UNIVERZITA PALACKÉHO V OLOMOUCI
SOUTĚŽNÍ PRÁCE O CENU DĚKANA 2012
INFORMATIKA, 2. ROČNÍK

Lukáš Havrlant

Vyhledávač založený na FCA

23. dubna 2012

Vedoucí práce: Mgr. Jan Outrata, PhD

Abstrakt

Napovídáním souvisejících dotazů může vyhledávač pomoci uživateli rychleji najít dokumenty, které potřebuje. Práce se zabývá tvorbou vyhledávače s webovým rozhraním, který pracuje nad uzavřenou sadou dokumentů. Po položení dotazu dokáže napovědět konkrétnější, obecnější a podobný dotaz, což je realizováno pomocí formální konceptuální analýzy.

1. Úvod

Současné vyhledávače si uchovávají obsah, nad kterým mají vyhledávat, ve formě indexu, což je struktura v principu podobná indexu v knize. Uživateli stačí vložit do vyhledávacího pole svůj dotaz a vyhledávač z indexu získá potřebná data a zobrazí uživateli výsledek, obvykle ve formě nějakého uspořádaného seznamu odkazů na dokumenty. Pokud není uživatel spokojený s výsledky, musí přeformulovat svůj dotaz tak, aby lépe vystihoval to, co chce najít.

Tento problém je typický pro slova, která mají několik významů. Například pokud ve webovém vyhledávači vyhledáme slovo „jaguár“, tak vyhledávač nemůže vědět, zda chceme hledat auto, zvíře nebo ještě něco jiného. Pokud má přístup k historii hledání daného uživatele, může pomocí ní přizpůsobit výsledky. Ale může také uživateli zobrazit návrhy na nový dotaz, například „jaguár auto“ nebo „jaguár zvíře“, po jejichž vyhledání se výsledky velmi zpřesní.

Otázkou je, jak tyto návrhy získat. Pokud si vyhledávač uchovává historii všech hledání, která uživatelé provádějí, může se je pokusit vytáhnout právě z této historie. Pokud ovšem tato data vyhledávač nemá, nebo jich má málo, musí se použít jiná metoda.

Práce popisuje metodu, jak získat podobné dotazy pomocí formální konceptuální analýzy (anglicky Formal concept analysis, dále jen FCA) pouze na základně znalosti obsahu dokumentů. FCA pracuje s tabulkovými daty, ve kterých hledá nějaké potenciálně zajímavé shluky dat. Tyto shluky pro nás budou představovat množiny dokumentů, které jsou nějakým způsobem podobné a které sdílí nějaká zásadní klíčová slova. Tato klíčová slova poté dále využijeme při generování návrhů na nové dotazy.

2. Information Retrieval

Tato sekce se zabývá budováním samotného vyhledávače, jehož výsledky budou zobrazeny uživateli a také budou použity jako vstup do formální konceptuální analýzy.

Cílem je popsat tvorbu vyhledávače, který bude umět stáhnout z webu požadované dokumenty, zaindexovat je a následně v nich vyhledávat. Dokumenty mohou být buď obyčejné webové stránky nebo složitější soubory, například PDF.

2.1. Jak funguje obecný vyhledávač

Moderní vyhledávače využívají index. Jedná se o podobnou strukturu, kterou můžeme najít v některých, obzvláště odborných, knihách. Protože v klasické tištěné knize nemůžeme nijak „vyhledávat“, dává se na konec knihy seznam nejdůležitějších slov, která kniha obsahuje, spolu s čísly stránek, na kterých se pojem vyskytuje. Takže chce-li uživatel nalézt stránky obsahující slovo „derivace“, podívá se do indexu, kde hned zjistí, že slovo se vyskytuje na té a té stránce.

Podobný princip můžeme použít i ve vyhledávačích. Nebudeme ale vytvářet index z důležitých slov, ale ze všech slov, která se v dokumentech vyskytují. Vytvoříme tak slovníkovou strukturu, kde klíčem bude slovo a hodnotou bude seznam dokumentů, které dané slovo obsahují. Při položení dotazu pak může vyhledávač rychle zjistit jaké dokumenty obsahují dané klíčové slovo prostým nahlédnutím do tohoto slovníku. Myšlenka indexu je popsána v [14], kapitola 1 a 2.

2.2. Vyhledávač CLaSeek

Součástí této diplomové práce je naprogramovaný vyhledávač CLaSeek (Concept Lattice Seeker), který bude v dalších částech textu popsán. CLaSeek je vyhledávač napsaný v Pythonu 3. Pracuje se statickou sadou dokumentů; předpokládá se, že se sada dokumentů bude měnit pouze nárazově jednou za čas. Samotná sada dokumentů, nad kterou má vyhledávač pracovat, nebude příliš velká, řádově stovky dokumentů.

CLaSeek má rozumět logickým operátorům AND, OR a NOT. Po zadání dotazu má vrátit výsledky seřazené podle relevance, kterou spočítá pomocí klasického tf-idf algoritmu. Výstup bude textový ve formátu JSON, se kterým pak mohou pracovat další programy, v tomto případě webové rozhraní, které je napsáno v PHP.

Další částí vyhledávače je hledání souvisejících dotazů. Tato část bude podrobně rozebrána v další kapitole. V této kapitole je dále popsáno, jak CLaSeek buduje index a jak vrací výsledky.

2.3. Předzpracování dokumentů

Při budování indexu máme na vstupu sadu dokumentů a na výstupu strukturu, která reprezentuje index této sady. Během samotného budování indexu procházíme jednotlivé dokumenty a upravujeme je do takové podoby, která se hodí pro uložení.

Na samotném začátku tak musíme upravit jednotlivé textové dokumenty do nějaké kanonické podoby. To budeme dělat postupnými úpravami jednotlivých dokumentů. S každým dokumentem budeme provádět identické operace v identickém pořadí. Jednotlivé operace budou popsány v takovém pořadí, v jakém se aplikují ve vyhledávači. Všechny operace jsou popsány v knihách [2], kapitola 7 a [14], kapitola 2.

Odstranění formátovacích prvků dokumentu jako například HTML značky.

Ponechání písmen Tj. odstranění interpunkce a jiných zbytečných znaků.

Odstranění bílých znaků Odstraníme zdvojené mezery, nové řádky apod.

Převod na malá písmena Velikost písmen nehraje u vyhledávání žádnou roli.

Odstranění stop slov jako například „ke“, „u“, „na“ a podobně.

Převod na stemy Stem je kořen, základ slova. Smyslem je, abychom si v indexu neuchovávali všechny tvary každého slova, ale abychom si od každé slova uchovávali ideálně jen jeden, základní tvar – místo „strom“, „stromy“, „stromu“ tak budeme mít pouze jeden tvar „strom“.

Odstranění diakritiky

2.4. Vyhledání atributů dokumentu

V druhé části vyhledávače, která se zabývá nalezením souvisejících dotazů, budeme potřebovat znát množinu slov, která nejvíce charakterizuje daný dokument. Těmto slovům budeme říkat „atributy dokumentu“.

2.4.1. Jak by měly atributy vypadat

Atributy jsou poměrně běžnou součástí různých vědeckých článků, kde je ale obvykle vyplňuje sám autor pod názvem „klíčová slova“. V případě vyhledávače stojíme před problémem, jak získat atributy z libovolného dokumentu nějakým obecným způsobem.

Jednoduchým způsobem je seřazení všech stemů v daném dokumentu podle jejich četnosti od nejvíce častého. Za atributy pak můžeme vzít ty stemy, které překročí nějakou absolutní hranici („alespoň 10 výskytů v dokumentu“) nebo nějakou relativní hranici („poměr počtu daného stemu ku počtu všech stemů v dokumentu je větší než 0,05“).

Tento postup může být úspěšný v případě, kdy máme pouze jeden dokument. Pokud ale máme sadu dokumentů, můžeme ještě zjistit vztah s dalšími dokumenty. Atribut pro daný dokument by totiž mělo být takové slovo, které je mezi ostatními dokumenty co možná nejvíce unikátní.

Problém vyřešíme tím, že při hledání atributů pro dokument vezmeme v potaz i to, jak často se daná slova vyskytují v ostatních dokumentech. Budeme tak hledat taková slova, která se v daném dokumentu vyskytují co nejčastěji a v ostatních dokumentech co nejméně často.

2.4.2. Algoritmus tf-idf

Tento postup má své jméno, jedná se o tf-idf algoritmus, viz [14], kapitola 6. Ten je rozdělený do několika částí. První je funkce $\mathrm{tf}_{t,d}$, která v základním nastavení vrací počet výskytů slova t v dokumentu d. Dále máme funkci df_t, která vrací počet dokumentů, které obsahují slovo t. Tuto funkci využijeme k tomu, abychom snížili skóre těch slov, která se vyskytují v příliš mnoha dokumentech.

Označme N počet všech dokumentů v naší sadě. Pak vydělením N/df_t získáme koeficient, který značí, jak moc je slovo t unikátní. Pokud se vyskytuje jen v jednom dokumentu, získáme maximální hodnotu N. Při vyšším výskytu slov v dokumentech by tento koeficient klesal příliš rychle, proto ještě použijeme logaritmus. Získáme funkci idf_t

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}.$$

Složením funkcí tf a idf získáme funkci tf-idf$'$ (za chvíli tuto funkci ještě vylepšíme, prozatím si ji označíme s apostrofem) definovanou jako

$$\text{tf-idf}'_{t,d} = \mathrm{tf}_{t,d} \cdot \mathrm{idf}_t.$$

2.4.3. Vylepšení algoritmu tf-idf

Problémem tohoto přístupu je, že příliš preferuje velké dokumenty, které obsahují mnoho slov. Máme-li například učebnici středoškolské matematiky, je pravděpodobné, že bude několikrát, řekněme 50krát, obsahovat slovo „kombinace“. Vedle toho můžeme mít desetistránkový dokument pojednávající čistě o kombinacích, ale slovo „kombinace“ obsahuje pouze 25krát. Podle stávající $\mathrm{tf}_{t,d}$ funkce bude učebnice na klíčové slovo „kombinace“ dvakrát relevantnější než článek přímo se zaměřující na kombinace.

Tento problém můžeme zkusit vyřešit tím, že hodnotu $\mathrm{tf}_{t,d}$ ještě vydělíme celkovým počtem slov v dokumentu. Získáme tak relativní zastoupení slova t mezi všemi slovy v dokumentu. Učebnice z předchozího příkladu pak bude mít mnohem nižší hodnotu $\mathrm{tf}_{t,d}$, protože je mnohonásobně větší než článek. Vzorec funkce tf-idf by pak vypadal takto:

$$\text{tf-idf}''_{t,d} = \frac{\mathrm{tf}_{t,d}}{|d|} \cdot \mathrm{idf}_t,$$

kde $|d|$ je počet slov v dokumentu d.

Problém jsme tím ale ve skutečnosti nevyřešili, jen jsme ho obrátili – už nejsou preferované velké dokumenty, ale malé dokumenty, které dané klíčové slovo obsahují. Pokud bychom zpracovali dokument, jehož obsahem by byla pouze věta „Sázky a kurzy na severskou kombinaci.“, pak by hodnota $\mathrm{tf}_{t,d}$ pro slovo „kombinace“ byla $\frac{1}{4}$ – v dokumentu jsou čtyři slova (slova „a“ a „na“ nepočítáme, jsou to stop slova) a jedno z nich je právě „kombinace“ (po převedení na stem). Pokud by měl mít zmíněný článek, který obsahuje 25krát slovo „kombinace“, alespoň stejnou hodnotu $\mathrm{tf}_{t,d}$, nesměl by mít více než sto slov.

CLaSeek nakonec hodnotu $\mathrm{tf}_{t,d}$ ještě dělí logaritmem počtu všech slov v dokumentu. Hodnota $\mathrm{tf}_{t,d}$ u velkých dokumentů tak bude vydělena větší hodnotou než u malých dokumentů. Zároveň ale tato hodnota, kterou dělíme, nebude růst lineárně, takže desetkrát větší dokument nepotřebuje i desetkrát více klíčových slov, aby dosáhl na stejné hodnocení.

Nyní můžeme napsat finální verzi funkce tf-idf:

$$\text{tf-idf}_{t,d} = \frac{\mathrm{tf}_{t,d}}{\log |d|} \cdot \mathrm{idf}_t,$$

kde $|d|$ je počet slov v dokumentu d. Tato modifikace je vlastní, v [14] jsou popsány jiné možné modifikace tf-idf algoritmu.

2.5. Odpovídání na dotazy

Hlavním účelem vyhledávače je samozřejmě odpovídání na dotazy. Uživatel vloží do rozhraní vyhledávače svůj dotaz, který je dále zpracován vyhledávačem, který vrátí nějaký seznam výsledků. Celý princip fungování boolean dotazů je popsán v [14], kapitola 1.

2.5.1. Syntax dotazu

Dotazem může být libovolný text, přičemž může obsahovat jisté řídicí příkazy, kterým říkáme operátory. CLaSeek podporuje tři operátory: AND, OR a NOT. Musí být psány velkými písmeny, takže pokud chce uživatel použít daná slova jako obyčejný text, stačí použít jejich variantu s malými písmeny.

Pokud nevložíme do textu žádný operátor, použije se implicitně mezi každým slovem AND operátor. Spojíme-li dvě slova operátorem OR, budou se hledat dokumenty, které obsahují alespoň jedno z těchto slov. Operátor NOT vylučuje ty dokumenty, které obsahují slovo, které se nachází za NOT.

Tyto operátory můžeme různě kombinovat a s pomocí závorek můžeme tvořit složitější dotazy, například „(spočetné OR nespočetné) množiny NOT (komplexní OR přirozená)“. Takový dotaz pak převedeme do stromové struktury, kterou nazveme syntaktický strom. Každý uzel obsahuje buď název operátoru a jeden nebo dva potomky, nebo klíčové slovo z dotazu.

2.5.2. Vyhledání odpovídajících dokumentů

Ve chvíli, kdy máme syntaktický strom, se můžeme pustit do hledání dokumentů. Pro účely popisu mechanismu si definujeme funkci R, která bere na vstupu dotaz (syntaktický strom) Q a na výstupu vrací množinu dokumentů, které odpovídají zadanému dotazu. Tato funkce se bude chovat odlišně v závislosti na tom, jak vypadá dotaz Q.

Dále označme $\mathbb{D}$ množinu všech dokumentů a α a β nechť označují nějaké dotazy.

- Je-li $Q = s$, kde s je nějaký stem, pak funkce R vrátí množinu dokumentů, které obsahují daný stem, což vyčteme z indexu. Formálně to můžeme zapsat jako:

$$\mathrm{R}_s = \{d \in \mathbb{D} \,|\, s \in d\}\,.$$

- Je-li dotaz ve tvaru Q = „α AND β“, pak $\mathrm{R}_Q = \mathrm{R}_\alpha \cap \mathrm{R}_\beta$.
- Je-li dotaz ve tvaru Q = „α OR β“, pak $\mathrm{R}_Q = \mathrm{R}_\alpha \cup \mathrm{R}_\beta$.
- Je-li dotaz ve tvaru Q = „NOT α“, pak $\mathrm{R}_Q = \mathbb{D} \setminus \mathrm{R}_\alpha$.

Postupnou aplikací těchto pravidel dostaneme množinu výsledných dokumentů R. V dalším kroku tyto dokumenty seřadíme podle relevance.

2.5.3. Seřazení dokumentů

V současné chvíli máme množinu dokumentů R_Q. Aby byl vyhledávač smysluplný, měl by tyto dokumenty seřadit podle toho, jak relevantní dané dokumenty vzhledem k položenému dotazu jsou. To je obecně nelehký úkol. Ve vyhledávači je pak použit standardní tf-idf algoritmus popsaný v části 2.4.2.

Abychom mohli ohodnotit jednotlivé dokumenty, potřebujeme znát slova, vůči kterým máme dokumenty ohodnocovat. Odstraníme tak z dotazu všechny operátory a dostaneme množinu všech slov S. Vůči těmto slovům budeme dokumenty hodnotit.

K tomu už využijeme tf-idf algoritmus – pro každé slovo z množiny S a pro každý dokument z množiny R_Q spočítáme jeho tf-idf skóre; poté tato skóre sečteme a dokumenty seřadíme sestupně podle tohoto skóre. Skóre $\mathrm{sc}_{d,S}$ dokumentu d tak udává vzorec:

$$\mathrm{sc}_{d,S} = \sum_{s \in S} \text{tf-idf}_{s,d}\,.$$

Toto základní skóre ještě dále upravíme podle toho, zda se klíčová slova z dotazu nevyskytují v důležitých částech dokumentu, konkrétně jde o název, adresu a popisek stránky. Hodnotu stránky ztrojnásobíme, pokud se klíčové slovo vyskytuje v titulku nebo v URL. Hodnotu stránky dále zdvojnásobíme, pokud se klíčové slovo vyskytuje v popisu. Výsledek si označíme jako $\mathrm{rank}_{d,S}$, kde d je dokument, pro který počítáme skóre a S jsou slova z dotazu Q.

Nyní seřadíme dokumenty R_Q do n-tice $\langle d_1, d_2, \ldots, d_n \rangle$, kde

$$n = |\mathrm{R}_Q| \quad \text{a} \quad \bigcup_{i=1}^{n} d_i = \mathrm{R}_Q$$

tak, aby platilo $\mathrm{rank}_{d_1,S} \geq \mathrm{rank}_{d_2,S} \geq \ldots \geq \mathrm{rank}_{d_n,S}$. Tato seřazená n-tice je výstupem algoritmu vyhledávání. Idea a postup řazení dokumentů je popsán v [14], včetně zvýhodňování určitých částí dokumentu.

2.6. Další vymoženosti vyhledávače

Vyhledávač si během budování indexu uchovává informace o stemech a nakonec vytvoří slovník, který funguje jako inverzní stemovací funkce. Vstupem je stem a výstupem je slovo, které se na tento stem zobrazovalo nejčastěji. Tato funkce je použita v dalších částech vyhledávače, abychom uživateli napovídali smysluplnější slova.

CLaSeek dále umí částečně opravovat překlepy. Během budování indexu si uchovává seznam všech slov ze všech dokumentů. Pokud uživatel vloží dotaz, který neodpovídá žádnému dokumentu, zkusí vyhledávač zjistit, zda uživatel neudělal v nějakém slově překlep. Funkce využívá interní funkci Pythonu, viz [1].

3. Hledání souvisejících dokumentů

V této kapitole bude popsána druhá část vyhledávače, která se stará o nalezení souvisejících dotazů. Nejprve uveďme stručný úvod do formální konceptuální analýzy.

3.1. Formální zavedení FCA

Všechny definice a všechny věty z této kapitoly jsou citovány z [3]. Další materiály o FCA: [11], [5], [6], [7].

Definice 1 (Formální kontext). *Formální kontext je trojice $\langle X, Y, I\rangle$, kde X je neprázdná množina objektů, Y je neprázdná množina atributů a I je binární relace mezi X a Y, tj. $I \subseteq X \times Y$.*

Definice 2 (Šipkové operátory). *Pro kontext $\langle X, Y, I\rangle$ definujeme operátory $^{\uparrow} : 2^X \to 2^Y$ a $^{\downarrow} : 2^Y \to 2^X$ tak, že pro každé $A \subseteq X$ a $B \subseteq Y$:*

$$A^{\uparrow} = \{y \in Y \mid \text{pro všechna } x \in A : \langle x, y\rangle \in I\} \quad (1)$$

$$B^{\downarrow} = \{x \in X \mid \text{pro všechna } y \in B : \langle x, y\rangle \in I\} \quad (2)$$

Definice 3 (Formální koncept). *Formální koncept v kontextu $\langle X, Y, I\rangle$ je dvojice $\langle A, B\rangle$, $A \subseteq X$, $B \subseteq Y$ tak, že:*

$$A^{\uparrow} = B \wedge B^{\downarrow} = A.$$

Množině A říkáme „extent“ a množině B „intent“.

Definice 4 (Uspořádání konceptů). *O konceptech $\langle A_1, B_1\rangle, \langle A_2, B_2\rangle$ kontextu $\langle X, Y, I\rangle$ řekneme, že*

$$\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle \quad \text{právě když} \quad A_1 \subseteq A_2 \quad (B_2 \subseteq B_1).$$

Definice 5 (Konceptuální svaz). *Pro kontext $\langle X, Y, I\rangle$ máme definovanou množinu všech formálních konceptů $\mathcal{B}(X, Y, I)$ z $\langle X, Y, I\rangle$. Tedy*

$$\mathcal{B}(X, Y, I) = \left\{\langle A, B\rangle \in 2^X \times 2^Y \mid A^{\uparrow} = B \wedge B^{\downarrow} = A\right\}.$$

Dvojice $\langle \mathcal{B}(X, Y, I), \leq\rangle$ se nazývá konceptuální svaz.

Definice 6 (Supremálně a infimálně hustá množina). *Množinu $K \subseteq V$ nazveme supremálně hustou ve V právě tehdy, když pro všechna $v \in V$ existuje $K' \subseteq K$ takové, že $v = \bigvee K'$. Každý prvek množiny V je tak supremem nějakých prvků z K.*

Duálně pro infimum: když pro všechna $v \in V$ existuje $K' \subseteq K$ takové, že $v = \bigwedge K'$.

Věta 7 (Hlavní věta konceptuálních svazů). *Věta má dvě části:*

1. *Nechť $\mathcal{B}(X, Y, I)$ je kompletní svaz. Pak suprema a infima získáme*

$$\bigwedge_{j\in J} \langle A_j, B_j\rangle = \left\langle \bigcap_{j\in J} A_j, \left(\bigcup_{j\in J} B_j\right)^{\downarrow\uparrow} \right\rangle, \quad (3)$$

$$\bigvee_{j\in J} \langle A_j, B_j\rangle = \left\langle \left(\bigcup_{j\in J} A_j\right)^{\uparrow\downarrow}, \bigcap_{j\in J} B_j \right\rangle. \quad (4)$$

2. *Dále, libovolný úplný svaz $\boldsymbol{V} = \langle V, \leq\rangle$ je isomorfní k $\mathcal{B}(X, Y, I)$ právě tehdy, když existují zobrazení $\gamma : X \to V, \mu : Y \to V$ tak, že*
   - *$\gamma(X)$ je supremálně hustá množina ve V a $\mu(Y)$ je infimálně hustá množina v V,*
   - *$\gamma(x) \leq \mu(y)$ právě tehdy, když $\langle x, y\rangle \in I$.*

□

3.2. Motivace

Pokud uživatel položí vyhledávači nějaký dotaz, vyhledávač odpoví nějakým seznamem dokumentů, které jsou podle něj nejvíce relevantní. Pokud má uživatel štěstí, bude v tomto seznamu dokument, který zrovna potřebuje najít. Pokud bude mít velké štěstí, pak bude tento dokument na předních místech v seznamu.

Pokud ale toto štěstí mít nebude a dokument se mu nepodaří nalézt, musí uživatel nějakým způsobem přeformulovat svůj dotaz tak, aby vyhledávač vrátil jinou sadu výsledků. Obecně může upravit dotaz třemi různými způsoby. Může

1. přidat k dotazu jedno či více slov, díky čemuž obdrží méně výsledků,
2. změnit jedno či více slov, díky čemuž obdrží podobné výsledky,
3. odebrat jedno či více slov, díky čemuž obdrží více výsledků.

V různých situacích se hodí různé postupy. Pokud jsme zadali příliš konkrétní dotaz, na který vyhledávač odpověděl málo dokumenty, bude vhodné odebrat některá klíčová slova dotazu, abychom získali více výsledků. Pokud jsme naopak zadali příliš obecný dotaz, můžeme přidat nějaká klíčová slova, abychom obdrželi méně dokumentů, která ale lépe odpovídají na náš dotaz.

3.3. Hlavní cíl

Hlavním cílem vyhledávače je nacházet zmíněné úpravy dotazu automaticky. Tyto úpravy můžeme hledat například pomocí historie dotazů, pokud danou historii máme. Pokud uživatel položí dotaz „hosting php“, můžeme se podívat do historie vyhledávání a nalézt všechny dotazy, které obsahují alespoň jedno ze slov v dotazu a z této množiny dotazů pak můžeme nějakým postupem vybrat související dotazy pro všechny tři kategorie úprav. Můžeme například zjistit, že z konkrétnějších dotazů je nejvíce hledaný dotaz „hosting php mysql“, z podobných „server php“ a podobně.

My ale použijeme jiný postup. Všechny související dotazy budeme hledat pouze na základě znalostí sady dokumentů. Nebudeme k tomu využívat žádné dodatečné informace, všechno si spočítáme pouze ze samotných dokumentů.

K tomu využijeme formální konceptuální analýzu, která v daných datech vyhledává jisté shluky potenciálně zajímavých dat a zároveň je ukládá do hierarchie, se kterou můžeme dále pracovat. Pro základní představu si můžeme představit výstup FCA jako Hasseův diagram, kde jeden z uzlů – ten musíme nějak najít – představuje aktuální výsledky, které nám vyhledávač zobrazil, „otcové“ tohoto uzlu jsou obecnější dotazy, „synové“ jsou konkrétnější dotazy a „sourozenci“ jsou podobné dotazy.

3.4. Stručný popis funkčnosti celého vyhledávače

Následující seznam ukazuje postup vyhledávače po zadání dotazu a nastiňuje spojení FCA s Information retrieval.

1. Uživatel vloží do vyhledávače dotaz. Vyhledávač odpoví seřazenou sadou výsledků. To zůstává stejné.
2. Pokud se jedná o dotaz s více než jedním klíčovým slovem, provedeme ještě jeden dotaz, kde všechna klíčová slova z dotazu spojíme operátorem OR.
3. Nyní přichází na řadu FCA část vyhledávače. Jako první se vytvoří kontext $\langle X, Y, I \rangle$. Objekty budou dokumenty, které nám vyhledávač vrátil v předchozím bodě. Atributy budou atributy těchto dokumentů plus klíčová slova z dotazu.
4. Relaci I definujeme takto: $\langle x, y \rangle \in I$ právě tehdy, když dokument x obsahuje slovo y.

5. Nyní musíme nalézt koncept dotazu. To je koncept, který odpovídá dotazu od uživatele. Můžeme ho nalézt tak, že vezmeme množinu dokumentů, které vyhledávač vrátil v prvním bodě, a spočítáme koncept, který těmto dokumentům odpovídá.

6. Jednotlivé návrhy poté nalezneme v intentech okolních konceptů. Specializaci nalezneme v dolních sousedech, generalizaci v horních sousedech a podobné dotazy v sousedních konceptech. Pokud například koncept dotazu má intent {limita, funkce} a nějaký dolní soused má intent {limita, funkce, vlastní}, odvodíme z toho specializaci `+vlastní`.

Tato idea není nijak nová a už byla implementována v několika prototypech jako například CREDO, FooCA a SearchSleuth, podrobnější popis těchto vyhledávačů, včetně popisu odlišností, se nachází v kapitole 3.8. Myšlenka vyhledávače, který spolupracuje s FCA je pak popsána například v [4]. Zde popisovaný vyhledávač je velice podobný vyhledávači SearchSleuth, přebírá některé vizuální prvky a FCA část je také téměř stejná. Vyhledávač CLaSeek je vlastní pokus o ověření funkčnosti celé ideje.

Všechny tři předchozí vyhledávače pracují, či pracovaly, nad dynamickou sadou dokumentů, typicky braly výsledky z nějakého webového vyhledávače. Naproti tomu vyhledávač popsaný v této práci pracuje primárně nad statickou sadou dokumentů; na vstupu bere seznam dokumentů, ty stáhne, zaindexuje a pak pracuje nad tímto statickým indexem. Nicméně skrze poskytované API je možné vytvořit vyhledávač, který bude pracovat nad dynamickou sadou dokumentů a bude se chovat prakticky stejně jako například SearchSleuth, viz kapitolu 4.4.2.

3.5. Jak vytvořit kontext

3.5.1. Sestavení kontextu

Označme dotaz, který uživatel položil vyhledávači, jako Q. Výsledný seřazený seznam dokumentů označíme D, přičemž D_1 je dokument na prvním místě, D_2 na druhém atp. Pro účely této kapitoly předpokládejme, že máme dokumenty v D reprezentovány jako množinu všech stemů, které dokument obsahuje.

Během budování seznamu jsme každému dokumentu spočítali jeho atributy, což byla nějaká slova, která tento dokument nejvíce charakterizovala. Tyto atributy máme seřazené podle jejich tf-idf skóre. Označme A_i seznam atributů pro dokument D_i.

Kontext $\langle X, Y, I \rangle$ sestavíme takto: ze seznamu dokumentů vezmeme prvních n dokumentů, kde n je hodnota daná nastavením vyhledávače. V základním nastavení platí $n = 50$. Tento seznam n dokumentů bude představovat objekty kontextu

$$X = \{D_i \mid i \leq n\}.$$

Dále nalezneme atributy. Vyhledávač umožňuje nastavit maximální počet atributů, které má u každého dokumentu vzít. Tuto hodnotu pojmenujeme m. Nyní z každého dokumentu z množiny X vezmeme maximálně m atributů daného dokumentu a tato slova sjednotíme do jedné množiny. Nejprve si nadefinujeme funkci, která nám pro každý dokument vrací m atributů dokumentu s nejvyšším tf-idf skórem

$$\alpha_m(A_i) = \left\{ A_i^j \mid j \leq m \right\},$$

kde A_i^j označuje j-tý atribut dokumentu D_i. Tedy A_i^1 je atribut dokumentu D_i, který má nejvyšší hodnocení. Nyní sjednotíme všechny atributy všech dokumentů do jedné množiny

$$Y_1 = \bigcup \{\alpha_m(A_i) \mid i \leq n\}.$$

K této množině ještě přidáme slova z dotazu Q převedená na stemy. Nepřidáváme ale slova, která se vyskytují v NOT operátoru. Zavedeme tak funkci β, která bere na vstupu dotaz a na výstupu vrací množinu slov, které se vyskytují v dotazu, ale nevyskytují se v NOT. Tato funkce se bude chovat různě v závislosti na tom, jak vypadá dotaz Q:

1. pokud je Q pouze jedno slovo, označme ho s, pak $\beta(Q) = \{s\}$,
2. pokud je Q tvaru „a AND b“ nebo „a OR b“, pak $\beta(Q) = \beta(a) \cup \beta(b)$,
3. pokud je Q tvaru „NOT a“, pak $\beta(Q) = \emptyset$.

Celou množinu atributů našeho kontextu tak získáme tak, že k současných atributům v Y_1 ještě přidáme klíčová slova dotazu

$$Y = Y_1 \cup \beta(Q).$$

Už zbývá pouze definovat relaci I. Křížek v tabulce bude tehdy, když daný dokument obsahuje dané slovo. Zapsáno formálně:

$$\langle x, y \rangle \in I \iff y \in x,$$

kde $x \in X$ a $y \in Y$.

3.5.2. Nalezení konceptu dotazu

Dalším úkolem je nalezení konceptu, který reprezentuje výsledek aktuálního dotazu. Máme v zásadě dvě možnosti, jak tento koncept nalézt. Vezmeme si slova z dotazu $\beta(Q)$ a spočítáme extent a zpět intent. Tím dostaneme koncept, který v intentu obsahuje slova z dotazu a zároveň je to nejmenší koncept, který tato slova obsahuje. Koncept dotazu budeme označovat $\mathbb{H}$:

$$\mathbb{H} = \left\langle \beta(Q)^{\downarrow}, \beta(Q)^{\downarrow\uparrow} \right\rangle.$$

Tento postup má ale zásadní nevýhodu v případě, kdy se snažíme nalézt koncept dotazu pro dotaz, který obsahuje jiný operátor než AND. Pokud má dotaz tvar $Q =$„derivace OR integrál“, nemůžeme hledat koncept dotazu tak, že spočítáme extent $\{\text{derivace, integrál}\}^{\downarrow}$, protože bychom tím získali dokumenty, které obsahují jak slovo „derivace“, tak slovo „integrál“. Což jistě nepředstavuje koncept dotazu „derivace OR integrál“.

Lepším postupem tak je začít počítat koncept dotazu přes dokumenty. Pokud vezmeme množinu dokumentů, které vyhledávač vrátil, přesněji tu podmnožinu, kterou jsme použili v kontextu, X, pokud jsme aplikovali omezení, a tuto množinu uzavřeme, dostaneme koncept, který přesně odpovídá našemu dotazu. Dostaneme tak koncept

$$\mathbb{H} = \left\langle X^{\uparrow\downarrow}, X^{\uparrow} \right\rangle = \left\langle X, X^{\uparrow} \right\rangle.$$

Problémem tohoto přístupu je, že vždy dostaneme ten největší koncept. Nemůžeme tak hledat obecnější dotazy v horních sousedech, ani související dotazy v okolních sousedech, protože ani horní, ani okolní sousedy tento koncept nemá. To vyřešíme jedině tak, že kontext, který používáme, sestavíme jinak.

3.5.3. Sestavení rozšířeného kontextu

Kontext, který byl popsán v kapitole 3.5.1. nebudeme vůbec používat. Sestavíme jiný kontext. Nicméně provedeme pouze malé změny.

Začneme opět tím, že vyhledáme dotaz Q. Dostaneme dokumenty D. První změnou bude, že v případě, že dotaz bude mít více než jedno slovo, $|\beta(Q)| > 1$, tak položíme vyhledávači ještě jeden dotaz. Nový dotaz označíme Q' a bude mít tento tvar: vezmeme všechna slova z dotazu Q a vložíme mezi každá dvě slova OR. Pokud byl původní dotaz $Q =$„formální konceptuální analýza“, tak nový dotaz bude mít tvar $Q' =$„formální OR konceptuální OR analýza“. Dotaz Q' položíme vyhledávači, čímž získáme dokumenty D'.

Další postup už je stejný jako v kapitole 3.5.1., pouze do kontextu vložíme dokumenty a atributy z množiny D', nikoliv z D. Za X' vezmeme prvních n dokumentů z D', za Y' jejich atributy sjednocené s $\beta(Q)$. Relaci I' sestavíme stejně

$$\langle x, y \rangle \in I' \iff y \in x.$$

Dostaneme rozšířený kontext $\langle X', Y', I' \rangle$.

3.5.4. Nalezení konceptu dotazu v rozšířeném kontextu

Koncept dotazu nalezneme jednoduše v případě, že dotaz Q se skládá pouze z AND operátorů. Pak je koncept dotazu roven

$$\mathbb{H} = \left\langle \beta(Q)^{\downarrow}, \beta(Q)^{\downarrow\uparrow} \right\rangle.$$

V případě, že dotaz obsahuje OR nebo NOT operátor, musíme koncept dotazu hledat jinak. Můžeme vzít všechny dokumenty, které vyhledávač vrátil během hledání prvního dotazu Q, tj. D. K těmto dokumentům nalezneme v kontextu $\langle X', Y', I' \rangle$ intent a zpět extent. Dostáváme tak koncept

$$\mathbb{H} = \left\langle D^{\uparrow\downarrow}, D^{\uparrow} \right\rangle.$$

Pokud bychom neomezovali velikost kontextu, postup by fungoval. Vzhledem k tomu, že množina objektů X' může být menší než množina navrácených dokumentů D', tak ještě provedeme průnik množin, abychom zajistili, že množina, ke které počítáme intent, je podmnožinou X'. Pak koncept dotazu je roven

$$\mathbb{H} = \left\langle (D \cap X')^{\uparrow\downarrow}, (D \cap X')^{\uparrow} \right\rangle.$$

Idea rozšířeného kontextu a hledání konceptu dotazu byla použita ve vyhledávači SearchSleuth a je popsána v [10]. SearchSleuth používá k tvorbě rozšířeného kontextu horní sousedy konceptu dotazu – nejprve sestaví nerozšířený formální kontext, spočítá koncept dotazu, nalezne horní sousedy a intenty těch sousedů znovu vyhledá a výsledné dokumenty připojí ke kontextu, čímž vznikne rozšířený kontext. CLaSeek používá jiný postup, hledá rozšířený koncept přes OR dotaz.

Zároveň také SearchSleuth nepřidává do kontextu klíčová slova z dotazu, protože předpokládá, že se slova vyskytnou v titulcích a popiscích jednotlivých odkazů, které vrátí webový vyhledávač. To nemusí vždy nastat, takže ve formálních kontextu mohou některá klíčová slova chybět. SearchSleuth dále odstraňuje atributy, které sdílí méně než 5 % objektů.

3.6. Lindigův algoritmus pro hledání horních sousedů

Lindigův algoritmus je algoritmus pro budování konceptuálního svazu z kontextu. Součástí je i algoritmus na spočítání horních sousedů nějakého konceptu. Tento algoritmus budeme potřebovat, takže si ho popíšeme. Nejprve si ale zadefinujeme, co je to horní soused konceptu.

Definice 8 (Horní soused konceptu)**.** *Mějme kontext $\langle X, Y, I \rangle$ a jemu příslušný konceptuální svaz $\mathcal{B}(X, Y, I)$. Dále mějme koncept $\langle A, B \rangle \in \mathcal{B}(X, Y, I)$, který není největším konceptem svazu, tj. $\langle A, B \rangle \neq \left\langle X, X^{\uparrow} \right\rangle$.*

Koncept $\langle A', B' \rangle$ nazveme horním sousedem konceptu $\langle A, B \rangle$, pokud platí, že $\langle A, B \rangle < \langle A', B' \rangle$ a zároveň platí, že neexistuje koncept $\langle A'', B'' \rangle$ takový, že $\langle A, B \rangle < \langle A'', B'' \rangle < \langle A', B' \rangle$.

Jinými slovy, horní koncepty jsou ty koncepty, které se nachází právě o jednu úroveň výše než koncept $\langle A, B \rangle$.

Myšlenka algoritmu je založena na tom, že když přidáme k extentu A konceptu $\langle A, B \rangle$ objekt z X, který není v A, a uzavřeme, získáme koncept, který je určitě větší. Pokud postupně přidáme všechny objekty z $X \setminus A$, dostaneme množinu konceptů, která obsahuje všechny horní sousedy. Množinu si pojmenujeme S:

$$S = \left\{ \left\langle (A \cup \{a\})^{\uparrow\downarrow}, (A \cup \{a\})^{\uparrow} \right\rangle \mid a \in X \setminus A \right\}$$

Tato množina sice obsahuje všechny horní sousedy, ale bohužel i některé navíc. Může se stát, že přidáním objektu $a \in X \setminus A$ „přeskočíme“ nějaký koncept a dostaneme koncept, který je o více než jednu úroveň výše. Abychom tak získali pouze množinu horních sousedů, musíme při každém přidání nového objektu zkontrolovat, zda jsme skutečně vygenerovali horního souseda.

To zkontrolujeme tak, že zjistíme, zda všechny objekty, které přibyly v novém konceptu $\langle A', B' \rangle$, generují tentýž koncept. Přesněji to popisuje následující věta.

```
1: function UpperNeighbors(⟨A, B⟩ , ℬ(X, Y, I))
2:     candidates ← X \ A
3:     neighbors ← ∅
4:     for all a ∈ X \ A do
5:         B′ ← (A ∪ {a})↑
6:         A′ ← B′↓
7:         if candidates ∩ (A′ \ A \ {a}) = ∅ then
8:             neighbors ← neighbors ∪ (⟨A′, B′⟩)
9:         else
10:            candidates ← candidates \ {a}
11:        end if
12:    end for
13:    return neighbors
14: end function
```

Obrázek 1. Algoritmus UpperNeighbors na počítání horních sousedů konceptu

Věta 9 (Jak nalézt horní sousedy)**.** *Nechť* $\langle A, B\rangle \in \mathcal{B}(X, Y, I)$ *a* $\langle A, B\rangle \neq \langle X, X^{\uparrow}\rangle$*. Pak* $(A \cup \{a\})^{\uparrow\downarrow}$*, kde* $a \in X \setminus A$*, je extent horního souseda konceptu* $\langle A, B\rangle$ *právě tehdy, když pro všechna* $x \in (A \cup \{a\})^{\uparrow\downarrow} \setminus A$ *platí* $(A \cup \{x\})^{\uparrow\downarrow} = (A \cup \{a\})^{\uparrow\downarrow}$*.* □

Pomocí této věty už můžeme napsat algoritmus, kterým vygenerujeme všechny horní koncepty konceptu $\langle A, B\rangle$ ve svazu $\mathcal{B}(X, Y, I)$, viz obrázek 1. Algoritmus byl původně popsán v [13].

3.7. Hledání návrhů ve svazu

V současné chvíli máme spočítaný koncept dotazu a víme, jak spočítat horní a dolní sousedy. To bude stačit k tomu, abychom zjistili všechny návrhy. Postup nalezení těchto návrhů bude vycházet z [10].

3.7.1. Specializace

Specializací budeme rozumět návrh, který konkretizuje dotaz tak, aby uživatel, z těch výsledků, které už má k dispozici, dostal nějakou významnou podmnožinu. Například pokud zadá slovo „jaguár“, aby ho specializace dovedla na podmnožinu dokumentů, které se zabývají autem, nebo zvířetem.

Jako první si spočítáme dolní sousedy našeho konceptu dotazu. Tím dostaneme množinu konceptů, kterou si označíme $\mathbb{L}_{\mathbb{H}}$

$$\mathbb{L}_{\mathbb{H}} = \text{LowerNeighbors}(\mathbb{H}).$$

Z každého konceptu budeme nejdříve potřebovat pouze intent, poté extent. Nechť tak funkce Int a Ext vrací z dané množiny konceptů K intenty a extenty.

$$\text{Int}(K) = \{B \mid \langle A, B\rangle \in K\} \tag{5}$$

$$\text{Ext}(K) = \{A \mid \langle A, B\rangle \in K\} \tag{6}$$

V tuto chvíli máme koncept dotazu, který má například intent {Jaguár} a dva dolní sousedy, které mají intenty {Jaguár, Zvíře} a {Jaguár, Auto}. Abychom získali návrhy `+Zvíře` a `+Auto`, odečteme od intentů dolních sousedů intent konceptu dotazu. Zároveň ještě odečteme slova z dotazu, protože v případě složitějších Booleovských dotazů by se mohlo stát, že by intenty dolních sousedů obsahovaly slovo, které už v dotaze je. Označme $\mathbb{H} = \langle \mathbb{H}_E, \mathbb{H}_I\rangle$. Dostaneme množinu možných návrhů na specializaci dotazu.

$$Spec' = \{B \setminus \mathbb{H}_I \setminus \beta(Q) \mid B \in \text{Int}(\mathbb{L}_{\mathbb{H}})\}.$$

Množina $Spec'$ obsahuje možné návrhy na rozšíření dotazu, ale některé z rozšíření mohou obsahovat více než jedno slovo. Pokud má koncept dotazu intent {Jaguár} a dolní soused má intent {Jaguár, rychlost, maximální}, pak v množině $Spec'$ bude návrh {rychlost, maximální}.

Nicméně je zbytečné navrhovat úpravu `+rychlost, maximální`, protože stačí, když uživateli navrhneme přidat pouze jedno z těchto slov a dovedeme ho ke stejnému konceptu. Z každé množiny v $Spec'$ tak vybereme vždy jen to slovo, které se ve všech dokumentech vyskytuje nejčastěji. Dostaneme finální množinu

$$Spec = \{\max(s) \mid s \in Spec'\},$$

kde max v tomto případě vybere to slovo, které se v dokumentech $\mathbb{D}$ vyskytuje nejčastěji. Množina $Spec$ už je finální a obsahuje všechny návrhy, které můžeme uživatel předložit. Zbývá jen definovat uspořádání, abychom uživateli navrhovali ty nejrelevantnější návrhy.

Řekneme, že návrh $s_1 \in Spec$ je relevantnější než návrh $s_2 \in Spec$, zapíšeme $s_1 \geq s_2$, právě tehdy, když pro koncepty $\langle A_1, B_1\rangle, \langle A_2, B_2\rangle \in \mathbb{L}_{\mathbb{H}}$, jejichž intenty generují s_1 a s_2, platí $|A_1| \geq |A_2|$.

V extentech A_1 a A_2 jsou jednotlivé dokumenty, ideou tak je, že nejrelevantnější návrh by měl zároveň vést na nejvíce dokumentů.

3.7.2. Generalizace

Generalizací budeme rozumět návrh, který zobecňuje dotaz tak, aby uživatel po této změně získal více dokumentů, které odpovídají nějakému obecnějšímu konceptu. Zobecnění bude prováděno odstraněním klíčových slov z dotazu.

Generalizace budeme hledat v horních sousedech konceptu dotazu, takže si je spočítáme

$$\mathbb{U}_{\mathbb{H}} = \text{UpperNeighbors}(\mathbb{H}).$$

Nyní jsme v opačné situaci než během hledání specializací. Koncept dotazu může mít intent {Jaguár, Zvíře} a horní sousedé intenty {Jaguár} a {Zvíře}. Tím, že odečteme intenty horních sousedů od intentu konceptu dotazu, získáme slova, která se nachází v intentu konceptu, ale nenachází se v intentu horního souseda. Jsou to ta slova, která vyhledávač navrhne uživateli k odstranění.

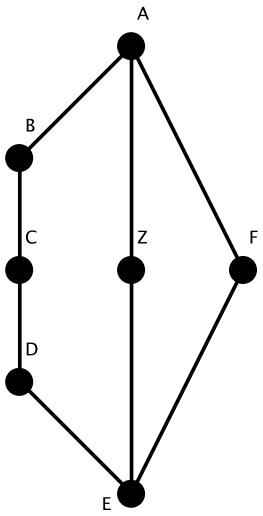
$$Gen' = \{\mathbb{H}_I \setminus M \mid M \in \text{Int}(\mathbb{U}_{\mathbb{H}})\}$$

Množina Gen' obsahuje množiny slov, která může uživatel odebrat. Ale může obsahovat i slova, která neobsahuje samotný dotaz a která tak nelze odstranit. Například pokud uživatel zadá dotaz „jaguár rychlost“, může být intent konceptu dotazu {jaguár, rychlost, maximální} a pokud má horní soused intent {jaguár}, pak by v množině Gen' bylo i slovo „maximální“ a vyhledávač by tak uživateli nabídl odstranit slovo „maximální“, aniž by toto slovo v dotazu bylo. Proto ještě provedeme průnik všech prvků v množině Gen' se slovy dotazu Q, abychom měli jistotu, že napovídáme odstranit jen ta slova, která uživatel skutečně do dotazu zadal.

$$Gen = \{g \cap \beta(Q) \mid g \in Gen'\}$$

Návrhy opět seřadíme tak, aby návrh na prvním místě vedl na co největší počet výsledků. Řekneme, že návrh $a_1 \in Gen$ je relevantnější než $a_2 \in Gen$, zapíšeme $a_1 \geq a_2$, právě tehdy, když koncepty pro koncepty $\langle A_1, B_1\rangle, \langle A_2, B_2\rangle \in \mathbb{U}_{\mathbb{H}}$, jejichž intenty generují a_1 a a_2, platí $|A_1| \geq |A_2|$.

Tento postup funguje správně v případě, kdy dotaz obsahuje pouze operátor AND. Pokud obsahuje i operátor OR nebo NOT, tak intent konceptu dotazu neodpovídá přesně struktuře dotazu. Pokud máme například dotaz „(spočetné OR nespočetné) AND (vysvětlení OR objasnění)“, tak koncept dotazu by mohl mít intent {spočetné, množiny} a algoritmus by nám mohl vygenerovat návrh `-spočetné`, čímž bychom získali dotaz „nespočetné AND (vysvětlení OR objasnění)“, který paradoxně povede k menšímu počtu výsledků. CLaSeek tak zobrazuje návrhy na odebrání klíčových slov jen v případě, kdy dotaz neobsahuje operátor OR nebo NOT.



Obrázek 2. Sourozenci konceptu Z

3.7.3. Podobné dotazy

V této části popíšeme, jak vygenerovat dotazy, které jsou podobné současnému dotazu, který uživatel zadal do vyhledávače. Když jsme hledali konkrétnější dotazy, hledali jsme je v dolních sousedech; obecnější dotazy jsme hledali v horních sousedech. Podobné dotazy budeme hledat v konceptech, které se nachází na stejné úrovni. Můžeme tak říci, že je budeme hledat v sourozencích konceptu dotazu.

Otázkou je, jak definovat sourozence. Když se podíváme na obrázek 2., jaké koncepty by měly být sourozenci konceptu Z? Když se budeme držet terminologie, která je běžná u binárních stromů, pak by sourozenci byli potomci rodiče, což by odpovídalo sourozencům $\{B, F\}$.

U svazů ale můžeme zvolit i opačnou cestu – rodičové potomků. To by odpovídalo sourozencům $\{D, F\}$. Dále můžeme říci, že sourozenci jsou všichni mezi potomky a rodiči, tj. všechny koncepty mezi konceptem A a E, tj. $\{B, C, D, F\}$.

Další rozumnou možností je vzít ty koncepty, které mají stejného rodiče jako koncept Z a zároveň i stejného potomka. Tomu odpovídá koncept F.

Pro jednu z těchto definic bychom se měli rozhodnout. Není vhodné uživateli napovídat všechny dotazy z konceptů B, C a D, protože D je konkrétnější dotaz než C a C je konkrétnější dotaz než B. Z této řady stačí uživateli navrhnout jeden dotaz.

Pokud budeme za sourozence brát potomky rodičů, budeme mu nabízet spíše obecnější dotazy. Pokud zvolíme rodiče potomků, pak mu budeme nabízet spíše konkrétnější dotazy.

Zvolíme tak zlatou střední cestu – budeme mu nabízet dotazy, které jsou generovány koncepty mající společného rodiče i potomka. Zadefinujeme si funkce, které spočítají všechny horní a dolní sousedy všem prvkům množiny.

$$\mathrm{UN}(X) = \bigcup \{\mathrm{UpperNeighbors}(C) \mid C \in X\} \tag{7}$$

$$\mathrm{LN}(X) = \bigcup \{\mathrm{LowerNeighbors}(C) \mid C \in X\} \tag{8}$$

Nyní můžeme sourozence $\mathbb{S}$ konceptu $\mathbb{H}$ zadefinovat jako

$$\mathbb{S}_{\mathbb{H}} = [\mathrm{LN}(\mathrm{UN}(\{\mathbb{H}\})) \cap \mathrm{UN}(\mathrm{LN}(\{\mathbb{H}\}))] \setminus \{\mathbb{H}\}\,.$$

Z této množiny už jen vytáhneme intenty a dostaneme množinu podobných dotazů

$$Sibl = \mathrm{Int}(\mathbb{S}_{\mathbb{H}}).$$

Tyto dotazy setřídíme tak, že spočítáme podobnost konceptů z množiny $\mathbb{S}_{\mathbb{H}}$ s konceptem dotazu. Podobnost dvou konceptů spočítáme přes podobnost jejich extentů a intentů. Podobnost dvou extentů pak jednoduše definujeme jako podíl počtu společných objektů a počtu všech objektů. Podobnost dvou konceptů $\langle A, B\rangle$ a $\langle C, D\rangle$ je rovna

$$\mathrm{sim}(\langle A, B\rangle, \langle C, D\rangle) = \frac{1}{2}\left(\frac{|A \cap C|}{|A \cup C|} + \frac{|B \cap D|}{|B \cup D|}\right).$$

Návrh $s_1 \in Sibl$ je relevantnější než $s_2 \in Sibl$, zapíšeme $s_1 \geq s_2$, právě tehdy, když pro koncepty $\langle A_1, B_1\rangle, \langle A_2, B_2\rangle \in \mathbb{S}_{\mathbb{H}}$, jejichž intenty generují s_1 a s_2, platí $\mathrm{sim}(\langle A_1, B_1\rangle, \mathbb{H}) \geq \mathrm{sim}(\langle A_2, B_2\rangle, \mathbb{H})$. Jako první budeme zobrazovat dotaz, který je nejvíce podobný současnému dotazu Q.

Téma sousedních konceptů a jejich podobnosti je blíže rozebráno v článku [10] a [9].

3.8. Podobné vyhledávače

V historii již existovaly vyhledávače, které fungovaly velice podobně jako CLaSeek. Jedná se o vyhledávače „CREDO“, „FooCA“ a „SearchSleuth“. V současné době CREDO a SearchSleuth nefungují vůbec a k použití FooCA je nutné zažádat o registraci tvůrce tohoto vyhledávače.

CREDO Zkratka pochází z „Conceptual REorganization of DOcuments“. Jeho autory jsou C. Carpineto a G. Romano. CREDO spolupracovalo s Googlem. Uživatel vložil do vyhledávače dotaz, CREDO tento dotaz nejprve poslalo pomocí SOAP API do Googlu, nechalo si vrátit prvních 100 výsledků a poté nad těmito daty provedlo FCA analýzu. CREDO počítalo pouze dolní sousedy a to do druhé úrovně. Začínalo se u největšího konceptu. Smyslem bylo, abych z těch sto výsledků, které dostal na začátku, postupně ukazoval uživateli ty odkazy, které odpovídají nějakému konceptu.

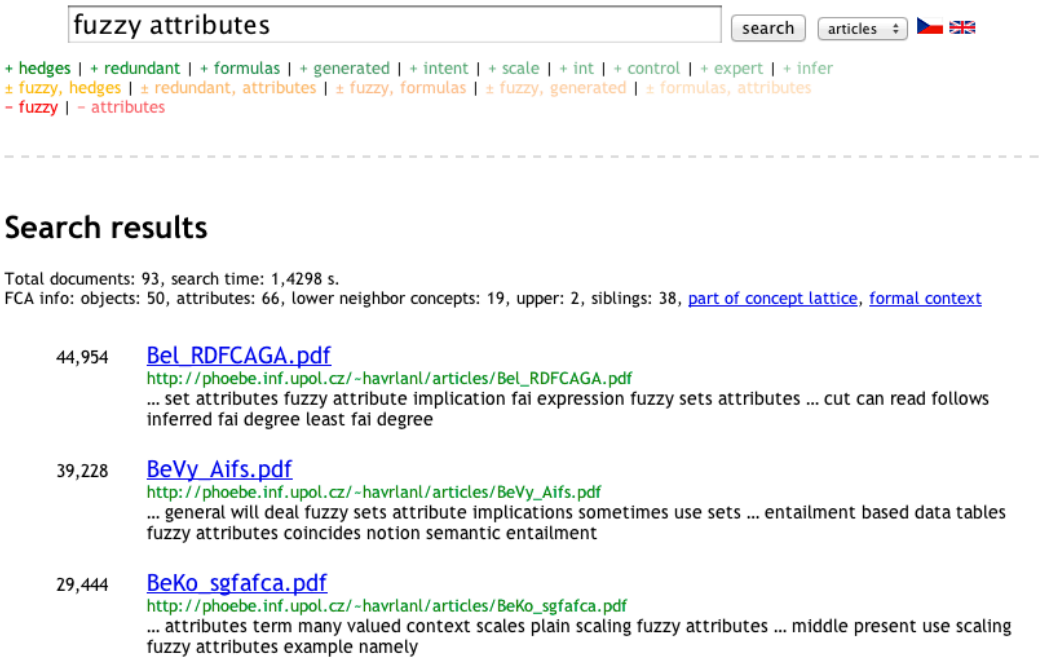
CREDO poté zobrazilo uživateli seznam odkazů spolu se stromovou strukturou návrhů na změnu dotazu. Tyto návrhy ale nebyly interaktivní, pouze po kliknutí na návrh se z té stovky vrácených dotazů vyfiltrovaly ty, které odpovídaly novému dotazu. Žádný nový dotaz do Google neproběhl. Detailnější informace jsou v článku [8].

FooCA Název vznikl ze spojení FCA a Google. Autorem je Bjoern Koester. Vyhledávač funguje na adrese `http://fooca.webstrategy.de`, ale vyžaduje registraci. FooCA, podobně jako CREDO, spolupracuje s webovými vyhledávači, konkrétně s Googlem a Yahoo. Uživatel vloží dotaz do FooCA, ten přepošle dotaz beze změny do Google API nebo do Yahoo API a dále pracuje s navrácenými výsledky. Po skončení FCA analýzy zobrazí uživateli výsledek ve formě tabulky – kontextu – nebo ve formě diagramu znázorňující konceptuální svaz.

Detailnější informace jsou v článku [12].

SearchSleuth je nejvíce podobný vyhledávači popsanému v této práci. Jeho autory jsou J. Ducrou a P. Eklund. SearchSleuth spolupracoval s Yahoo a jeho API. Po obdržení výsledků od vyhledávače sestaví kontext, nalezne koncept dotazu a znovu vyhledá ve vyhledávači obsah intentů horních sousedů, čímž SearchSleuth získá více výsledků a může sestavit rozšířený kontext. CLaSeek toto provádí pomocí dodatečného OR dotazu. SearchSleuth také do kontextu nepřidává klíčová slova z dotazu. Jinak je FCA část obou vyhledávačů velice podobná.

Detailnější informace jsou v článku [10] a [9]. Podobný systém byl také použit například pro analýzu CHM souborů, proprietárního formátu Microsoftu pro nápovědu. Více informací v [15].



Obrázek 3. Výsledek vyhledávače na dotaz „fuzzy attributes“

4. Výsledky vyhledávače

4.1. Uživatelské prostředí vyhledávače

Současná verze vyhledávače CLaSeek běží na adrese `http://phoebe.inf.upol.cz/claseek/`. Do textového pole se vloží dotaz, který chceme položit vyhledávači, vybereme jazyk, databázi, nad kterou chceme dotaz provádět a odešleme. Výsledek vyhledávání je vidět na obrázku 3.

4.2. Technické požadavky na HTML stránky

Přestože CLaSeek podporuje další typy souborů jako PDF nebo ODT, je především stavěn pro HTML stránky. Aby vyhledávač dobře fungoval, měl by HTML dokument splňovat některá elementární pravidla.

Vyhledávač potřebuje u každého dokumentu znát jeho název. V případě HTML stránek ho zjišťuje v elementu `TITLE`. Nicméně pokud ho neobsahuje nebo je jeho obsah prázdný, tak se použije poslední část URL. Obsah titulku by měl být pro každou stranu jiný. Není vhodné, aby mělo více stránek, nebo dokonce celý web, stejný titulek, protože pak je ve výsledcích vyhledávání nepůjde rozlišit jinak než jejich adresou. Toto pravidlo je důležité nejen pro tento vyhledávač, ale i pro běžné vyhledávače jako Google nebo Seznam. Zároveň se připravujeme o možnost zvýšit hodnocení stránky, pokud by klíčové slovo, které uživatel hledal, bylo v titulku.

Další informací, kterou může z webu získat, je popisek. Ten se zapisuje do elementu `<META NAME='description'>`. Jeho obsahem by měl být ne příliš dlouhý text, který popisuje danou stránku. Opět by měl být unikátní pro každou stránku. Pokud stránka neobsahuje popisek, vyhledávač se pokusí ze stránky vytáhnout nějaký popisek sám, ale téměř jistě to bude horší výsledek, než ručně psaný popisek pro každou stránku zvlášť.

HTML kód nemusí být nutně validní, ale musí dodržovat syntaxi jazyka. Nehledí se například na to, jestli element `IMG` obsahuje povinný atribut `ALT`, ale pokud nebude hodnota atributu správně ukončena uvozovkami, parser si na tom vyláme zuby. Pro převod HTML stránky do obyčejného textu se ve vyhledávači používají dvě funkce – první používá vestavěný parser, který postupně

parsuje stránku a neodpouští tak žádné chyby proti syntaxi jazyka; druhá funkce používá regulární výrazy a je tak benevolentnější, ale její výsledek je zase méně přesný. Druhá funkce se použije až v případě, kdy selže první funkce.

4.3. Hodnocení úspěšnosti vyhledávače

Ohodnotit, jak je CLaSeek úspěšný v řazení nebo v nabízených návrzích, nijak exaktně a automaticky nejde. Zbývá jen ruční zkoumání a slovní ohodnocení.

Ve vyhledávači je uloženo několik testovacích sad dokumentů. Jedná se o weby

- `jakpsatweb.cz`, který se zabývá tvorbou webových stránek a podobných věcí okolo,
- `matweb.cz`, který se zabývá především středoškolskou matematikou,
- `jakpodnikat.cz`, který se zabývá podnikáním, jak danit příjmy a podobně,
- `inf.upol.cz`, což je web katedry informatiky UP

a nakonec je ve vyhledávači použita i sada volně stažitelných odborných článků ve formátu PDF, které napsali zaměstnanci katedry informatiky UP.

Kromě webu katedry informatiky se jedná o weby, které mají jasné zaměření každé stránky a bohatý obsah, takže na jejich webech vypadá analýza nejlépe. Web katedry informatiky je stručnější, obsahuje jisté chyby (například titulky pro každou stranu zvlášť byly vytvořeny až během psaní této práce) a obsahuje nepříjemnost v podobě opakujících se slov na každé stránce, takže jeho analýza neprobíhá nejlépe.

4.3.1. Příklady dobrých výsledků

Analýza webu jakpodnikat.cz dává několik dobrých výsledků, pro příklad:

- Na dotaz „živnostenský list“ vrací seznam specializací `+ studenti | + příjmy | + provozovna | + nemocenské | + účetnictví | + jednotný | + dědická | + odkazy | + nemovitostí | + auto`.
- Když k dotazu přidáme například slovo „provozovna“, vyhledávač odpoví novými specializacemi `+ studenti | + příjmy | + zahájení | + služby | + autorský | + nemovitostí`. Zároveň přidá i podobné dotazy `+/- list, příjmy, živnostenský | +/- list, zahájení, živnostenský | +/- autorský, list, živnostenský | +/- nemovitostí, list, živnostenský`.
- Po vyhledání dotazu „slevy na dani“ dostaneme specializace `+ příjmy | + studenti | + základ | + doklady | + tisíc | + příloha | + minimální`, z nichž všechny kromě slova „tisíc“ dávají nějaký smysl. Seznam podobných dotazů: `+/- dani, příjmy | +/- dani, základ | +/- dani, doklady | +/- tisíc, dani | +/- příloha, dani`. Opět kromě dotazu, který obsahuje klíčové slovo „tisíc“ to má nějaký smysl.

Dobré výsledky vrací CLaSeek i pro web jakpsatweb.cz:

- Pro dotaz „hosting“ vrátí seznam specializací `+ php | + serveru | + seo | + google | + weblogu | + zdarma | + nástroje | + statistiky | + domény | + anglicky`.
- Pokud přidáme slovo „php“, dostaneme nový seznam specializací `+ serveru | + knihy | + google | + zdarma | + faq | + díky | + weblogu | + xml | + seo | + statistiky` a seznam podobných dotazů: `+/- hosting, serveru | +/- google, hosting | +/- hosting, zdarma | +/- knihy, php | +/- php, zdarma`.

4.3.2. Příklady špatných výsledků

Ze současných databází dává většinou nejhorší výsledky web katedry informatiky. Příkladem budiž dotaz „studium“, na který získáme specializace `+ zimní | + absolvent | + letní | + skupina | + jan | + martin | + arnošt | + činnosti | + szz | + bartl`. Většina těchto dotazů nejspíše k žádnému významnému konceptu nepovede.

Dalším příkladem může být složitější dotaz „jak vložit obrázek na stránku“ nad databází jakpsatweb.cz. Vyhledávač odpoví specializacemi `+ obrázky | + soubor | + google | + mail`. Nabízí to dokonce slovo „obrázky“, protože použitý stemmer zvolí jiný základ pro slovo „obrázek“ a pro „obrázky“. Další návrhy nejsou o nic rozumnější.

4.4. API vyhledávače

API aplikace je rozhraní, pomocí něhož může uživatel s danou aplikací komunikovat. Uživatel aplikaci posílá nějaké dotazy ve specifickém formátu a aplikace na ně opět ve specifickém formátu odpovídá. Uživatel tak může danou aplikaci používat, ačkoliv ji přímo nezkompiluje do své aplikace.

Jak jádro vyhledávače, tak webové rozhraní poskytuje API pro uživatele. Webové API je přístupné komukoliv, konzolové API je přístupné tomu, kdo si nainstaluje vyhledávač k sobě. Webové API je RESTové, to znamená, že využívá HTTP protokol a běžné metody jako GET nebo POST. Všechna data, která vyhledávač vrací na výstupu, jsou ve formátu JSON. Pro detaily viz kapitolu 4.5.

4.4.1. Získávání dat z indexu

CLaSeek umožňuje získávat data o existujících indexech. Ta se získávají pomocí HTTP GET požadavku s požadovanými parametry. Je možné zjistit například informace o daném dokumentu, jaký má titulek, jakou adresu a podobně. Zároveň lze zjistit další informace o indexu, například počet všech zaindexovaných dokumentů nebo seznam URL adres všech dokumentů.

Například dotazem `/api.php?d=jpw&docinfo=147&title` zjistíme název dokumentu z databáze `jpw`, který má ID 147.

4.4.2. Posílání vlastních dat na server

API vyhledávače umožňuje poslat na server vlastní dokumenty a provést nad nimi FCA analýzu. To se děje HTTP POST požadavkem. Poslaná data musí být ve formátu JSON. CLaSeek tyto dokumenty přijme, zpracuje je, tj. provede úplně stejné operace, jako by budoval klasický index, vytvoří dočasný index a následně nad těmito daty provede standardní FCA analýzu, jaká byla popsána v předchozích částech dokumentu.

Výsledkem budou opět data ve formátu JSON, která budou obsahovat výsledek FCA analýzy. Nalezneme tam specializace, generalizace a podobné dotazy a některá další data o kontextu a konceptuálním svazu. Dále tam bude seznam dokumentů, který bude seřazený dle relevance. Výstup je prakticky stejný, jako je grafický výstup vyhledávače, pouze máme výsledky v JSON podobě.

Tohoto můžeme využít na vylepšení výsledků běžného vyhledávače, například Googlu. Napíšeme webovou aplikaci, na které bude textové pole pro zadání dotazu. Uživatel zadá dotaz, aplikace tento dotaz přepošle API Googlu a získá výsledky pro zadaný dotaz. Následně převede výsledky z Googlu do takového formátu, kterému rozumí CLaSeek.

4.5. Dokumentace

Dokumentace k jednotlivým částem programu se nachází na webu spolu se zdrojovými kódy aplikací. K distribuci zdrojového kódu byl použit web `www.github.com`, přesné adresy jednotlivých projektů: Jádro vyhledávače: `https://github.com/havrlant/fca-search`, webové rozhraní: `https://github.com/havrlant/fca-search-web`.

5. Závěr

Výsledkem této práce je prototyp webového vyhledávače CLaSeek, který primárně pracuje nad statickou sadou dokumentů. Umí prohledávat několik typů dokumentů, automaticky z dokumentů získávat název a popisek, zvládá booleovské dotazy, kontroluje překlepy a výsledné dokumenty vrací seřazené dle relevance.

Po navrácení výsledných dokumentů provádí jejich analýzu a prostřednictvím formální konceptuální analýzy nabízí tři skupiny návrhů na úpravu dotazu: konkrétnější dotazy, obecnější dotazy a podobné dotazy. CLaSeek tak pomáhá uživateli upravovat dotaz tak dlouho, dokud není spokojen s výsledky vyhledávání.

CLaSeek dále poskytuje veřejné webové API, díky kterému je možné vyhledávači zasílat vlastní data. Vyhledávač pak sestaví dočasný index a provede nad zaslanými daty stejnou analýzu, jako v případě statického indexu.

Kvalita výsledků je přímo úměrná kvalitě zaindexovaných dokumentů. Pokud do vyhledávače vložíme obsahově bohaté dokumenty, jsou návrhy většinou dobré. Celá analýza je poměrně rychlá, CLaSeek pro daný dotaz vrátí všechny výsledky obvykle za několik sekund. Všechny kódy jsou volně přístupné pod BSD licencí a vyhledávač tak lze případně nainstalovat na vlastní server.

Další výzkum může proběhnout v části, která zpracovává samotné dokumenty. Například vylepšit algoritmy, které hledají atributy dokumentů, zdokonalit stemmer, zjišťovat synonyma a podobně. V části hledající návrhy je možné zavést fuzzy FCA, kdy do formálního kontextu neuložíme pouze informace o tom, zda dokument dané slovo obsahuje, ale i informaci o tom, jak moc je slovo pro dokument důležité. Je možné také rozšířit API, které CLaSeek nabízí a nad tímto API postavit další externí služby.

Literatura

[1] Module difflib :: Class SequenceMatcher. 2008, [Online].
URL `http://epydoc.sourceforge.net/stdlib/difflib.sequencematcher-class.html`

[2] Baeza-Yates, R. A.; Ribeiro-Neto, B.: *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999, ISBN 020139829X.

[3] Bělohlávek, R.: Introduction to Formal Concept Analysis. 2008.
URL `http://phoenix.inf.upol.cz/esf/ucebni/formal.pdf`

[4] Bělohlávek, R.; Outrata, J.: Improving web search with FCA.
URL `http://phoenix.inf.upol.cz/~outrata/download/texts/fcawebsearch-slides.pdf`

[5] Bělohlávek, R.; Outrata, J.: Relational Data Analysis 1. 2008.
URL `http://phoenix.inf.upol.cz/esf/ucebni/rda_fca_i.pdf`

[6] Bělohlávek, R.; Outrata, J.: Relational Data Analysis 2. 2008.
URL `http://phoenix.inf.upol.cz/esf/ucebni/rda_fca_ii.pdf`

[7] Bělohlávek, R.; Outrata, J.: Relational Data Analysis 3. 2008.
URL `http://phoenix.inf.upol.cz/esf/ucebni/rda_fca_iii.pdf`

[8] Carpineto, C.; Romano, G.: Exploiting the Potential of Concept Lattices for Information Retrieval with CREDO. *j-jucs*, aug 2004: s. 985–1013.
URL `http://www.jucs.org/jucs_10_8/exploiting_the_potential_of`

[9] Dau, F.; Ducrou, J.; Eklund, P.: Concept Similarity and Related Categories in Information Retrieval using Formal Concept Analysis. In *International Journal of General Systems*, Taylor & Francis Group, 2011.

[10] Ducrou, J.; Eklund, P.: SearchSleuth: The Conceptual Neighbourhood of an Web Query. In *CLA'07*, 2007.

[11] Ganter, B.; Wille, R.: *Formal Concept Analysis: Mathematical Foundations*. Berlin/Heidelberg: Springer, 1999.

[12] Koester, B.: Conceptual Knowledge Retrieval with FooCA: Improving Web Search Engine Results with Contexts and Concept Hierarchies. *Advances in Data Mining*, 2006: s. 176–190.
URL `http://dx.doi.org/10.1007/11790853_14`

[13] Lindig, C.: Fast Concept Analysis. In *Working with Conceptual Structures - Contributions to ICCS 2000*, Shaker Verlag, August 2000, s. 152–161.

[14] Manning, C. D.; Raghavan, P.; Schtze, H.: *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008, ISBN 0521865719, 9780521865715.

[15] Wormuth, B.; Eklund, P.: Restructuring Help Systems using Formal Concept Analysis. In *3rd Int. Conference on Formal Concept Analysis*, LNAI3403, Springer Verlag, 2005, s. 129–144.