PV251 Vizualizace

Jaro 2017

Výukový materiál

---

7. přednáška: Stromy, grafy, sítě, texty, dokumenty

Stromy, grafy, sítě

Zatímco dosud jsme se zabývali vizualizačními technikami z pohledu zobrazování hodnot dat a jejich atributů, nyní se podíváme na další důležitou oblast aplikace vizualizace – zprostředkování informace o relaci mezi daty, tedy jakým způsobem spolu datové položky či záznamy souvisí.

Vztahy mezi daty mohou nabývat několika forem:

- Hierarchické relace (část vs. podčást, rodič/potomek,…)
- Propojenost, spojitost (např. jak jsou města spojeny silnicemi, počítače propojeny do sítě, …)
- Odvození (například sekvence kroků nebo fází)
- Sdílená klasifikace
- Podobnost mezi hodnotami
- Podobnost mezi atributy (např. prostorové, časové, …)

Vztahy mohou být jednoduché i komplexní: jednosměrné nebo obousměrné, nevážené nebo vážené, … Kromě toho mohou vztahy mezi daty poskytnout mnohem více informací než jenom ty, co jsou obsaženy přímo v záznamech.

Aplikace zobrazující relační informace mohou být velmi odlišné – od kategorizace biologických druhů přes prozkoumávání archivu dokumentů po monitorování teroristické sítě.

V této přednášce popíšeme řadu technik pro vizualizaci relační informace. Nicméně náš přehled bude pouze zlomkem toho, co v této oblasti existuje, protože vizualizaci stromů a grafů je věnována řada obsáhlých publikací.

Zobrazení hierarchických struktur

Hierarchické struktury (často označované jednoduše termínem „stromy“) jsou jedny z nejběžnějších struktur používaných pro uchování informací o vzájemných vztazích mezi daty. Proto vznikla řada vizualizačních technik pro zobrazení právě těchto struktur a uchovaných vztahů. Tyto techniky a jejich algoritmy můžeme rozdělit do dvou tříd:

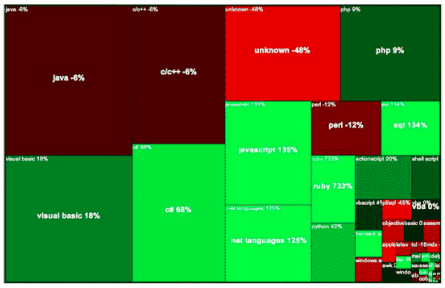
- **Space-filling** – maximálně využívají prostor obrazovky
- **Non-space-filling** – maximální využití prostoru výstupního zařízení není jejich hlavním kritériem

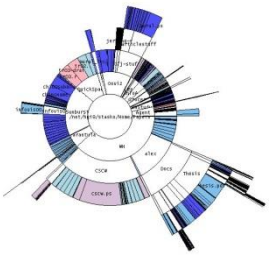
Při seznamování s technikami pro zobrazení hierarchických struktur se tedy zaměříme na tyto dva typy algoritmů.

Space-filling metody

Jak už název napovídá, tyto techniky se snaží o maximální využití prostoru obrazovky. To je často doprovázeno využíváním metody juxtapositioningu pro zobrazení relací. Dva nejběžnější přístupy ke generování tohoto typu hierarchie jsou **obdélníkové** a **radiální rozložení**.

Mezi nejoblíbenější formu space-filling rozložení patří stromové mapy (treemap) a jejich různé varianty. Ty jsou příkladem obdélníkového rozložení prostoru.





Zástupcem druhé třídy, tedy radiálního rozložení, jsou například tzv. „sunburst“ zobrazení (ve tvaru slunce). U nich je kořen hierarchie zobrazen ve středu a pro zobrazení dalších vrstev jsou využívány vnořené prstence.

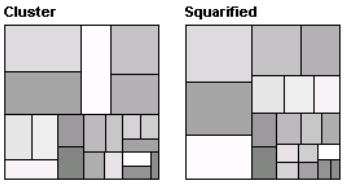
Treemaps

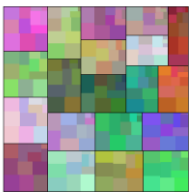
V základní variantě je obdélník rekurzivně dělen do segmentů, přičemž je střídáno horizontální a vertikální dělení obdélníka, které je odvozeno od populace podstromů v dané úrovni. Jako příklad si uvedeme následující pseudokód:

Označení:

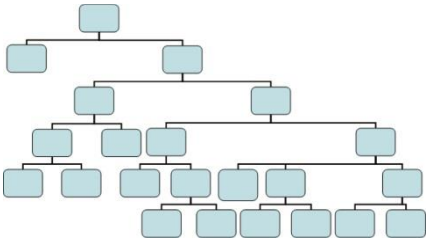
```
Width = šířka obdélníka
Height = výška obdélníka
Node = kořenový uzel stromu
Origin = pozice obdélníka (např. [0, 0])
Orientation = směr řezů – střídání horizontálního a vertikálního řezu

treemap(Node n, Orientation o, Position orig, Width w, Height h)


treemap(Node n, Orientation o, Position orig, Width w, Height h)
   if n je koncový uzel (nemá potomky)
      vykresli_obdélník(orig, w, h);
      return;
   for each potomek uzlu n(child_i)získej počet koncových uzlů v podstromu;
   sečti počet koncových uzlů;
   spočti procentuální poměr koncových uzlů v každém podstromu(percent_i);
   if orientace je horizontální
      for each podstrom
         spočti offset počátku na základě počátku a šířky (offset_i);
          treemap(child_i, vertical, orig + offset_i, w * percent_i, h);
   else
      for each podstrom
         spočti offset počátku na základě počátku a výšky (offset_i);
         treemap(child_i, horizontal, orig + offset_i, w, h * percent_i);
```

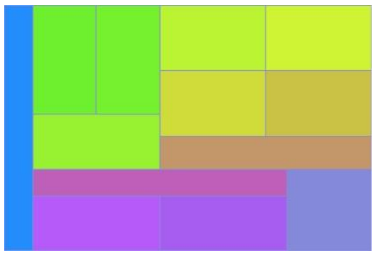
Byla vyvinuta řada variant treemap, jako například squarified treemaps (redukování dlouhých tenkých obdélníků) nebo vnořené (nested) treemapy (pro zdůraznění hierarchické struktury).





Následující obrázek ukazuje příklad vizualizace dané hierarchie ve formě treemapy, která byla vytvořena pomocí algoritmu popsaného předcházejícím pseudokódem.





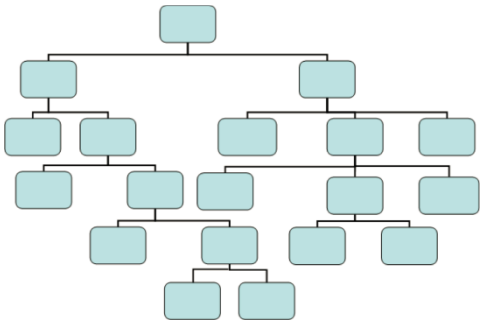
Radiální rozložení

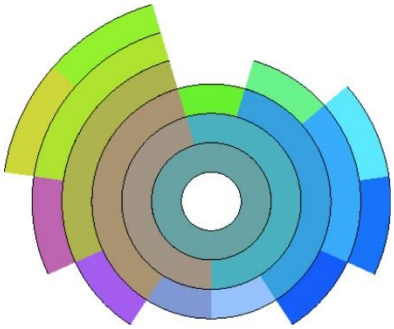
Metody pro radiální rozložení hierarchie, často označované jako „sunburst displays“, mají kořen hierarchie umístěn ve středu radiálního zobrazení a jednotlivé vrstvy hierarchie jsou reprezentovány soustřednými prstenci. Každý prstenec je rozdělen na základě počtu uzlů v dané úrovni hierarchie. Tyto techniky využívají podobnou strategii jako treemapy – počet koncových uzlů daného podstromu určuje velikost obrazovky, která bude alokována pro tento podstrom.

Avšak zatímco treemapy věnují většinu prostoru obrazovky zobrazení koncových uzlů, radiální techniky zobrazují rovněž vnitřní uzly. Pseudokód tohoto algoritmu je následující:

Označení:

```
Start = počáteční úhel uzlu (iniciálně 0)
End = koncový úhel uzlu (iniciálně 360)
Origin = pozice středu radiálního zobrazení (např.[0, 0])
Level = aktuální stupeň hierarchie (iniciálně 0)
Width = tloušťka každého radiálního pásu – založena na maximální hloubce a
velikosti displeje

sunburst(Node n, Start st, End en, Level l)


sunburst(Node n, Start st, End en, Level l)
   if n je koncový uzel (nemá  potomky)
      vykresli_radiální_sekci(Origin, st, en, l * Width,(l+1) * Width);
      return;
   for each potomek n(child_i) získej počet koncových uzlů v podstromu;
   sečti počet koncových uzlů;
   spočti procentuální poměr koncových uzlů v každém podstromu(percent_i);
   for each podstrom
      spočti počáteční/koncový úhel na základě velikosti podstromů, jejich
      uspořádání a rozsahu úhlů;
      sunburst(child_i, st_i, en_i, l+1);
```





Využití barvy u hierarchických technik

U zmíněných i dalších space-filling technik může být barva využita pro zvýraznění mnoha atributů, jako například hodnoty každého uzlu (např. klasifikace) nebo může zesílit zobrazení hierarchických vztahů (např. sourozenci a jejich předchůdci mohou mít podobnou barvu – např. odstín, což je pozorovatelné např. na předchozím obrázku).

Další vlastnosti dat mohou být sděleny například využitím symbolů a dalších označení umístěných do obdélníkových či kruhových segmentů.

Non-space-filling metody

Jednou z nejběžnějších reprezentací hierarchických vztahů jsou spojové (node-link) diagramy. Struktura organizace, rodokmeny či párování do dvojic při turnajích jsou pouze několika příklady běžných aplikací těchto diagramů. Zobrazení těchto stromů je nejvíce ovlivněno dvěma faktory: **stupněm větvení** (např. maximální počet sourozenců pro jednoho rodiče) a **hloubkou** (určena nejvzdálenějším uzlem od kořene).

Různé typy stromů jsou omezeny v jednom nebo obou těchto faktorech, jako například binární stromy nebo stromy povolující pouze tři nebo čtyři vrstvy.

Při návrhu algoritmů pro vykreslování spojového diagramu (nejen stromů) musíme vzít v potaz tři kategorie pravidel, které jsou často protichůdné:

- Konvence vykreslování
- Omezení
- Estetika

Konvence mohou zahrnovat omezující hrany, které mohou být definovány pouze jednou rovnou čarou, sadou přímek, lomených čar či obecných křivek. Další konvencí může být umístění uzlů do fixní mřížky či umístění sourozenců na stejnou vertikální pozici.

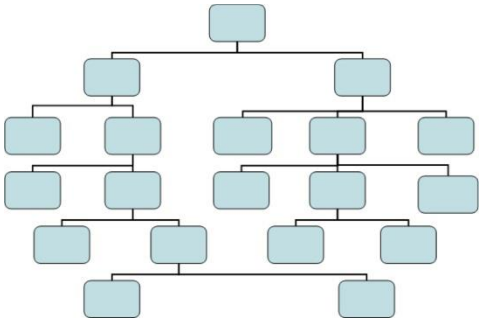
Omezení mohou zahrnovat požadavek na umístění příslušného uzlu do středu obrazovky, požadavek na umístění skupiny uzlů ve vzájemné těsné blízkosti nebo požadavek na orientaci čar.

Estetika má významný vliv na interpretaci daného stromu či obecně grafu. Některá typická estetická pravidla zahrnují:

- Minimalizaci křížení čar
- Udržení „rozumného“ poměru výšky a šířky obrazu
- Minimalizace celkové plochy použité pro vykreslení
- Minimalizace délky hran
- Minimalizace počtu ohybů hran
- Minimalizace počtu různých úhlů nebo zakřivení
- Snaha o udržení symetrie

Pro stromy, zvláště ty vyvážené, je relativně snadné navrhnout algoritmy, které vyhovují většině uvedených pravidel. Příkladem může být procedura vykreslení, kterou nyní uvedeme:

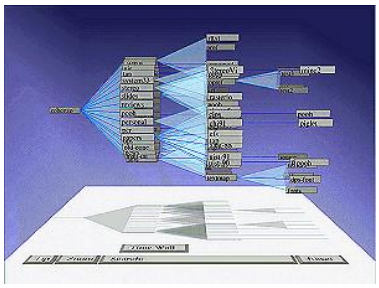
1. Rozdělit plochu pro vykreslení do plátů o stejné výšce – toto rozdělení je odvozeno od hloubky stromu
2. Pro každou úroveň stromu určit, kolik uzlů musí být vykresleno
3. Rozdělit každý plát na obdélníky stejné velikosti – odvozeno od počtu uzlů v dané úrovni
4. Vykreslit každý uzel do středu odpovídajícího obdélníka
5. Vykreslit spojovací čáru vedoucí ze středu spodní hrany každého uzlu do středu horní hrany jeho potomka(ů)



Tento základní algoritmus může být doplněn o řadu optimalizací, které zlepšují využití prostoru obrazovky přesunutím potomků blíže jejich rodičům. Mezi tyto optimalizace patří např.:

- Místo stejnoměrného dělení prostoru a vystředění je každá úroveň rozdělena podle počtu koncových uzlů patřících danému podstromu.
- Rovnoměrné rozložení koncových uzlů v prostoru určeném pro vykreslování a vystředění jejich rodičovských uzlů.
- Přidání dodatečných oddělujících prostor mezi sousední uzly, které nejsou sourozenci. Cílem je zvýraznit vztahy.
- Pokud je to možné, přeorganizujeme podstromy uzlů pro dosažení větší symetrie a vyvážení.
- Umístíme kořenový uzel do středu obrazovky a kolem něj radiálně rozmístíme potomky.

Pro velké stromy je populárním přístupem využití třetí dimenze, který je doplněn o nástroje pro rotaci, translaci a zoomování. Asi nejznámější z těchto technik jsou tzv. **cone trees** (kuželové stromy). V tomto rozložení jsou potomci uzlu rovnoměrně radiálně rozloženi kolem rodiče a potom posunuti kolmo na rovinu.



Dva zásadní parametry jsou poloměr a vzdálenost posunutí – jejich změna ovlivňuje hustotu zobrazení a stupeň vzájemného překrytí.

Minimálně by tyto stromy měly splňovat podmínku, že jejich oddělené větve nejsou umístěny do stejného místa v 3D prostoru. Jednou z metod dosažení této podmínky je určení poloměru pomocí inverzní úměry zanoření hloubky daného uzlu ve stromu. Tímto způsobem jsou uzly blízko kořene blíže k sobě.

Zobrazování libovolných grafů/sítí

Stromy jsou pouze jedním typem více obecné reprezentace vztahů mezi daty, která se nazývá **graf**. Strom je tedy příkladem spojového neváženého acyklického grafu. Je zřejmé, že existuje řada různých reprezentací, jako například grafy s váženými hranami, neorientované grafy, grafy obsahující cykly, nesouvislé grafy atd.

Spíše než popis algoritmů pro tyto jednotlivé typy grafů se budeme zabývat vizualizací grafů, jejichž struktura není předem známa. Nazýváme je **arbitrary graphs**. V našem případě budeme předpokládat, že graf je neorientovaný, i když většina prezentovaných technik může být jednoduše rozšířena na orientované grafy. Zaměříme se na dva odlišné přístupy k vykreslování:

- **Node-link diagrams** (spojové diagramy)
- **Matrix displays** (maticová zobrazení)

Node-link graphs

Force-directed grafy popisované na minulé přednášce využívají analogii pružin pro reprezentaci spojů, kdy jsou pozice uzlů iterativně upravovány, dokud není minimalizována hodnota tzv. *stress* (např. multidimensional scaling). Příklad je uveden na obrázku.

Pro každou dvojici spojených uzlů jsou spočteny dvě síly: f_{ij} – síla odvozena od struny mezi nimi, g_{ij} – elektrický odpor zabraňující uzlům dostat se příliš blízko sebe. Jednoduchým modelem je použití Hookova zákona pro reprezentaci síly f_{ij} a zákon převrácených čtverců (inverse square law) pro reprezentaci g_{ij}. Pokud je jako d(i, j) označena Euklidovská vzdálenost mezi uzly i a j, s_{ij} je délka struny (v klidu) a k_{ij} je napnutí struny, pak x-ová komponenta síly f_{ij} mezi těmito uzly je spočtena jako

$$f_{ij}(x) = k_{ij} * (d(i, j) - s_{ij}) * (x_i - x_j) / d(i, j)$$

Pokud r_{ij} označuje sílu odporu mezi uzly i a j, pak x-ová komponenta síly g_{ij} je spočtena jako:

$$g_{ij}(x) = (r_{ij} / d(i, j)^2) * (x_i - x_j) / d(i, j)$$

Každý krok procesu úpravy pozice spočte součet všech sil pro každý uzel (x, y a z-ové komponenty) a úměrně výsledné síle posune příslušně pozici. Samozřejmě po úpravě pozice musíme všechny síly znovu přepočítat a opět dochází k posunu pozice. Je nutné se vyhnout oscilaci, tudíž sílu upravujeme, abychom konvergovali k bodu, kde jsou síly minimalizovány. Počáteční pozice jsou přiřazeny náhodně. Je pravděpodobné, že daný bod skončí spíše v lokálním než v globálním energetickém minimu. Proto je běžnou praxí spouštět algoritmus několikrát za sebou s různými počátečními konfiguracemi a poté vybereme nejlepší výslednou konfiguraci. „Vhodnost“ umístění bodů v dané vrstvě může být spočtena na základě součtu velikosti sil v dané konfiguraci.

Planární (rovinné) grafy

Techniky pro vykreslování rovinných grafů vychází z předpokladu, že základní graf je **planární (rovinný)**, tedy jeho hrany se neprotínají. Algoritmy pro planární grafy jsou z mnoha důvodů velmi populární. Zaprvé, teorie planárních grafů má dlouhou historii a existuje řada studií věnujících se tomuto tématu. Zadruhé, křížení hran obecně komplikuje interpretaci grafů, proto je výhodné tato překřížení minimalizovat, nejlépe zcela odstranit. A nakonec, planární grafy jsou často řídké: Eulerova věta pro planární grafy říká, že při n vrcholech mohou mít nejvýše 3n-6 hran.

Omezení se na práci s planárními grafy není tak restriktivní, jak by se zdálo. Grafy s křížením hran totiž mohou být převedeny na planární graf následujícím způsobem. Do průsečíků hran umístíme falešné (dummy) uzly, spustíme algoritmus pro planární rozvržení pozic uzlů a poté falešné uzly opět odstraníme.

Navíc budeme předpokládat, že graf je **souvislý**, tedy že z každého uzlu existuje cesta do všech ostatních uzlů. Grafy, které nejsou souvislé, mohou být rozděleny do podgrafů, které jsou vykresleny odděleně jako souvislé.

Podgraf, který je **maximálně spojitý** (všechny uzly jsou vzájemně propojeny), se nazývá **spojitá komponenta** grafu.

Další definice:

- **Ploška** (face) – část roviny uzavřená sadou spojených vrcholů.
- **Sada sousedů** (neighbor set) – sada vrcholů čtených proti směru chodu hodinových ručiček, které sousedí s daným uzlem.
- **Rovinné zanoření** (planar embedding) – třída vykreslení rovinných grafů se stejnou sadou sousedů pro každý vrchol. Rovinný graf může mít exponenciálně mnoho takovýchto zanoření.
- **Cutvertex** – jakýkoliv uzel, který když odstraníme, tak se graf stane nespojitým.
- **Biconnected graph** – graf neobsahující cutvertices.
- **Blok** (block) – maximální biconnected podgraf grafu.
- **Oddělující pár** (separating pair) – dvojice vrcholů, jejichž odstranění způsobí, že biconnected graf se stane nespojitým.
- **Triconnected graph** – graf bez oddělujících párů. Rovinný triconnected graph má jednoznačné zanoření.

Nejdříve musíme určit strategii, podle které rozhodneme, zda je daný graf planární. Existuje několik algoritmů řešících tento problém, avšak ty časově efektivní jsou velmi komplexní a naopak jednoduché algoritmy jsou výpočetně náročné.

Proto si problém trochu zjednodušíme. Řekneme, že graf je planární, pokud všechny jeho spojité komponenty jsou rovněž planární. Podobně můžeme tvrdit, že spojitý graf je planární, pouze pokud všechny biconnected komponenty jsou planární. Proto nám pouze stačí algoritmus, který určí, zda biconnected graf je planární nebo ne.

Algoritmus funguje následovně. Využijeme přístup rozděl a panuj. Pokud náš graf obsahuje cyklus, který v sobě nemá žádný jiný cyklus, který neobsahuje hranu původního cyklu (jinými slovy, pokud odstraníme hrany původního cyklu, nezůstanou žádné další cykly), pak zůstávají cesty (označované jakokusy - *pieces*), které začínají a končí v jednom z vrcholů tohoto cyklu (nazývané *attachments*). Ty vzniknou takto: Dvě hrany e a e' patří do stejného kusu, jestliže existuje cesta začínající v e a končící v e', která neobsahuje vrcholy z cyklu jako svoje „vnitřní" vrcholy.

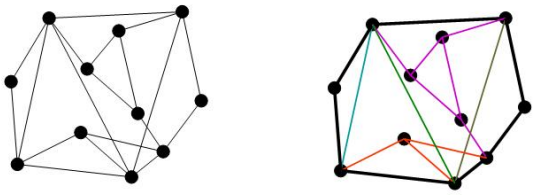
Kus může být:

- Jedna hrana mezi dvěma vrcholy cyklu
- Spojitý graf, který obsahuje alespoň jeden vrchol, který neleží v cyklu

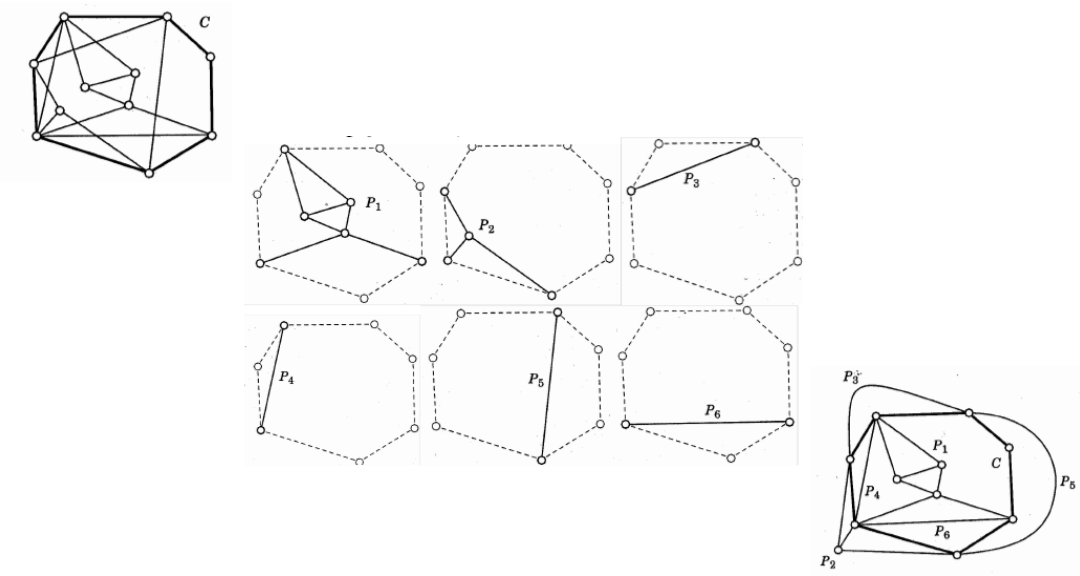
Dva takovéto kusy grafu se „proplétají“ (*interlace*), pokud oba začínají a končí v uzlech cyklu a dva konce jednoho kusu jsou odděleny koncem druhého kusu. Abychom tyto kusy mohli vykreslit planárně, jeden z těchto kusů musí být vykreslen uvnitř cyklu a druhý vně.

Když nyní vytvoříme graf všech kusů, kdy dva protínající se kusy jsou odděleny hranou a takovýto graf je bipartitní (rozdělitelný do dvou sad vrcholů takovým způsobem, že mezi příslušníky dané sady neexistuje žádná hrana), pak původní graf je planární.

Obrázek ukazuje příklad biconnected grafu. Vpravo je cyklus znázorněn černě a barevně je znázorněno pět kusů.



Pokud graf obsahuje po odstranění hran původního cyklu více cyklů, znamená to, že jeden nebo více kusů obsahuje cyklus (viz růžový kus na obrázku). V tomto případě vytvoříme podgraf obsahující tento kus a část původního cyklu spojující koncové body a rekurzivně zavoláme algoritmus na testování planarity grafu.



Nyní si pro úplnost uvedeme ještě pseudokód tohoto algoritmu. Označme si jako **oddělující cyklus** (separating cycle) takový cyklus, který generuje alespoň dva kusy.

```
Mějme biconnected graf G a oddělující cyklus C.
1. Spočteme všechny kusy G vzhledem k C.
2. Pro každý kus P, který není jednoduchou cestou (obsahuje
   cyklus)
      a. Vytvoříme graf G' skládající se z P a C.
      b. Vytvoříme cyklus C' skládající se z cesty skrz P a
         z části C spojující konce.
      c. Aplikujeme tento algoritmus na (G', C'). Pokud je
         výsledek nerovinný, pak je i G nerovinný.
3. Spočítáme proložení grafu I kusů G.
4. Jestliže I není bipartitní, G je nerovinný; jinak je G rovinný.
```

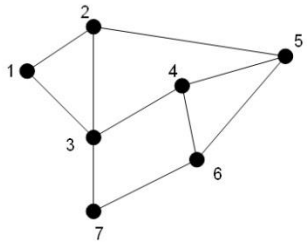
Pokud je graf nerovinný, můžeme z něj udělat rovinný pomocí následující strategie:

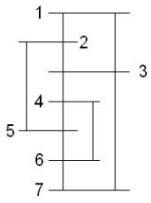
1. Určíme největší rovinný podgraf původního grafu.
2. Zbývající vrcholy umístíme do plošek tak, že minimalizujeme křížení hran.
3. Pro každý průsečík hran rozdělíme odpovídající hrany na dvě části a v průsečíku vytvoříme nový „falešný“ (dummy) vrchol.

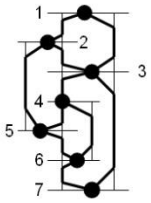
Vykreslení planárních grafů

Poté, co jsme daný graf označili jako planární nebo jsme jej rozšířili, abychom dosáhli planarity, můžeme využít mnoho technik pro generování vykreslení těchto grafů. Jednou z těchto technik je tzv. **visibility approach** (technika viditelnosti), která se skládá ze dvou kroků:

1. **Visibility step** – vytvoříme tzv. visibility representation (reprezentaci viditelnosti). V této reprezentaci je každý vrchol vykreslen jako horizontální úsečkový segment a každá hrana je vykreslena jako vertikální čára spojující odpovídající segmenty vrcholů. Je zřejmé, že pro planární grafy je vždy možné vykreslit takovouto reprezentaci bez protínajících se hran kromě těch, ve kterých se setkávají segmenty vrcholů. Je zřejmé, že existuje mnoho možných uspořádání úsečkových segmentů – jedna ze strategií může být ta, která segmenty uspořádá tak, aby byla minimalizována délka vertikálních spojnic.
2. **Replacement step** – každý segment odpovídající vrcholu se „smrskne“ do jediného bodu a každá vertikální spojnice je nahrazena polyčárou, která se snaží zachovat „vzhled“ původní hrany, jak je to jen možné. Opět pro tento krok existuje řada variant, jako například využití zakřivených čar, použití jednoho vs. více segmentů atd.







Maticová reprezentace grafů

Alternativní vizualizační reprezentací grafu je tzv. **matice sousednosti**, což je mřížka o velikosti NxN, kde N je počet uzlů. Pozice (i, j) reprezentuje přítomnost či nepřítomnost spojnice mezi uzly i a j. Matice sousednosti může být binární maticí nebo hodnota na dané pozici může odpovídat síle nebo váze spojnice mezi odpovídajícími dvěma uzly.

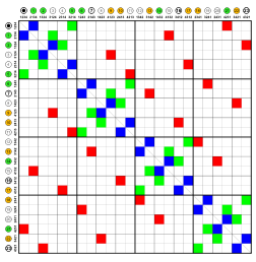
Tato metoda překonává jeden z největších problémů spojových diagramů, což je křížení hran. Není však vhodná pro příliš velké grafy (s tisíci uzlů).

Bertin byl jedním z prvních vědců, kteří zkoumali silné stránky této reprezentace. Zejména se soustředili na různé strategie organizace řádek a sloupců za účelem odhalení zajímavých struktur v grafu. Význam přeskládání je zřejmý z obrázků, kde každá z matic reprezentuje stejný graf o osmi uzlech.

	a	b	c	d	e	f	g	h
a		●	●			●		
b	●			●		●		
c	●				●		●	●
d	●	●				●		
e			●				●	●
f	●	●		●				
g			●		●			●
h			●		●		●	

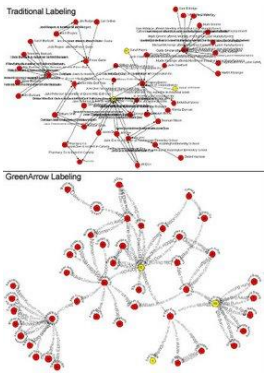
	p	q	r	s	t	u	v	w
p		●	●	●				
q	●		●	●				
r	●	●		●				
s	●	●	●		●			
t				●		●	●	●
u					●		●	●
v					●	●		●
w					●	●	●	

Pro reorganizaci řádek a sloupců existuje řada různých algoritmů. Některé z nich jsou primárně řízeny uživatelem, jiné jsou čistě automatické. Jako pro každý optimalizační problém, nalezení optimálního uspořádání řádek a sloupců je NP-úplný problém (přesněji, žádný polynomiální ani rychlejší algoritmus nebyl nalezen). Proto bylo nutné zavést řadu heuristik, které poskytují kvalitní výsledky, zejména pro některé třídy grafů.



Labeling

Odpovídající označení (labeling) vizualizace je nezbytný pro pochopení, jaká data vlastně zobrazujeme. Například mapa bez popisků by nebyla příliš přínosnou. Podobně graf využívající obarvení jednotlivých prvků reprezentujících data by nebyl pochopitelný, pokud bychom neměli definováno, co barvy představují. Při vykreslování stromů a grafů není vykreslování označení jednoduché zejména proto, že mohou obsahovat velké množství uzlů a rovněž mohou být označeny jednotlivé hrany.



Pokud používáme pouze malé množství odlišných značek, je lepší v tomto případě využít „netextových“ značek, jako jsou barva, velikost či tvar uzlu nebo barva, tloušťka nebo styl čáry reprezentující hranu. Nezabereme tím totiž plochu displeje a obvykle jsou tyto reprezentace interpretovány jednoznačně i v případě velkého množství křížení hran či překrytí uzlů.

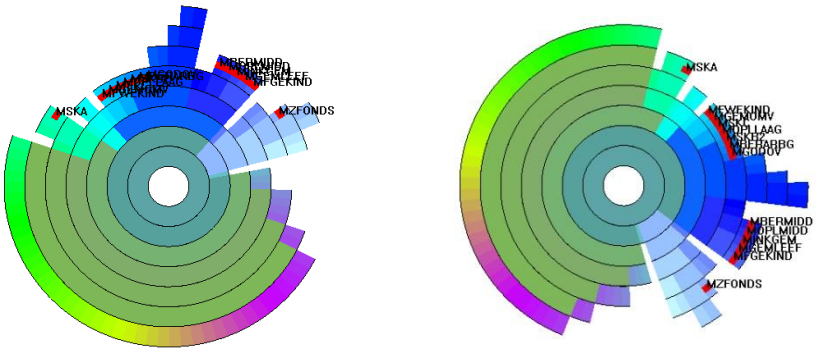
Avšak v případě, kdy počet různých značek překročí pět nebo šest, je pravděpodobnost špatné interpretace poměrně vysoká.

Pro malé grafy je běžnou strategií pro umístění označení uzlů jejich vložení přímo do uzlů – používají se obdélníkové či oválné tvary uzlů, aby bylo možné text umístit přehledně. Abychom se vyhnuli chybnému vnímání jednotlivých uzlů, měla by být velikost uzlů nastavena podle délky nejdelšího označení. V případech, kdy jsou označení příliš dlouhá, je jednou z možností využití zkratek nebo číselných označení s legendou pro vysvětlení.

Podobná strategie může být zvolena pro označování hran, kdy jsou označení umístěna blízko středu hrany. Pro hrany, které jsou převážně vertikální, by měla být označení umístěna vlevo nebo vpravo od hrany. Naopak pro horizontální hrany jsou označení umístěna nad nebo pod hrany. Měla by se však dodržovat konvence, která umístí všechna označení buď vlevo/nahoru nebo vpravo/dolů od hrany. Jinak by mohlo dojít ke špatnému přiřazení označení k hraně.

Při velkém množství různých označení, která mají být zobrazena, nebo při jejich přílišné délce je zřejmé, že souběžné zobrazení všech označení je neefektivní. Pro řešení tohoto problému můžeme využít některou z existujících strategií.

Běžným řešením je zobrazení označení pouze v malém regionu grafu, například v určitém okolí aktuální pozice kurzoru. Pokud je hustota zobrazení příliš velká, jsou vyžadovány různé deformace vizualizace (viz dále), abychom mohli dané sekci grafu věnovat větší oblast prostoru obrazovky. Alternativou k tomuto řešení může být rotování grafů za účelem redukování překryvů označení (viz obrázek).



Dalším zajímavým řešením je zobrazení pouze náhodné podmnožiny označení po krátký časový okamžik a poté zobrazení jiné náhodné podmnožiny atd. Pozadí této myšlenky spočívá v tom, že krátkodobá paměť pozorovatele umožní zapamatovat si větší množství označení než při použití statického zobrazení.

Interakce

I když se technikami interakce s různými vizualizačními prostředími budeme seznamovat později, zmíníme zde několik technik interakce, které jsou nejčastěji spojeny se stromy a grafy. Některé typy interakce, jako například sledování kamerou a zoomování, jsou společné všem typům vizualizace, a proto je zde zmíníme pouze pro úplnost. Další, jako například **focus+context**, mohou být sice aplikovatelné na celou škálu vizualizací, nicméně prvotně byly vyvinuty pro vizualizaci stromů a grafů. Proto se jim zde budeme věnovat podrobněji.

Interakce s virtuální kamerou

Běžné interakce, jako například sledování kamerou (panning), zoomování a rotace, jsou považovány za jednoduché změny aplikované na virtuální kameru, která zabírá určitý segment scény. To umožňuje uživateli inkrementálně pozorovat celou scénu a vytvořit si mentální model objektů ve scéně a jejich vztahů. Operace tohoto typu jsou často řízeny manuálně, ačkoliv existují i automatizované techniky, jako například řízené průlety nad scénou či automatická rotace 3D objektů.

Interakce s prvky grafu

Většina interakcí tohoto typu začíná výběrem (selekcí), kdy je izolována jedna nebo více komponent grafu, na kterých je provedeno zvýraznění, mazání, maskování, posun či

získání detailů. Například grafy s nepřehlednými shluky dat mohou být upraveny takovým způsobem, že vybereme sadu uzlů a táhneme je do oblasti obrazovky, která je méně obsazena uzly, přičemž samozřejmě udržujeme jejich spojení s ostatními uzly.

Podobně můžeme vybrat a posunout či změnit tvar hran za účelem eliminace jejich křížení nebo za účelem zvýšení estetické hodnoty grafu.

Selekce mohou obsahovat jeden objekt, všechny objekty v daném regionu nebo vzdálenosti nebo objekty vyhovující omezením definovaným uživatelem (např. všechny uzly přímo spojené s daným uzlem). Jeden z největších problémů při výběru prvků v grafu se objevuje v hustých regionech, kde jsou jednotlivé prvky tak blízko sebe, že jednoznačný výběr je obtížný nebo dokonce nemožný. To vede k potřebě zavedení dalších typů interakce, jako například zoomování nebo techniky deformace, kterými se budeme zabývat později.

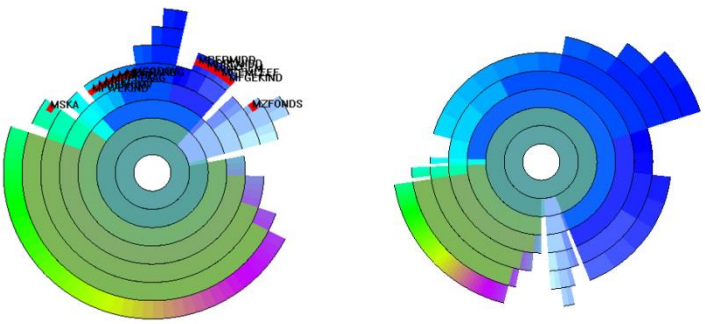
Interakce se strukturou grafu

Rozlišujeme dvě základní třídy interakce, které jsou směrovány přímo na grafové struktury.

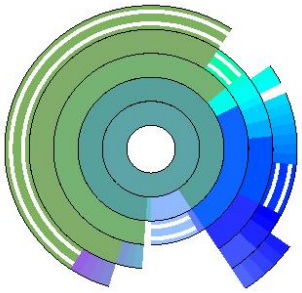
- První třída mění samotnou strukturu grafu. Například přeskládání pořadí větví stromu mohou odhalit vztahy, které nebyly v původním uspořádání viditelné. Přeskládání sloupců a řádek v maticové vizualizaci může opět zvýraznit nové vlastnosti či vztahy mezi daty.

- Druhá třída interakcí spojená se strukturou grafu jsou takzvané **focus+context** techniky, kdy je určitá podmnožina dané struktury (focus) prezentována detailně, zatímco zbytek struktury je zobrazen pouze v obrysech, aby měl uživatel příslušný kontext. Jednou z nejpopulárnějších technik tohoto typu je rybí oko, kdy části grafu spadající do oblasti zájmu jsou zvětšeny pomocí nelineárního škálování, zatímco části mimo oblast zájmu jsou proporčně zmenšeny. Tento typ deformace může být proveden v prostoru obrazovky (odvozeno od pixelů) nebo v prostoru struktury (odvozeno od komponent grafu). Druhý případ je vhodný právě pro vizualizaci grafů, kdy můžeme například zvětšit jednu větev stromu na úkor ostatních větví nebo můžeme zvýraznit všechny hrany vycházející z daného uzlu, abychom lépe odhalili jeho sousední uzly.

Modrá oblast grafu vlevo byla zvýrazněna na obrázku vpravo, což umožňuje jednodušší prozkoumání a interaktivní selekci v této oblasti.



Technika, která může být považována za podobnou k oběma zmíněným třídám interakce je selektivní schovávání či odstraňování sekcí grafu. Například pokud již byla daná větev stromu prozkoumána, uživatel ji může chtít odstranit z obrazovky, aby měl více prostoru pro neprozkoumané regiony. V tomto smyslu to může být považováno za změnu struktury (smazání komponenty) nebo za redukci detailů. Obrázek znázorňuje několik podstromů, které byly „srolovány“ a dvojitý bílý pás uživateli oznamuje, že pod těmito uzly se nachází detailnější informace.



Vizualizace textu a dokumentů

V současné době máme k dispozici obrovské množství zdrojů informací – od knihoven přes archivy po veškeré aplikace běžící na internetu. Pro analýzu těchto dat je nenahraditelným pomocníkem právě vizualizace. Blog, wiki, biliony slov, soubor papírů či digitální knihovnu lze zobrazovat různými způsoby. Protože zvolený způsob vizualizace je závislý na daném úkolu, podíváme se, s jakými úkoly se musíme vypořádat, když pracujeme s textu, dokumenty či objekty na webu.

Jedněmi z nejběžnějších operací nad texty a dokumenty je vyhledávání slova, fráze či tématu. Pokud jsou data alespoň částečně strukturována, můžeme rovněž vyhledávat vztahy mezi slovy, frázemi, tématy či dokumenty. Pro strukturované texty nebo soubory dokumentů je často klíčovým úkolem vyhledávání vzorů či osnovy v textech nebo dokumentech.

Sadu dokumentů definujeme jako **korpus** (soubor dokumentů). Poté pracujeme se objekty uvnitř těchto korpusů. Tyto objekty mohou být slova, věty, odstavce, celé dokumenty nebo dokonce další sady dokumentů. Za objekty můžeme považovat i obrázky a videa. Tyto objekty jsou často považovány za atomické vzhledem k danému úkolu, analýze či vizualizaci.

Texty a dokumenty jsou často strukturovány a obsahují množství atributů a metadat. Například dokumenty mají daný formát a zahrnují metadata o dokumentu (např. autor, datum vytvoření, datum modifikace, komentáře, velikost, …).

Systémy pro dolování informací z dokumentů pracují na bázi dotazování, kdy jako výsledek obdržíme relevantní dokument nebo jeho část, která odpovídá danému dotazu. To však předpokládá předzpracování dokumentu a interpretaci sémantiky textu.

Dále můžeme počítat statistiky o dokumentech, jako například počet slov nebo odstavců, rozložení slov či jejich frekvence. Takto můžeme například rozpoznat autora.

Můžeme rovněž identifikovat vztahy mezi odstavci nebo dokumenty v korpusu. Například se můžeme zeptat: „Které dokumenty se týkají rozšíření chřipky?“ Toto není jednoduchý dotaz, protože nám nestačí pouze hledat výskyt slova „chřipka“.

Podobnost může být definována například i formou citací, spoluautorství, tématu atd.

Definujeme tři stupně reprezentace textu: **lexikální**, **syntaktický** a **sémantický**. Každý stupeň požaduje jistou konverzi nestrukturovaného textu do nějaké formy strukturovaných dat.

Lexikální stupeň

Lexikální stupeň spočívá v transformaci řetězce znaků do sekvence atomických entit, nazývaných **tokens** (znaky). Lexikální analyzéry zpracovávají sekvenci znaků s danou sadou pravidel do nové sekvence znaků (tokens), které mohou být využity pro následné analýzy. Tokeny mohou obsahovat znaky (characters), n-gramy, slova, lexémy (jednotka slovní zásoby), fráze atd., všechny s asociovanými atributy. Pro extrahování tokenů se používá mnoho typů pravidel, přičemž nejběžnější jsou konečné stavové automaty definované regulárními výrazy.

Syntaktický stupeň

Syntaktický stupeň se zabývá identifikací a označováním (anotací) funkcí každého tokenu. Takto přiřazujeme různé značky, jako například pozice věty nebo zda je slovo podstatným jménem, přídavným jménem, doplňkem, spojkou, … Tokeny mohou mít rovněž atributy jako například zda jsou v jednotném nebo množném čísle nebo jejich příbuznost k jiným tokenům. „Bohatší“ tagy obsahují datum, místo, osobu, organizaci či čas. Proces extrahování těchto anotací se nazývá **NER – named entity recognition**. Bohatost a velká rozmanitost jazykových modelů a gramatik poskytují širokou škálu přístupů.

Sémantický stupeň

Sémantický stupeň obsahuje extrakci významu a vztahů mezi poznatky odvozenými ze struktur identifikovaných v syntaktickém stupni. Cílem tohoto stupně je definovat analytickou interpretaci celého textu v daném kontextu nebo i nezávisle na kontextu.

Vector Space Model (VSM)

Vector space model je algebraický model pro reprezentaci textových dokumentů (a obecně jakýchkoliv objektů) ve formě vektorů identifikátorů. Je využíván pro filtrování informace, dolování informace, indexování atd.

Dokumenty jsou reprezentovány vektory. Každá dimenze ve vektoru odpovídá určitému termu (slova, klíčová slova nebo i delší fráze). Pokud se term vyskytuje v dokumentu, je jeho hodnota ve vektoru nenulová.

Počítání tzv. „term vektorů“ je zásadním krokem při vizualizaci dokumentů a korpusů a technikách analýzy.

Ve Vector Space Modelu je jako ***term vector*** objektu zájmu (odstavec, dokument, sada dokumentů) označen vektor, ve kterém každá dimenze reprezentuje váhu daného slova v dokumentu. Dále se typicky pro odstranění šumu odfiltrovávají tzv. „stop words“ (např. the, a) nebo se agregují slova se stejným základem (kořenem).

Následující pseudokód počítá výskyty jednoznačných tokenů, kromě stop words. Jako vstup je brán proud tokenů generovaný z jednoho dokumentu pomocí lexikálního analyzéru.

Proměnná označená jako **terms** obsahuje hašovací tabulku, která mapuje unikátní termy na počet jejich výskytů v dokumentu.

```
Count-Terms(tokenStream)
   1. terms := Ø; /*inicializace terms na prázdnou hašovací tabulku*/
   2. for each token t in tokenStream
   3.   do if t není stop word
   4.      do inkrementuj (nebo inicializuj na 1) terms[t];
   5. return terms;
```

Příklad:

Vzorový text:

There is a great deal of controversy about the safety of genetically engineered foods. Advocates of biotechnology often say that the risks are overblown. ‘‘There have been 25,000 trials of genetically modified crops in the world, now, and not a single incident, or anything dangerous in these releases,’’ said a spokesman for Adventa Holdings, a UK biotech firm. During the 2000 presidential campaign, then-candidate George W. Bush said that ‘‘study after study has shown no evidence of danger.’’ And Clinton Administration Agriculture Secretary Dan Glickman said that ‘‘test after rigorous scientific test’’ had proven the safety of genetically engineered products.

- Předchozí odstavec obsahuje 98 řetězcových tokenů, 74 termů a 48 termů po odstranění stop words
- Příklad term vektoru generovaného pseudokódem:

genetically	said	safety	engineered	study	test	great	deal	controversy	foods
3	3	2	2	2	2	1	1	1	1

VSM – počítání vah

Tento vector space model vyžaduje schéma pro přiřazování vah termům v dokumentu. Existuje řada metod pro přiřazování vah, nejznámější z nich je „**term frequency inverse document frequency**“ – **tf-idf**. Nechť Tf(w) je term frequency neboli počet výskytů slova w v dokumentu a Df(w) je document frequency neboli počet dokumentů, které obsahují slovo w. Nechť dále N je počet dokumentů. Pak definujeme TfIdf(w) pomocí vzorce

$$TfIdf(w) = Tf(w) * \log\left(\frac{N}{Df(w)}\right)$$

Takto je určena relativní důležitost daného slova v dokumentu, která odpovídá našemu intuitivnímu pohledu na důležitost slov. Zajímají nás slova, která se vyskytují často v jednom dokumentu, ale nejsou tak častými v ostatních dokumentech korpusu. Tabulka ukazuje term vektory pro skupinu dokumentů využívajících tf-idf váhy.

id	men	entered	bank	charlotte	missiles	masks	aryan	guns	witnesses	reported	silver	suv	august
seg1.txt	0.239441	0	0.153457	0.195243	0	0.237029	0	0.195243	0.237029	0.140004	0.195243	0.237029	0
seg13.txt	0	0	0	0	0	0	0	0	0	0	0	0	0
seg14.txt	0	0.192197	0	0	0	0	0	0	0	0	0	0	0.172681
seg15.txt	0	0	0	0	0	0	0	0	0	0	0	0	0.149652
seg16.txt	0	0	0	0	0	0	0	0	0	0	0	0	0
seg17.txt	0	0	0	0	0	0	0	0	0	0	0	0	0
seg18.txt	0	0.158432	0	0	0	0	0	0	0	0	0	0	0
seg19.txt	0	0	0	0.197255	0	0	0	0	0	0.141447	0	0	0.155038
seg2.txt	0	0	0	0	0	0	0	0	0	0	0	0	0
seg20.txt	0	0.234323	0	0	0	0	0	0	0	0	0	0	0
seg21.txt	0	0	0	0	0	0	0	0	0	0	0	0	0
seg22.txt	0	0	0	0	0.139629	0	0.127389	0	0	0	0	0	0
seg23.txt	0	0	0	0	0	0	0	0	0	0.180656	0	0	0
seg24.txt	0	0	0	0	0	0	0.117966	0	0	0.117966	0	0	0
seg25.txt	0	0	0	0	0	0	0	0	0	0	0	0	0
seg26.txt	0	0	0	0	0	0	0	0	0	0	0	0	0
seg27.txt	0	0	0.235418	0	0	0	0.214781	0	0	0	0	0	0
seg28.txt	0	0	0	0	0.151753	0	0	0	0	0	0	0	0
seg29.txt	0	0	0	0	0	0	0.129852	0	0	0	0	0	0.142329
seg3.txt	0	0	0	0	0.18432	0	0	0	0	0	0	0	0
seg30.txt	0.078262	0	0	0	0	0	0	0	0	0	0	0	0
seg31.txt	0	0	0.213409	0	0	0	0.194701	0	0	0	0	0	0
seg32.txt	0	0	0	0	0	0	0	0	0	0	0	0	0

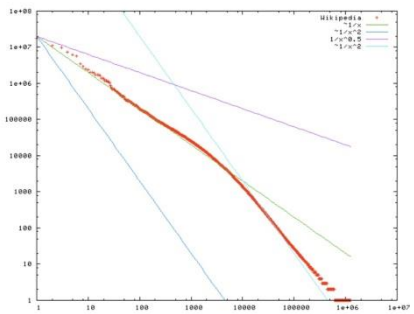
Pseudokód počítá tf-idf vektory každého dokumentu v dané sadě dokumentů. Využívá funkci Count-terms popsanou v předchozím pseudokódu. První část iteruje přes všechny dokumenty a počítá a ukládá si term frekvence a document frekvence. Druhá část počítá tf-idf vektory pro každý dokument a ukládá je do tabulky.

```
Compute-TfIdf(documents)
1    termFrequencies ← ∅  // Looks up term count tables for document names.
2    documentFrequencies← ∅  // Counts the documents in which a term occurs.
3    uniqueTerms← ∅  // The list of all unique terms.
4    for each document d in documents
5              do docName ← Name(d) // Extract the name of the document.
6      tokenStream ← Tokenize(d) // Generate document token stream.
7      terms ← Count-Terms(tokenStream) // Count the term frequencies.
8      termFrequencies[docName] ← terms // Store the term frequencies.
9      for each term t in Keys(terms)
10            do increment (or initialize to 1) documentFrequencies[t]
11           uniqueTerms← uniqueTerms ∪ t
12
13     tfIdfVectorTable ← ∅ // Looks up tf-idf vectors for document names.
14     n ← Length(documents)
15     for each document name docName in Keys(termFrequencies)
16             do tfIdfVector ← create zeroed array of length
Length(uniqueTerms)
17     terms ← termFrequencies[docName]
18      for each term t in keys(terms)
19             do tf ← terms[t]
20           df ← documentFrequencies[t]
21           tfIdf ← tf * log(n/df)
22           tfIdfVector[index of t in uniqueTerms]← tfIdf
23           tfIdfVectorTable[docName] ← tfIdfVector
24    return tfIdfVectorTable
```

Zipfův zákon

Nejčastěji se setkáváme s normálním a uniformním typem rozložení hodnot. Ekonom Vilfredo Pareto tvrdil, že výnos společnosti je inverzně úměrný jejímu postavení – klasický mocninný zákon, jehož výsledkem je známé 80-20 pravidlo, kdy 20% populace vlastní 80% světového bohatství.

Harvardský lingvista George Kingsley Zipf označil distribuci slov v přirozených jazykových korpusech za použití diskrétní mocninné distribuce za Zipfovu distribuci. Zipfův zákon říká, že v typickém dokumentu v přirozeném jazyce je frekvence jakéhokoliv slova inverzně úměrná jeho kategorii (ranku) ve frekvenční tabulce. Vykreslení Zipfovy křivky v logaritmickém měřítku v obou osách má tvar rovné čáry s náklonem -1.



Jednou z okamžitých implikací tohoto zákona je skutečnost, že malý počet slov popisuje většinu klíčových konceptů v malých dokumentech.

Úkoly využívající Vector Space Model

Vector space model doplněný o nějakou vzdálenostní metriku umožňuje provádět celou řadu úkolů. Můžeme použít tf-idf v kombinaci s vector space modelem pro identifikaci zajímavých dokumentů. Můžeme tak odpovídat na dotazy, jako například: Které dokumenty jsou podobné danému dokumentu? Které dokumenty jsou relevantní dané sadě dokumentů? Které dokumenty jsou nejvíce relevantní danému vyhledávacímu dotazu? Výsledkem jsou dokumenty, jejichž term vektory jsou co nejvíce podobné zadanému dokumentu, průměrný term vektor skrz sadu dokumentů a vektor vyhledávacího dotazu.

Dalším nepřímým úkolem může být poskytnutí uživateli významu celého korpusu. Uživatel může hledat vzory nebo klastry a rozložení témat skrz sadu dokumentů. To často zahrnuje vizualizaci korpusu ve 2D rozložení či prezentace uživateli ve formě grafu se spoji mezi dokumenty pro lepší navigaci mezi nimi.

Vizualizační pipeline se dobře mapuje na vizualizaci dokumentů: získáme data (korpus), transformujeme jej na vektory, poté spustíme příslušné algoritmy založené na daném úkolu (podobnost, vyhledávání, klastrování, …) a generujeme vizualizaci.

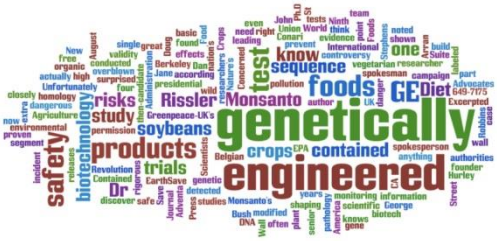
Vizualizace jednotlivých dokumentů

Nyní si ukážeme několik technik pro vizualizaci jednoho dokumentu.

Tag clouds – známé též jako text clouds nebo word clouds, jsou rozložení jednotlivých tokenů, které jsou obarveny, a je jim nastavena velikost podle jejich frekvence v dokumentu.

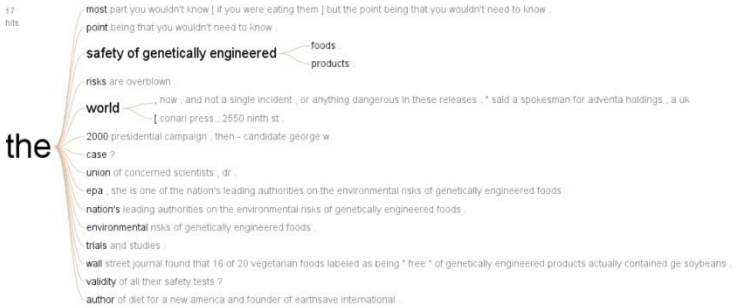
Velikost fontu a odstín jsou úměrné frekvenci výskytu slov v dokumentu.

Text clouds a jejich variace (jako např Wordle – viz obrázek dole), jsou příklady vizualizace, která využívá pouze frekvenci term vektorů a algoritmy pro rozvržení vrstev.



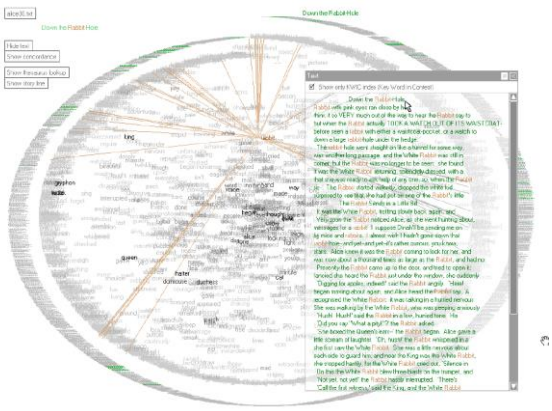
Velikost textu odpovídá frekvenci výskytu slov v dokumentu.

WordTree je vizuální reprezentace jak frekvence termů, tak jejich kontextu (viz obrázek). Velikost slouží k reprezentaci frekvence termu nebo fráze. Kořenem stromu je uživatelem definované slovo nebo fráze a větve reprezentují různé kontexty, ve kterých je slovo nebo fráze použita v dokumentu.

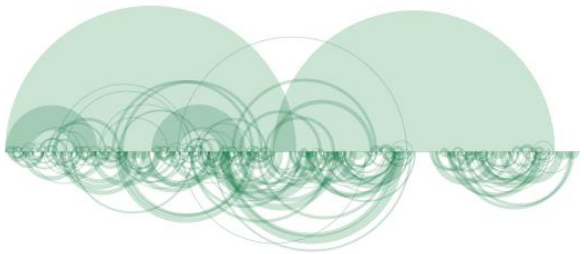


TextArc

Reprezentaci rozložení slov můžeme rozšířit o zobrazení konektivity. Existuje několik způsobů, jak spočítat konektivitu. TextArc je vizuální reprezentace toho, jakým způsobem jsou termy vztaženy k řádkům textu, ve kterých se objevují (viz obrázek). Každé slovo textu je vykresleno okolo elipsy. Podobně jako u Text Clouds, slova s vyšší frekvencí výskytu jsou vykresleny jako větší a světlejší. Slova s vyšší frekvencí jsou vykresleny uvnitř elipsy v závislosti na jejich výskytu na kružnici (podobně jako RadViz technika).



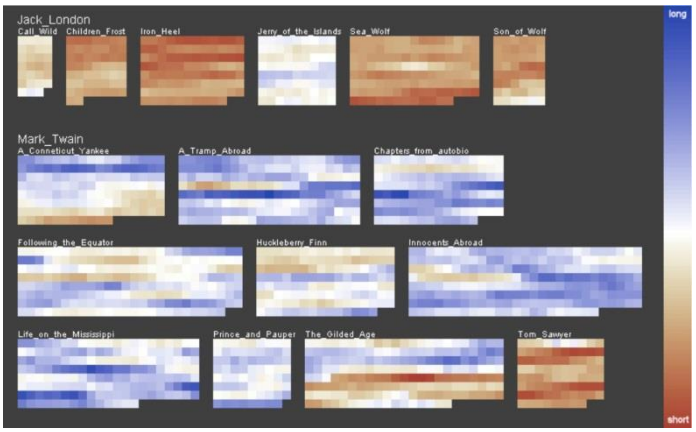
Arc Diagrams je technika zaměřující se na zobrazení opakování v textu nebo jakékoliv sekvenci. Opakující se podsekvence jsou identifikovány a spojeny polokruhovými oblouky. Tloušťka oblouku reprezentuje délku podsekvence, výška oblouku reprezentuje vzdálenost mezi podsekvencemi.



Obrázek vizualizuje Bachův menuet v G dur, kde je vidět klasická struktura menuetu. Obsahuje 2 části, každá se skládá z dlouhé pasáže, která se hraje dvakrát. Části nejsou v přílišném vztahu. Překryv těchto dvou částí ukazuje, že konec první části je stejný jako začátek druhé části.

Literature fingerprinting je metoda zobrazování vlastností charakterizujících daný text. Místo počítání hodnot pouze jedné z vlastností nebo vektoru reprezentujícího text spočítáme sekvenci hodnot vlastností pro celý text a prezentujeme ji uživateli jako charakteristický „otisk“ dokumentu.To umožní uživateli nahlédnout do dokumentu a analyzovat vývoj hodnot skrz celý text. Navíc je strukturní informace o dokumentu využita k vizualizaci dokumentu v různých úrovních rozlišení.

Tato metoda se používá například pro ověřování autorství.



Obrázek ukazuje právě příklad ověřování autorství. Každý pixel reprezentuje blok textu a pixely jsou shlukovány do knížek. Barva je zde mapována na průměrnou délku vět. Je vidět, že otisky knih Jacka Londona a Marka Twaina jsou vizuálně odlišné.

Vizualizace sady dokumentů

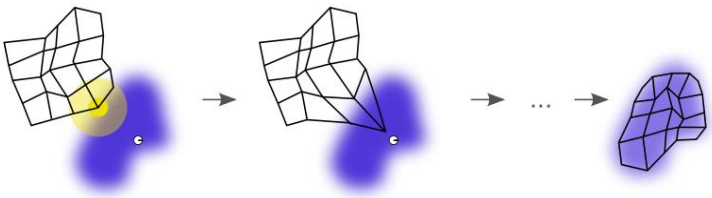
Při vizualizaci sady dokumentů je většinou cílem umístit podobné dokumenty blízko sebe a naopak ty nepodobné co nejdále od sebe. Toto je problém nalezení minimálních a maximálních hodnot – $O(n^2)$. Spočteme podobnost mezi všemi páry dokumentů a tak určíme jejich rozložení. Běžné přístupy jsou graph spring layouts, multidimensional scaling, clustering a self-organizing maps.

Self-organizing mapa (SOM) je vytvářena pomocí algoritmu strojového učení využívajícího kolekci typicky 2D uzlů, do kterých umisťujeme dokumenty. Každý uzel má v sobě asociován vektor stejné dimenzionality, jakou mají vstupní vektory (vektory dokumentů), které se použily na natrénování mapy.

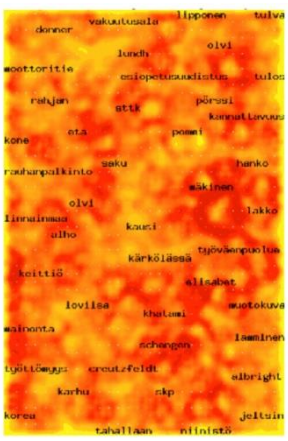
SOM je tedy typem umělé neuronové sítě, která vytváří nízkodimenzionální (typicky 2D) diskrétní reprezentaci vstupního prostoru vzorků, kterou nazýváme mapa. Pro zachování topologie vstupního prostoru je využívána funkce beroucí ohled na sousedy.

Na počátku jsou inicializovány SOM uzly – typicky jsou jim nastaveny náhodné hodnoty. Poté je vybrán náhodný vektor ze sady vstupních vektorů a spočte se jeho vzdálenost od všech ostatních uzlů. Přiřadíme váhy nejbližším uzlům (v daném poloměru) – nejbližší uzel má největší váhu. Při iteraci přes vstupní vektory se postupně zmenšuje poloměr.

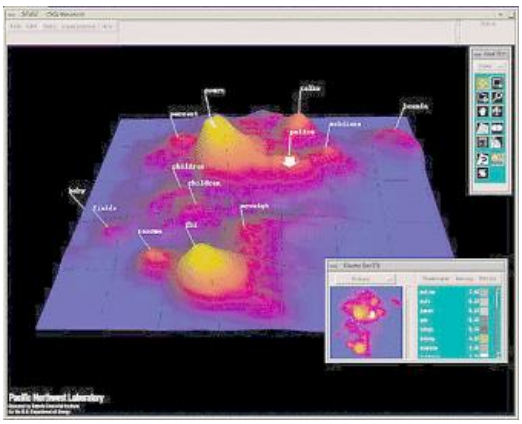
Trénování neuronové sítě s ohledem na sousedy probíhá následovně. Modrý mrak reprezentuje rozložení trénovacích dat a malý bílý kruh označuje aktuální trénovací vzorek. Na počátku jsou SOM uzly rozmístěny libovolně v datovém prostoru. Uzel nejblíže k aktuálnímu trénovacímu vzorku (žlutý) je posunut k tomuto vzorku. Po několika iteracích se dostaneme k výsledku vpravo.



Příklad použití SOM pro textová data je uveden na obrázku, který ukazuje milion dokumentů sesbíraných z 83 diskusních skupin ve Finsku. Labely označují oblasti témat diskusí, barva reprezentuje počet dokumentů – světlejší oblasti znamenají vyšší výskyt.

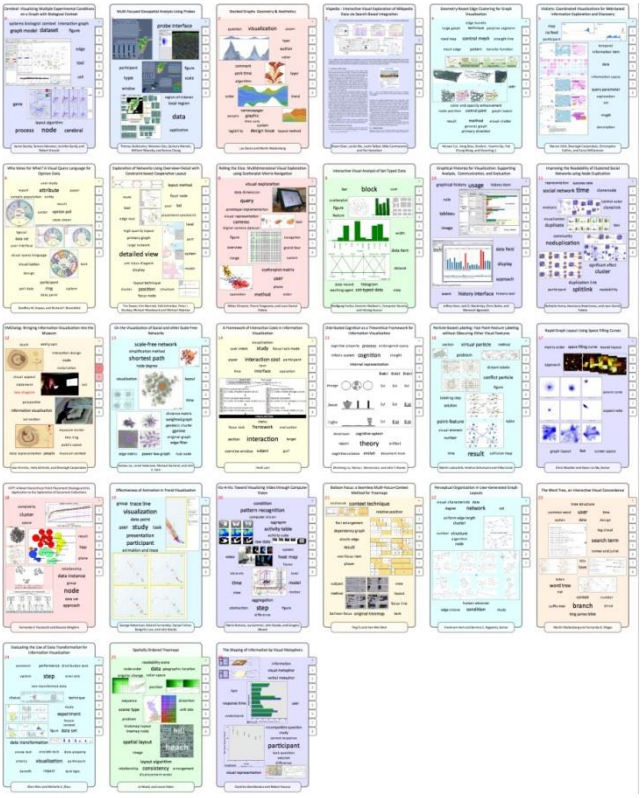


Themescapes jsou v podstatě souhrnné informace o korpusech ve formě 3D terénů (landscapes), kde jsou výška a barva využity pro reprezentaci hustoty podobných dokumentů.



Document cards (záložky dokumentů) jsou kompaktní vizualizací reprezentující klíčovou sémantiku dokumentů ve formě směsi obrázků a klíčových termů, které jsou získány pomocí pokročilých algoritmů pro dolování textu (založeny na automatickém extrahování struktury dokumentu). Obrázky a jejich popisky jsou odvozeny použitím grafických heuristik.

Navíc je využit barevný histogram pro obrázky, které tímto klasifikuje a třídí do jednotlivých tříd (třída 1: fotografie/renderované obrázky, třída 2: diagramy/náčrty/grafy, třída 3: tabulky) a poté je zobrazen alespoň jeden reprezentant každé neprázdné třídy.



Obrázek znázorňuje korpus IEEE Infovis 2008 proceedings, který je reprezentován maticí karet dokumentů. Frekvence termu na každé stránce je zvýrazněna na pravé straně karty (čím červenější, tím vyšší frekvence – např. levý dokument v řadě 3).

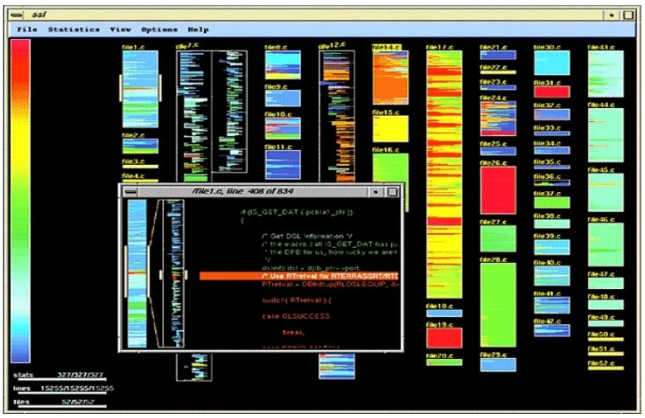
Rozšířené metody vizualizace textu

Nyní se ještě zaměříme na několik vizualizačních technik pro texty, které zahrnují i metadata.

- Software Visualization
- Search Result Visualization
- Temporal Document Collection Visualizations
- Representing Relationships

Software Visualization

Eick a spol. vyvinuli vizualizační nástroj nazvaný SeeSoft, se kterým jsme se již setkali a který vizualizuje statistiky pro každou řádku kódu (např. „stáří“ a počet modifikací, jméno programátora, datum). Na obrázku každý sloupec reprezentuje soubor se zdrojovým kódem, kde výška sloupce odpovídá velikosti souboru. Pokud je soubor větší než je výška obrazovky, pokračuje se dalším sloupcem. Každý řádek ve sloupcích reprezentuje jeden řádek kódu. Pokud by byl zdrojový soubor příliš velký, je každý řádek kódu reprezentován pixelem v řádku ve sloupci. To výrazně zvyšuje počet řádků, které můžeme zobrazit. Barva je mapována na počet volání. Čím je řádek červenější, tím častěji je volán. Naopak řádky volány zřídka jsou modré. Barva může být využita i na jiné parametry, jako například čas poslední modifikace nebo počet modifikací.

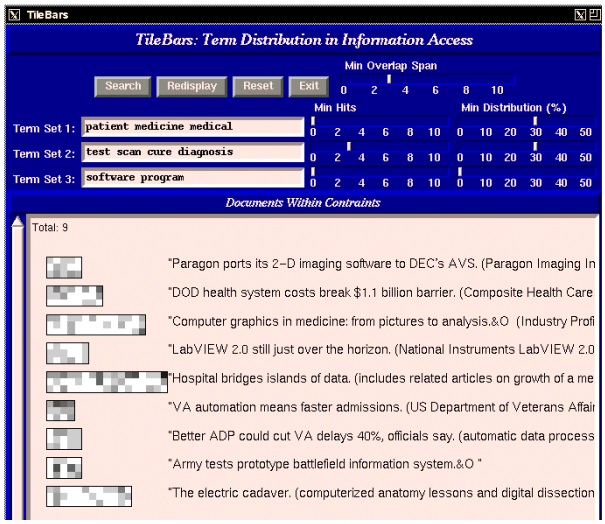


Při velikosti okna 100x100 pixelů je SeeSoft schopen zobrazit více než 50 000 řádků kódu. Screenshot obsahuje 52 souborů s 15 255 řádky kódu.

Search Result Visualization

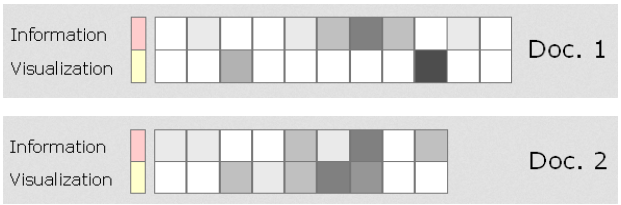
Marti Hearst vyvinul jednoduchou vizualizaci pro výsledky dotazování, která je v principu velmi podobná Keimovým pixel displays. Nazval ji TileBars. Zobrazuje řadu statistik týkajících se termů, jako například frekvenci a rozložení termů, délku dokumentu a další.

Každý dokument výsledné množiny je reprezentován obdélníkem, kde šířka naznačuje relativní délku dokumentu a navrstvené čtverce uvnitř odpovídají segmentům textu. Každý řádek daného zásobníku obsahujícího segmenty reprezentuje sadu dotazovacích termů a odstín čtverečků označuje jejich frekvenci. Tituly a první slova dokumentu se objevují vedle jejich TileBaru.



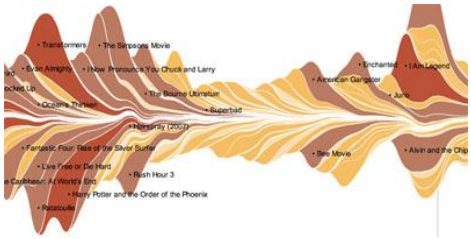
Každý velký obdélník reprezentuje jeden dokument a každý malý čtverec uvnitř dokumentu reprezentuje textový segment. Čím tmavší je čtverec, tím je větší frekvence sady dotazovacích termů. To produkuje velmi kompaktní reprezentaci a poskytuje informace o struktuře dokumentu odrážející relativní délku dokumentu, frekvenci dotazovacích termů a rovněž jejich rozložení.

Příkladem může být vyhledávání slov Information a Visualization ve dvou různých dokumentech. Dokument 1 je delší a jednotlivé čtverečky a jejich barva (odstín) označují frekvenci výskytů daného slova v odpovídající části textu. Je vidět, že v prvním dokumentu se tyto dvě slova nevyskytují v žádné části zároveň. Naopak ve druhém, kratším, dokumentu, se tyto slova zároveň vyskytují hned ve třech částech. Proto pokud hledáme slovní spojení Information Visualization, bude pro nás druhý dokument vhodnější.

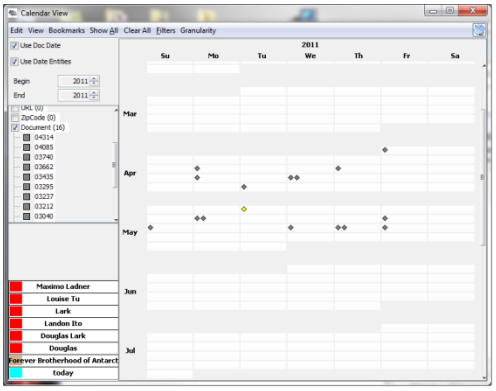


Temporal Document Collection Visualizations

ThemeRiver, nazývaný též stream graph, je využíván pro vizualizaci tematických změn v čase pro sadu dokumentů. Tato vizualizace předpokládá, že se vstupní data mění v čase. Tematické změny jsou vizuálně reprezentovány barevnými horizontálními pásy, jejichž vertikální tloušťka v daném horizontálním místě reprezentuje jejich frekvenci v daném časovém okamžiku.

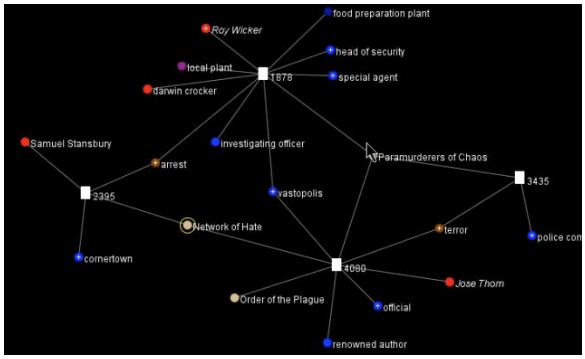


Jigsaw je nástrojem pro vizualizaci a prozkoumávání textových korpusů. Využívá vzhled kalendáře, kdy objekty dokumentů jsou umístěny do kalendáře na základě data, kdy byly dané entity v textu identifikovány. Když uživatel klikne na nějaký dokument, entity vyskytující se v tomto dokumentu se okamžitě zobrazí.



Representing Relationships

Jigsaw rovněž obsahuje reprezentaci v podobě grafu entit. Entity jsou spojeny s dokumenty, ve kterých se vyskytují. Tato reprezentace neukazuje entity celé kolekce dokumentů, ale uživatel může interaktivně měnit oblasti zájmu.



Jigsaw List view je akternativou ke graph view. Pokud uživatel vybere položky, které jej zajímají, list view vykreslí spojnice ukazující jejich vztahy.