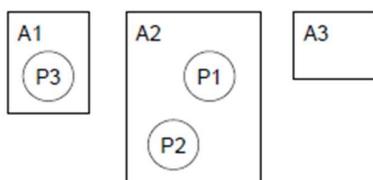


RELAČNÍ DATA

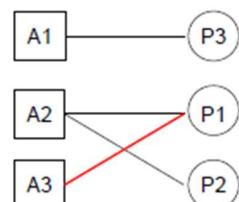
- podobná tabulárním s tím, že ale obsahují **relace (links)** mezi **položky/items (řádky)** pro vykomunikování nějaké hierarchie nebo vztahů v rámci jedné nebo více tabulek
- struktura:
 - **vrcholy** – nodes (položky/items)
 - **hrany** – links
 - **atributy** – nodes a links můžou obsahovat několik různých atributů
- jsou to **abstraktní data**, ale nodes můžou obsahovat souřadnicový systém
 - např. města a lety
 - nemůžeme data obohatovat, protože to nedává smysl

Vizuální kanály

- používají se **seskupovací (grouping) kanály**
 - containment, connection, jiné identity kanály (odstín, blízkost)



Visualization using containment
1:M binary relation

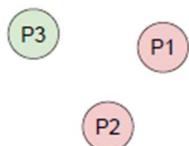


Visualization using connections
1:M, N:M binary relations

Máme definované regiony a v regionech jsou entity.

např. regiony jsou adresy a lidé jsou ty bublinky v těch regionech

Spojení je reprezentována jako spojení mezi entitami.

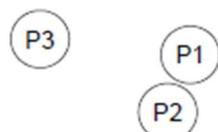


Visualization using similarity (color hue)
1:M binary relation
Not very scalable

Např. lidé a jejich adresy.

Pro každou adresu máme jiný barevný odstín; kolečka – lidé. Obarvíme podle adres.

Není to moc škálovatelné, protože lidé neumí rozeznávat moc odstínů. S výším množstvím adres už nebude fungovat.



Visualization using proximity
1:M binary relation

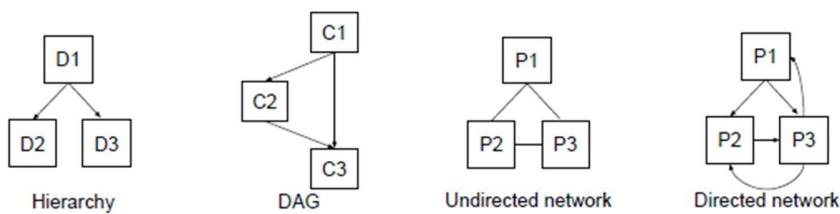
Podobná jako ohrazení. Jednotlivé grafické elementy jsou dány blízko k sobě, pokud lidé bydlí na stejné adrese.

Pouze vizualizace pomocí spojení (connections) je schopna vyjádřit vztah N:M. Žádná z ostatních není schopná toto vyjádřit.

Vizuální enkódování položek

- na geometrie
- mapování na **tvar, barvu, velikost/area**
- pokud používáme vizualizace pomocí spojení/connections, můžeme mapovat atributy těch relací – ty čáry můžeme mapovat na **tloušťku, barvu, typ čárky** – jestli bude čárkovaná, tečkovaná, atd.

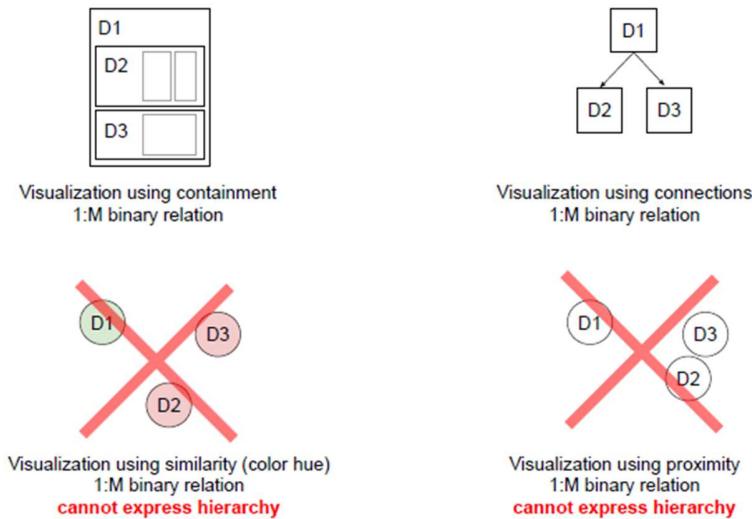
Příklady relačních dat:



- **systém souborů** – hierarchie - adresáře jsou položky/nodes
- **prerekvizity kurzu** – DAG – kurzy jsou nodes, prerekvizity kurzu jsou links
- **sociální síť** – neorientovaný network – lidi jsou items/nodes, friendships jsou links
- **world wide web** – orientovaný network – stránky jsou items/nodes, hyperlinks jsou links

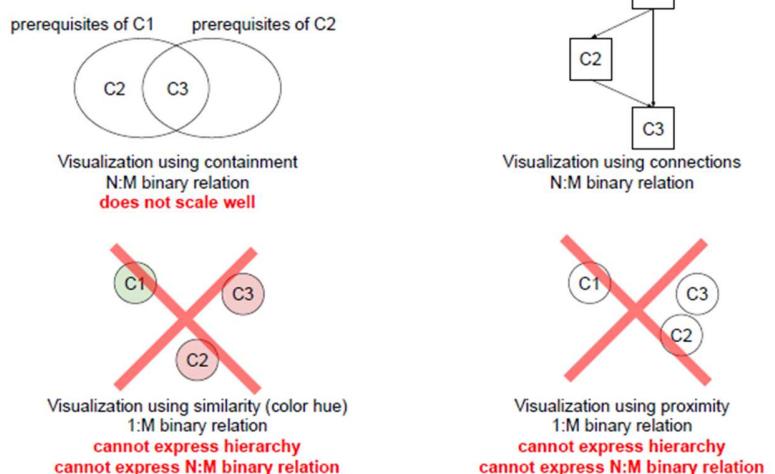
Vizuální enkódování hierarchie

- použití **containment, connections**

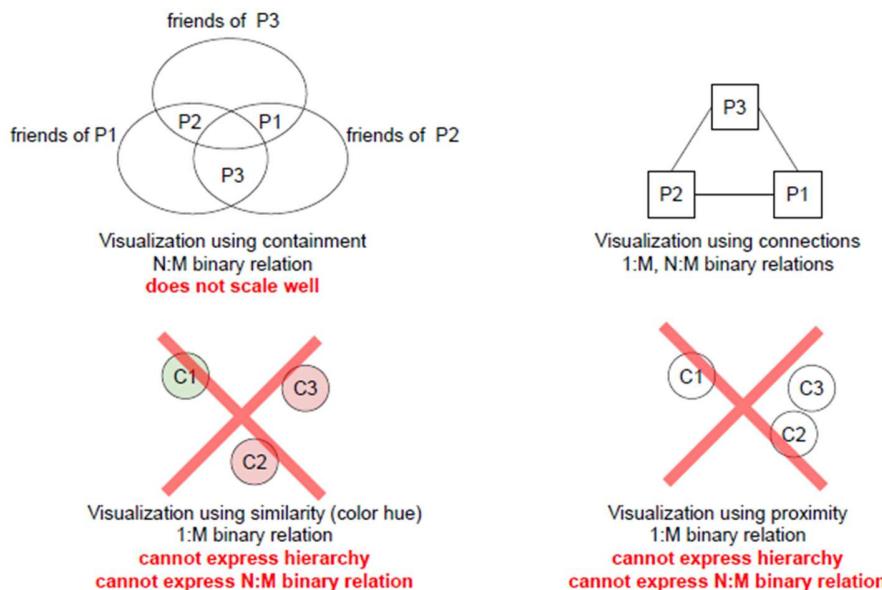


Vizuální enkódování DAG

- čím více se bude protínat elips, tím hůř se v tom vyznáme
 - o máme 6 prerekvizit a už je to v háji
- vhodnější pomocí uzlů a hran



Vizuální enkódování sociální sítě



Tasky ve vizualizaci relačních dat

- Identifikační task – target je jeden atribut
 - o Vizualizace odpovídá na otázky ohledně hodnot individuálních atributů
 - o Jaká je hodnota atributu X pro danou položku
- Identifikační task – target jsou položky
 - o Jaké položky mají hodnotu X daného atributu?
 - o Jaké položky mají hodnotu daného atributu v rangi [X, Y]
 - o Jaké položky jsou incidentní danému linku?
 - o Jaké položky mají více než X incidentních links?
- Identifikační task – target jsou links
 - o Jaké links mají hodnotu X daného atributu?
 - o Jaké links mají hodnotu daného atributu v rangi [X, Y]

- Jaké links jsou incidentní dané položce?
- Identifikační tasky – target, které jsou charakteristické pro network
 - Path: Jaké cesty kratší, než X links jsou mezi dvěma danými položkami?
 - Componenty: kolik jich je a jak velké jsou komponenty v networku?
 - Levels: kolik úrovní tam je? Kolik položek je na určité úrovni?

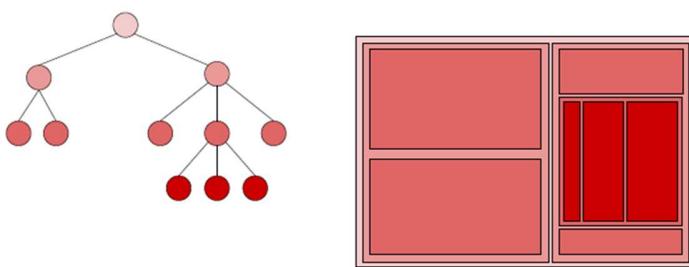
Vizualizace hierarchií s containment

Treemap

- používá containment k enkódování hierarchických relací
- mapuje jeden atribut položky v hierarchii **na velikost/oblast plochy**
 - předpoklad: pro **vnitřní nodes** v hierarchii (ne listy) je jejich hodnota atributů **součet všech hodnot atributů jejich potomků**
- dává overview celé hierarchie
 - Více prostoru je dáné položkám s vyšší hodnotou atributu (co je mapován na velikost/plochu)
- užitečné tam, kde potřebujeme identifikovat položky hierarchie s určitou hodnotou v jednom atributu
- e.g. jaké adresáře jsou velké

Treemap algoritmus

- založený na dělení prostoru
- **začíná se na ose X (dělící osa), startovací node je kořen hierarchie**
- dělíme obdélník reprezentující node v hierarchii do menších obdélníčků, které reprezentují děti rootu
 - velikost obdélníku je podle hodnot v určitých atributech
 - takže když dělíme root, který má potomky 1 a 3, tak rozdělujeme v poměru 1:3 a první obdélník bude o dost menší, atd.
 - pro vnitřní nodes hierarchie, hodnota atributu je pořad sumu těch dětí
- změníme dělící osu
- pokračujeme dělením v DFS nebo BFS pořadí, dokud už nemáme obdélníky, co bychom dělili (dokud není žádný node s potomky)



- ohrazení (containment) můžem ukázat šírkou čáry
- další atributy položek, nebo nějaké vlastnosti stromu můžeme mapovat na barvy obdélníků
- např. hierarchie nodu může být mapována na barvu

Zlepšení treemap

- problém je, že obdélníky bývají úzké a to je hnusný
- trochu změníme algoritmus, když dělíme obdélník reprezentující určitý node:
 - dělící osa je furt X

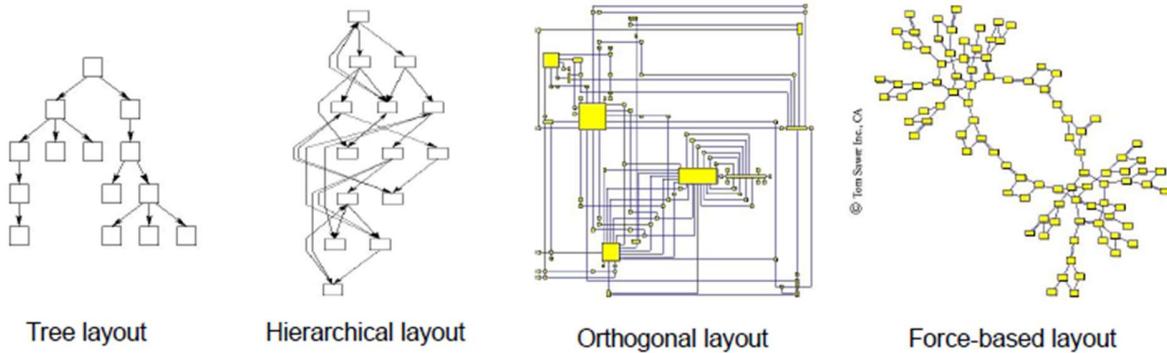
1. rozdělíme potomky do dvou skupin, aby ta **suma byla co nejpodobnější a potomci měli co nejpodobnější velikost**
 2. rozdělíme obdélník na obdélníčky podobných velikostí
 3. změníme dělící osu
 4. rekursivně dělíme nové obdélníky
- Congratulations, máme hezký pěkný obdélníčeky

Vizualizace hierarchie a networků pomocí grafů

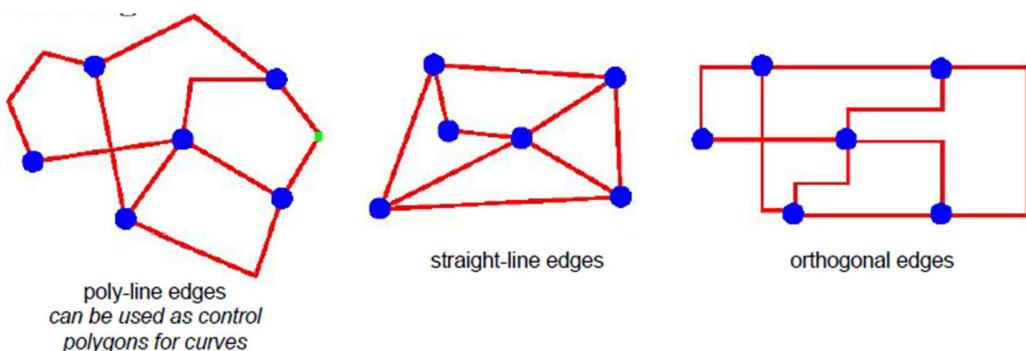
- hierarchie a networks (DAG, orientované, neorientované) mohou být reprezentovány jako grafy
- položky – nodes, links – hrany
- můžeme mapovat **atributy položek** na vlastnosti nodů
 - o tvar, barva, velikost
- můžeme mapovat atributy relací na vlastnosti hran
 - o šířka, barva

Layout grafů

- je jich hodně
- jsou definovány tím,
 - o jak jsou nody distribuovány v prostoru (typicky 2D)
 - o jak jsou nody spojené s hranami

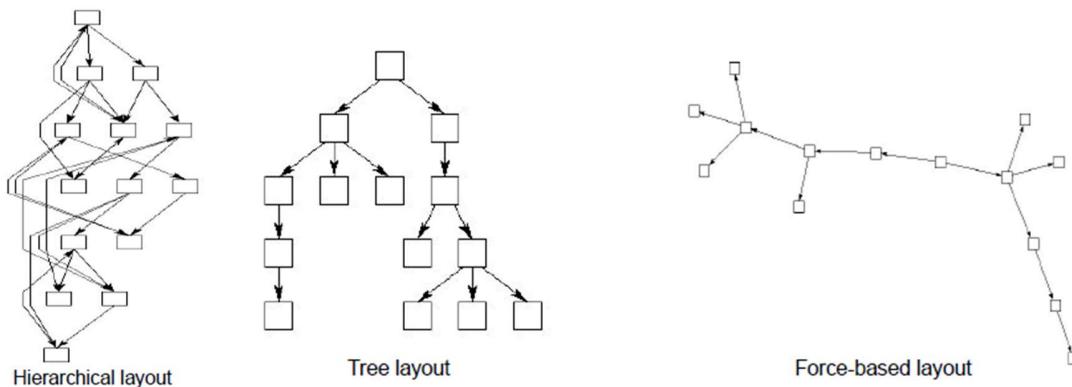


- typy hran:



- různé layouts jsou dobré pro různé tasky
- tree layout a hierarchický layout naznačují hierarchii mnohem líp, než force-based layout – ten je založený na působení nějakých sil mezi uzly

- Hierarchický layout je vhodný pro DAG – lépe komunikuje acyklický graf než force-based layout



E s t e t i k a grafového layoutu

- bez ohledu na layoutu by grafy, ale měli dosahovat určitých estetických kvalit
- myšlenka je taková, že graf pak bude přehledný a strukturu pak uvidíme lépe, než když jich nedosáhneme
- kritéria:

Crossings (křížení hran)

- o typicky bychom chtěli minimalizovat křížení hran, ale ne vždy se nám to podaří
- o to je možné ale jen pro **planární grafy** (nemají žádný křížení hran)

Minimalizace oblasti, co graf zabírá

- o efektivní využití místo na obrazovce
- o minimalizace poměru stran

Maximalizace nejmenšího úhlu

- o maximalizace nejmenšího úhlu mezi dvěma hrany incidentní na tom samém uzlu
- o myšlenka je, že se můžeme snadno splést, pokud máme moc úzké úhly – sledujeme jednu hranu a pak omylem přeskočíme najinou

Délka hran

- o minimalizace sumy délky hran (**total edge length**)
- o minimalizace maximální délky hran (**maximum edge length**)
- o minimalizace variance délky hran (**uniform edge length**)
- o to všechno pomáhá zmenšit grafovou oblast
- o sousední uzly budou více spolu

Počet zlomů

- o minimalizace počtu zlomu hran (**total bends**)
- o minimalizace maximálního počtu zlomu na hraně (**maximum bends**)
- o minimalizace variance počtu zlomu na hraně (**uniform bends**)
- o důležité hlavně pro ortogonální kresby
- o triviálně splněno použitím pouze rovných hran – ale tam je zase křížení hran

Symetrie

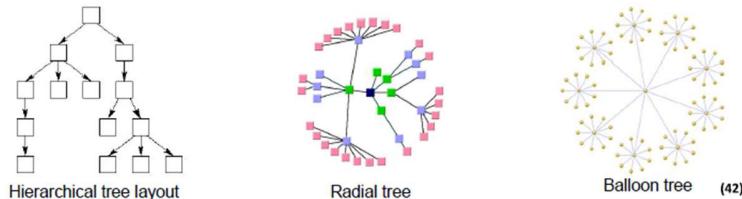
- o Zobrazit symetrii grafu, pokud taková symetrie existuje

- kritéria jsou někdy odporužící – minimalizace délky hran může způsobit zmenšení úhlů
- chceme najít nějakou rovnováhu – optimalizační úloha – výpočetně náročná

- abychom zvětšili rychlosť výpočtu grafového layoutu, často ignorujeme určitá kritéria a používáme approximační strategie a heuristiky (např. gradient descent)
 - o Heuristika se často zaseknou na lokálním minimu

Vizualizace hierarchie s grafy

- všechny layouty komunikují hierarchii grafu
- **Hierarchical tree layout**
 - o pokud je hodně listů a hodně úrovní, tak graf může být i vysoký, i široký
 - o scrolling a zooming bude potřeba
 - o bude těžší sledovat cestu od kořenu k listu, atd.
- **Radial tree**
 - o využívá toho, že **obvod kružnice je větší než šířka obrazovky** – tím se snaží bojovat proti drawbacku hierarchického layoutu, který potřebuje scrollovat a zoomovat
 - o můžeme ukázat jen limitovaný počet úrovní hierarchie
 - o je dobré uzly podle úrovně barvit, abychom o tom neztratili přehled
 - o uživatel může změnit jaký node bude v centru té kružnice
- **Balloon tree**
 - o podstrom je vždycky blízko rootu
 - o uživatel může zoomovat, aby viděl jen jeden podstrom
 - sémantický zoom – např. pouze indikujeme barvou další úrovně grafu



- **Hyperbolický strom**
 - o varianta radial tree
 - o používá **hyperbolický prostor** – máme více místa pro nodes v centru hyperbolického prostoru
 - takže pro uzel uprostřed lze zobrazit nějaké detaile, atd.
 - o technika focus + context
- **Cone tree**
 - o 3D varianta balloon tree
 - o většinou není úplně účinné vizualizovat v těchto případech ve 3D prostoru, ale cone tree je *one special boi*
 - o pomáhá, když uzly mají příliš velké množství dětí – z uzlu „spustíme“ kužel, potomky pověsíme na něj
 - o výhodou je, že protože zobrazujeme potomky na té kuželové kružnici, tak jich tam lze pověsit více
 - o pokud chceme vidět potomky, tak můžeme se stromem můžeme interagovat, rotovat ho, atd. a poměrně dobře prozkoumat ty, které nás zajímají

Sugiyama framework

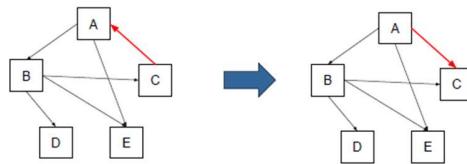
- algoritmus pro hierarchický layout stromů, nebo DAGy
- pokud input graf není DAG, tak ho na DAG překonvertujeme
- pro stromy bude výsledek hierarchický tree layout a DAG – hierarchický layout

- algoritmus lze lehce modifikovat, protože kroky jsou jen naznačené

Kroky algoritmu:

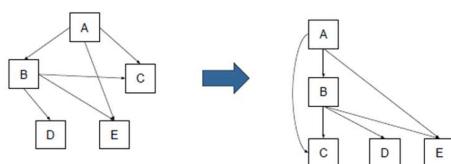
1. Odstraň cykly

- Pokud graf není DAG, vyměň jeden edge v každém cyklu, aby odstranil cykly



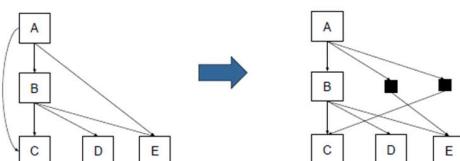
2. Přiřad' úrovňě uzlům

- Každý node je přiřazený úrovni na základě **topologického ocíslování** grafu – **každý potomek je v nižší úrovni než je jeho rodič**



3. Redukuj dlouhé hrany

- Pokud je hrana mezi dvěma uzly v ne-adjacentní úrovni (např. 1 a 4 – **hrana přeskakuje přes úrovne až dolů**), tak tam mezi ně dej virtuální nody (např. 2 a 3)



4. Agreguj virtuální uzly

- Volitelný krok, můžeme agregovat několik virtuálních nodů – **pokud hrana přišla ze samého uzlu**, tak lze dva uzly nahradit jedním, ze kterého vedou dvě hrany do další vrstvy



5. Redukuj křížení hran

- Optimizační krok, redukujeme počet křížení hran tím, že vyměňujeme pozice uzlů

6. Přiřad' souřadnice

- Přiřazujeme souřadnice nodům na základě jejich velikostí a počet a velikosti jejich potomků
- Odendáme virtuální nodes

- Hrany, u kterých jsme měli orientaci v kroku 1 – změníme orientaci zpět

Force-based methods (metody na základě silového pole)

- používá fyzikální analogii – jak grafy kreslí příroda
- bere grafy jako systém objektů, na které působí síla
- V podstatě hledáme systémovou konfigurace s lokální minimální energií
 - **Suma sil na každém objektu je 0 (objekt se nebude už nikam přesouvat)**
 - Takový stav často není globální minimum, ale jen lokální
- Obecně, metody na bázi silového pole mají dvě části:
 - **Model** – silový systém definovaný uzly a hrany
 - **Layout algoritmus** – hledáme rovnovážný stav, lokální minimální energii
- fyzikální analogie je, že každý node v grafu je **nabitá elektrická částice** a každá hrana je **elastická pružina**
 - uzly spojené hranou se přitahují – kvůli té pružině
 - všechny uzly se navzájem taky odpuzují, bez ohledu na to, zda jsou propojené či ne
 - pro každé dva uzly spojené hranou, hledáme **rovnováhu**
 - ve vzdálenosti, kde se odpudivá a přitažlivá síla navzájem vyruší

Force-based layout algoritmus

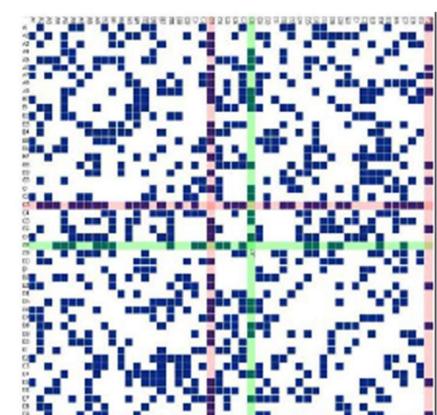
- začneme s pseudonáhodnou (vychází z nějakého seedu, který jsme předali tomu algoritmu – takže bychom měli být schopni tu sekvenci vygenerovat znova, pokud tam zadáme stejný seed) pozicí uzel
- opakujeme
 - pro každý node, spočti celkovou sílu, která na node působí
 - pohni každý uzel ve směru síly, která na něj působí
- dokud součet sil, které působí na všechny uzly není menší než je nějaký threshhold

Problémy:

- vůbec nepokrývají křížení hran
- pro husté grafy (stupeň uzel jsou vysoké, uzly mají mnoho hran) nám to dělá hairball výsledek
 - mnoho uzel působí na mnoho uzel a shluknou se v centru, hodně křížení, struktura je pak nečitelná
- výsledek je často lokální minimum

Matrix-based vizualizace networku

- node-link diagramy neškálují dobře se zvyšujícím se číslem uzel a hran kvůli křížení hran, zastiňování atd.
- jedna reprezentace networku škáluje dobře a je založená na **matici sousednosti networku**
- myšlenka za tím je:
 - ukázat network jako tabulku reprezentující matici sousednosti
 - uzly jsou řádky/sloupce tabulky
 - hrany jsou buňky tabulky
 - např. pokud hrana neexistuje mezi uzly, tak buňku obarvím bílou barvou, pokud mezi nody je hrana, tak modrou
- nevidíme ale strukturu grafu
- existují techniky, které se to snaží tyhle matrix-based vizualizace zlepšit:
 - přehazováním sloupců a řádků, aby lépe zobrazili shluky, ...
 - lepší orientace ve struktuře grafu



- Technika se dá dále rozvíjet
 - **Matice s na podmaticemi**
 - shluky můžeme vyjádřit pomocí matice sousednosti a pak je propojit navzájem pomocí hran

