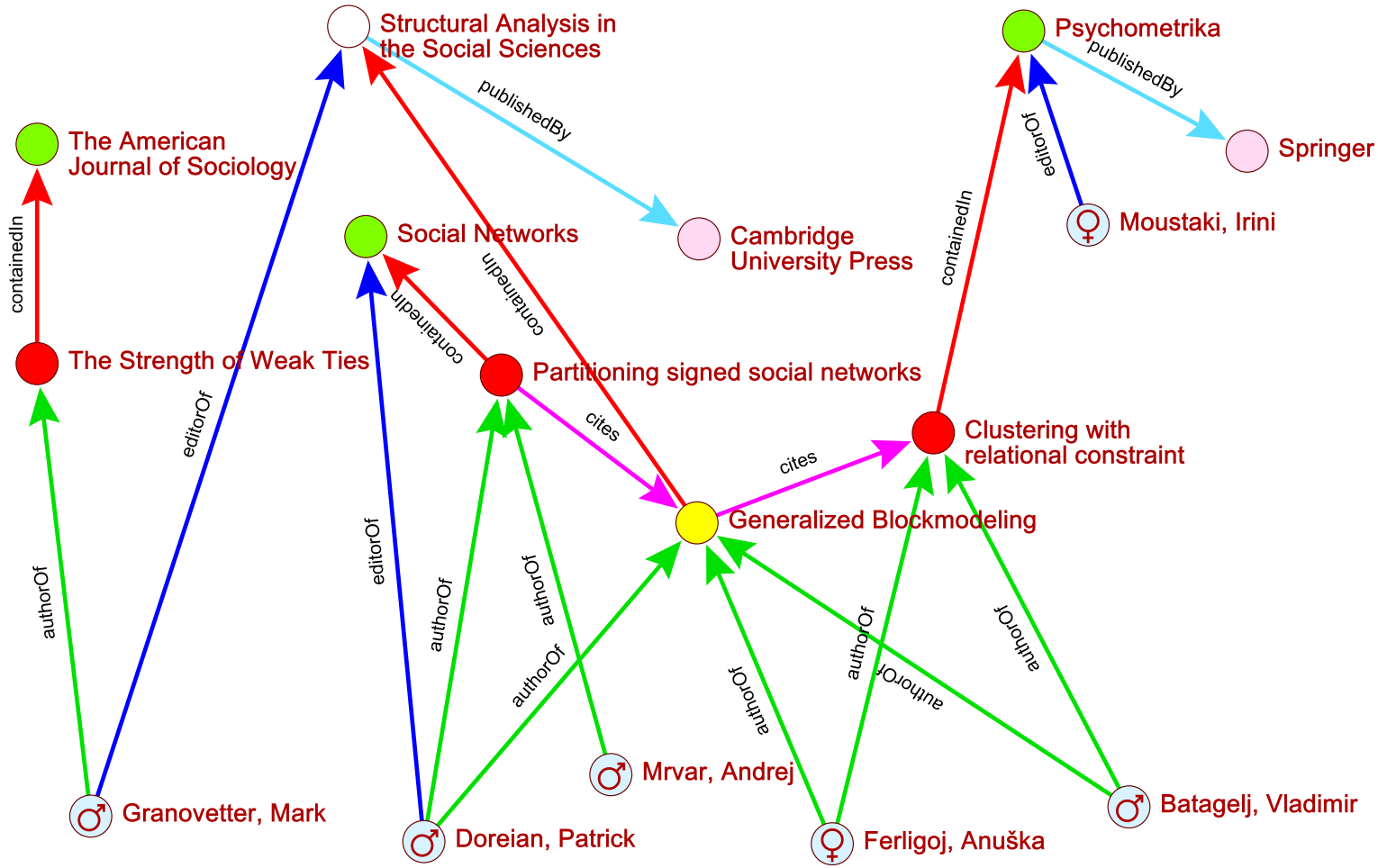
bib.net

```
*vertices  16
1 "Batagelj, Vladimir"
2 "Doreian, Patrick"
3 "Ferligoj, Anuška"
4 "Granovetter, Mark"
5 "Moustaki, Irini"
6 "Mrvar, Andrej"
7 "Clustering with relational constraint"
8 "The Strength of Weak Ties"
9 "Partitioning signed social networks"
10 "Generalized Blockmodeling"
11 "Psychometrika"
12 "Social Networks"
13 "The American Journal of Sociology"
14 "Structural Analysis in the Social Sciences"
15 "Cambridge University Press"
16 "Springer"
*arcs :1 "authorOf"
*arcs :2 "cites"
*arcs :3 "containedIn"
*arcs :4 "editorOf"
*arcs :5 "publishedBy"
```

```
*arcs
1: 1 10 1 l "authorOf"
1: 2 10 1 l "authorOf"
1: 3 10 1 l "authorOf"
1: 1 7 1 l "authorOf"
1: 3 7 1 l "authorOf"
1: 4 8 1 l "authorOf"
4: 4 14 1 l "editorOf"
1: 2 9 1 l "authorOf"
1: 6 9 1 l "authorOf"
4: 5 11 1 l "editorOf"
4: 2 12 1 l "editorOf"
3: 10 14 1 l "containedIn"
3: 7 11 1 l "containedIn"
3: 8 13 1 l "containedIn"
3: 9 12 1 l "containedIn"
2: 9 10 1 l "cites"
2: 10 7 1 l "cites"
5: 14 15 1 l "publishedBy"
5: 11 16 1 l "publishedBy"
```

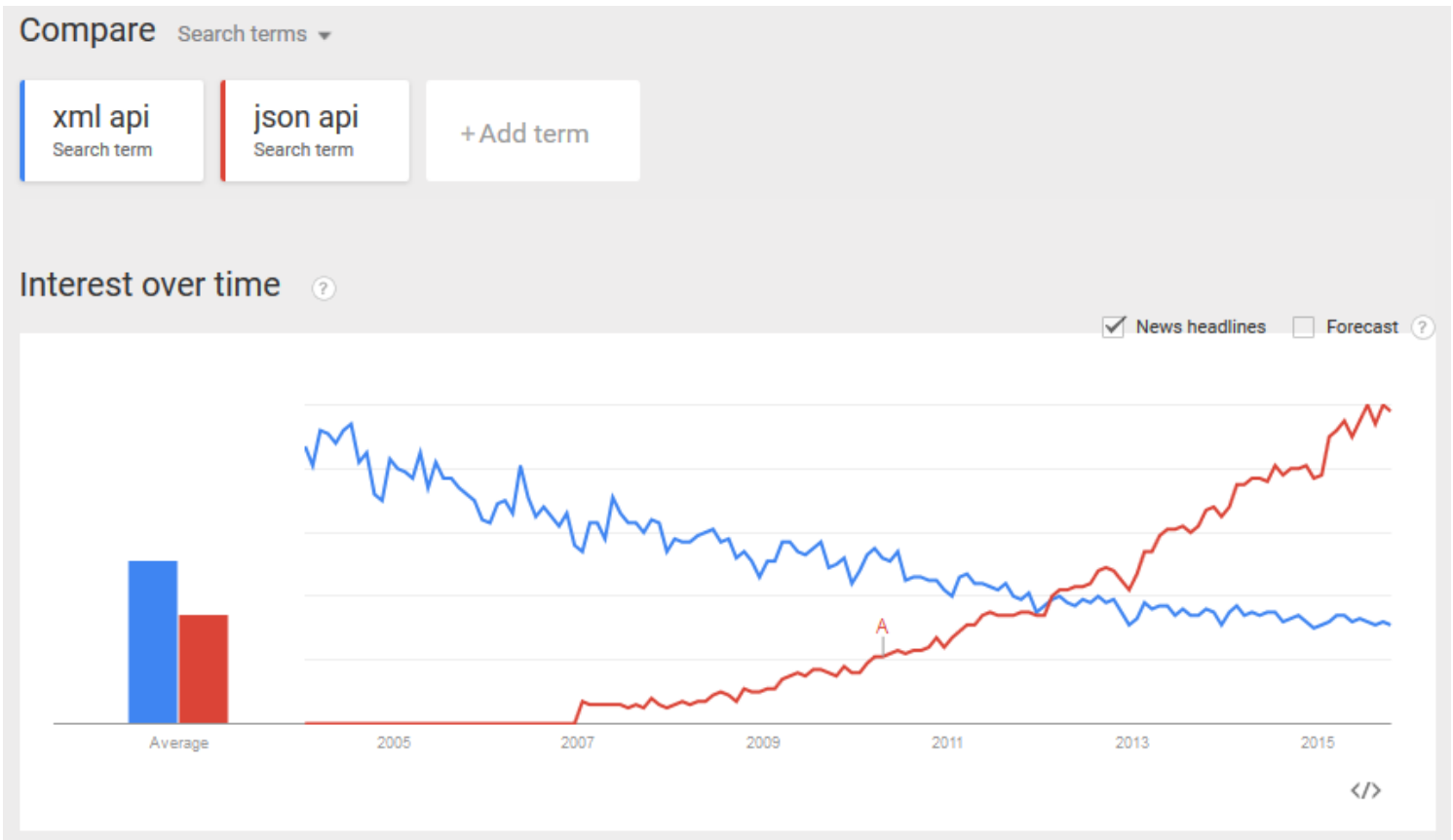
bib.net, bibMode.clu, bibSex.clu,

Bibliographic network – picture / Pajek



XML api – JSON api

In near past, for description of structured data the XML (Extensible Markup Language) was mostly used. In last five years a JSON format started to replace it. Google trends



JSON

JSON (JavaScript Object Notation) is a text data format that preserves the structure of data objects. It is "compatible" with basic data structures in modern programming languages.

The initial version of JSON vas developed by Douglas Crockford (around 2002). He based it on the Javascript notation. The principal idea is: if we apply on a string (sequence of characters) containing a description of a data object, the Javascript function `eval` we get as its result the corresponding data object. JSON is a programming language independent, open code standard for exchange of data among programs.

Two JSON standards exist:

- The JSON Data Interchange Format. Standard ECMA-404, October 2013.
- The JavaScript Object Notation (JSON) Data Interchange Format Request for Comments: 7159, March 2014.

JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Wikipedia

JSON

XML is appropriate for describing the structure of textual data, JSON is becoming the first choice for describing structured data. JSON has much simpler grammar, is more readable and compatible with basic data structures in modern programming languages. All keys (names of fields) are in double quotes.

JSON files are by default based on the encoding Unicode (UTF-8).

The MIME type for JSON files is `application/json`, the recommended file extension is `.json`.

For work with JSON there exists supporting libraries for all important programming languages http://www.json.org/.

JSON grammar

value
- *object*
- *array*
- *string*
- *number*
- `true`
- `false`
- `null`

object
- `{}`
- `{` *members* `}`

members
- *pair*
- *pair* , *members*

pair
- *string* : *value*

array
- `[]`
- `[` *elements* `]`

elements
- *value*
- *value* , *elements*

string
- `""`
- `"` *chars* `"`

chars
- *char*
- *char chars*

char
- *any-Unicode-character-except-"-or-\-or-control-character*
- `\"`
- `\\`
- `\/`
- `\b`
- `\f`
- `\n`
- `\r`
- `\t`
- `\u` *four-hex-digits*

number
- *int*
- *int frac*
- *int exp*
- *int frac exp*

int
- *digit*
- *digit1-9 digits*
- *- digit*
- *- digit1-9 digits*

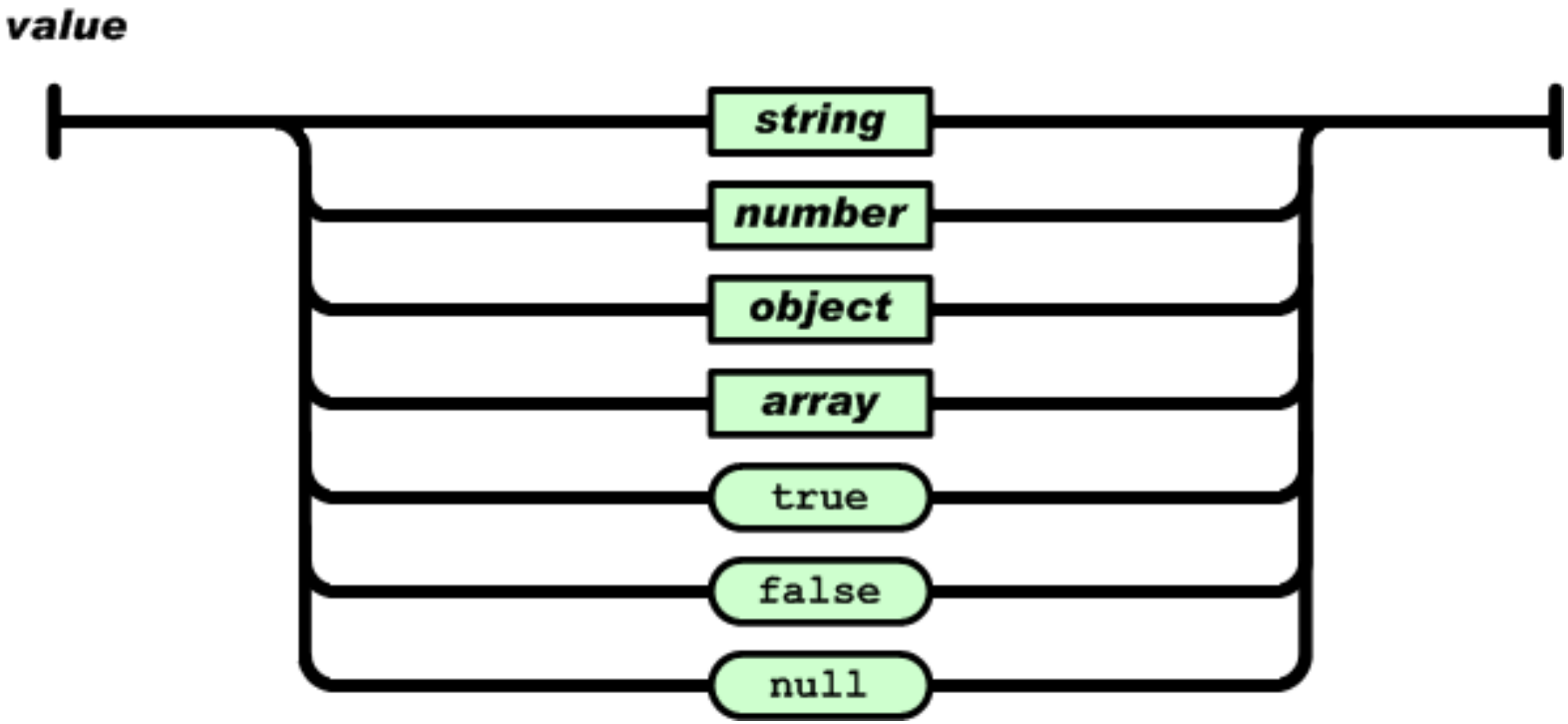
frac
- *. digits*

exp
- *e digits*

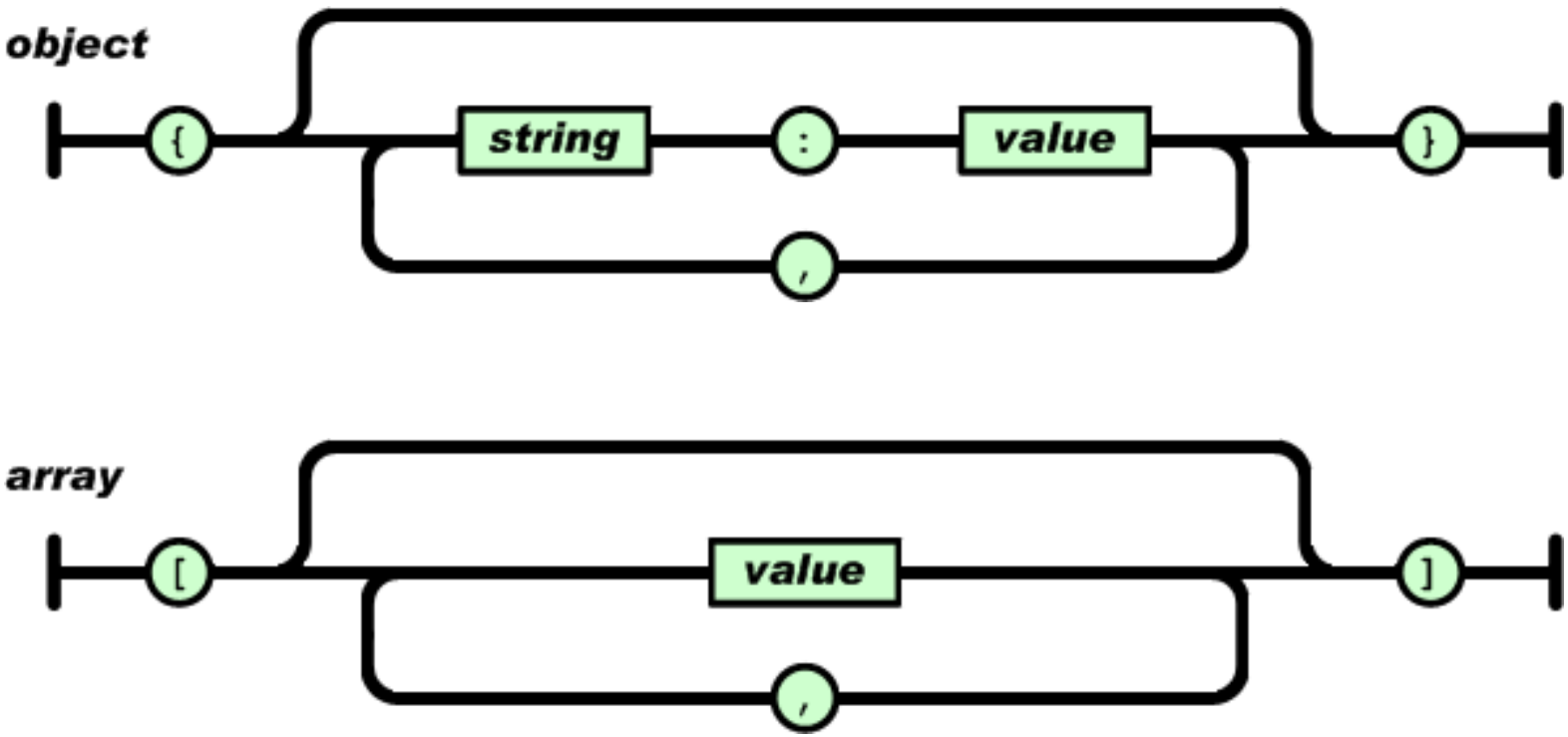
digits
- *digit*
- *digit digits*

e
- `e`
- `e+`
- `e-`
- `E`
- `E+`
- `E-`

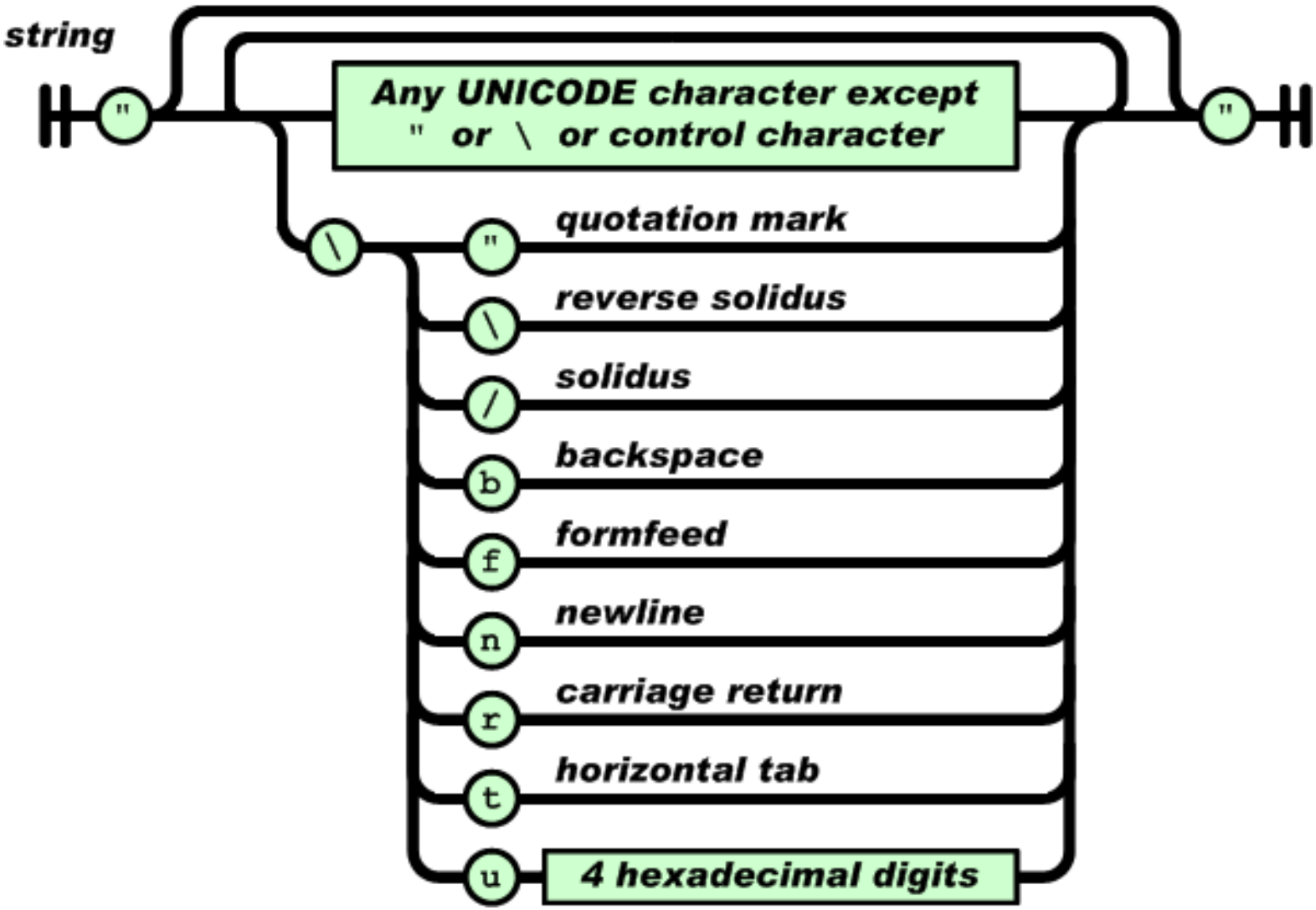
JSON grammar



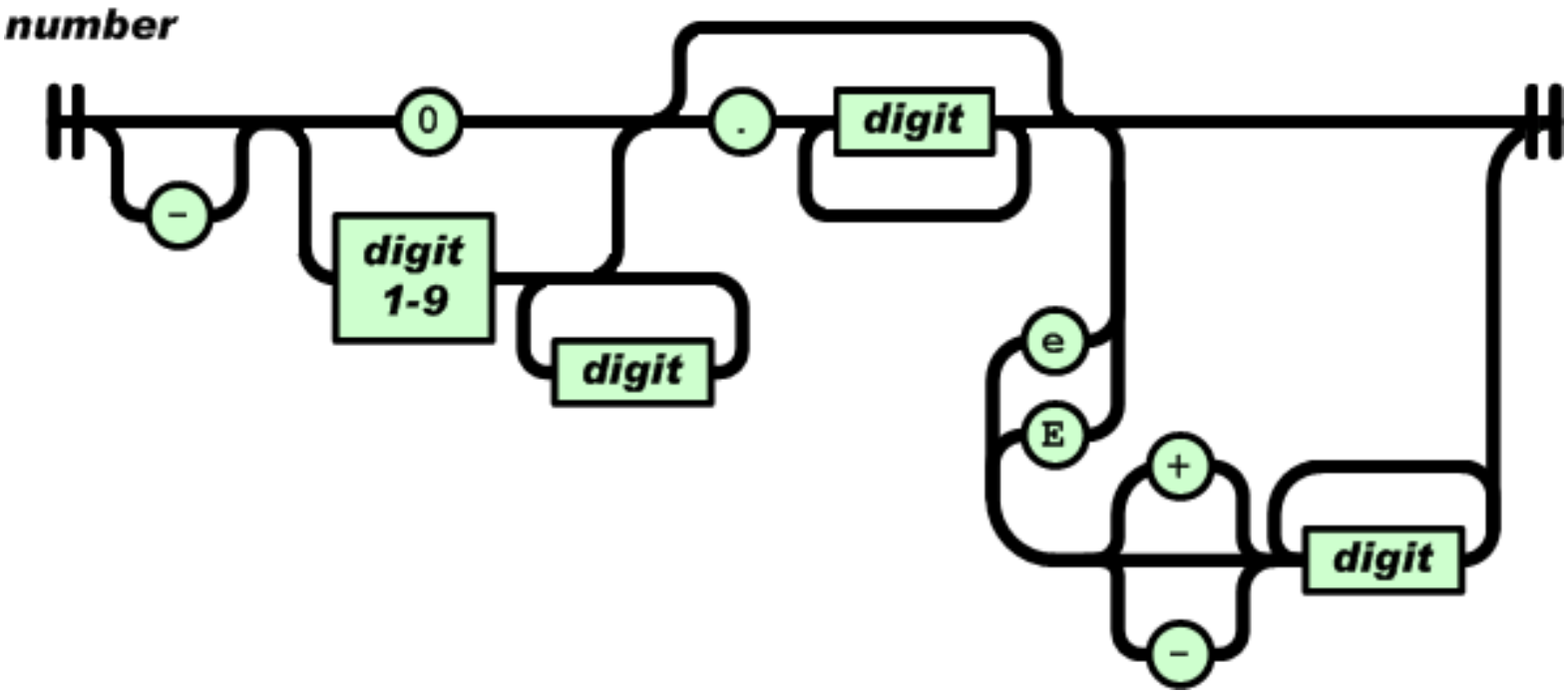
JSON grammar



JSON grammar



JSON grammar



eval and JSON.parse

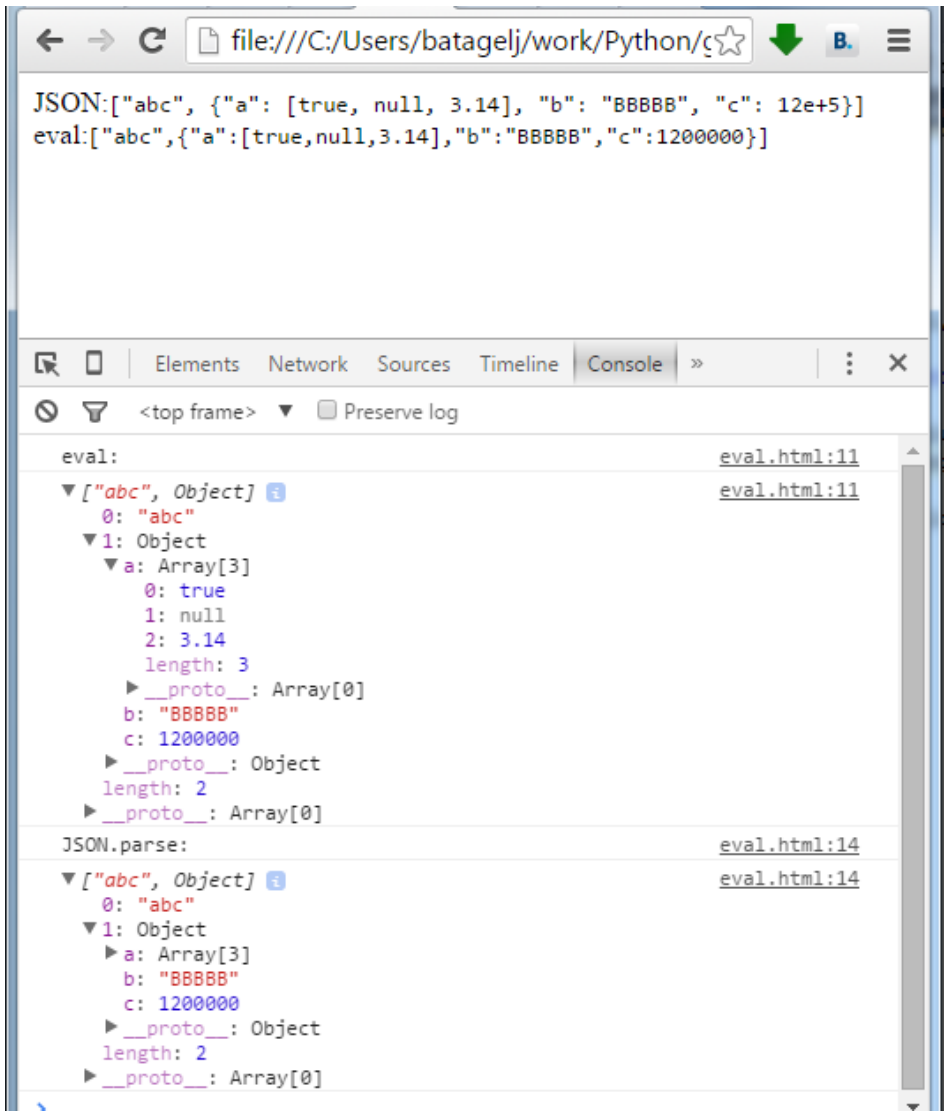
eval.html

```
<html>
<head>
<title>JSON: eval and parse</title>
</head>
<body>
<script>
var data='["abc",{"a":[true, null,3.14],"b":"BBBBB","c":12e+5}]'
alert("JSON: " + data);
document.write("JSON:<tt>"+data+"</tt><br>");
var value = eval('(' + data + ')');
console.log("eval:"); console.log(value);
document.write("eval:<tt>"+JSON.stringify(value)+"</tt><br>");
var json = JSON.parse(data);
console.log("JSON.parse:"); console.log(json);
</script>
</body>
</html>
```

To run the Javascript code I used the Google Chrome browser. To get the console we select

Customize and Control GC / More tools / Developer tools

eval and JSON.parse



Importing data as Javascript assignment

import.html

```
<html>
<head>
<title>JSON import</title>
<script src="./person.js"></script>
</head>
<body>
<script>
document.write("JSON:<tt>"+person+"</tt><br>");
document.write("string:<tt>"+JSON.stringify(person)+"</tt><br>");
console.log("JSON:"); console.log(person);
</script>
</body>
</html>
```

person.js

Well formed and valid JSON files

A JSON file is *well formed* iff it respects JSON's grammar. Is my file well formed? service. JSONlint - another checker.

JSON editor

Similar to XML's DTD files or schema, we can impose additional restrictions to the structure of JSON files describing special types of data using JSON schema – the JSON files respecting these additional restrictions are called *valid*.

Github, validation, JSON Schema Lint, JSON Schema validator.

Simple example

graph.json

```
{
  "info":{"org":0,"nNodes":4},
  "nodes":[
    {"name":"Ann","x":0.2,"y":0.2,"Num":1,"Size":100},
    {"name":"Ben","x":0.2,"y":0.8,"Num":4,"Size":500},
    {"name":"Tim","x":0.8,"y":0.2,"Num":2,"Size":200},
    {"name":"Zoe","x":0.8,"y":0.8,"Num":3,"Size":400}
  ],
  "links":[
    {"source":0,"target":1,"Count":1,"Weight":100},
    {"source":1,"target":2,"Count":1,"Weight":100},
    {"source":2,"target":3,"Count":1,"Weight":100},
    {"source":1,"target":3,"Count":2,"Weight":300}
  ]
}
```

In graph.js the JSON description is assigned to the variable graph:

```
graph = {
  "info":{"org":0,"nNodes":4},
  "nodes":[
...
  ]
}
```

Circular layout

adapted from Brath and Jonker, p. 257-258

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<script src="./graph.js"></script>
<!-- script src="./class.js"></script -->
<script src="http://d3js.org/d3.v3.min.js"></script>
</head>
<body>
<script>
// set up the drawing area
var width = 500,
  height = 500;
var svg = d3.select("body").append("svg")
  .attr("width", width)
  .attr("height", height)
  .attr("xmlns", "http://www.w3.org/2000/svg");
// angle and radius for layout assistance
var ang = 2 * Math.PI / graph.nodes.length;
var rad = width / 2.5;
```

. . . Circular layout

```
// create the links
var link = svg.selectAll("line")
  .data(graph.links).enter().append("line")
  .style("stroke","blue")
  .attr("stroke-width",2)
  .attr("x1",function(d){return(rad*Math.cos(d.source*ang)+.5*width);})
  .attr("y1",function(d){return(rad*Math.sin(d.source*ang)+.5*width);})
  .attr("x2",function(d){return(rad*Math.cos(d.target*ang)+.5*width);})
  .attr("y2",function(d){return(rad*Math.sin(d.target*ang)+.5*width);});
// create the nodes and set out in a circular layout
var node = svg.selectAll("circle")
  .data(graph.nodes).enter().append("circle")
  .attr("r",10)
  .attr("cx",function(d,i){return(rad*Math.cos(i*ang)+.5*width);})
  .attr("cy",function(d,i){return(rad*Math.sin(i*ang)+.5*width);})
  .attr("fill","yellow")
  .attr("stroke","red");
</script>
</body>
```

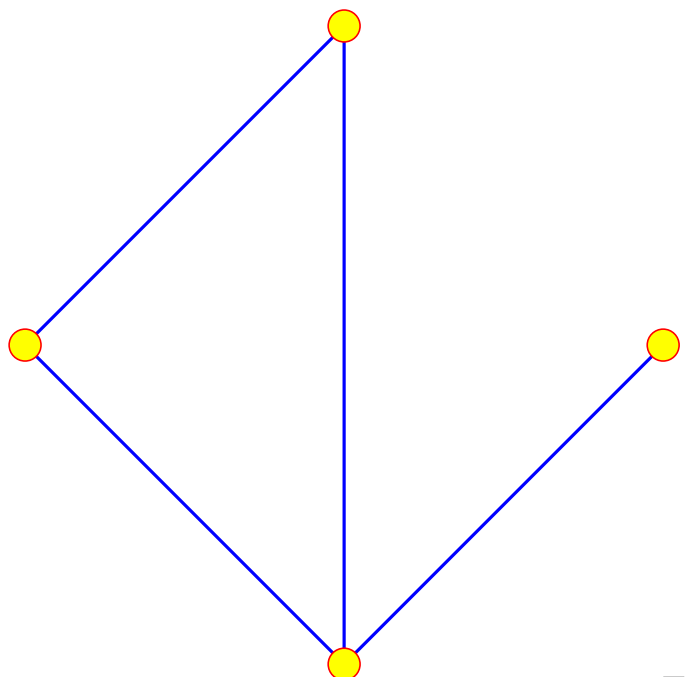
graphCircle.html; class.json, class.js, classCircle.html

+ labels: classCircleL.html

+ permutation: classP.json, classP.js, classCircleP.htm

SVG

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<svg width="500" height="500" xmlns="http://www.w3.org/2000/svg">
<line stroke-width="2" x1="450" y1="250" x2="250" y2="450" style="stroke: blue;"></line>
<line stroke-width="2" x1="250" y1="450" x2="50"  y2="250" style="stroke: blue;"></line>
<line stroke-width="2" x1="50"  y1="250" x2="250" y2="50"  style="stroke: blue;"></line>
<line stroke-width="2" x1="250" y1="450" x2="250" y2="50"  style="stroke: blue;"></line>
<circle r="10" cx="450" cy="250" fill="yellow" stroke="red"></circle>
<circle r="10" cx="250" cy="450" fill="yellow" stroke="red"></circle>
<circle r="10" cx="50"  cy="250" fill="yellow" stroke="red"></circle>
<circle r="10" cx="250" cy="50"  fill="yellow" stroke="red"></circle>
</svg>
```



Networks in JSON format / basic netJSON

class.json

```
{ "netJSON": "basic",
  "info":{ "network": "class", "org": 1, "nNodes": 15,
     "nArcs" : 30, "nEdges": 13, "nWeak" : 1,
     "title" : "borrowing study materials",
     "meta"  : { "date": "October 2015", "author": "V. Batagelj" }
  },
  "nodes": [
     { "id":  1, "short": "m02", "x": 0.1857, "y": 0.2781, "size": 1 },
     { "id":  2, "short": "m03", "x": 0.5482, "y": 0.6169, "size": 1 },
     { "id":  3, "short": "w07", "x": 0.2219, "y": 0.4526, "size": 2 },
     { "id":  4, "short": "w09", "x": 0.8078, "y": 0.3223, "size": 2 },
...
     { "id": 14, "short": "m89", "x": 0.4000, "y": 0.8469, "size": 1 },
     { "id": 15, "short": "m96", "x": 0.3482, "y": 0.1778, "size": 1 }
  ],
  "links": [
     { "type": "arc",  "source": 6, "target": 15, "weight": 1 },
     { "type": "arc",  "source": 2, "target":  7, "weight": 1 },
...
     { "type": "arc",  "source":15, "target":  3, "weight": 1 },
     { "type": "edge", "source": 6, "target": 12, "weight": 1 },
...
     { "type": "edge", "source": 4, "target": 12, "weight": 1 },
     { "type": "edge", "source": 6, "target": 13, "weight": 1 }
  ]
}
```

JSON and R

Transforming Pajek NET and CLU files in to JSON

```
{"nodes":[{"name":"Myriel","group":1},{"name":"Napoleon","group":1}, ...
    {"name":"Brujon","group":4},{"name":"Mme.Hucheloup","group":8}],
 "links":[{"source":1,"target":0,"value":1},{"source":2,"target":0,"value":8}, ...
    {"source":76,"target":48,"value":1},{"source":76,"target":58,"value":1}]}
```

```
setwd("C:/Users/Batagelj/test/python/2012/amazon")
library(rjson)

net2json <- function(netF,cluF,jsonF){
  net <- file(netF,"r");  clu <- file(cluF,"r")
  b <- unlist(strsplit(readLines(net,n=1)," "))
  n <- as.integer(b[length(b)])
  N <- readLines(net,n=n);  nam <- character(n)
  for(i in 1:n) nam[i] <- unlist(strsplit(N[i],'"'))[2]
  skip <- readLines(clu,n=1);  C <- as.integer(readLines(clu,n=n))
  skip <- readLines(net,n=1);  L <- readLines(net,n=-1)
  M <- matrix(as.integer(unlist(strsplit(sub('^\\s+','',L),'\\s+'))),ncol=3,byrow=TRUE)
  nods <- vector('list',n)
  for(i in 1:n) nods[[i]] <- list(name=nam[i],group=C[i])
  m <- nrow(M); lnks <- vector('list',m)
  for(i in 1:m) lnks[[i]] <- list(source=M[i,1]-1,target=M[i,2]-1,value=M[i,3])
  data <- list(nodes=nods,links=lnks)
  jstr <- toJSON(data)
  json <- file(jsonF,"w");  cat(jstr,file=json)
  close(json);  close(net); close(clu)
}

net2json("islands.net","islands.clu","islands.json")
```

islands, island 1, island 4, force: islands

CSV2JSON.R

```
# transforming CSV files to JSON file
# by Vladimir Batagelj, June 2016
setwd("C:/Users/batagelj/work/Python/graph/SVG/EUSN")
library(rjson)
colC <- c(rep("character",4),rep("numeric",5)); nas <- c("","NA","NaN")
nodes <- read.csv2("bibNodesXY.csv",encoding='UTF-8',colClasses=colC,na.
M <- factor(nodes$mode); mod <- levels(M); M <- as.numeric(M)
S <- factor(nodes$sex); sx <- levels(S); S <- as.numeric(S); S[is.na(S)]
links <- read.csv2("bibLinks.csv",encoding='UTF-8',colClasses="character
F <- as.numeric(factor(links$from,levels=nodes$name,ordered=TRUE))
T <- as.numeric(factor(links$to,levels=nodes$name,ordered=TRUE))
R <- factor(links$relation); rel <-levels(R); R <- as.numeric(R)
n <- nrow(nodes); nods <- vector('list',n)
for(i in 1:n) nods[[i]] <- list(id=i,name=nodes$name[i],mode=M[i],
   sex=S[i],x=as.numeric(nodes$x[i])/1000,y=as.numeric(nodes$y[i])/1000)
m <- nrow(links); lnks <- vector('list',m)
for(i in 1:m) lnks[[i]] <- list(type="arc",source=F[i],target=T[i],
   rel=R[i],weight=1)
meta <- list(date="June 11,2016",author="Vladimir Batagelj")
leg <- list(mode=mod,sex=sx,rel=rel)
inf <- list(network="bib",org=1,nNodes=n,nArcs=m,
  title="Example for EUSN'16",legend=leg,meta=meta)
data <- list(netJSON="basic",info=inf,nodes=nods,links=lnks)
json <- file("bib.json","w"); cat(toJSON(data),file=json); close(json)
```

bib.json


```
{"netJSON":"basic",
 "info":{"network":"bib","org":1,"nNodes":16,"nArcs":19,"title":"Example for EUSN'16",
  "legend":{
   "mode":["book","journal","paper","person","publisher","series"],
   "sex":["f","m"],
   "rel":["authorOf","cites","containedIn","editorOf","publishedBy"]},
  "meta":{"date":"June 11,2016","author":"Vladimir Batagelj"}},
 "nodes":[
  {"id":1,"name":"Batagelj, Vladimir","mode":4,"sex":2,"x":0.8091,"y":0.6537},
  {"id":2,"name":"Doreian, Patrick","mode":4,"sex":2,"x":0.3585,"y":0.6791},
  {"id":3,"name":"Ferligoj, Anu\u0161ka","mode":4,"sex":1,"x":0.6195,"y":0.6807},
  {"id":4,"name":"Granovetter, Mark","mode":4,"sex":2,"x":0.1456,"y":0.6605},
  {"id":5,"name":"Moustaki, Irini","mode":4,"sex":1,"x":0.783,"y":0.228},
  {"id":6,"name":"Mrvar, Andrej","mode":4,"sex":2,"x":0.478,"y":0.6301},
  {"id":7,"name":"Clustering with relational constraint","mode":3,"sex":0,"x":0.6841,"y":0.3801}
...
  {"id":15,"name":"Cambridge University Press","mode":5,"sex":0,"x":0.5343,"y":0.2382},
  {"id":16,"name":"Springer","mode":5,"sex":0,"x":0.8846,"y":0.174}],
 "links":[
  {"type":"arc","source":1,"target":10,"rel":1,"weight":1},
  {"type":"arc","source":2,"target":10,"rel":1,"weight":1},
  {"type":"arc","source":3,"target":10,"rel":1,"weight":1},
  {"type":"arc","source":1,"target":7,"rel":1,"weight":1},
  {"type":"arc","source":3,"target":7,"rel":1,"weight":1},
  {"type":"arc","source":4,"target":8,"rel":1,"weight":1},
  {"type":"arc","source":4,"target":14,"rel":4,"weight":1},
...
  {"type":"arc","source":10,"target":7,"rel":2,"weight":1},
  {"type":"arc","source":14,"target":15,"rel":5,"weight":1},
  {"type":"arc","source":11,"target":16,"rel":5,"weight":1}
]}
```


bib.json, picture: bib

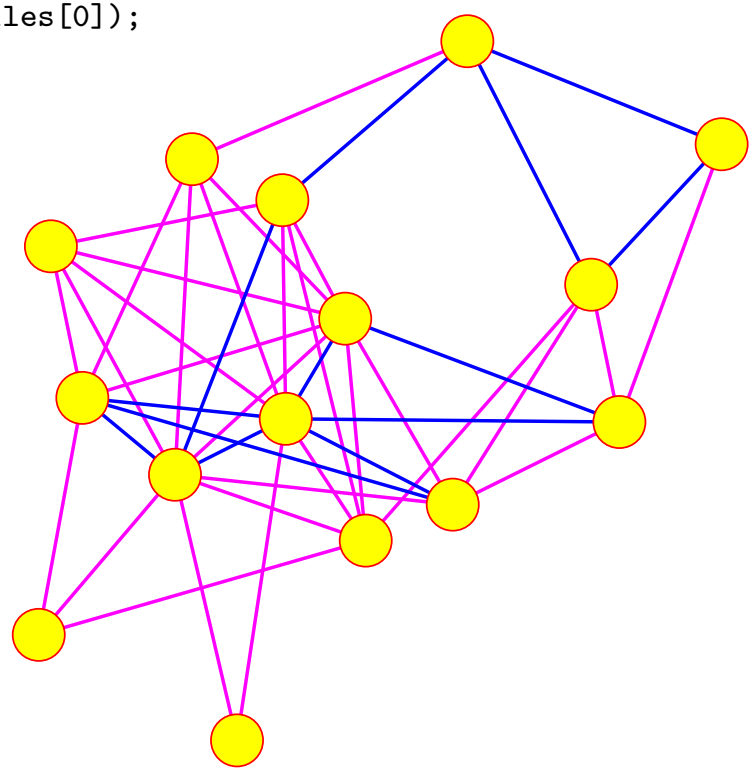
Reading JSON files and displaying a network with given nodes' coordinates


```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<script src="http://d3js.org/d3.v3.min.js"></script>
</head>
<body>
<input type='file' accept='application/json' onchange='openFile(event)'>
<script>
function process(graph) {
// set up the drawing area
var width = 500,
  height = 500; s = graph.attributes.org;
var svg = d3.select("body").append("svg")
  .attr("width", width)
  .attr("height", height)
  .attr("xmlns", "http://www.w3.org/2000/svg");
// draw the links
var link = svg.selectAll("line")
  .data(graph.links).enter().append("line")
  .style("stroke", function(d,i) {return((d.type=="arc" ? "magenta" : "blue"))})
  .attr("stroke-width", 2)
  .attr("x1", function(d) {return(graph.nodes[d.source-s].x*width);})
  .attr("y1", function(d) {return(graph.nodes[d.source-s].y*height);})
  .attr("x2", function(d) {return(graph.nodes[d.target-s].x*width);})
  .attr("y2", function(d) {return(graph.nodes[d.target-s].y*height);});
// draw the nodes
var node = svg.selectAll("circle")
  .data(graph.nodes).enter().append("circle")
  .attr("r", 15)
  .attr("cx", function(d,i) {return(d.x*width);})
  .attr("cy", function(d,i) {return(d.y*height);})
  .attr("fill", "yellow")
  .attr("stroke", "red");
}
```

. . . reading

adapted from Matt West Reading Files Using The HTML5 FileReader API

```
var openFile = function(event) {
  var input = event.target;
  var reader = new FileReader();
  reader.onload = function(){
    process(JSON.parse(reader.result));
  };
  reader.readAsText(input.files[0]);
};
</script>
</body>
```



graphRead.html

Reading JSON file from server

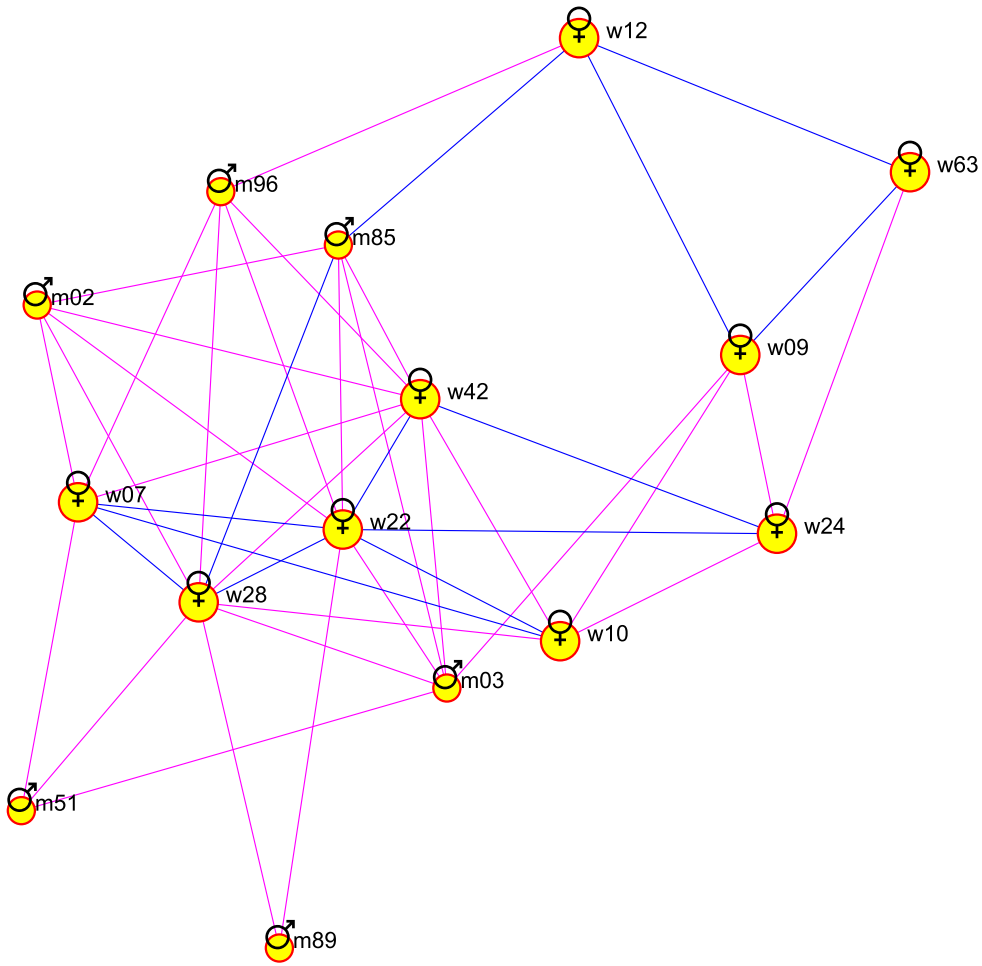
adapted from Ying Kit Yuen jQuery & Javascript -– Read JSON files on server

```
<!DOCTYPE html>
<head>
<meta charset="utf-8">
<title>Load JSON file from server</title>
<!-- http://eureka.ykyuen.info/2013/09/25/jquery-javascript-read-json-files-on-server/ -->
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
<script src="http://d3js.org/d3.v3.min.js"></script>
</head>
<body>
<select id="data">
  <option value="graph.json">graph.json</option>
  <option value="class.json">class.json</option>
</select>
<button id="btn">Read</button>
<script>
function process(graph) {
// set up the drawing area
  ... the body of function process is the same as in the previous example
  .attr("stroke", "red");
}

$("#btn").click(function(){
  $.getJSON($("#data").val(), function(json) {process(json);});
});
</script>
</body>
```

graphLoad.html

Network displayer graphXY



drawNet.js

graphXYgen

```
<!DOCTYPE html>
<meta charset="utf-8">
<body>
<!-- script src="./graphA.js"></script -->
<script src="./graphB.js"></script>
<!-- script src="./classS.js"></script -->
<script src="./drawNet.js"></script>
<script src="./d3/d3.js"></script>
<script>
var
s = graph.info.org,
width = graph.style.canvas.width,
height = graph.style.canvas.height;
var lw = 1;
if ( typeof graph.style !== 'undefined' && graph.style) {
  if ( typeof graph.style.link !== 'undefined' && graph.style.link ) {
    if ( typeof graph.style.link.width !== 'undefined' && graph.style.li
      var lw = graph.style.link.width
} } };
if(graph.netJSON=='general') {
  var OK = drawNet(graph.persons.data,graph.friend.data)
} else {
  var OK = drawNet(graph.nodes,graph.links)
}
</script>
</body>
```

Saving SVG picture

S. Murray: Interactive Data Visualization for the Web, p. 235

In Google Chrome we draw a picture and open the Developer Tools. In the Elements we identify the SVG subobject, select it, copy it into some text editor, and save with extension `.svg`.

To enhance the picture or to transform it to other picture formats we process it using some vector graphics editor.

To do

- displayer templates can be based on GUI;
- for a description of multirelational networks we use the `rel` attribute; in the `general` version they can be treated as separate sets of links;
- properties are considered as attributes; in the `general` version they can be represented as special objects `vector`, `partition`, `permutation` and `cluster`;
- use temporal quantities for describing temporal networks;
- extend the list of info-attributes: `attributes: type` (simple, temporal), `twoModeOrg`, `nStrong`, `nRelations`, `planar`, ...
- icons can be used for visualization of nodes Font Awesome, Material Icons , ..., tests;
- links can be visually represented in many different ways that can be described in `style`;

. . . to do

- `style` can be attached as an attribute also to an element (node, link) thus changing the default settings;
- add the visualization of arcs with arrows Directed Graph Editor, D3 Tips and Tricks;
- to be included in netD3.js: vzmetno risanje Force in urejanje Springy. Matrix with permutations.
- can some attributes be renamed: : `from`, `tail`, `nodeA` $\rightarrow$ `source` ; `to`, `head`, `nodeB` $\rightarrow$ `target`; . . . , may be the simplest solution is a replace in some text editor;
- implement saving of the obtained SVG picture to a file: Export SVG with Style, d3js/SVG Export demo, . . .
- include in netJSON elements useful in some applications, such as, hooks or in/out-ports; background image, etc.

Sources

Ideas for visualization styles can be found in GoJS - Interactive Diagrams for JavaScript and HTML, Vis.js in Visual Complexity.

GoJS: —Sankey Diagram; Family Tree; Logic Circuit; Dynamic Ports
Vega - a visualization grammar.

References I


Vladimir Batagelj: Complex Networks, Visualization of. R.A. Meyers, ed., Encyclopedia of Complexity and Systems Science, Springer 2009: 1253-1268.


Vladimir Batagelj, Andrej Mrvar: Pajek manual.


Jernej Bodlaj: Network Data File Formats. in Reda Alhajj, Jon Rokne (eds.): Encyclopedia of Social Network Analysis and Mining. Springer, New York, 2014, p. 1076-1091.


Richard Brath, David Jonker: Graph Analysis and Visualization: Discovering Business Opportunity in Linked Data. John Wiley & Sons, Indianapolis, Indiana, 2015.


Emden Gansner, Eleftherios Koutsofios, Stephen North: Drawing graphs with dot, January 26, 2006


Scott Murray: Interactive Data Visualization for the Web. O'Reilly, Sebastopol, 2013.

References II


Wouter De Nooy, Andrej Mrvar, Vladimir Batagelj: Exploratory Social Network Analysis with Pajek; Revised and Expanded Second Edition. Structural Analysis in the Social Sciences, Cambridge University Press, September 2011.


Matthew Roughan, Jonathan Tuke: Unravelling Graph-Exchange File Formats. arXiv:1503.02781, submitted on 10 Mar 2015.


Wikipedia: JSON


Leland Wilkinson: The Grammar of Graphics. Springer-Verlag, New York, 2005.