Introduction to Complex Networks

Network Application Diagnostics
B2M32DSA

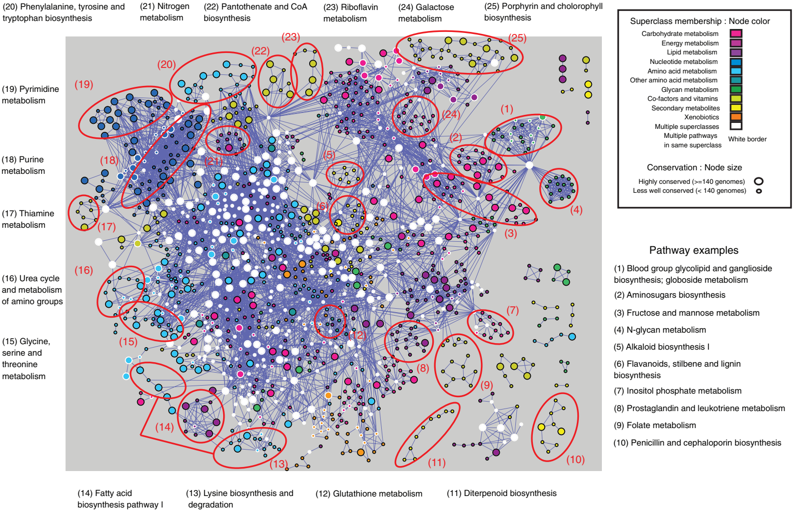
Radek Mařík

Czech Technical University
Faculty of Electrical Engineering
Department of Telecommunication Engineering
Prague CZ

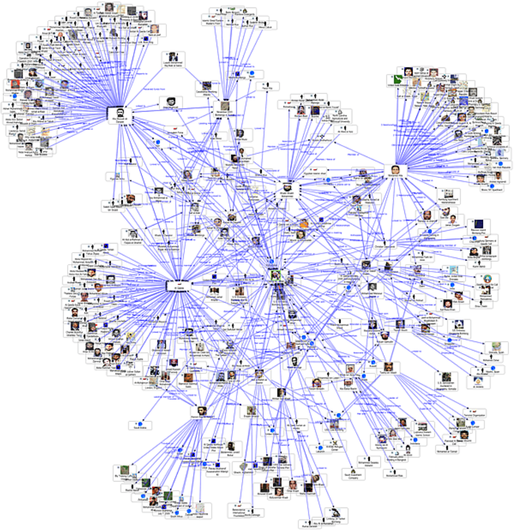
October 1, 2017

Outline

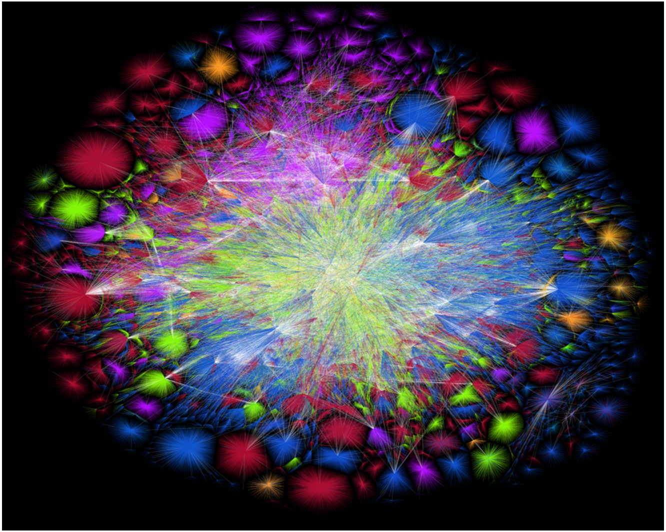
Conservation within the global metabolic network [PASP09]



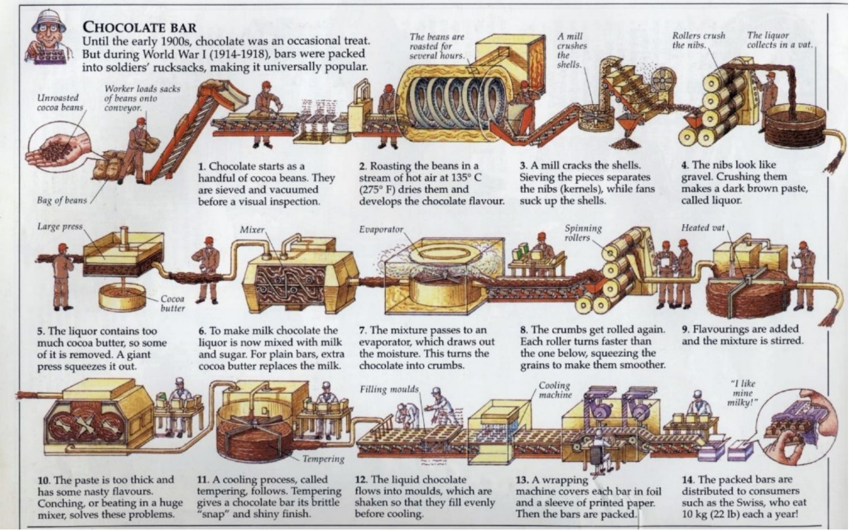
Link Analysis of the Al Qaeda Terrorist Network [FMS]



Internet Map in 2015 [Blo14, Opt17]



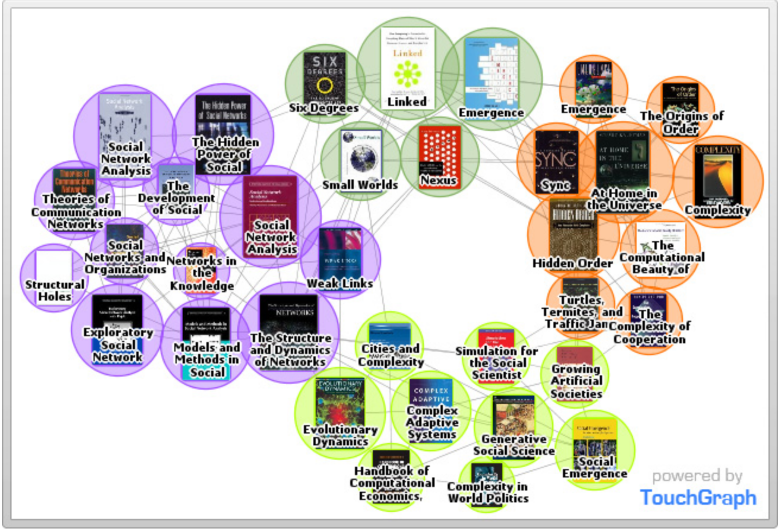
Chocolate Making Process Dependencies [Fre14]



More Examples

- Biological networks
  - gene regulation networks
  - protein-protein interaction networks
  - metabolic networks
  - the food web
  - predator-prey relations
  - brain network
- Social networks:
  - networks of acquaintances
  - collaboration networks
  - phone-call networks
  - citation networks
  - opinion formation
  - society/community/party networks
- Technological networks:
  - the Internet
  - telephone networks
  - transportation networks
  - sensor networks
  - energy grid networks
- Informational networks:
  - the World Wide Web
  - Twitter
  - Facebook
  - peer-to-peer

SNA Books


Six Degrees
Linked
Emergence
Emergence
The Origins of Order
Social Network Analysis
The Hidden Power of Social
Small Worlds
Nexus
Sync
At Home in the Universe
Complexity
Theories of Communication Networks
The Development of Social
Social Network Analysis
Hidden Order
The Computational Beauty of
Social Networks and Organizations
Networks in the Knowledge
Weak Links
Structural Holes
Turtles, Termites, and Traffic Jam
The Complexity of Cooperation
Exploratory Social Network
Models and Methods in Social
The Structure and Dynamics of Networks
Cities and Complexity
Simulation for the Social Scientist
Growing Artificial Societies
Evolutionary Dynamics
Complex Adaptive Systems
Generative Social Science
Social Emergence
Handbook of Computational Economics
Complexity in World Politics
powered by TouchGraph

Approach to Complex Networks

- One needs to distinguish between **analysis** and **production** phases
- Some phenomena appear only with sufficiently large data volumes (emergent events)
- Volume
  - A number of suitable tools ... HDF5, ElasticSearch, Clouds
  - Capable to operate with terabytes of data
- Visualization
  - Critical if anomaly features are not known
  - At present, there is no obvious choice of a tool and a network layout given a particular problem.
  - Tools do not often scale with data volumes (> 10.000 nodes, 10^5 edges)
  - GGobi, Pajek, NetworkX, SNAP, Tulip, Gephi, Cytospace, yEd, D3.js
  - Aspects: data volume, interactions with the user

Popular software packages [HLDS13]

- Analysis
  - **UCINET** (http://www.analytictech.com/ucinet.htm)
  - **ENET** (http://analytictech.com/e-net/e-net.htm)
  - **Pajek** (http://pajek.imfm.si/doku.php?id=pajek)
  - **RSIENA**
  - **R**
  - **NodeXL**
  - **NetworkX** . . . a Python library
  - **iGraph** . . . a C/Python library
- Visualization
  - **yEd**
  - **Gephi**
  - **Cytospace**
  - **Tulip**
  - **NetDraw** (2D, embedded in UCINET, see above)
  - **Mage** (3D, embedded in UCINET, see above)
  - visit www.netvis.org/resources.php for more

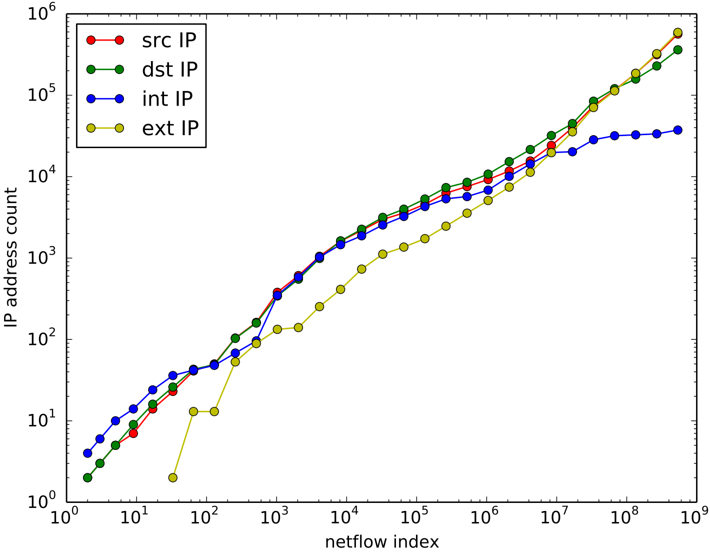
NETFLOW Primary Statistics

- **Netflow**
  - Condensed records on a packet flow
  - Several packets are merged into one netflow record
  - Only 14-20 aggregated metrics

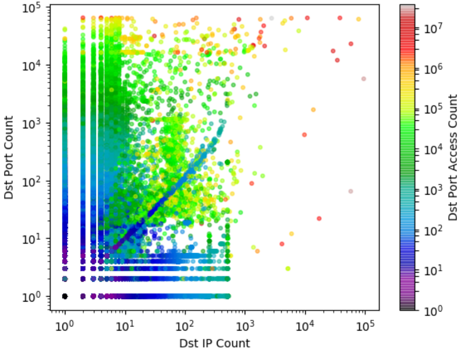
An enterprise traffic as a netflow sample taken during 9 days:

Statistics	Value
Total transported data volume	13,995,690,457,765 $[B]$
Packet count	20,131,367,095
Netflow count	617,326,053
IP address count	686,168
Source IP address count	614,150
Destination IP address count	392,881
Different P2P connections count	2,412,481

Is the Sample of IP addresses reprezentative?

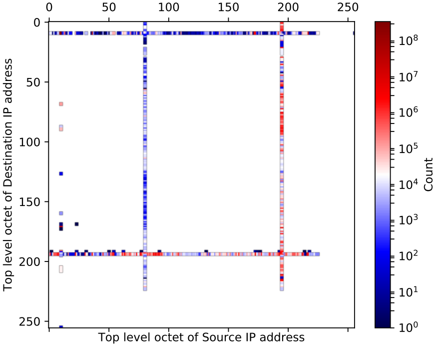


A Data Projection Focused on Services



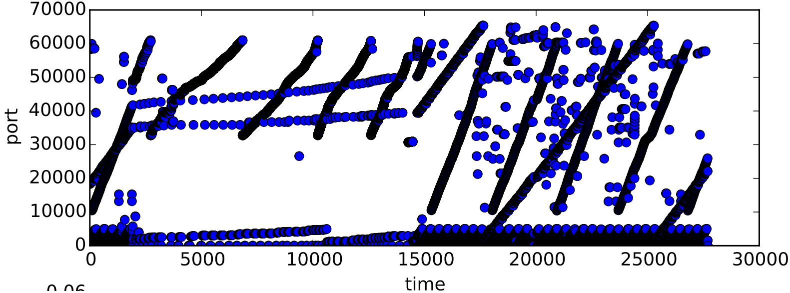
- Destination IP vs. destination port (space of services and their locations)
- Some counts of accesses are exceptional (red)

Top Level IP Network Projection - Data Sparsity



- Focused on the network of source and destination IP addresses
- Top level octets of IP addresses (160.30.29.17 $\Longrightarrow$ 160)
- A very sparse space
- A rather restricted source-destination IP connections (as expected)

Port Scanning from xxx.xxx.18.120 - Logical Time Progress



- 617,326,053 netflows $\approx$ 60,000 samples $\times$ sample size 10.000
- $\Longrightarrow$ 60,000 samples might be still visualized with difficulties
- $\Longrightarrow$ 1.000 events can be easily missed with 10,000 sample size

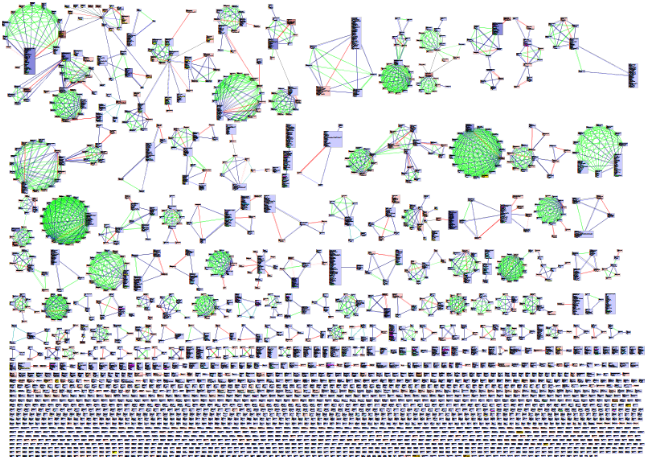
Masters of Social Network Analysis [RP13, Weh13]

- US National Security Agency
- Maintains large programs in social network analysis
- Believed to process 2×10^{10} node and tie updating events per day
- Result:
  "Better Person Centric Analysis"

Types

- **94 entity/node** types
  (*phone numbers*, *e-mail addresses*, *IP addresses*, etc.)
- **164 relationship** types to build "community of interest" profiles
  (*travelsWith*, *hasFather*, *sentForumMessage*, *employs*, etc.)

Egypt Data - Family Recognition

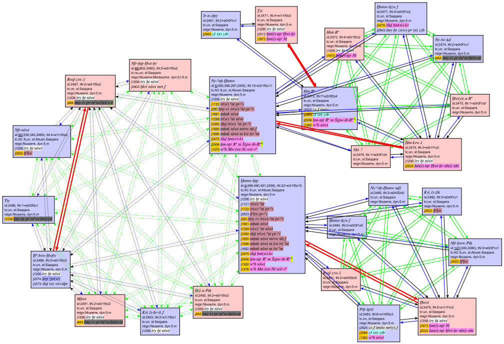


circular layout (yEd)

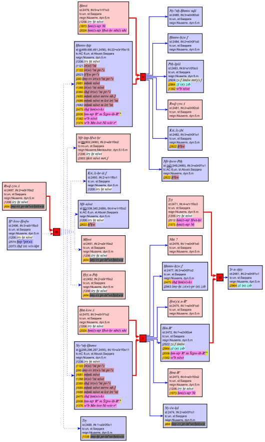
A family:

- Using family designation
  - husband, wife, son, etc.
- A connected graph component
- Sparse data assumed
- Transformed into family tree using marriage nodes

Egypt Data - Transformation into Family Tree

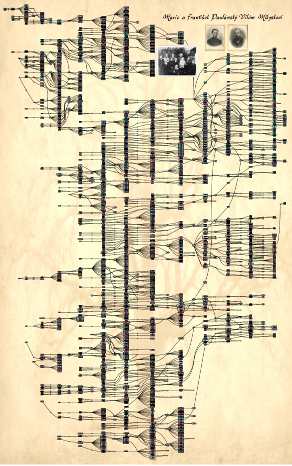


A family as a connected component
circular layout (yEd)

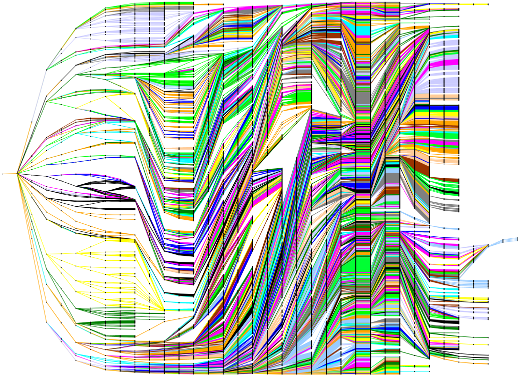


A family tree
hierarchical layout (yEd)

Family Trees[Mar17]



multitree-like tree driven layout, Graphviz

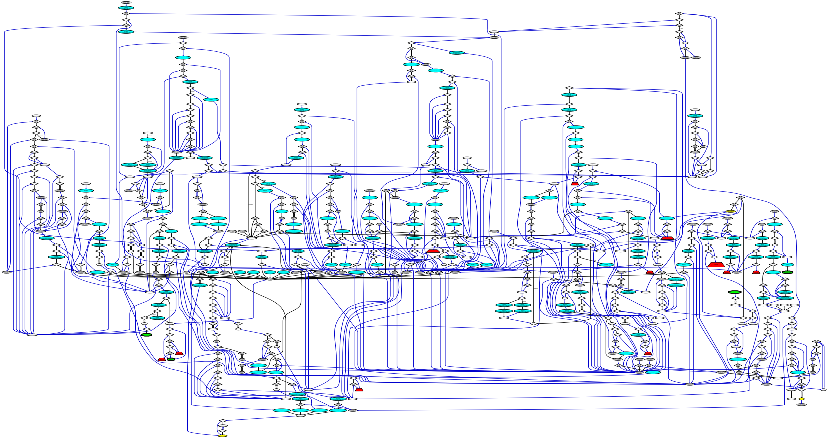


- Taxonomic information ITIS on plants, animals, fungi, and microbes,
- A phylogenetic tree with 945.352 nodes
- **multitree-like tree driven layout**

HLASM Mainframe Assembly

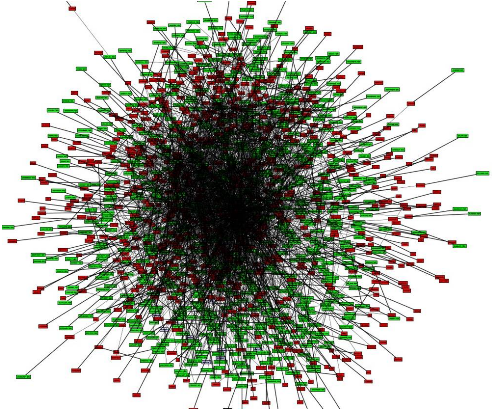
```
◂ Active Usings: IHIFSARA,R13
◂ Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                          HLASM R5.0  2008/04/18 03.22
+00028A 1A3B                        646          AR    R3,PBT           ADDRESS OF PBT-ENTRY          11980000
+00028C 910C 3006      00006        647          TM    6(R3),BETABM     PROCEDURE CALLED              12000000
+000290 4780 D2A4            002A4  648          BZ    PROLOG1          NO                            12020000
+000294 0000 0000            00000  649          C     ADR,ASTLOC(0,FSA) COMP.CONT.OF ADR.WITH ADDR.OF 12040000
+** ASMA044E Undefined symbol - ASTLOC
+** ASMA435I Record 619 in KOTEM01.CBT310.ASM(IHIFSA) on volume: TSU011
+                                   650 *                               *FUNCTIONVALUESTORAGE         12060000
+000298 0000 0000            00000  651          BE    OERR21(0,FSA)    BRANCH IF EQUAL               12080000
+** ASMA044E Undefined symbol - OERR21
+** ASMA435I Record 621 in KOTEM01.CBT310.ASM(IHIFSA) on volume: TSU011
+00029C 9110 3006      00006        652          TM    6(R3),CODEPRM    CODE PROCDURE CALLED          12100000
+0002A0 4710 D47A            0047A  653          BO    PROLOG2          YES                           12120000
+0002A4 4803 0004            00004  654 PROLOG1  LH    R0,4(R3)         LENGTH OF DSA TO REG 0        12140000
+0002A8 184F                        655          LR    R4,BRR           SAVE BRR DURING GETMAIN       12160000
+                                   656          GETMAIN R,LV=(0)       GETMAIN FOR DSA               12180000
+0002AA 4510 D2AE            002AE  658+         BAL   1,*+4           INDICATE GETMAIN      @Z30EN9G 01-GETMA
+0002AE 0A0A                        659+         SVC   10              ISSUE GETMAIN SVC              01-GETMA
+0002B0 18F4                        660          LR    BRR,R4                                         12200000
+0002B2 5802 B000            00000  661          L     R0,0(R2,PBT)     LOAD POINTER OF LAST GENERATION 12220000
+0002B6 5000 1000            00000  662          ST    R0,0(0,R1)       AND STORE IT IN DSA           12240000
+0002BA 50A0 1004            00004  663          ST    CDSA,4(0,R1)     STORE POINTER OF EMBRACING PB. 12260000
+0002BE 4020 1008            00008  664          STH   R2,8(0,R1)       STORE PBT DISPLACEMENT        12280000
+0002C2 9200 100A      0000A        665          MVI   10(R1),X'00'     ZEROS TO VALUE ARRAY AND      12300000
+0002C6 D204 100B 100A 0000B 0000A  666          MVC   11(5,R1),10(R1)  *ARRAY POINTERS               12320000
+0002CC 5012 B000            00000  667          ST    R1,0(R2,PBT)     STORE CURR.DSA POINTER IN PBT 12340000
+0002D0 18A1                        668          LR    CDSA,R1          SET CDSA POINTER              12360000
+0002D2 90BC A010            00010  669          STM   PBT,LAT,16(CDSA)                               12380000
+0002D6 0000 0000            00000  670          L     STH,RASPT(FSA)   RAS-POINTER TOP               12400000
```

CHALLENGE: Complex Control Flow, a typical case



layered layout - Graphviz dot

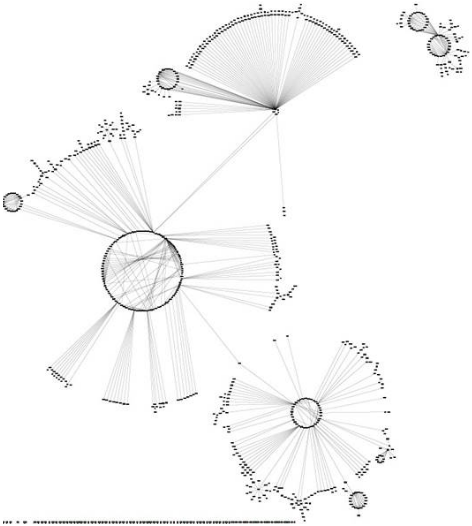
Dependancy of External Symbols in Mainframe Assembly Software



Fruchterman-Reingold force-driven layout

- A software product . . . over 10.000.000 lines of code
- Over 400 modules . . . red
- External symbols . . . green
- Thick line . . . the definition of a symbol
- Thin line . . . a reference to a symbol
-
- *Where should the developer start with a bug analysis?*

Assembly Software - Recovered Architecture

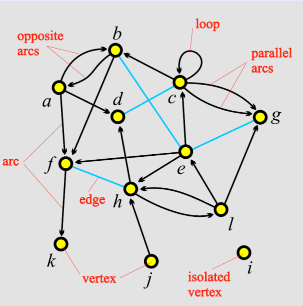


double-circular layout - yEd

Company Network of People - 3D Hyperbolic Tree Layout (Walrus)

Graph [Weh13]

A **graph** is a set of vertices and a set of lines between pairs of vertices.



- **Actor** - **vertex**, **node**, point
- **Relation** - line, edge, arc, link, tie
  - **Edge** $=$ undirected line, $\{c, d\}$
    c and d are **end** vertices
  - **Arc** $=$ directed line, (a, d)
    a is the **initial** vertex, (source, start)
    d is the **terminal** vertex, (target, end)
  - Parallel (multiple) arcs/edges are only allowed in **multigraphs** with more than one relation (set of lines).
  - **Loop** (self-choice)

We focus on simple graphs!

A **simple** undirected graph has no loops and no parallel edges.
A simple directed graph has no parallel arcs.

Network [EK10, New10, Weh13, Erc15]

Network

A **network** consists of a graph and additional information on the vertices or the lines of the graph.

Formally, 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 vertices,
  - $\mathcal{A}$ is the set of arcs,
  - $\mathcal{E}$ is the set of edges, and
  - $\mathcal{L} = \mathcal{E} \cup \mathcal{A}$ is the set of lines.
- $\mathcal{P}$ vertex value functions / properties: $p : \mathcal{V} \rightarrow A$
- $\mathcal{W}$ line value functions / weights: $w : \mathcal{L} \rightarrow B$

- **Long range dependencies** vs. multidimensional space
- **Specific topological properties**
- **Large/Huge volumes** of **sparse** data records

Asymptotic Notation [CLRS09, Erc15]

Let $c, c_1, c_2 \in \mathbb{R}^{>0}$, $n_0, n \in \mathbb{N}$, $f, g \in \mathbb{N} \rightarrow \mathbb{R}^+$

Asymptotic upper bound (CZ horní asymptotický odhad)

$f(n) \in O(g(n))$, if $(\exists c > 0)(\exists n_0)(\forall n > n_0) : |f(n)| \leq |c \cdot g(n)|$

Asymptotic lower bound (CZ dolní asymptotický odhad)

$f(n) \in \Omega(g(n))$, if $(\exists c > 0)(\exists n_0)(\forall n > n_0) : |c \cdot g(n)| \leq |f(n)|$

Asymptotic tight bound (CZ optimální asymptotický odhad)

$f(n) \in \Theta(g(n))$, if $\Theta(g(n)) \stackrel{\text{def}}{=} O(g(n)) \cap \Omega(g(n))$
$(\exists c_1, c_2 > 0)(\exists n_0)(\forall n > n_0) : |c_1 \cdot g(n)| < |f(n)| < |c_2 \cdot g(n)|$

NP-Completeness [CLRS09, Erc15]

P and NP

- **P - Polynomial**. Problems that can be solved in polynomial time.
- **NP - Nondeterministic Polynomial**. A problem is in NP if you can in polynomial time by a *certifier* test whether a solution is correct without worrying about how hard it might be to find the solution.
    - Nondeterministic is a fancy way of talking about guessing a solution.
- $\mathrm{P} \subseteq \mathrm{NP}$ (??? $\mathrm{P} = \mathrm{NP}$???)

NP-complete and NP-hard

- **NPH - NP-hard**. An NPH problem is a problem which is as hard as any problem in NP
    - An NPH problem does not need to have a certificate.
- **NPC - NP-complete**. A problem is NPC if it is NP and is as hard as any problem in NP
    - A problem A is NPC if it is both NPH and in NP, $\mathrm{NPC} = \mathrm{NP} \cap \mathrm{NPH}$.

Complexity Classes Other Than NP [CLRS09, Erc15]

Complexity classes harder than NP

- **PSPACE**. Problems that can be solved using a reasonable amount of memory
  - defined formally as a polynomial in the input size
  - without regard to how much time the solution takes.
- **EXPTIME**. Problems that can be solved in exponential time.
- **Undecidable**. For some problems, we can prove that there is no algorithm that always solves them, no matter how much time or space is allowed.

Tree Search [BM08]

- A systematic procedure, or **algorithm**,
that generates a sequence of rooted trees in G,
starting with the trivial tree consisting of a single root vertex r, and terminating either with a spanning tree of the graph
or with a nonspanning tree whose associated edge cut is empty,
is called **tree-search** and
the resulting tree is referred to as a **search tree** [BM08].
- **Depth-first search** is a tree-search in which the vertex added to the tree T at each stage is one which is a neighbor of as recent an addition to T as possible.
- The resulting spanning tree is called a **depth-first search tree** or **DFS-tree**.

DFS-tree Search Edge Classification [BM08]

- There are two times associated with each vertex $v \in G$ during the construction of its DFS-tree T:
  - the **discovery time** $\tau_d(v)$ when v is incorporated into T and
  - the **finish time** $\tau_f(v)$ when all the neighbors of v are found to be already in T.
- In particular, $\tau_d(r) = 1$, $\tau_f(v) = \tau_d(v) + 1$ for every leaf v of T, and $\tau_f(r) = 2|V|$.
- Based on Proposition 1 and Theorem 1 any edge $e = uv$ in a graph G having a DFS-tree T with $\tau_d(u) < \tau_d(v) < \tau_f(v) < \tau_f(u)$ can be oriented as $\vec{e} = \overrightarrow{uv} = (u, v)$ and classified as:
  - **tree edge**, if $e \in T$, i.e. the vertex u is an ancestor of v in T,
  - **back edge**, if $e \notin T$.

Tree Search Times - Properties

Proposition 1 (Proposition 6.5 [BM08], p.141)

Let u and v be two vertices of G, with $\tau_d(u) < \tau_d(v)$.

a) *If u and v are adjacent in G, then $\tau_f(v) < \tau_f(u)$.*

b) *u is an ancestor of v in T if and only if $\tau_f(v) < \tau_f(u)$.*

Theorem 1 (Theorem 6.6 [BM08], p.142)

Let T be a DFS-tree of a graph G. Then every edge of G joins vertices which are related in T.

Lemma 1 (Lemma 22.11 [CLRS09], p.614)

A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

Tree Search Times - Properties

Proposition 2 (Proposition 1.5.6 [Die05], p.16)

Every connected graph contains a normal spanning tree, with any specified vertex as its root.

Breadth-first Search [CLRS09, Erc15]

Algorithm 1 BFS

```
Input: G(V, E), a source node s
Output: d_v, pred[v], ∀v ∈ V
        ▷ distance and place of a vertex in BFS
Q ...a queue
for all u ∈ V \ {s} do
    d_u ← ∞
    pred[u] ← ⊥        ▷ undetermined value
end for
d_s ← 0
pred[s] ← s
```

BFS ...the main loop

```
Q ← s
while Q ≠ ∅ do
    u ← deque(Q)
    for all (u, v) ∈ E do
        if d_v = ∞ then
            d_v ← d_u + 1
            pred[v] ← u
            enqueu(Q, v)
        end if
    end for
end while
```

Theorem 2 (Theorem 3.1 [Erc15], p.35)

The time complexity of BFS algorithm is $\Theta(N + M)$ for a graph of order N and size M.

Depth-first Search [CLRS09, Erc15]

Algorithm 2 DFS_Forest

Input: $G(V, E)$, directed or undirected
Output: pred$[v]$, firstVis$[v]$, secVis$[v]$, $\forall v \in V$
int $time \leftarrow 0$; $visited[1:n] \leftarrow 0$
for all $u \in V$ **do**
  $visited[u] \leftarrow false$
  pred$[u] \leftarrow \perp$ ▷ undetermined value
end for
for all $u \in V$ **do**
  if $\neg visited[u]$ **then**
    $DFS(u)$
  end if
end for

DFS procedure

procedure $\mathrm{DFS}(\mathrm{u})$
  $visited[u] \leftarrow true$
  $time \leftarrow time + 1$
  $firstVis[u] \leftarrow time$
  for all $(u, v) \in E$ **do**
    if $\neg visited[v]$ **then**
      pred$[v] \leftarrow u$
      $DFS(u)$
    end if
  end for
  $time \leftarrow time + 1$
  $sectVis[u] \leftarrow time$
end procedure

Asymptotic complexity of the DFS algorithm

The time complexity is $\Theta(N + M)$ for a graph of order N and size M.

Dijkstra's Single Source Shortest Paths [CLRS09, Erc15]

Algorithm 3 Dijkstra_SSSP

Input: $G(V, E)$, directed or undirected,
Input: positive weights l_e on edges,
Input: a source node s
Output: d_v, pred$[v]$, $\forall v \in V$
for all $u \in V \setminus \{s\}$ **do**
    $d_u \leftarrow \infty$
    pred$[u] \leftarrow \perp$ ▷ undetermined value
end for
$d_s \leftarrow 0$
pred$[s] \leftarrow s$

SSSP ...the main loop

$S \leftarrow [V]$ ▷ insert all vertices
while $S \neq \emptyset$ **do**
    $u \leftarrow min(S)$
    $S \leftarrow S \setminus \{u\}$
    for all $(u, v) \in E$ **do**
        if $d_v > d_u + l(u, v)$ **then**
            $d_v \leftarrow d_u + l(u, v)$
            pred$[v] \leftarrow u$
        end if
    end for
end while

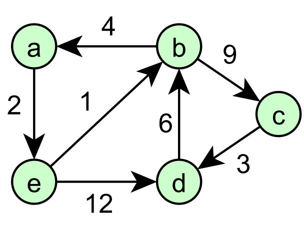
Theorem 3 (Theorem 5.1 [Erc15], p.84)

The time complexity of the Dijkstra's_SSSP is $O(N^2)$ for a graph of order N.

Floyd-Warshall All Pairs Shortest Paths [CLRS09, Erc15]

- The approach
  - Dynamic programming approach
  - Comparing all possible paths between each pair of nodes in G
  - Improving the shortest path between them at each step until the result is optimal.
- Distance matrix $D[N, N]$ between nodes u and v
- Matrix $P[N, N]$ with the first node on the current shortest path from u to v

Example 1



FW APSP Algorithm [CLRS09, Erc15]

Algorithm 4 FW_APSP

Input: $G(V, E)$,
Input: weights w_e on edges,
no negative-weight cycles
Output: $D[N, N]$, $P[N, N]$
for all $\{u, v\} \in V$ **do**
  if $u = v$ **then**
    $D[u, v] \leftarrow 0$; $P[u, v] \leftarrow \perp$
  else if $(u, v) \in E$ **then**
    $D[u, v] \leftarrow w_{uv}$; $P[u, v] \leftarrow v$
  else
    $D[u, v] \leftarrow \infty$; $P[u, v] \leftarrow \perp$
  end if
end for

APSP . . . the main loop

$S \leftarrow \emptyset$
while $S \neq V$ **do**
  pick w from $V \setminus S$ ▷ Select a pivot
  for all $u \in V$ **do**
    for all $v \in V$ **do**
      if $D[u, w] + D[w, v] < D[u, v]$ **then**
        $D[u, v] \leftarrow D[u, w] + D[w, v]$
        $P[u, v] \leftarrow P[u, w]$
      end if
    end for
  end for
  $S \leftarrow S \cup \{w\}$
end while

Asymptotic complexity of the FW_APSP algorithm

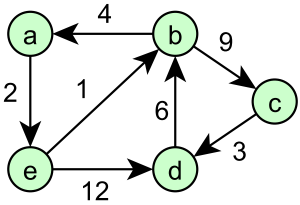
The time complexity is $\Theta(N^3)$ for a graph of order N.

FW APSP Algorithm Example [Erc15]

$$D = \begin{bmatrix} 0 & \infty & \infty & \infty & 2 \\ 4 & 0 & 9 & \infty & \infty \\ \infty & \infty & 0 & 3 & \infty \\ \infty & 6 & \infty & 0 & \infty \\ \infty & 1 & \infty & 12 & 0 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 0 & 3 & \infty & 14 & 2 \\ 4 & 0 & 9 & 12 & 6 \\ \infty & 9 & 0 & 3 & \infty \\ 10 & 6 & 15 & 0 & \infty \\ 5 & 1 & 10 & 12 & 0 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 0 & 3 & 12 & 14 & 2 \\ 4 & 0 & 9 & 12 & 6 \\ 13 & 9 & 0 & 3 & 10 \\ 10 & 6 & 15 & 0 & 12 \\ 5 & 1 & 10 & 12 & 0 \end{bmatrix}$$



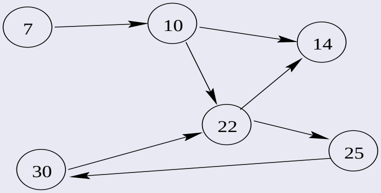
Summary

- An introduction to complex networks
- Several practical application domains shown
- Software tools overview
- Demonstration of two issues
  - Network data volume
  - Network visualization
- Graph Terminology Reminder
- Graph Path Algorithms Reminder

Appendix

Graph Representation [Bei95]

Graph



List

7: 10
10: 14, 22
14:
22: 14, 25
25: 30
30: 22

Adjacency matrix (Table)

	7	10	14	22	25	30
7	.	1	.	.	.	.
10	.	.	1	1	.	.
14	.	.	.	.	.	.
22	.	.	1	.	1	.
25	.	.	.	.	.	1
30	.	.	.	1	.	.

Graph (Formal Definitions) [Die05, BM08, Wil98]

- A **graph** is a pair $G = (V, E)$ of sets such that $E \subseteq [V]^2$, $V \cap E = \emptyset$, together with an **incidence function** ψ_G that associates with each edge of G an unordered par of not necessarily distinct vertices of G.
- The number of vertices of a graph G is its **order** $N = v(G) = |V| = |G|$.
- A graph with vertex set V is said to be a **graph on** V.
- The vertex set of a graph G is referred to as $V(G)$, its edge set as $E(G)$, independently of any actual names of these two sets.
- We also write $v \in G$ instead of $v \in V(G)$, similarly $e \in G$.
- The number of edges of a graph G is its **size** denoted by $M = e(G) = |E| = ||G||$.

Graph Operations [Die05, BM08, Wil98]

- Let $G = (V, E)$ and $G' = (V', E')$ be two graphs.
- If $V' \subseteq V$ and $E' \subseteq E$, then G' is a **subgraph** of G, written as $G' \subseteq G$.
- If $G' \subseteq G$ and G' contains all the edges $xy \in E$ with $x, y \in V'$, then G' is an **induced subgraph** of G; i.e. V' **induces** or **spans** G' in G and $G' := G[V']$.
- $G' \subseteq G$ is a **spanning** subgraph of G if V' spans all of G, i.e. if $V' = V$.
- If U is any set of vertices, we write $G - U$ for $G[V \setminus U]$.
- If $U = \{v\}$ is a singleton, we write $G - v$ rather than $G - \{v\}$.
- For a subset $F \subseteq [V]^2$ we write
  $G - F := (V, E \setminus F)$ and
  $G + F := (V, E \cup F)$;
  $G - \{e\}$ and $G + \{e\}$ are abbreviated to $G - e$ and $G + e$.

Graph Maximality [Die05]

- A graph G is **edge-maximal** with a given graph property if G itself has the property byt no graph $G + uv$ does for non-adjacent vertices $u, v \in G$.
- When we call a graph **minimal** or **maximal** with some property but hove not specified any particular ordering, we refer to the subgraph relation.
- We speak of minimal or maximal sets of vertices or edges if the reference is made to set inclusion.

Graph Edges [Die05, BM08, Wil98]

- Let e be an edge and u and v are vertices such that $\psi_G(e) = \{u, v\}$.
- A vertex v is **incident** with an edge e if $v \in e$; then e is an edge **at** v.
- The set of all the edges in E at a vertex v is denoted by $E(v)$.
- The two vertices v_1 and v_2 incident with an edge $e = \{v_1, v_2\}$ are its **endvertices** or **ends**, and an edge **joins** its ends.
- An edge $\{u, v\}$ might be written as uv (or vu).
- If $u \in U \subseteq V$ and $w \in W \subseteq V$ then uw is an $U - W$ **edge**.
- The set of all $U - W$ edges in a set E is denoted by $E(U, W)$.
- Two vertices $u, v \in G$ are **adjacent**, or **neighbors**, if $uv \in G$.
- Two edges $e \neq f$ are **adjacent** if they have an end in common.
- If $\{V_1, V_2\}$ is a partition of V, the set $E(V_1, V_2)$ of all the edges of G **crossing** this partition is called a **cut**.

Graph Neighborhood [Die05, BM08, Wil98]

- Let $G = (V, E)$ be a (non-empty) graph.
- The set of **neighbors** of a vertex v **in** G is denoted by $N_G(v)$, or briefly by $N(v)$.
- The **neighbors of** U for $U \subseteq V$, denoted by $N(U)$, is the set of the neighbors $V \setminus U$ of vertices in U.
- The **degree** (or **valency**) $d_G(v) = d(v)$ of a vertex v is the number $|E(v)|$ of edges at v.
- Let $r \geq 2$ be an integer.
- A graph $G = (V, E)$ is called r-**partite** if V admits a partition into r classes such that every edge has its ends in different classes: vertices in the same partition class are not adjacent.
- If $r = 2$ then such a graph is denoted as **bipartite**.
  - $V = V_1 \cup V_2,\ V_1 \cap V_2 = \emptyset$

Graph Path [Die05, BM08, Wil98]

- A **path** is a non-empty graph $P = (V, E)$ of the form $V = \{v_0, v_1, \ldots v_k\}$, $E = \{v_0 v_1, v_1 v_2, \ldots v_{k-1} v_k\}$, where the v_i are all distinct.
- The vertices v_0 and v_k are **linked** by P and are called its **ends**, the vertices $v_1, \ldots v_{k-1}$ are the **inner** vertices of P.
- A path P can often be identified by its natural sequence of its vertices, i.e. $P = v_0 v_1 \ldots v_k$ and called a path **from** v_0 **to** v_k (or **between** v_0 *and* v_k).
- Given sets A, B of vertices, we call $P = v_0 v_1 \ldots v_k$ an $A - B$ **path** if $V(P) \cap A = \{v_0\}$ and $V(P) \cap B = \{v_k\}$.
- We write $a - B$ path rather than $\{a\} - B$, etc.
- If $P = v_0 \ldots v_{k-1}$ is a path and $k \geq 3$, then the graph $C := P + v_{k-1} v_0$ is called a **cycle**.

Graph Subpath [Die05, BM08, Wil98]

- For $P = v_0 v_1 \dots v_k$ and $0 \leq i \leq j \leq k$ we write

$$Pv_i := v_0 \dots v_i, \text{ and} \tag{1}$$
$$v_i P := v_i \dots v_k, \text{ and} \tag{2}$$
$$v_i P v_j := v_i \dots v_j \tag{3}$$

and

$$\mathring{P} := v_1 \dots v_{k-1}, \text{ and} \tag{4}$$
$$P\mathring{v}_i := v_0 \dots v_{i-1}, \text{ and} \tag{5}$$
$$\mathring{v}_i P := v_{i+1} \dots v_k, \text{ and} \tag{6}$$
$$\mathring{v}_i P \mathring{v}_j := v_{i+1} \dots v_{j-1} \tag{7}$$

for the appropriate subpaths of P.

- A **concatenation** of three paths $Px \cup xQy \cup yR$ is denoted as $PxQyR$

Graph Walk [Die05, BM08, Wil98]

- A **walk** in a graph G is a sequence $W := v_0 e_1 v_1 \ldots v_{\ell-1} e_\ell v_\ell$, whose terms are alternately vertices and edges of G, such that v_{i-1} and v_i are the ends of e_i, $1 \le i \le \ell$.
- If $v_0 = x$ and $v_\ell = y$, we say that W **connects** x to y and refer to W as an xy-**walk**.
- The vertices x and y are called the **ends** of the walk, x being its **initial vertex** and y its **terminal vertex**, the vertices $v_1, \ldots, v_{\ell-1}$ are its **internal vertices**.
- The integer ℓ (the number of edge terms) is the **length** of W.
- An x-**walk** is a walk with initial vertex x.
- If there is an xy-walk in a graph G, then is also an xy-path.
- The length of a shortest such xy-path is called the **distance** between x and y and denoted $d_G(x, y)$.
- The greatest distance between any two vertices in G is called the **diameter of** G, denoted by $diam(G) = \max_{y,v} d_G(u, v)$.

Graph Component [Die05, BM08, Wil98]

- A non-empty graph G is called **connected** if any two of its vertices are linked by a path in G, otherwise the graph is **disconnected**.
- If $U \subseteq V(G)$ and $G[U]$ is connected, we call U itself connected (in G).
- A maximal connected sugraph of G is called a **component** of G.

Graph Separator [Die05, BM08, Wil98]

- If $A, B \subseteq V$ and $X \subseteq V \cup E$ are such that every $A - B$ path in G contains a vertex or an edge from X, we say that X **separates** the sets A a B in G.
- X **separates** G if $G - X$ is disconnected, that is, if X separates in G some two vertices that are not in X.
- A separating set of vertices is a **separator**.
- A vertex which separates two other vertices of the same component is a **cutvertex**, and an edge separating its ends is a **bridge**.
- The unordered pair $\{A, B\}$ is a **separation** of G if $A \cup B = V$ and G has no edge between $A \setminus B$ and $B \setminus A$.
- The number $|A \cap B|$ is the **order** of the separation $\{A, B\}$.

Graph Block [Die05, BM08, Wil98]

- G is k-**connected** (for $k \in \mathbb{N}$) if $|G| > k$ and $G - X$ is connected for every set $X \subseteq V$ with $|X| < k$.
- A maximal connected subgraph without a cutvertex is called a **block**.
- Thus, every block of a graph G is either a maximal 2-connected subgraph, or a bridge (with its ends), or an isolated vertex.
- By their maximality, different blocks of G overlap in at most one vertex, which is then a cutvertex of G.
- Every edge of G lies in a unique block, and G is the union of its blocks.
- Let A denote the set of cutvertices of G, and $\mathcal{B}$ is set of its blocks.
- A bipartite graph on $A \cup \mathcal{B}$ formed by the edges aB with $a \in A \cap B$ and $B \in \mathcal{B}$ is called a **block graph** of G.

Graph Tree [Die05, BM08, Wil98]

- An **acyclic graph** is a graph that does not contain any cycle.
- An acyclic graph is also called a **forest**.
- A connected forest is called a **tree**.
- The vertices of degree 1 in a tree are its **leaves**.
- One vertex of a tree can be selected as special; such a vertex is then called the **root** of this tree.
- A tree T with a fixed root r is a **rooted tree**.
- A **spanning tree** of a graph G is a minimal connected spanning subgraph $T \subset G$
  - by the equivalence of (i) and (iii) of Theorem 4.

Proposition 3 (Proposition 3.1.2 [Die05], p.56)

The block graph of a connected graph is a tree.

Tree Properties I

Theorem 4 (Theorem 1.5.1 [Die05], p.14)

The following assertions are equivalent for a graph T:

1. T *is a tree;*
2. *Any two vertices of* T *are linked by a unique path in* T*;*
3. T *is minimally connected, i.e.* T *is connected but* $T - e$ *is disconnected for every edge* $e \in T$*;*
4. T *is maximally acyclic, i.e.* T *contains no cycle but* $T + uv$ *does, for any two non-adjacent vertices* $u, v \in T$*.*

Corollary 1 (Corollary 1.5.3 [Die05], p.14)

A connected graph with N *vertices is a tree if and only if it has* $N - 1$ *edges.*

Tree Properties II

Corollary 2 (Corollary 1.5.2 [Die05], p.14)

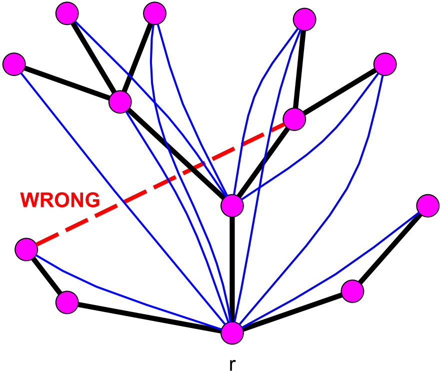
The vertices of a tree can always be enumerated, say as $v_1, \ldots, v_N$, so that every v_i with $i \geq 2$ has a unique neighbor in $\{v_1, \ldots, v_{i-1}\}$.

Tree Order [Die05]

- We write uTv for the unique path in a tree T between two vertices u, v
  - with regard to (ii) of Theorem 4.
- The **tree-order** associated with T and its root r defines a partial ordering on $V(T)$ as $u \leq v$ for $u \in rTv$.
- If $u < v$ we say that
  - u **lies below** v in T,
  - $\lceil v \rceil := \{u | u \leq v\}$ is the **down-closure of** v, and
  - $\lfloor u \rfloor := \{v | u \leq v\}$ is the **up-closure of** u.
- The root r is the least element in the tree order.
- The leaves of T are the maximal elements of its tree order.
- The ends of any edge of T are comparable.
- The down-closure of every vertex is a **chain**, a set of pairwise comparable elements.
- The vertices at distance k from r have **height** k and form the **kth level of T**.

Normal Spanning Tree [Die05]

- A rooted tree T contained in a graph G is called **normal in G** if the ends of every T-path in G are comparable in the tree-order of T.
- Normal spanning trees are also called **depth-first search trees**.



Normal Spanning Tree - Properties

Lemma 2 (Lemma 1.5.5 [Die05], p.15)

Let T be a normal tree in G:

1. *Any two vertices $u, v \in T$ are separated in G by the set $\lceil u \rceil \cap \lceil v \rceil$.*
2. *If $S \subseteq V(T) = V(G)$ and S is down-closed, then the components of $G - S$ are spanned by the sets $\lfloor u \rfloor$ with u minimal in $T - S$.*

Proposition 4 (Proposition 1.5.6 [Die05], p.16)

Every connected graph contains a normal spanning tree, with any vertex specified as its root.

Example 2 (Rapid Spanning Tree Protocol (802.1w) by Cisco [Cis17])

- A network protocol that builds a logical loop-free topology.

Digraph [Die05, BM08, Wil98]

- A **directed graph** (or **digraph**) is a pair (V, E) of disjoint sets (of **vertices** and **arcs**) together with two maps $\text{init} : E \to V$ and $\text{ter} : E \to V$ assigning to every arc e an **initial vertex** $\text{init}(e)$ and a **terminal vertex** $\text{ter}(e)$.
- In some references, vertices of directed graphs are called **nodes**.
- The arc e is said to be **directed from** $\text{init}(e)$ **to** $\text{ter}(e)$.
- Both maps $\text{init}(e)$ and $\text{ter}(e)$ are often combined into an **incidence function** ψ_D that associates with each arc of D an ordered pair of vertices of D, $\psi_D(e) = (u, v)$.

Digraph Arc [Die05, BM08, Wil98]

- If a is an arc and $\psi_D(a) = (u, v)$,
  then the vertex u is also referenced as the **tail** of a,
  and the vertex v its **head**;
  they are the two **ends** of a,
  and we also say that u **dominates** v.
- If the orientation of an arc is irrelevant to the discussion,
  we refer to the arc as **edge** of the directed graph.
- If $\text{init}(e) = \text{ter}(e)$, the edge e is called a **loop**.
- Note that a directed graph may have several arcs between the same two vertices u, v.

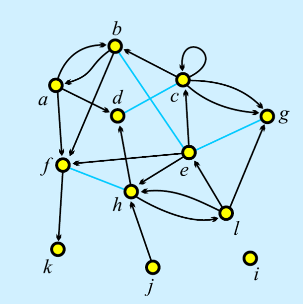
Graph Orientation [BM08, Wil98]

- A directed graph D is an **orientation** of an (undirected) graph G if $V(D) = V(G)$ and $E(D) = E(G)$ and if $\{\text{init}(e), \text{ter}(e)\} = \{u, v\}$ for every $e = uv \in G$.
- Sometimes, it is necessary to distinguish an oriented version of a given graph from its (undirected) graph.
- We denote the (undirected) version as $G = (V, E) = \overline{G}$ and the related graph orientation by $\vec{G} = (V, \vec{E})$ where each oriented edge $\vec{e} = (u, v) \in \vec{G}$ is mapped to the edge $e = \{u, v\} \in G$.
- We say that $\overline{G} = G(\vec{G})$ is the **underlying graph** of $\vec{G}$ [Wil98, BM08].
- A digraph D is **connected** if it cannot be expressed as the union of two digraphs, i.e. the underlaying graph of D is a connected graph.

Digraph Degree [Die05, BM08, Wil98]

- The **degree** of a vertex v in a digraph D is simply the degree of v in the underlying graph $G(D)$ of D.
- The **indegree** $d_D^-(v)$ of a vertex $v \in D$ is the number of arcs with head v,
- the **outdegree** $d_D^+(v)$ of a vertex $v \in D$ is the number of arcs with tail v.
- A vertex of indegree zero is called a **source**, one of outdegree zero a **sink**.

Vertex Degree [Weh13]



- **Degree** of vertex i,
  $deg(i) = d_i = k_i = \sum_{j=1}^{N} A_{ij}$
  $=$ the number of lines with i as end-vertex,
  (end-vertex is both initial and terminal)
- **Indegree** of vertex i, $indeg(i), deg^{+}(i)$
  $= k_i^{\text{in}} = \sum_{j=1}^{N} A_{ij}$ the number of lines with v as terminal vertex
- **Outdegree** of vertex j, $outdeg(j), deg^{-}(j)$
  $= k_j^{\text{out}} = \sum_{i=1}^{N} A_{ij}$ the number of lines with j as initial vertex.

Example 3

$N = 12,\ M = 23,\ deg^{+}(e) = 3,\ deg^{-}(e) = 5,\ deg(e) = 6$

$$\sum_{v \in \mathcal{V}} deg^{+}(v) = \sum_{v \in \mathcal{V}} deg^{-}(v) = |\mathcal{A}| + 2|\mathcal{E}|$$

Digraph Walk [Die05, BM08, Wil98]

- A **directed walk** in a digraph D is an alternating sequence of vertices and arcs $W := (v_0, a_1, v_1, \ldots, v_{\ell-1}, a_\ell, v_\ell)$ such that v_{i-1} and v_i are the tail and head of a_i, respectively, $1 \leq i \leq \ell$.
- If x and y are the initial and terminal vertices of W, we refer to W as a directed (x, y)**-walk**.
- A **directed path** or **directed cycle** is an orientation of a path or cycle in which each vertex dominates its successor in the sequence.
- We say that a vertex y is **reachable** from a vertex x if there is a directed (x, y)-path.

Digraph Strong Connectivity [Die05, BM08, Wil98]

- In a digraph D, two vertices x and y are **strongly connected** if there is a directed (x, y)-walk and also a directed (y, x)-walk.
- Strong connection is an equivalence relation on the vertex set of a digraph.
- The subdigraphs of D induced by the equivalence classes with respect to this relation are called the **strong components** of D.
- The **condensation** $C(D)$ of a digraph D is the digraph
  whose vertices correspond to the strong components of D,
  two vertices of $C(D)$ being linked by an arc
  if and only if there is an arc in D linking the corresponding strong components and with the same orientation.
- The condensation of any digraph is acyclic.

References I

[Bei95] Boris Beizer. *Black-Box Testing, Techniques for Functional Testing of Software and Systems*. John Wiley & Sons, Inc., New York, 1995.

[Blo14] David A. Bloom. Matula thoughts december 5, 2014, 2014.

[BM08] J.A. Bondy and U.S.R. Murty. *Graph Theory*. Springer, 2008.

[Cis17] Cisco. Understanding rapid spanning tree protocol (802.1w), 2017.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[Die05] Reinhard Diestel. *Graph Theory*. Springer, 2005.

[EK10] David Easley and Jon Kleinberg. *Networks, Crowds, and Markets. Reasoning About a Highly Connected World*. Cambridge University Press, July 2010.

[Erc15] Kayhan Erciyes. *Complex Networks, An Algorithmic Perspective*. CRC Press, 2015.

[FMS] FMS. Social network analysis (SNA) diagram, al qaeda terrorist network, accessed 28.1.2014.

[Fre14] Fremantle. Celebrating a soy-free easter with amedei chocolate, accessed 28.1.2014. `http://infonolan.hubpages.com/hub/Celebrating-a-Soy-Free-Easter-with-Amedei-Chocolate`, 2014.

[HLDS13] Dan Halgin, Joe Labianca, Rich DeJordy, and Maxim Sytch. Introduction to social network analysis. `http://www.danhalgin.com/slides`, August 2013.

[Mar17] Radek Marik. *Complex Networks & Their Applications V: Proceedings of the 5th International Workshop on Complex Networks and their Applications (COMPLEX NETWORKS 2016)*, chapter Efficient Genealogical Graph Layout, pages 567–578. Springer International Publishing, Cham, 2017.

[New10] M. Newman. *Networks: an introduction*. Oxford University Press, Inc., 2010.

References II

[Opt17] The OPTE project: The internet 2015; accessed 2017.09.17, 2017.

[PASP09] Jose M Peregrin-Alvarez, Chris Sanford, and John Parkinson. The conservation and evolutionary modularity of metabolism. *Genome Biology*, 10(6), June 2009.

[RP13] James Risen and Laura Poitras. N.S.A. gathers data on social connections of U.S. citizens, September 2013.

[Weh13] Stefan Wehrli. Social network analysis, lecture notes, December 2013.

[Wil98] Robin J. Wilson. *Introduction to Graph Theory*. Longman, fourth edition, 1998.