TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Disertační práce

Liberec 2009

Mgr. Jiří Vraný

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: P3901 Aplikované vědy v inženýrství
Studijní obor: 3901V025 Přírodovědné inženýrství

Lokalizace algoritmů pro řazení výsledků vyhledávání informací na webu
Localized algorithms for web pages ranking

Mgr. Jiří Vraný

Školitel: doc. RNDr. Pavel Satrapa PhD.
Pracoviště: Ústav nových technologií a aplikované informatiky

Rozsah práce:

Počet stran: 131
Počet obrázků: 19
Počet tabulek: 17

15.3.2009

Anotace

Tato disertační práce se zabývá problematikou algoritmů a technik, používaných pro uspořádání výsledků vyhledávání informací na webu, pomocí fulltextového vyhledávače. Konkrétně jde o algoritmy které pracují s kontextem vyhledávaného dotazu a mají možnost přizpůsobit uspořádání tak, aby zvýhodňovalo určité téma, odrážející preference uživatele, který s vyhledávačem pracuje. Zároveň ale tyto algoritmy stále provádějí uspořádání datové kolekce před samotným vyhledávacím procesem, což umožňuje jejich nasazení i na rozsáhlé datové kolekce.

Práce se zabývá problematikou urychlení těchto algoritmů pomocí paralelních implementací a zejména ověřením, zda je tyto algoritmy – vyvinuté a otestované pro anglický jazyk – možné použít také pro jiný jazykový model, konkrétně pro češtinu.

V práci jsou uvedeny výsledky lokalizace dvou algoritmů jejichž teoretické návrhy byly experimentálně ověřeny na několika datových kolekcích. Proces lokalizace se skládá z aplikace českého jazykového modelu, urychlení výpočtu pomocí numerických metod lineární algebry a paralelního výpočtu hodnotících vektorů které jsou částmi výsledného řešení. Výsledky jednotlivých experimentů prokázaly, že díky navrženým modifikacím lze originální algoritmy Intelligent Surfer a Topic Sensitive PageRank úspěšně použít i pro Český jazyk.

Klíčová slova: fultextové vyhledávání, PageRank, personalizace, lokalizace, řazení výsledků

Annotation

This dissertation thesis is concerned with the algorithms for web pages ranking, especially with the personalized algorithms. Those algorithms allow to modify basic ranking, based on hyperlink structure of the web, by a user preferences and search query context. Still, there is a lot of ordering done offline, in advance before the search process. That allows usage of the algorithms on large datasets.

This thesis is focused on the speedup of personalized algorithms by using the methods of numeric linear algebra, by applications of the language model and by the parallel computation. The main focus of the thesis is to verify a possibility of localization of the algorithms originally designed and tested on the English language. The Czech language model is used for the localization testing.

The results of the localization process of two personalized algorithms are included. A theoretical proposal of localization was experimentally tested on several datasets. The results of those experiments are included. The experiments proved that localization of the Intelligent Surfer and the Topic Sensitive PageRank for the Czech language is possible.

Keywords: fultext search, PageRank, personalization, localization, results ranking

Prohlášení

Byl jsem seznámen s tím, že na mou disertační práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé disertační práce a prohlašuji, že **souhlasím** s případným užitím mé disertační práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své disertační práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Disertační práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací se školitelem.

V Liberci 20. března 2009

Rád bych na tomto místě v první řadě poděkoval celé své rodině, zejména však Zuzaně a Filipovi, bez kterých bych patrně neměl dost motivace práci dokončit.

Dále bych rád poděkoval svému školiteli doc. RNDr. Pavlu Satrapovi, PhD. za trpělivé vedení práce, prof. Ing. Miroslavu Tůmovi, CSc. za cenné rady a kolegům Ing. Jiřímu Hnídkovi a Mgr. Davidu Kmochovi za podporu a konzultace.

Obsah

Seznam obrázků

Seznam tabulek

Seznam symbolů a zkratek

V tomto seznamu jsou uvedeny pouze symboly používané soustavněji v celé práci. Dílčí symboly jednotlivých vzorců jsou vždy vysvětleny v textu u příslušného vzorce.

W graf reprezentující web

V množina všech stránek – uzlů grafu W

E množina všech odkazů – hran grafu W

$\mathbf{M}$ matice sousednosti grafu W

$\mathbf{G}$ stochastická neredukovatelná matice pro výpočet PageRank vektoru

$\mathbf{v}$ personalizační vektor PageRanku

α teleportační/tlumící koeficient PageRanku

Q množina slov tvořících dotaz

D množina dokumentů (stránek)

S slovník - množina slov tvořících určitou kolekci dokumentů

w_{sd} ohodnocení (váha) slova s v dokumentu d

$\mathbf{e}$ jednotkový vektor

s slovo - základní prvek většiny datových množin

d dokument (stránka)

fs_d frekvence slova s v dokumentu d

$\mathbf{rank}$ vektor ohodnocení množiny stránek

$outlink(i)$ počet odchozích odkazů ze stránky i

$inlink(i)$ počet příchozích odkazů na stránky i

$P_q(i \rightarrow j)$ pravděpodobnosti přechodu ze stránky na stránku pomocí odkazu

$P'_q(j)$ pravděpodobnosti přechodu na stránku přímým zadáním adresy - též teleportace

CZDIS algoritmus Czech Distributed Intelligent Surfer - lokalizovaná verze IS

DMOZ jiná zkratka pro ODP

DN Dangling Node - koncový uzel v grafu, stránka která nikam neodkazuje

HITS algoritmus Hypertext Induce Topic Selection

IS algoritmus Intelligent Surfer

MAPR modifikovaný adaptivní PageRank algoritmus

ODP Open Directory Project - otevřený katalog stránek

PageRank Algoritmus pro řazení stránek

SALSA Stochastic Algorithm for Link Structure Analysis

SPAM původně nevyžádaná elektronická pošta, dnes označení většiny neetických reklamních aktivit na Internetu.

SEO Search engine optimization - optimalizace stránek pro lepší výsledky vyhledávání. Zahrnuje jak postupy legální a doporučené, tak techniky nelegální označované jako SEO SPAM

TKC Tigtly Knit Community

TSPR algoritmus Topic Sensitive PageRank

TSPR lokalizovaná verze TSPR

ZI Získávání Informací

ZIW Získávání Informací na Webu

Kapitola 1

Úvod

WWW stránky uložené na serverech po celém světě v rámci sítě Internet představují nejrozsáhlejší soubor dokumentů v lidských dějinách. Přitom je celá tato rozsáhlá kolekce dosažitelná z jakéhokoliv z připojených počítačů a nemá žádnou centrální autoritu, která by měla na starost správu či katalogizaci dokumentů tak, jak ji známe například z knihoven.

Do jisté míry suplují tuto roli vyhledávací stroje pro WWW, které jsou v současnosti velmi rozvíjenou oblastí vědy o získávání informací. Jsou tak zároveň i jednou z jejích nejznámějších aplikací u široké veřejnosti. Tato mezioborová disciplína v sobě spojuje poznatky z teorie informací, matematiky, teorie programování, ale i z humanitních oborů jako je lingvistika, psychologie či sociologie.

Obrovský rozmach WWW stránek jako dalšího publikačního média ve druhé polovině 90. let s sebou, krom jiného, přinesl také masivní rozvoj vyhledávacích technologií a metod pro získávání informací. Klasické modely dokumentů, používané dlouhou řadu let, přestávaly být na obrovské množině dat efektivně použitelné. Proto se poměrně brzy objevilo hned několik nových teorií a algoritmů ve všech částech procesu získávání informací.

Nový mediální trh – Internetová reklama – vzbudil značnou pozornost investorů, i když očekávání bylo mnohdy větší než reálné možnosti. To bylo také patrně jedním z důvodů krachu technologického akciového trhu NASDAQ v roce 2000 [6]. Přesto zůstala řada společností, které se na trhu úspěšně zabydlely a v současné době v oblasti reklamy úspěšně konkurují tištěným médiím. Jednou z ekonomicky nejsilnějších společností je největší a nejznámější fulltextový WWW vyhledávací server - Google.com.

Google je dílem L. Page a S. Brina a od počátku byl tento projekt velmi inovativní ve všech částech procesu získávání informací. Jak v oblasti získávání dokumentů i jejich následného zpracování a indexace, tak v oblasti uspořádání výsledků vyhledávání dle relevance. Základní algoritmus pro hod-

nocení relevance nazvaný PageRank přinesl v oblasti WWW zcela nový pohled na tuto problematiku a měl překvapivě efektivní výsledky. Google se záhy stal světovou jedničkou, jeho autoři a jejich spolupracovníci pak velmi bohatými lidmi.

Právě mohutný komerční úspěch vyhledávače Google byl dalším impulsem pro detailní výzkum zaměřený na zvýšení efektivity vyhledávání i větší relevanci odpovědí vzhledem k dotazu uživatele. Vzniklo tak několik dalších modifikací základního algoritmu PageRank, ale rozvíjeny byly také další algoritmy jako Hits či Salsa. Některé z těchto teorií doplnily základní algoritmy, využívající metod lineární algebry a teorie grafů, o prostředky umožňující sémantickou či lexikální analýzu dat i dotazů. To s sebou na jednu stranu přináší nesporné výhody, ovšem je to zaplaceno tím, že algoritmus přestává být nezávislý na textu dotazu. A právě díky tomuto omezení se nabízí otázka, zda jsou tyto teorie publikované a navrhované pro anglický jazyk, platné i v českém jazykovém prostředí.

Práce je dále organizována následujícím způsobem: teoretický úvod představuje kapitola 2., jejímž cílem je seznámit čtenáře se základními pojmy teorie vyhledávání informací. Následující dvě kapitoly shrnují současný stav problematiky v oblasti algoritmů pro řazení výsledků vyhledávání založených na analýze hypertextové struktury webu – kap. 3. – a v oblasti efektivního výpočtu PageRank vektoru – kap. 4. Cíle práce, zformulované na základě provedených rešerší jsou uvedeny v kapitole 5.

Výsledky disertační práce jsou rozděleny do čtyř kapitol. První z nich – kap. 6. – ve stručnosti popisuje návrh a implementaci experimentálního systému. V kapitole 7. jsou prezentovány výsledky experimentů s jednotlivými metodami efektivního výpočtu PageRanku. Kapitoly 8. a 9. přinášejí výsledky lokalizace algoritmů IS a TSPR. Celou práci ukončuje srovnání těchto dvou lokalizovaných algoritmů – kapitola 10.

Kapitola 2

Úvod do získávání informací na webu

Získávání informací je mezioborová disciplína spojující informatiku, matematiku, knihovnictví a informační vědy, lingvistiku, psychologii, statistiku a další vědy. Jako většina oborů využívajících určitým způsobem počítačů, vznikla v 60. letech v USA. Za jednoho z jejích zakladatelů je považován prof. Gerard Salton, mimo jiné autor vektorového modelu dokumentu. Během vývoje systému nazvaného SMART řešil jako první řadu problémů spojených s automatickou indexací a následným získáváním informací z textových dokumentů. V systému SMART byla poprvé použita celá řada řešení a modelů do té doby využívaných v oblasti teoretické lingvistiky, jako jsou například hledání kořenů slov, statistická analýza textů, seznamy stop slov a mnoho dalších. Salton je také autorem teze, že cílem systému pro získávání informací je najít v kolekci dokumentů tu její část, která je nejvíce relevantní k hledanému dotazu.

Různé kolekce dokumentů, které Salton a další zkoumali, měly jeden společný rys – ze současného pohledu šlo o ideální data. Jednotlivé dokumenty byly kontrolované, takže tyto kolekce prakticky neobsahovaly žádné dokumenty podvodného charakteru (SPAM). Byly také relativně malé a do značné míry neměnné.

Tento stav se změnil v polovině 90. let nástupem služby WWW. Web představuje rozsáhlou heterogenní kolekci dokumentů – kromě textů v různých jazycích a kódováních, jsou zde také audio a video dokumenty, obrázky a další. Web lze charakterizovat jako nekontrolovanou kolekci dokumentů, kdokoliv se může stát autorem. Díky tomu se tato kolekce prakticky neustále mění, stránky vznikají, zanikají, přesouvají se. Některá URL změní svůj obsah několikrát za den, některá nikdy. Tradiční získávání informací tedy přestalo stačit a musela nastoupit nová disciplína – získávání informací

(ZI) na webu. Pochopitelně využívá řadu principů tradičního ZI, ale přináší také řadu vlastních a nových. Zpětně pak některé principy ZIW mohou být využity i pro zpracování jiných kolekcí dokumentů než WWW.

Tato kapitola popisuje základní principy tradičního ZI i ZIW a dále některé důležité rozdíly mezi oběma disciplínami.

2.1 Tradiční získávání informací

Základní podmínkou pro strojové zpracování dat je jejich reprezentace vhodným modelem. V teorii získávání informací existují tři základní směry návrhů modelu, jejichž kombinací pak vznikají modely pokročilejší. Tyto směry se opírají o následující vědní obory:

- teorie množin
- lineární algebra
- teorie pravděpodobnosti

Další dělení modelů vychází z přístupu konkrétního modelu ke vzájemným vazbám mezi jednotlivými termíny. Základní modely tyto vztahy vůbec neuvažují, pokročilejší modely pak buď tyto vztahy přímo řeší (imanentní modely) a nebo předpokládají existenci vztahů mezi termíny s tím, že tyto vazby jsou řešeny mimo vlastní model (transcendentní modely).

Z velkého množství dosud publikovaných modelů si uvedeme pouze dva nejzákladnější, ze kterých je odvozena většina dalších pokročilejších modelů. Jedná se o Booleovský a Vektorový model.

2.1.1 Standardní Booleovský model

Standardní (základní) Booleovský model patří mezi nejjednodušší a nejstarší. Jeho výhodou je snadná implementovatelnost, díky které v minulosti byl součástí většiny komerčních řešení pro vyhledávání. V současné době je obvykle nahrazen některou pokročilejší modifikací, ale základní princip zůstává stejný. Model, jak je z názvu patrné, vychází z Booleovské algebry a teorie množin.

Definujme tři základní množiny T,D a Q. Přičemž $T = \{t_1, t_2, t_3, \ldots, t_n\}$ je množina všech možných termínů (slov), $D = \{D_1, D_2, D_3, \ldots, D_m\}$ množina všech prohledávaných dokumentů a $Q = \{q_1, q_2, q_3, \ldots, q_p\}$ množina slov tvořících konkrétní dotaz.

Vlastní dotaz je pak logický výraz v úplné disjunktivní normální formě složený z $q_i \in Q$ a logických operátorů AND a OR. Pomocí logického operátoru NON je dále možné určit, že se mají vyhledávat pouze takové dokumenty, které konkrétní prvek $q_i \in Q$ neobsahují.

Vyhledávání dokumentů probíhá ve dvou krocích:

1. Pro každý $q_i \in Q$ i=1..p nalezneme množinu $S_i \subset D$.
   Platí, že $D_k \in S_i \Leftrightarrow q_i \in D_k$. Pokud se dané slovo v dokumentu vyskytovat naopak nemá, určí se výsledná podmnožina jako doplněk k nalezené podmnožině S_i.

2. Výsledek dotazu je pak tvořen sjednocením průniku množin S_i v závislosti na konkrétním logickém výrazu.

Chování Booleovského modelu nazýváme absolutní shoda. Konkrétní dokument dotazu buď zcela vyhovuje a výsledkem je 1 nebo nevyhovuje třeba jen v jednom z hledaných slov a pak už je výsledkem vždy 0. Z toho plyne i jeho další důležitá vlastnost - model neurčuje relevanci jednotlivých výsledků, vrátí pouze množinu dokumentů, které dotazu vyhovují. Pro uspořádání této množiny je tedy nutné použít nějaký externí algoritmus.

2.1.2 Vektorový model

Tento model je založený na aplikaci lineární algebry. Jak již bylo uvedeno, jeho autorem je prof. Gerard Salton [63] a poprvé byl model použit v systému SMART na Cornellově univerzitě.

Každý dokument je reprezentován vektorem složeným z reprezentantů jednotlivých klíčových slov. Vektory všech zkoumaných dokumentů společně tvoří n-rozměrný lineární vektorový prostor. Dimenze prostoru n je rovna počtu prvků množiny všech možných klíčových slov (termů). Příkladem takové množiny může být například český národní korpus [25].

Reprezentantem klíčového slova ve vektoru může být buď četnost jeho výskytu v dokumentu nebo jeho důležitost vzhledem k ostatním slovům v dokumentu. Důležitost slova se odvozuje z jeho četnosti, sémantické a syntaktické analýzy konkrétního dokumentu na základě předem určených pravidel. Salton určuje ve svém modelu důležitost (váhu) $w_{s,d}$ konkrétního slova s ve vektoru dokumentu V_d takto:

$$V_d = [w_{1,d}, w_{2,d}, ..., w_{n,d}]^T$$

$$w_{s,d} = fs_{s,d}. \log \frac{|D|}{|s \in D|} \tag{2.1}$$

Kde $fs_{s,d}$ je četnost slova s v dokumentu d a $\log \frac{|D|}{|s \in d|}$ je inverzní frekvence dokumentu,$|D|$ je počet všech dokumentů,$|s \in D|$ počet všech dokumentů obsahujících slovo s.

Podobnost dvou dokumentů je pak určena na základě předpokladu, že čím jsou si dokumenty podobnější, tím mají jejich vektory menší odchylku. Z vlastností vektorového prostoru vyplývá, že dva ortogonální vektory reprezentují dva zcela odlišné dokumenty. Dotaz je, stejně jako dokumenty, sestaven ze slov přirozeného jazyka a je reprezentován dalším vektorem.

Nejefektivnější způsob jak určit odchylku ϕ vektorů $\mathbf{q}$ (dotaz) a $\mathbf{d}$ (dokument), je vypočítat $\cos\phi$ jako podíl skalárního součinu obou vektorů a součinu jejich velikostí:

$$\cos\phi = \frac{\mathbf{q}.\mathbf{d}}{||\mathbf{q}||.||\mathbf{d}||}$$

$$||\mathbf{d}_i|| = \sqrt{\sum_j w_{i,j}^2}$$

Čím je $\cos\phi$ blíže jedné, tím jsou si dokument a dotaz podobnější. Díky tomu je možné stanovit pomocí tohoto modelu relevanci jednotlivých výsledků bez nutnosti použít nějaký externí algoritmus, případně použít tuto relevanci jako vstupní data pro zpracování dalším algoritmem pro zpřesnění výsledků.

Základní vektorový model má několik limitujících vlastností. Špatně pracuje s dlouhými dokumenty (vektory mají velkou dimenzi a malé skalární součiny). Dále nedokáže určit, že jsou dva dokumenty podobné, pokud používají jinou slovní zásobu - tedy nerozlišuje např. synonyma. Stejně tak si model neporadí se sémantikou. Pro reálná nasazení je tedy nutné model modifikovat, například takto:

- stanovením omezené množiny základních klíčových slov každého dokumentu.
- vyloučením tzv. stop slov. Tedy krátkých slov jako jsou spojky, předložky, částice, které se sice vyskytují v každém dokumentu, ale neovlivňují jeho faktický obsah.
- nalezením kořenů slov (stemming), nalezením všech tvarů daného slova (lemmatizace).
- omezit vektory pouze na podstatná jména, případně některá přídavná jména a slovesa.

- nezahrnovat do výsledků $\cos\phi$ menší než předem stanovená minimální hodnota.

Řada z těchto modifikací ovšem představuje netriviální problémy, které mohou značně zvýšit výpočetní náročnost celého procesu.

2.1.3 Měření kvality získaných informací

Pro testování kvality jednotlivých metod a nástrojů pro ZI bylo vytvořeno hned několik různých metrik. Pro každé slovo $q \in Q$, kde Q je množina slov v dotazu, nejprve ručně vytvoříme kolekci relevantních dokumentů $D_q \subseteq D$. Testovaný nástroj vrátí na dotaz q kolekci dokumentů $(d_1, d_2, \ldots, d_n)$. Pro měření *přesnosti* nástroje vypočteme nejprve vektor relevance $\mathbf{r}$, přičemž platí, že:

$$r_i = \begin{cases} 1 \text{ pokud je } d_i \in D_q \\ 0 \text{ jinak} \end{cases}$$

Přesnost nástroje pak definujeme jako poměr mezi počtem relevantních vrácených dokumentů a celkovým počtem dokumentů. Pro k dokumentů vypočteme přesnost P jako:

$$P(k) = \frac{\sum_{i \leq k} r_i}{k}$$

Informační hodnota výsledku (recall) je metrika definovaná jako podíl počtu vrácených relevantních dokumentů a všech relevantních dokumentů. Pro k dokumentů vypočteme tuto hodnotu H jako:

$$H(k) = \frac{\sum_{i \leq k} r_i}{|D_q|}$$

Přesnost a informační hodnota spolu souvisí. Pro k=0 bude přesnost rovna 1, zatímco informační hodnota bude nulová. Zvýšením počtu zkoumaných dokumentů logicky zvedneme informační hodnotu výsledku, ale zároveň nám klesne jeho přesnost. Kombinací přesnosti a informační hodnoty v jednom vztahu je tzv. **F**-metrika vypočtená jako:

$$F_b = \frac{(b^2 + 1)(P \times H)}{b^2 P + H}$$

Hodnota b nám umožňuje manipulovat s **F**-metrikou a měnit poměr mezi důrazem na přesnost a informační hodnotu. Jestliže $b = 0$, pak je hodnota F stejná jako přesnost P pro pevně dané k, pokud $b \to \infty$, pak je F stejná jako informační hodnota H pro pevně dané k. Pro $b = 1$ je kladen stejný důraz

na přesnost i informační hodnotu. Pro $b = 2$ má informační hodnota 2x větší význam než přesnost atd.

Pro potřeby ZIW jsou pak tyto metriky použitelné jen velmi obtížně, neboť pro běžné dotazy je nemožné ručně sestavit dostatečnou kolekci relevantních dokumentů.

2.2 Získávání informací na webu

Při získávání informací na webu lze využít pouze část z tradičních prostředků. Díky povaze tohoto prostředí bylo ale nutné vyvinout a implementovat celou řadu zcela nových metod a nástrojů. Jedním z nejdůležitějších nástrojů jsou vyhledávací servery. Uvádí se [83], že až 80% uživatelů získává nové adresy stránek právě pomocí těchto vyhledávačů. Proto si v této části práce popíšeme základní schema fulltextového vyhledávacího stroje, jeho důležité části a některé datové struktury.

2.2.1 Struktura WWW vyhledávače

Internetový fulltextový vyhledávač je komplexní aplikace složená z mnoha samostatně pracujících programů. Tyto programové části lze rozdělit podle toho, v jaké fázi vyhledávacího procesu přicházejí na řadu.

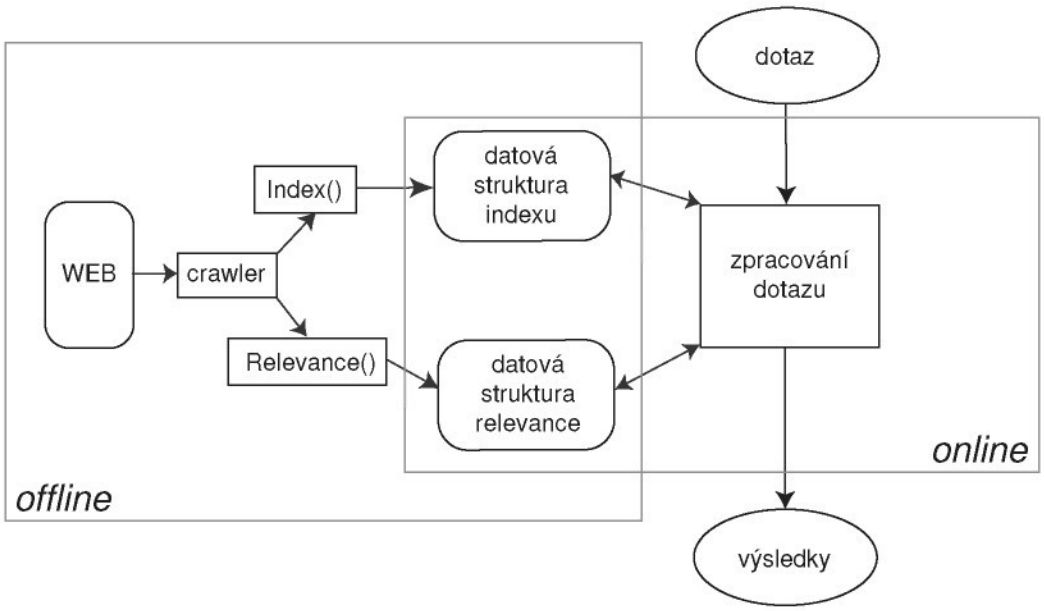
Vlastnímu vyhledávání, tedy situaci, kdy uživatel zadá dotaz a čeká na zobrazení výsledků, předchází celá řada procesů - výpočty, získávání a zpracování dat. Interaktivní proces mezi vyhledávačem a uživatelem probíhá v reálném čase - odtud označení *online*. Přípravné procesy probíhají mimo vědomí uživatele, ať už paralelně na jiných počítačích, nebo v době, kdy je většina uživatelů neaktivních a výkon počítačů určených pro interakci s uživatelem lze využít k jiné činnosti. Z pohledu uživatele tedy probíhají v době, kdy není připojen k vyhledávacímu serveru - *offline*.

Každý vyhledávací stroj se obvykle skládá z následujících základních komponent:

Offline komponenty

Jak již bylo uvedeno, nejsou tyto komponenty součástí vlastního uživatelského procesu vyhledávání, ale připravují a poskytují data nezbytná pro tento proces.

- Crawler - též spider. Jde v podstatě o automatický prohlížeč, který prochází web na základě předem stanového algoritmu. Při návrhu tohoto algoritmu je nutné vyřešit problém rychlosti získávání dat.



Obrázek 2.1: Blokové schema internetového vyhledávače.

Na jedné straně se počet stránek zvyšuje rychleji než je možné je strojově zpracovat. Na druhé straně, může crawler odesílající několik desítek požadavků za sekundu přivodit přetížení a kolaps slabšího WWW serveru. Proto je nutná paralelizace procházení i zpracování získaných dat. Dalším problémem může být frekvence návratů - některé stránky se nezmění od doby svého vzniku, další mění svůj obsah několikrát za den.

Problematikou crawlerů se zabývá řada akademických článků a prací - například [35] nebo [39].

- Index - základní datová struktura každého vyhledávače obsahuje data získaná crawlerem a zpracovaná indexovacím programem (na obr. 2.1 jako funkce Index()). Data jsou uložena ve struktuře odpovídající zvolenému vyhledávacímu modelu. Běžnou praxí je i použití více datových struktur zároveň, například běžný a zároveň obrácený frekvenční index a další kombinace.

  S indexací je spojena většina problémů, které se v rámci získávání informací na webu řeší. Z těch nejzásadnějších vyjmenujme alespoň:

  - Řídká datová struktura obrovských rozměrů - problémy spojené s kompresí a paralelizací řešení.

  - Problémy spojené se zpracováním přirozeného jazyka, tedy například nalezení kořenů slov, či všech tvarů daného slova a další.
  - Problém aktualizace indexu. Jak často aktualizovat a jak zajistit konzistenci indexu.

- Relevance - výsledky vyhledávání je potřeba uspořádat před tím, než dojde k jejich vrácení uživateli. Kritériem pro uspořádání je relevance nalezeného dokumentu vzhledem k dotazu. Přestože řazení dle určitého uživatelského dotazu q musí být logicky součástí online procesu, existuje řada algoritmů, které umožňují uspořádat indexované stránky předem, za použití různých funkcí pro ohodnocení kvality. Detailně popisuje tyto algoritmy kapitola 3.

  Data získaná v rámci výpočtu relevance jsou uložena ve zvláštní datové struktuře, která je doplňkem k základnímu indexu vyhledávače. Na obr. 2.1 jde o funkci Relevance() a příslušnou datovou strukturu.

Online komponenty

Online komponenty zajišťují interaktivní uživatelský proces v reálném čase. I když i offline komponenty je nutné optimalizovat na maximální výkon, zde to platí dvojnásob. Zpravidla bývá řešeno jako vícevrstvá architektura - uživatelské rozhraní v některém ze skriptovacích jazyků pro web (PHP, Python, ASP), zpracování dotazu v některém z vyšších jazyků (C,C++).

- Uživatelské rozhraní - přijímá a zobrazuje dotazy od uživatele. Obvykle je to domácí stránka každého vyhledávače, jedna z komerčně nejcennějších částí celého stroje. O to zajímavější je minimalistický přístup Google právě k titulní stránce. Drtivá většina dalších vyhledávačů, světových i domácích, využívá domácí stránku jako svou nejúčinnější (a nejdražší) reklamní plochu. Z čistě technického hlediska jde o poměrně jednoduchý problém. O to více problémů ale nabízí domácí stránka z pohledu ergonomie, psychologie, designu a dalších oborů řešících určitým způsobem interakci uživatele a vyhledávacího stroje.

- Zpracování dotazu - dotaz je rozložen na jednotlivá klíčová slova, případně další operátory a dále zpracován podle použitého modelu. Výsledky zpracování jsou následně seřazeny dle hodnotící funkce a vráceny uživateli. V případě, že je výsledků malý počet, může být dotaz automaticky (bez zásahu uživatele) doplněn, například o vyhledání synonym, dalších slovních tvarů apod. Naopak v případě velkého počtu

výsledků může být jejich počet před vrácením uživatelskému rozhraní redukován.

Proces nemusí probíhat pouze automaticky. Uživatel může být také požádán o zpřesnění dotazu, pokud je výsledků mnoho, nebo mu naopak může být automaticky nabídnuta korekce překlepů, pokud je výsledků velmi málo.

2.2.2 Používané datové struktury

Nyní si popíšeme dvě základní datové struktury, používané pro indexaci různých částí WWW stránek. Detailně popisují tyto i další datové struktury použitelné pro indexaci například práce [9] nebo [47].

Inverzní index

Tato datová struktura se používá pro rychlé vyhledávání dokumentů obsahujících určitý dotaz q. Základní myšlenka je poměrně prostá. Jednotlivá slova s_i nalezená při zpracování množiny dokumentů D jsou uložena jako slovník S. Prvky tohoto slovníku, tvoří jednotlivé klíče inverzního indexu.

Každý prvek ve slovníku odkazuje na seznam s proměnlivou délkou. Tento seznam pozic obsahuje identifikátory konkrétních dokumentů. Může také obsahovat další informace, například frekvenci slova q v tomto dokumentu či jeho pozici. Příklad inverzního indexu je zobrazen na obrázku č. 2.2.

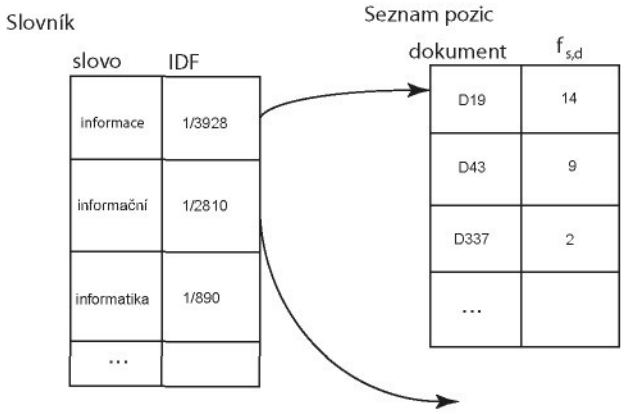
Inverzní index je opakem klasického indexu, ve kterém klíče tvoří identifikátory jednotlivých dokumentů a v odkazovaných seznamech jsou pak identifikátory jednotlivých slov.

Index web grafu

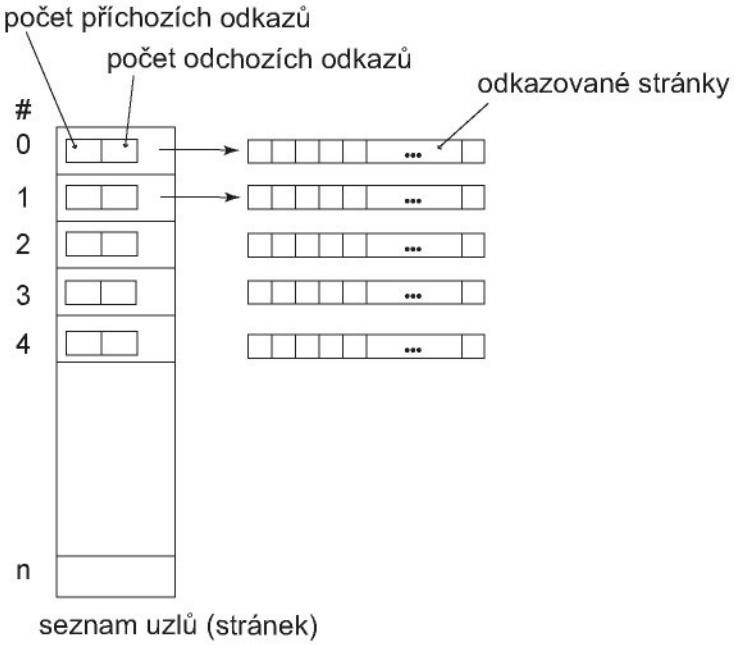
Pro indexování web grafu, se využívá *citační index* známý z bibliometriky. Možností jak ho vytvořit je několik. Triviální formou je seznam jednotlivých hran (u,v). Pokročilejší formou indexace grafů je pak seznam jednotlivých uzlů grafu W, položky tohoto seznamu tvoří pak opět seznamy uzlů připojených odchozím či příchozím odkazem – orientovanou hranou grafu. Příklad datové struktury pro uložení Web grafu ukazuje obrázek č. 2.3.

2.2.3 SEO

Na několika místech této práce se mluví o optimalizaci stránek pro vyhledávače – SEO. Jde o úpravu kódu WWW stránky tak, aby při sestavování výsledků dostala od vyhledávače co nejlepší hodnocení a zobrazila se pokud



Obrázek 2.2: Příklad inverzního indexu.



Obrázek 2.3: Příklad indexace grafu W.

možno mezi prvními deseti výsledky. V případě algoritmů založených na vyhodnocování odkazové struktury, pak jde o modifikaci kódu hned několika stránek za účelem získání co největšího množství příchozích odkazů.

Vztah společností provozujících vyhledávací servery k SEO je ambivalentní. Na jedné straně jsou pro ně optimalizované stránky přínosem, protože díky tomu vracejí kvalitní výsledky a uživatelé jsou spokojeni. Dá se tedy říct že vyhledávače SEO potřebují.

Tento přínos ovšem končí v okamžiku, kdy je díky SEO vysoce hodnocená stránka, kterou následně uživatel subjektivně vyhodnotí jako zcela nepotřebnou. Je-li takových stránek více, uživatel je nespokojený a může odejít ke konkurenci. Některé SEO techniky jsou tedy označovány za neetické a ze strany vyhledávacích společností mohou být různě penalizovány.

Z obavy před zneužíváním SEO také vychází snaha těchto společností co nejpečlivěji chránit skutečnou podobu hodnotící funkce a často jí měnit.

2.3 Velikost Webu

Na několika místech se v této práci píše o tom, že hlavním problémem získávání informací na webu je jeho velikost. Kolik přesně existuje webových stránek je nemožné určit, stejně jako je nemožné určit přesný počet lidí na zemi. Je to dáno jednak dynamickou povahou webu, nové stránky vznikají, jiné zanikají, a také tím, že existuje řada stránek, které nejsou přístupné prostými odkazy.

Tyto stránky se označují jako tzv. skrytý web. Ten mohou tvořit stránky izolované, které vůbec nejsou cílem odkazů, stránky vytvářené dynamicky a přístupné pouze po vyplnění formuláře (např. přihlašovacího) a další. Existuje také jiná skupina dynamicky vytvářených stránek s velmi limitovanou životností. Konkrétně může jít například právě o výsledky vyhledávání na webu.

Z těchto důvodů jsme schopni odhadnout pouze velikost tak zvaného *indexovatelného webu* – tedy stránek, které udržuje ve svém indexu některý z velkých vyhledávacích serverů.

2.3.1 Metodika odhadování velikosti

Pro odhadování velikosti indexů se obvykle využívá metodika [11] vycházející z jazykové statistiky a teorie pravděpodobnosti. Základním předpokladem je, že žádný z vyhledávačů neindexuje všechny stránky a že se jednotlivé indexy překrývají. Překrytí jednotlivých vyhledávačů se určuje následujícím postupem.

Představme si dva různé vyhledávací servery X a Y, přičemž velikosti indexu označíme jako $v(X)$, $v(Y)$. Nechť $Pr(X)$ představuje pravděpodobnost, že stránka i je součástí indexu X. $Pr(X \cap Y|X)$ je podmíněná pravděpodobnost, že stránka je součástí obou indexů. Dále platí, že $Pr(X \cap Y|X) \approx v(X \cap Y)/v(X)$ a obdobně $Pr(X \cap Y|Y) \approx v(X \cap Y)/v(Y)$. Poměr velikosti obou indexů $v(A)/v(B)$ je tedy přibližně $Pr(X \cap Y|Y)/Pr(X \cap Y|X)$. Poměr $v(A)/v(B)$ odhadneme tak, že otestujeme zda Y indexuje stránky, které X vrátí jako výsledek dotazu a naopak, zda X indexuje stránky, které na daný dotaz vrátí Y.

Pro implementaci metodiky potřebujeme vytvořit proces připravující vzorky (url) z indexu prvního vyhledávače. Tuto část nazýváme *vzorkování*. Druhým krokem metodiky je *testování* existence daného url ve druhém indexu.

Vzorkování : originální metodika vytváří vzorky za pomoci dotazů. Testovanému vyhledávači je odeslána množina dotazů a pro každý z dotazů $q \in Q$ je z prvních 100 výsledků náhodně vybráno jedno URL. Množina dotazů je sestavena jako frekvenční slovník. Bahrat a Broder použili pro vytvoření slovníku 300 tisíc dokumentů a slovník měl cca 400 tisíc výrazů. Celkový počet dotazů pak byl $|Q| = 35000$. Gulli a Signorini [29] použili jako kolekci dokumentů stránky uvedené v Open Directory Project. Kolekce dokumentů tak měla více než 2 mil. položek v 75 různých jazycích. Frekvenční slovník byl rozdělen do bloků po 20 slovech a z každého bloku bylo vybráno jedno ze slov jako reprezentant. Celkový počet dotazů $|Q|$ narostl na 438 tisíc.

Testování : testování připravených vzorků probíhá opět pomocí dotazů. Množina dotazů Q se tentokrát sestavuje ze stránek získaných při vzorkování. Pro každou stránku u je vybráno k nejvíce charakteristických slov. Jako určující metrika pro volbu charakteristických slov je použita TFIDF – inverzní frekvence slova v dokumentu známá z vektorového modelu. Těchto k slov se postupně odesílá vyhledávači, a testuje se, zda mezi vrácenými výsledky je či není také stránka u. V současné době umí většina moderních vyhledávačů potvrdit, zda zadané URL indexuje či nikoliv. Pro testování je proto možné použít i snadnější metodu, tedy odesílání normalizovaných URL tomuto rozhraní vyhledávače.

Odhadu velikosti indexu jednoho konkrétního vyhledávače se provádí za využití frekvenčního slovníku vytvořeného z dostatečně velkého textového korpusu. Známe-li frekvenci určitého slova a počet dokumentů použitých k vytvoření korpusu, můžeme z počtu indexovaných dokumentů, které toto slovo obsahují, odhadnout velikost celého indexu konkrétního vyhledávače.

2.3.2 Publikované odhady velikosti WWW

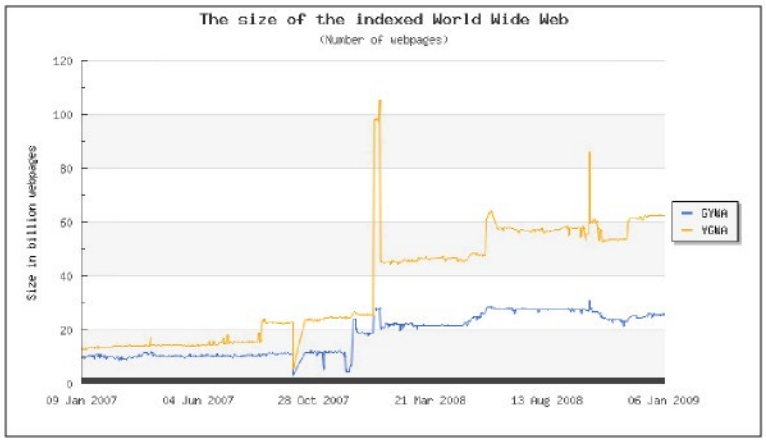
Jedním z prvních publikovaných odhadů je práce [11] z roku 1997. Bahrat a Broder odhadují, že počet stránek indexovaných v té době vyhledávači HotBot, Altavista, Excite a Infoseek je přibližně 200 milionů. Stejnou metodiku použili v roce 2005 Gulli a Signorini [29], výrazně ovšem zvýšili počet testovacích dotazů a jako referenční servery použili pochopitelně špičku vyhledávání v roce 2005 tedy Google, Yahoo, MSN a Ask/Teoma. Odhadovaná velikost webu v roce 2005 byla 11,5 miliardy stránek. Kromě těchto prací se velikostí webu zabývají například také stránky [79], [80]. Všechny tyto odhady jsou ovšem s přibývajícím časem stále více a více nepřesné.

Na práce [11] a [29] navazuje projekt worldwidewebsize.com [81]. Tento projekt vznikl na univerzitě v Tilburgu. Používá stejnou metodiku jako předchozí práce, ale jeho výhodou je, že data jsou neustále aktualizována. Na počátku roku 2009 byla odhadovaná velikost indexovatelného webu přibližně 25,5 miliardy stránek. Obrázek č. 2.4 ukazuje jak rostla velikost odhadu v posledních 2 letech. Obrázek č. 2.5 pak ukazuje velikost pouze indexu vyhledávače Google. Prudké změny velikosti jsou způsobeny několika výpadky během zpracovávání dat uvedenými na stránkách projektu. Jeho autor bohužel v anglické verzi dokumentace nijak nekomentuje, čím je zaviněn prudký nárůst velikosti odhadu na konci roku 2007.

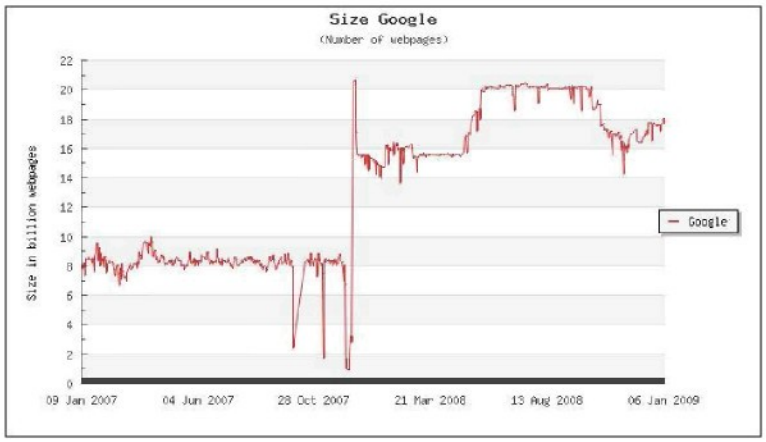
Výsledky zveřejňované na [81] jsou vypočteny za pomoci výše uvedné metodiky tak, že postupně dochází k testování překrytí 4 testovaných vyhledávačů. Jde o vyhledávače Google (G), Yahoo (Y), Windows Live Search (W) a Ask (A). Je zřejmé, že toto pořadí má na celkový odhad značný vliv. Viz rozdíl mezi YGWA a GYWA na obrázku č. 2.4. Ten je způsoben tím, že dle publikovaných odhadů právě Google indexuje nejvíce stránek. Pokud odhad začneme jiným vyhledávačem (křivka YGWA), dojde ke značnému nadhodnocení velikosti indexu druhého vyhledávače v pořadí. Podrobněji viz. [29]. Za přesnější lze tedy považovat křivku GYWA, kde je Google použit jako počátek odhadu. Současná velikost webu je tedy minimálně 25 miliard stránek, spíše však více.

2.3.3 Počet česky psaných stránek

Jednoduchou aplikací výše uvedené metodiky se můžeme pokusit o odhad počtu česky psaných stránek. Jako korpus pro zjištění frekvence využijeme množinu stránek CZdat2. Podrobnosti o této datové kolekci jsou uvedeny dále v kapitole 6.1.3 – experimentální systém. CZdat2 obsahuje přibližně 5 milionů stránek s poměrně širokým záběrem kategorií a témat. Lze jí tedy považovat za dostatečně reprezentativní množinu dat, pro vytvoření frek-



Obrázek 2.4: Růst velikosti indexovatelného webu za poslední dva roky. Zdroj [81].



Obrázek 2.5: Růst velikosti Google indexu za poslední dva roky. Zdroj [81].

	počet výsledků	*odhad velikosti indexu*
nebo - seznam	67 012 852	108 085 245
jak - seznam	73 988 427	134 524 413
nebo - google	56 500 000	91 129 032
jak - google	61 000 000	110 909 091

Tabulka 2.1: Odhad počtu česky psaných stránek za použití vyhledávačů Seznam a Google.

venčního slovníku.

Pro odhad použijeme dvě často používaná slova „jak“ a „nebo“. První z nich se vyskytuje v 55% všech dokumentů v korpusu, druhé v 62% dokumentů. Obě slova zadáme do vyhledávačů Seznam a Google, které mají největší podíl na českém trhu. V obou případech zvolíme vyhledávání pouze v česky psaných stránkách.

Z výsledků uvedených v tabulce 2.1 lze vyslovit odhad, že počet česky psaných WWW stránek je větší než průměrná hodnota odhadů v tabulce 2.1. Tedy že českých stránek je minimálně 111 milionů. K výsledkům je potřeba poznamenat, že byly zaznamenány 9.2.2009 a s přibývajícím časem se budou měnit, spolu se změnou počtu stránek v indexu.

Díky datům poskytnutým společností Seznam v únoru 2008 v celkem 12 souborech SBarell, můžeme provést ještě další měření počtu česky psaných stránek, tentokráte bez odhadu. Tato datová kolekce obsahuje celkem 103 541 347 různých URL. Celková velikost všech 12 barelů pak překračuje 540 GB.

U lokalizovaných algoritmů a dalších metod aplikovaných na česky psané stránky, můžeme tedy předpokládat, že počet stránek, na které se budou tyto metody aplikovat bude větší než 110 milionů.

Kapitola 3

Algoritmy pro řazení výsledků

V předchozí kapitole byl uveden odhad, že v současné době indexují světové vyhledávače více než 25 miliard WWW stránek. Přitom již v počátcích vývoje fulltextových vyhledávačů, kdy se indexovalo kolem čtyř milionů stránek, šlo o datovou strukturu do té doby nevídaných rozměrů. Proto je v současné době stále větší důraz ve vývoji kladen na co nejlepší uspořádání výsledků vyhledávání tak aby byl uživatel služby spokojen. Pokud možno tak, aby zároveň nepřišel o svůj komfort – tedy se například nemusel učit složité syntaxe dotazů, nebo vyplňovat formuláře apod.

Jelikož se Internet v průběhu vývoje stal plnohodnotným komunikačním médiem i podnikatelským prostředím, kde svou významnou roli hraje reklama a maximální snaha o to být viděn, musí se vyhledávací servery potýkat také s problematikou podvodných SEO technik - tedy snahou autorů stránek dostat se v hodnocení relevance odpovědi co nejvýše i za cenu technik zakázaných nebo alespoň neetických.

Pro uspořádání výsledků vyhledávání se v současné době využívají kombinace výsledků příslušného modelu (za předpokladu, že model výsledky uspořádá), dalších hodnotících funkcí, které vycházejí například ze syntaktické či sémantické analýzy dokumentu a algoritmů zabývajících se analýzou hypertextových odkazů.

Výsledné hodnocení stránky je pak složeno z desítek různých položek. Cílem je co nejpřesnější a zároveň nejstabilnější (neovlivnitelná spamem) hodnotící funkce. Tyto složené hodnotící funkce patří mezi nejvíce střežené technologie vyhledávačů. Z toho krom jiného plyne i to, že nejde o uzavřené algoritmy bez dalšího vývoje. Tak, jak prochází neustálým vývojem celý WWW prostor, prochází vývojem i tyto funkce.

V této kapitole si uvedeme přehled nejznámějších publikovaných algoritmů a jejich modifikací. Zaměříme se zejména na algoritmy, které pro uspořádání stránek využívají informace uložené v hypertextové struktuře webu.

Proto bývají označovány jako algoritmy odkazové analýzy. Jejich společným rysem je možnost uspořádat rozsáhlé datové kolekce předem – offline.

Nejprve si uvedeme některé základní pojmy a nezbytný matematický aparát.

3.1 Základní pojmy

Nechť $W = (V, E)$ je grafem Webu. V je množina všech stránek. Počet všech stránek, uzlů uvažovaného grafu, označme $N = |V|$. E je množina všech odkazů mezi stránkami, reprezentovaných orientovanými hranami grafu. Platí, že hrana $(i, j) \in E$ pokud existuje odkaz ze stránky i na stránku j. Každá stránka může být cílem i zdrojem odkazů.

Odkazy směřující ze stránky obvykle označujeme jako odchozí, odkazy směřující na stránku jako příchozí. Počet odchozích odkazů ze stránky i označme $outlink(i)$. Počet příchozích odkazů $inlink(i)$.

Graf W můžeme reprezentovat pomocí matice sousednosti $\mathbf{M}$. Pro jednotlivé prvky matice m_{ij} platí:

$$m_{ij} = \begin{cases} 1 \text{ pokud } (i,j) \in E \\ 0 \text{ v opačném případě} \end{cases} \tag{3.1}$$

Skalár λ a nenulový vektor x označují vlastní číslo matice $\mathbf{M}$ a k němu příslušející pravý vlastní vektor, jestliže platí, že $\mathbf{Mx} = \mathbf{x}\lambda$. Pokud platí, že $\mathbf{x}^T\mathbf{M} = \lambda\mathbf{x}$, představuje $\mathbf{x}$ levý vlastní vektor matice $\mathbf{M}$. Platí, že levý vlastní vektor je pravým vlastním vektorem transponované matice.

Říkáme, že matice $\mathbf{M}$ je redukovatelná, pokud existuje permutační matice $\mathbf{P}$ taková, že platí:

$$\mathbf{P}^T\mathbf{M}\mathbf{P} = \begin{bmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} \\ 0 & \mathbf{M}_{13} \end{bmatrix}$$

Platí-li, že řádky matice jsou nezáporné a součet prvků každého řádku je roven jedné, říkáme, že matice je řádkově stochastická. Pro stochastické matice platí, že jejich dominantní vlastní číslo je rovno jedné a příslušný vlastní vektor $\mathbf{x} = (k, k, ..., k)^T$ pro libovolnou konstantu k.

Stochastická matice $\mathbf{M}$ může reprezentovat přechodovou matici Markovova řetězce. Každý z prvků matice m_{ij} pak představuje pravděpodobnost přechodu ze stavu i do stavu j. Perioda stavu i je definována jako $p(i) = gcd\{n >= 1 : m_i i^n > 0\}$. Stav i je neperiodický, pokud platí, že $p(i) = 1$. Markovův řetězec je neperiodický právě tehdy, pokud jsou neperiodické všechny jeho stavy.

Podle Perron-Frobeniovy věty pro Markovovy řetězce má neperiodická, neredukovatelná, stochastická matice **M** unikátní stacionární rozložení reprezentované vektorem π a platí, že $\pi^T \mathbf{M} = \pi$. Stacionární rozložení Markovova řetězce může být vypočteno jako levý vlastní vektor stochastické matice **M**.

3.2 Pagerank

Tento algoritmus z roku 1998 je dílem autorů Brina a Page (odtud Pagerank) [56], kteří ho úspěšně použili ve struktuře vyhledávače Google [19]. Přestože Google nebyl zdaleka prvním vyhledávačem, svou konkurenci během několika let předstihl a získal dominantní postavení na trhu. Patrně i díky tomu patří Pagerank mezi nejstudovanější algoritmy současnosti, přinejmenším v oblasti získávání informací.

Již v základním návrhu počítá s možností personalizace, tedy přizpůsobení výsledků podle konkrétních preferencí uživatele nebo i vyhledávače. Díky tomu bylo v této oblasti již publikováno několik prací, které se personalizací zabývají. Na algoritmus Pagerank se tedy podíváme poněkud podrobněji, protože se jím bude zabývat podstatná část disertační práce.

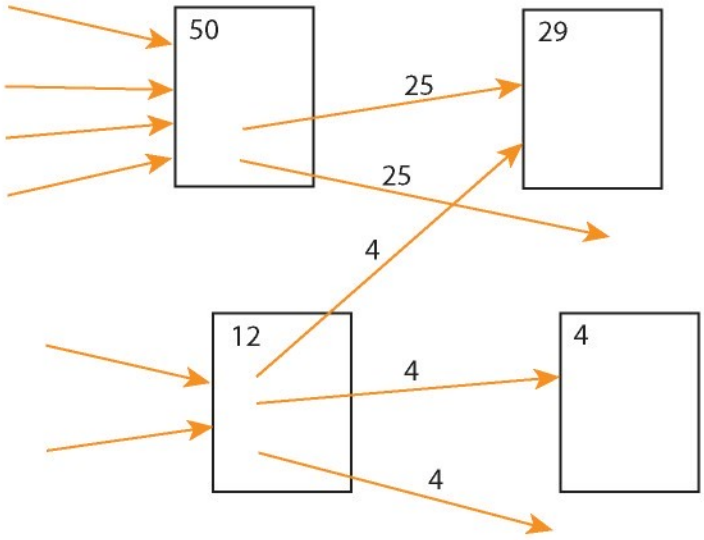
3.2.1 Základní definice

Algoritmus vychází z algoritmů pro sčítání citací v publikacích. Základní myšlenkou algoritmu je„„že stránka je důležitá, pokud na ní odkazují jiné důležité stránky". Pagerank (PR) konkrétní stránky *p* - *Rank(p)*, by tedy mohl být součtem PR všech stránek, které na ni odkazují. Důležité stránky obvykle odkazují na více míst, proto pro zajištění proporcionality dostává každá stránka pouze zlomek ranku odkazující stránky. Jestliže N_u je počet odkazů vedoucích ze stránky u, pak odkaz *(u,v)* přináší stránce v pouze $Rank(u)/N_u$. Princip proporcionálního předávání ukazuje obrázek č. 3.1.

Celkem jednoduše tak dojdeme k rekurzivnímu vztahu pro výpočet PR každé stránky na webu. Označme množinu všech stránek V, pak $moh(V)$ je počet všech stránek v množině V. Počáteční hodnotu PageRanku každé stránky i nastavíme na $Rank(i) = 1/moh(V)$. Dále nechť B_v je množina všech stránek odkazujících na stránku v.

$$\forall v \in V : Rank_{i+1}(v) = \sum_{u \in B_v} \frac{Rank_i(u)}{Nu} \; pro \; i = 1, 2, 3, \ldots$$

Iterace opakujeme dokud nedosáhneme předem požadované konvergence. Výsledkem výpočtu je vektor **Rank**, který obsahuje PR všech stránek v množině V.



Obrázek 3.1: PageRank - příklad proporcionálního předávání Rank()

3.2.2 Definice pomocí matice sousednosti

Celý algoritmus lze také převést na problém výpočtu vlastního vektoru matice sousednosti **M** definované za podmínek uvedených ve vzorci (3.1). Tato $n \times n$ matice je extrémně řídká, protože pro celý web je $n \approx 2,5.10^{10}$, zatímco průměrný počet nenulových prvků na jednom řádku nepřekračuje řád desítek.

Problém mohou představovat koncové uzly grafu označované jako *dangling nodes* – tedy takové stránky, které nemají ani jeden odchozí odkaz. Každý takovýto uzel vytvoří v matici **M** nulový řádek. Celou matici je tedy potřeba upravit tak, aby šlo o matici stochastickou. Tato úprava spočívá v tom, že každý nulový řádek je nahrazen řádkovým vektorem $\mathbf{e}/n$. Jde o velmi malá čísla, protože $n > 3.10^{10}$. Nevýhodou tohoto přístupu je, že nová matice bude obsahovat velké množství hustých řádků. Alternativní přístup jak zajistit, aby matice byla stochastická, ale zároveň zůstala extrémně řídká, navrhli Meyer a Langville v [48]. Zmíníme se o něm v následující části.

Nahrazením dangling nodes (DN) získáváme stochastickou matici **S**, která ale v případě webu je redukovatelná. Pro redukovatelnou matici nelze zaručit konvergenci výpočtu a proto zahrnuli autoři do algoritmu tzv. tlumící koeficient $\alpha : 0 < \alpha < 1$. Obvykle se udává, že Google používá $\alpha = 0,85$, a bylo provedeno několik výzkumů ověřujících různá nastavení tohoto koeficientu. Vlivem koeficientu α na celý výpočet, zejména na rychlost konvergence, se detailně zabývá například [31]. V této práci se k tomuto tématu vrátíme

v kapitole o výpočtu PageRank vektoru.

Použitím koeficientu α se dostáváme k výsledné Google matici $\mathbf{G}$.

$$\mathbf{G} = \alpha\mathbf{S} + (1 - \alpha)\mathbf{E} \tag{3.2}$$

Platí, že $\mathbf{E} = \mathbf{e}\mathbf{v}^T$, přičemž $\mathbf{v}$ je *personalizační vektor*, který umožňuje ovlivnit výsledný Pagerank libovolným směrem, ať už jsou důvody komerční či jiné.

Všechny dosavadní úpravy matice vedly k tomu, že na stochastickou matici $\mathbf{G}$ lze nyní aplikovat Perron-Frobeniovu větu. Výsledný Pagerank vektor je levým Perron-Frobeniovým vektorem $\mathbf{x}^T$ matice $\mathbf{G}$.

Výpočet vlastního vektoru se provádí pouze jednou pro každou množinu stránek a vypočtený rank je pak používán pro uspořádání stránek při sestavování odpovědi.

Zásadním přínosem Pageranku je jeho nezávislost na textu dotazu. Právě ta umožňuje vypočítat hodnocení stránek offline, což byla v době vzniku převratná technologie.

3.2.3 PageRank jako náhodná procházka

Jiným teoretickým přístupem k výpočtu Pageranku, je realizovat výpočet nalezením stacionárního vektoru pravděpodobnostního rozdělení Markovova řetězce reprezentujícího náhodnou procházku grafem web grafem W. Jde tedy opět o problém nalezení vlastního vektoru, ale pro sestavení matice $\mathbf{G}$ se využívá teorie pravděpodobnosti.

Jestliže ze stránky u vede na stránku v odkaz, potom bude v grafu W reprezentován orientovanou hranou $u \rightarrow v$. Počet odchozích odkazů ze stránky u vyjádřeme funkcí $outlink(u)$. Nyní si představme náhodného uživatele, který v čase k navštíví stránku u. Předpokládejme, že si zvolí jeden z odchozích odkazů jako další směr svého surfování sítí. V čase $k + 1$ si tedy s pravděpodobností $1/outlink(u)$ otevře stránku $v_i \in \{v|u \rightarrow v\}$.

PageRank stránky i pak vyjadřuje pravděpodobnost s jakou se náhodný uživatel v určitém čase $k > K$ dostane na stránku i. Je-li K dostatečně velké, je tato pravděpodobnost unikátní. Představme si Markovův řetězec reprezentující proces náhodné procházky grafem W. Jednotlivé stavy představují uzly grafu W a stochastická přechodová matice $\mathbf{P}$ popisující přechody z uzlu i do uzlu j je definována takto $p_{ij} = 1/outlink(i)$. Pokud je matice $\mathbf{P}$ neperiodická a neredukovatelná, pak je podle Ergodické věty stacionární rozložení unikátní[7].

Jak zajistit, aby šlo o stochastickou neredukovatelnou matici, již víme. Přidáme novou množinu odchozích linek s velmi malou pravděpodobností ke

všem uzlům grafu. Tím vytvoříme neperiodický a neredukovatelný přechodový graf. Pro odstranění dangling nodes pak můžeme použít dvě metody. Tou první je propojení DN se všemi ostatními uzly grafu. Tento přístup ovšem do řídké matice sousednosti zavádí mnoho hustých řádků a zbytečně tak zvyšuje paměťové nároky výpočtu.

Meyer & Langville navrhují v [48] vložit do grafu speciální uzel, který propojuje všechny dangling nodes (dále DN). Namísto n hran, tak z příslušného uzlu grafu vychází pouze jedna. Protože podle některých odhadů představují tyto uzly až 80% všech WWW stránek, je úspora nároků oproti klasické metodě opravdu podstatná. Tento nový uzel je reprezentován vektorem $\mathbf{a}$, pro který platí, že $a_i = 1$ pokud je stránka i DN a $a_i = 0$ pokud jde o stránku, ze které vedou odchozí odkazy.

Základní matici $\mathbf{P}$ tedy rozšíříme následujícím způsobem tak, abychom zajistili, že půjde o matici stochastickou.

$$\mathbf{S} = \mathbf{P} + \mathbf{a}\mathbf{v}^T \tag{3.3}$$

Nyní opět zavedeme matice $\mathbf{E} = \mathbf{e}\mathbf{v}^T$, kde $\mathbf{e}$ je jednotkový vektor a všechny prvky vektoru $\mathbf{v}$ jsou nezáporná čísla, jejichž součet je jedna. Markovův řetězec A pak můžeme definovat takto:

$$A = [\alpha\mathbf{S} + (1 - \alpha)\mathbf{E}]^T$$

Náhodná procházka může teď díky matici $\mathbf{E}$ probíhat následovně. S pravděpodobností $(1 - \alpha)$ přeskočí náhodný uživatel ze stránky i na náhodnou stránku l - nebude tedy sledovat některý z odkazů ze stránky, ale zvolí si úplně jinou (ze záložek, historie prohlížeče apod.). Pravděpodobnost, na kterou stránku l se uživatel vypraví, je určena pravděpodobnostním rozložením vektoru $\mathbf{v}$. Matice $\mathbf{E}$ se proto také někdy nazývá teleportační maticí. Jestliže zvolíme prvky vektoru $\mathbf{v}$ navzájem různé, můžeme pomocí něj ovlivnit chování teleportační matice. Jde tedy opět o *personalizační vektor Pageranku* známý z předchozího vysvětlení algoritmu.

3.3 HITS

Algoritmus HITS ("hypertext induced topic selection") je dílem Jona Kleinberga a byl to jeden z prvních algoritmů navržených pro stanovení určitého pořadí (rankingu) WWW stránek. Každá stránka je ohodnocena podle toho, kolik odkazů na ni směruje (inlinks) a na kolik stránek odkazuje ona sama (outlinks). Je tak zároveň zdrojem (Hub) i cílem (Authority) odkazů. Samozřejmě existují stránky, které spíše odkazují (například portály) a stránky, na které je spíše odkazováno.

Principem algoritmu je jednoduchá myšlenka "dobré zdroje, odkazují na dobré cíle". Pro každou stránku i je tedy stanoveno dvojí hodnocení a_i - authority score a h_i - hub score. Za použití matice sousednosti $\mathbf{N}$ můžeme vyjádřit vektory obou hodnocení takto:

$$\begin{cases} \mathbf{a} = \mathbf{N}^T\mathbf{h} \\ \mathbf{h} = \mathbf{N}\mathbf{a} \end{cases} \tag{3.4}$$

Matice sousednosti $\mathbf{N}$ je sestavena jako okolí dotazu q. Okolí dotazu q tvoří množina m stránek. Jde o všechny stránky, které dotazu vyhovují + všechny stránky, které jsou s nimi propojeny odkazem (oběma směry). Počet sousedních stránek může být tedy v závislosti na dotazu velmi variabilní. Tato množina stránek vytváří graf H, jehož incidenční maticí je právě matice $\mathbf{N}$. Je tedy zřejmé, že algoritmus HITS pracuje s kontextem dotazu.

Jednoduchou substitucí můžeme z rovnice (3.4) získat následující vztahy

$$\begin{cases} \mathbf{a} = \mathbf{N}^T\mathbf{N}\mathbf{a} \\ \mathbf{h} = \mathbf{N}\mathbf{N}^T\mathbf{h} \end{cases} \tag{3.5}$$

Pokud mají vektory $\mathbf{a}$,$\mathbf{h}$ konvergovat ke stabilnímu řešení, pak hledaným vektorem je vlastní vektor matice $\mathbf{N}^T\mathbf{N}$, respektive $\mathbf{N}\mathbf{N}^T$.

Kleinberg uvádí dvě zajímavé vlastnosti tohoto algoritmu. První z nich je, že pokud v každém iteračním kroku provedeme normalizaci výsledků, lze pro výpočet vektorů $\mathbf{a}$,$\mathbf{h}$ použít Mocninnou iterační metodu.

Druhá vlastnost je velmi dobře využitelná pro techniku shlukování dokumentů nazývanou destilace tématu. Tato technika je založena na struktuře odkazů mezi stránkami, nikoliv na jejich samotném obsahu. Kleinberg prokázal, že pro většinu dotazů q lze pomocí vektorů $\mathbf{a}$,$\mathbf{h}$ vytvořit bipartitní podgraf grafu G, který ukazuje na sémantický význam dotazu q.

Algoritmus HITS má několik problémů, které jeho použití v reálném prostředí velmi limitují, nebo dokonce vylučují.

Prvním podstatným problémem je značná nestabilita řešení. I malá změna grafu G může vést ke značným změnám vektorů $\mathbf{a}$,$\mathbf{h}$. Přidáme-li do množiny N jednu stránku – tedy jeden uzel grafu a dvě hrany, dojde k podstatné změně bipartitních podgrafů G a tím i k ovlivnění finálních výsledků. Web graf je nestabilní, k jeho změnám dochází prakticky neustále. Stránky vznikají, zanikají, případně se přesouvají na jiný server apod. Proto je nutné vždy spočítat hodnocení pro konkrétní dotaz, což je ale velmi náročné na výpočetní výkon vyhledávacího stroje. V případě komerčního vyhledávače, který řeší desítky dotazů za vteřinu, je pak nasazení algoritmu HITS prakticky nemožné.

Druhým problémem je poměrně vysoká možnost úspěšného ovlivnění výsledků pomocí SPAMu. Představme si stránku, která odkazuje na několik

dobrých cílů (autorit). Díky tomu získá tento uměle vytvořený zdroj vysoké hub score, které může předat dalším stránkám – falešným autoritám.

Další špatnou vlastností algoritmu HITS je takzvaný TKC efekt – problém těsně propojených komunit (Tigtly-Knit Community). TKC je určitá malá, ale hustě propojená množina stránek. Díky vysoké hustotě odkazů dochází k ovlivnění algoritmu – stránky dostávají vysoké hodnocení, ačkoliv nejsou autoritami vzhledem k určitému tématu.

Přes uvedené slabiny byl algoritmus HITS ve své době přínosem, zejména proto, že některé z jeho principů posloužily při vývoji modifikací dalších algoritmů.

3.4 SALSA

Salsa (Stochastic Algorithm for Link Structure Analysis) navrhli Moran a Lempel jako kombinaci některých vlastností algoritmů PageRank a HITS. Principem je určení bipartitních vazeb v prostředí náhodné procházky grafem W. Hlavním cílem tohoto návrhu je minimalizace TKC efektu. Algoritmus pracuje ve dvou krocích:

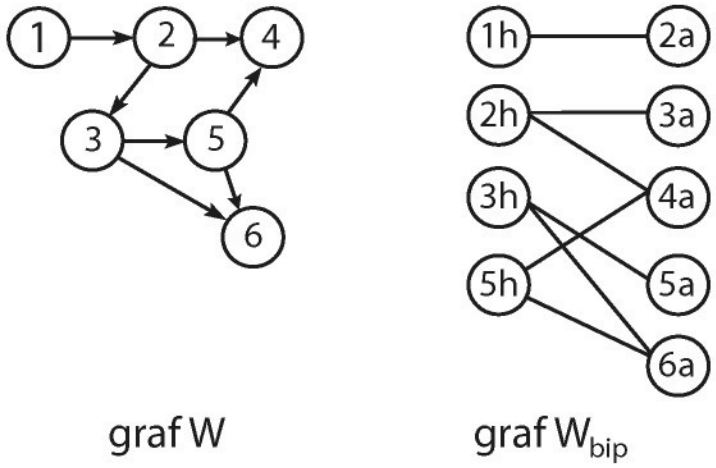
1. Z každé stránky v se náhodný uživatel přesune zpět na stránku u, která na stránku v odkazuje. Jako stránka v je zvolena náhodná stránka z množiny $\{v : (u, v) \in W\}$.

2. Ze stránky u může náhodný uživatel pokračovat na libovolnou stránku w, za předpokladu, že existuje hrana (u, w).

Obrázek č. 3.2 ukazuje princip transformace grafu W na bipartitní graf $W_{bip} = (V_a, V_h, E)$ definovaný jako:

$$\begin{cases} V_a = \{s_a : s \in W \wedge inlink(s) > 0\} \\ V_h = \{s_h : s \in W \wedge outlink(s) > 0\} \\ E = \{(s_h; r_a) : s \rightarrow r \in W\} \end{cases}$$

Kde a značí cíle (authority) a h značí zdroje (hub). Každá stránka $s \in W$, která není izolovaným bodem je v grafu W_{bip} reprezentována uzly s_h a s_a nebo jen jedním z nich (pokud postrádá příchozí - $inlink()$ či odchozí $outlink()$ odkazy). Každý z odkazů je reprezentován neorientovanou hranou spojující s_h a r_a. Na nově zkonstruovaném bipartitním grafu provádíme dvě různé náhodné procházky. Každá z nich zahrnuje pouze uzly z jedné poloviny grafu. V každém kroku procházíme vždy dvě hrany grafu W_{bip}. Každá z cest délky 2 v grafu W_{bip} reprezentuje buď přechod hypertextovým odkazem, pokud přecházíme ze strany zdrojů na stranu cílů, nebo naopak reprezentuje krok



Obrázek 3.2: SALSA - sestavení bipartitního grafu

zpět, pokud přecházíme ze strany cílů na stranu zdrojů. Tento pohyb označují autoři ve slovní hříčce jako jeden taneční krok.

Kvalitní zdroje i cíle příslušné k určitému tématu t jsou z grafu W_{bip} dobře patrné, protože jsou snadno dosažitelné z mnoha dalších uzlů buď přímo, nebo velmi krátkou cestou.

Salsa zkoumá dva různé Markovovy řetězce příslušející k daným náhodným procházkám. První řetězec reprezentuje stranu cílů v grafu W_{bip}, druhý pak stranu zdrojů. Každý řetězec definuje svou stochastickou přechodovou matici následovně:

Matici sousednosti grafu W označme jako $\mathbf{M}$. Jako $\mathbf{M}_r$ označme matici, kterou získáme, pokud každý nenulový prvek matice $\mathbf{M}$ vydělíme součtem prvků na jeho řádku. Obdobně matici označenou $\mathbf{M}_c$ získáme tak, že každý nenulový prvek matice $\mathbf{M}$ vydělíme součtem prvků v příslušném sloupci. Dále sestavíme matici zdrojů $\mathbf{Z}$ za použití nenulových řádků a sloupců matice $\mathbf{M}_r\mathbf{M}_c^T$ a matici cílů $\mathbf{C}$ za použití nenulových řádků a sloupců matice $\mathbf{M}_c^T\mathbf{M}_r$. Nulové řádky a sloupce matic $\mathbf{Z}$ a $\mathbf{C}$ algoritmus ignoruje, protože podle definice musí mít každý z uzlů grafu W_{bip} alespoň jednu hranu. Díky tomu jsou matice $\mathbf{Z}$ a $\mathbf{C}$ neredukovatelné a graf W_{bip} spojitý. Z matic $\mathbf{Z}$ a $\mathbf{C}$ se vypočítá vektor hodnocení stejně jako v případě algoritmu HITS.

Autoři algoritmu prokázali, že konvergenci algoritmu lze garantovat pomocí Ergodické věty. Rychlost konvergence zdrojů závisí na počtu odchozích odkazů, rychlost konvergence cílů pak na počtu příchozích. Dále autoři prokázali, že algoritmus SALSA lépe zvládá TKC efekt. Toho lze dosáhnout tím, že se ručně odstraní odkazy generované dynamicky (např. pomocí cgi

skriptů), odkazy reklamní a odkazy podezřelé jako SPAM. Bohužel tento krok prakticky znemožňuje zpracování reálných dat v akceptovatelném čase a algoritmus Salsa tak pravděpodobně zůstane pouze teoretickou studií bez reálné aplikace v některém z komerčních vyhledávačů.

3.5 Hilltop

Posledním z algoritmů, které lze označit za základní, je algoritmus Hilltop. Jeho hlavním utorem je, již několikrát citovaný, K. Bharat [12]. Algoritmus Hilltop vznikl v roce 1998 a byl použit ve stejnojmenném vyhledávacím stroji. Vývoj tohoto vyhledávacího stroje byl ale záhy ukončen, protože Bharat, stejně jako Henzingerová, se kterou vytvořil již zmíněný algoritmus Topic Distilation, se záhy stal zaměstnancem Google. Tato společnost si také v roce 2003 algoritmus Hilltop patentovala.

Stejně jako tři již představené algoritmy vychází i Hilltop z analýzy web grafu W tvořeného hypertextovou strukturou webu. K řazení dokumentů slouží tzv. expertní stránky – tedy referenční autority pro dané téma. Expertní stránka se vyznačuje tím, že je velmi relevantní k určitému tématu, a zároveň má řadu propojení s nepříbuznými stránkami. Nepříbuzná stránka je pak definována jako stránka, jejímž autorem je člen jiné organizace. Pro rozlišení organizací slouží hostname serveru. Definice je tedy částečně poplatná době vzniku, v dnešní době je běžnou praxí, že práce stejného autora (a tedy stránky příbuzné) se vyskytují na řadě různých serverů.

Jako nejkvalitnější stránky z kolekce dokumentů, tedy „vrchol kopce", který je v názvu algoritmu, jsou pak označeny stránky, na které odkazuje nejvíce expertních stránek k danému dotazu. Podobně jako u algoritmu HITS jde tedy i zde o uspořádání navázané na dotaz a část výpočtu probíhá online.

Klíčovou částí algoritmu je detekce expertních stránek pro dané téma. Pro tuto detekci je nutné definovat relaci *příbuznosti*. Originální definice této relace je následující: dva servery označíme jako příbuzné, jestliže jsou součástí stejné IP adresní třídy C (první tři oktety IPv4 adres jsou stejné), nebo jestliže se jejich doménová jména shodují v doménovém jméně druhé úrovně. Tedy například www.tul.cz a www.vslib.cz by byly vyhodnoceny jako nepříbuzné stránky, ale stránky www.tul.cz a www.tul.com jako stránky příbuzné. Relace je definována jako tranzitivní. Jako expertní stránka je v originále definována ta, která odkazuje na nejméně pět nepříbuzných stránek.

Stránky vyhodnocené jako expertní se dále indexují. Používá se klasický inverzní index, který přiřazuje klíčová slova ke stránkám obsahujícím tato slova. Podstatným rozdílem a urychlením zpracování je výběr klíčových slov pouze z významných částí stránky tzv. *klíčových frází* – konkrétně jde o ti-

tulek <title>, nadpis <h1>a texty odkazů <a>. Počet klíčových slov pro každou stránku je omezen, což omezuje možnost nežádoucího ovlivnění klíčových slov. Vyhledání expertních stránek v kolekci dokumentů, stejně jako jejich indexace, probíhá offline.

Po zadání dotazu Q přichází ke slovu online část algoritmu. Nejprve je nutno setřídit expertní stránky a zvolit ty, které jsou nejvíce relevantní k zadanému dotazu. Dalším krokem po ohodnocení expertních stránek je ohodnocení zbytku výsledků – tzv. cílů. K tomu se využívá vybraných odkazů z expertních stránek. Odkazy se vybírají dle jejich relevance k dotazu.

Do výpočtu hodnocení expertních stránek jsou zahrnuty pouze ty, které jsou indexovány pro alespoň jedno slovo $q \in Q$. To tedy znamená, že na stránce e existuje alespoň jedna klíčová fráze P, která obsahuje alespoň jedno slovo $q \in Q$. Značného urychlení výpočtu lze pak dosáhnout, pokud zahrneme pouze stránky indexované pro všechna slova v dotazu. Pro vlastní výpočet hodnocení je pak rozhodujícím faktorem umístění klíčového slova na stránce – nejvyšší hodnocení má slovo v titulku, následuje nadpis a nejmenší hodnocení má text odkazu. Toto hodnocení klíčové fráze P se označuje jako $LevelScore(P)$.

Dále hodnocení zohledňuje počet indexovaných slov, která nejsou součástí dotazu tzv. $FulnessFactor(P,Q)$. Čím menší rozdíl je mezi množinou slov klíčové fráze P a slov v dotazu, tím lepšího hodnocení stránka dosáhne. Jestliže počet slov, která tvoří frázi P a zároveň netvoří dotaz Q označíme jako m, pak $FulnessFactor(P,Q)$ určíme ze vztahu:

$$FulnessFactor(P,Q) = \begin{cases} 1 \text{ pro } m <= 2 \\ 1 - \frac{(m-2)}{|P|} \text{ pro } m > 2 \end{cases}$$

Postupně jsou pro každou expertní stránku e vypočtena tři různá hodnocení (S_0, S_1, S_2). S_0 je hodnocení vypočtené pro fráze, které jsou indexovány pro všechna slova v dotazu. S_1 nemusí obsahovat jedno z nich a S_2 dvě z nich. Počet slov v dotazu Q označme jako k. Pro každou indexovanou klíčovou frázi P je score vypočteno dle vztahu:

$$S_i(e) = \sum_P LevelScore(P) FulnessFactor(P,Q)$$

Tento výpočet provedeme pro všechny klíčové fráze P obsahující $k - i$ hledaných slov. Závěrečné hodnocení expertní stránky e se pak sestavuje tak, aby nejvíce zvýhodněny byly stránky indexované pro všechna slova v dotazu:

$$ScExpert(e) = 2^{32} S_0(e) + 2^{16} S_1(e) + S_2(e)$$

Jakmile máme k dispozici uspořádaný seznam ohodnocených expertních stránek, můžeme přistoupit k dalšímu kroku – vyhledávaní cílů. Cíle jsou stránky, které jsou cílem odkazů expertních stránek. Aby byla stránka zahrnuta mezi cíle, musí na ni odkazovat nejméně dvě další nepříbuzné stránky. Pro stránky splňující tyto podmínky vypočteme hodnocení cíle. Výpočet probíhá ve třech krocích:

1. Pro každou expertní stránku e odkazující na cíl c, sestrojíme orientovanou hranu (e, c). Pro výběr odkazů (hran) na základě klíčových slov slouží následující kritéria.

   - Odkaz je platný, pokud je slovo s v titulku.
   - Odkaz je platný, pokud se v dokumentu vyskytuje po již zpracovaném nadpisu. Další nadpis, stejné nebo větší úrovně, platnost odkazu ruší.
   - Odkaz je platný, pokud je slovo s v textu odkazu.

   Pro každé slovo $q \in Q$ definujeme $occ(q, c)$, jako počet všech různých klíčových prvků (titulek, nadpis, odkaz), které obsahují slovo q a odkaz (e, c) je díky nim platný. Na základě počtu těchto prvků můžeme provést ohodnocení jednotlivých odkazů (hran). K tomu slouží následující vztah:

   $$ScHrana(e, c) = \begin{cases} 0 \text{ jestliže } occ(q, c) = 0 \text{ pro některé z hledaných slov q} \\ ScExpert(e) * \sum_{q \in Q} occ(q, c) \end{cases}$$

   Hodnocení hrany, je tedy závislé na hodnocení expertní stránky, ze které tato hrana vychází.

2. Dalším krokem je kontrola příbuzných odkazů. Pokud dvě příbuzné expertní stránky odkazují na stejný cíl c, vyřadíme z hodnocení jednu z hran. Pokud má jedna z nich menší hodnocení než 2, vyřadíme tu. Jinak je výběr náhodný.

3. Hodnocení cíle c je součtem hodnocení všech hran $(e, c) \forall e \in E$.

Výstupem vyhledávače je pak seznam i cílů seřazený dle hodnocení c_i.

Hilltop byl po patentování v roce 2003 zakomponován do hodnotící funkce vyhledávače Google a je tak vedle PageRanku druhým z popsaných algoritmů, který je využíván v komerčním vyhledávači.

3.6 Modifikace základních algoritmů

SALSA není jediným publikovaným algoritmem, který vychází z HITS či PageRank. Následníků a rozšíření těchto algoritmů bylo publikováno více. V této práci jsou uvedeny ty významnější modifikace, zabývající se zejména zlepšením stability hodnotící funkce.

Modifikace HITS

Záhy po publikaci základního algoritmu publikovali Bahrat a Henzingerová [10] rozšíření algoritmu HITS nazvané topic distilation. Tato práce kombinuje informaci obsaženou v textu s odkazovou strukturou a z této kombinace vychází nová funkce pro určení důležitosti jednotlivých uzlů a hran grafu W.

Náhodné chování uživatele, podobné jako je u PageRanku či SALSy, přidává k algoritmu HITS algoritmus Randomized HITS [54]. Použitelnost těchto náhodných skoků pro detekci tématu stránky pak prezentuje práce [59].

Zcela jiný přístup – aplikaci teorie pravděpodobnosti pro výpočet hodnocení zdrojů a cílů pak přináší algorimus nazvaný PHITS [24].

Podobně jako originální algoritmus HITS jsou i tyto jeho modifikace v současné době použitelné spíše jako jedna z technik pro shlukování dokumentů.

Modifikace Pageranku

Pagerank byl podroben daleko většímu zkoumání než kterýkoliv jiný podobný algoritmus. Zásadní práce zabývající se personalizací výsledků si podrobně popíšeme v následující kapitole, ale PageRank byl zkoumán a modifikován i v mnoha jiných směrech.

Boldi a Santini se pokusili odstranit náhodnost teleportačního skoku při výpočtu PageRanku. Základem tohoto přístupu je zcela nový vztah pro výpočet PageRank vektoru, kdy je PageRank definován jako racionální funkce koeficientu α [15]. Efektivní aproximace této funkce pak posloužila pro definici TotalRank [16] algoritmu, který provádí výpočet průměrné hodnotící funkce pro všechny možné hodnoty koeficientu α.

Velkým problémem snižujícím kvalitu PageRanku jako hodnotící funkce je odkazový spam. Kromě metod jak detekovat stránky vytvořené pro umělé ovlivňování odkazové struktury, bylo publikováno také několik algoritmů, které by měly být proti spamu odolnější než originální PageRank.

Prvním z nich je algoritmus T-Rank (temporal rank) [13]. Menší ovlivnitelnost spamem a lepší výsledky hodnotící funkce dosahují jeho autoři pomocí kombinace PageRanku a časových informací. Časovou informací rozumíme

čas poslední změny konkrétní stránky, frekvenci, s jakou je obsah stránky aktualizován, ale také informace o tom, kdy a s jakou frekvencí jsou aktualizovány příchozí odkazy na tuto stránku.

Autoři algoritmu vycházejí z předpokladu, že zatímco odchozí odkazy jsou snadno ovlivnitelné autorem stránky, ovlivnění příchozích odkazů je o něco složitější. Rozlišení odkazů na příchozí a odchozí pak dovedlo autory k modifikaci pravděpodobnostního modelu náhodného uživatele. Pravděpodobnost, že si náhodný uživatel zvolí stránku y, jestliže aktuálně má zobrazenu stránku x, je určena kombinací třech funkcí – aktuálnosti stránky y : $f(y)$, aktuálností propojení stránek x, y : $f(x, y)$ a průměrné aktuálnosti všech příchozích odkazů stránky y. Podobně jako pravděpodobnost přechodu pomocí odkazu, je nově definována i pravděpodobnost teleportačního skoku na stránku y.

Přímo jako algoritmus odolný vůči spamu byl navržen Trust Rank[30]. Zaměřuje se na detekci podvodných technik jako jsou odkazové farmy, dvojitá domácí stránka (jedna pro uživatele, druhá vysoce optimalizovaná pro vyhledávače) a dalších. Takové stránky jsou dobře rozpoznatelné pokud se jejich identifikací zabývá člověk, ale jejich strojové rozpoznávání je složitější. Při velikosti webu je ovšem ruční zpracování prakticky nemožné. Autoři tedy nejprve ručně definují malý počet důvěryhodných zdrojů (konkrétně 200) a následně se analyzováním odkazové struktury snaží detekovat další kvalitní stránky. Při vhodné volbě počátečních zdrojů se publikované výsledky zdají být velmi slibné.

Pokusem určovat kvalitu stránek z modelu jejich návštěvnosti je práce [69] J. Tomlina. K modelování návštěvnosti stránek používá fyzikálního modelu maximalizace entropie. Návštěvnost webů je tedy modelována jako proudění tepla.

3.7 Algoritmy s podporou personalizace

Použijeme-li pro řazení výsledků vyhledávání algoritmus s podporou personalizace, znamená to, že dva různí uživatelé A a B dostanou na stejný dotaz Q výsledky seřazené v různém pořadí. Tedy, že řazení výsledků, je přizpůsobeno konkrétnímu uživateli a jeho preferovanému tématu. Ekonom hledající výraz *koruna* bude preferovat jiné řazení výsledků než historik či zahrádkář. Přesnějších výsledků by určitě každý z uvedených uživatelů dosáhl zpřesněním svého dotazu, ale trendem egonomie ovládání vyhledávače je maximální jednoduchost. Ideální by tedy bylo, aby toto zpřesnění provedl vyhledávací server sám.

Personalizace výsledků má dva základní problémy:

1. Jak určit téma zpracovávané stránky? Pokud možno automaticky a

rychle vzhledem k počtu zpracovávaných dat.

2. Jak získat od uživatele informaci o jeho preferovaných tématech? Ideálně tak, aby nemusel dělat navíc nic, než obvykle při své práci dělá.

Problematiku personalizace, tedy oba uvedené problémy, teoreticky řeší a zkoumá řada vědeckých článků. Velkou pozornost ale této oblasti věnují také komerční vyhledávače. Jejich řešení jsou obvykle předmětem obchodního tajemství a zveřejněny jsou pouze náznaky či základní principy. Ve stručnosti se tedy nejprve podívejme na řešení používaná v tomto produkčním prostředí.

Personalizace a komerční vyhledávače

Google potvrzuje pozici světové jedničky i na poli personalizovaného vyhledávání. Personalizace je uživatelům k dispozici již od konce roku 2005 [82]. Založena je na historii konkrétního uživatele, přičemž sběr dat pro tuto historii probíhá automaticky, pokud má uživatel založený účet u Google, případně používá i navigační nástroj Google Toolbar [84]. Těmito nástroji řeší Google problém č.2. – sběr dat od uživatele. Řešení problému č.1. je dle dostupných informací založeno na aplikaci teorie *modular pagerank* [40]. Tento algoritmus je podrobněji popsán v další části práce.

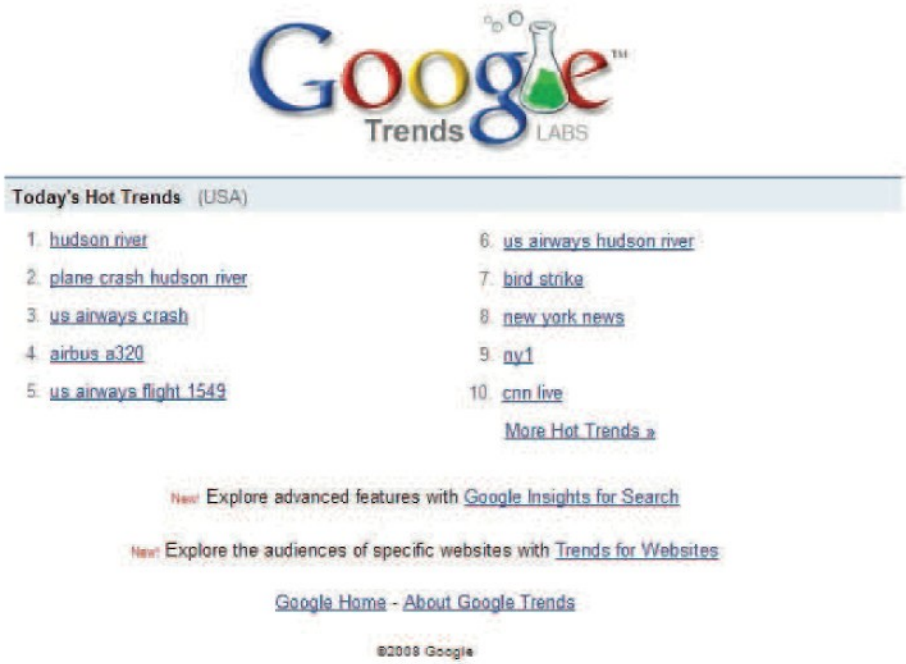
Obdobné služby nabízejí i další velké vyhledávače např. Yahoo [87], nebo Ask [88]. Sběr informací o chování uživatelů provádí i Seznam a patrně i většina dalších společností.

Společným rysem většiny v současnosti použitých řešení je nutnost mít u konkrétní společnosti založen účet, případně si danou službu vůbec aktivovat. Jde jednoznačně o nejsnáze implementovatelné řešení bodu 2. Nutnost registrace a přihlášení ale může být pro některé uživatele překážkou k použití personalizovaného vyhledávání.

Akademické práce o personalizovaném vyhledávání

Tato část práce popisuje aktuální stav problematiky – publikované články a jednotlivé oblasti výzkumu. Algoritmy významné z hlediska této práce jsou pak podrobně rozebrány níže.

Několik autorů si položilo otázku, do jaké míry je vlastně personalizace důležitá a jejich výzkum se zaměřil na analýzu uživatelských dotazů. Co lidé vyhledávají, jaké dotazy zadávají do vyhledávače apod. Za zmínku stojí práce [67], jejímž hlavním autorem je J. Teevan, která poukazuje na fakt, že uživatelé často dávají přednost jednoduchým, méně přesným dotazům, které je jednoduché specifikovat. Na tuto práci navázal Teevan v [68], kde rozebírá



Obrázek 3.3: Google - nejhledanější výrazy dne 16.1.2009

různé motivace, které uživatele vedou k použití vyhledávacího serveru. Poukazuje také na to, že hodnocení řazení výsledků, tedy hodnocení, jak moc je nalezená stránka relevantní vzhledem k dotazu, je silně individuální záležitost. Jedním ze závěrů práce je, že vyhledávače jsou velmi účinné v samotném procesu nalezení výsledků, ale individualizace jejich řazení je nedostatečná. Další prací, která se zabývá problematikou toho, co uživatelé vyhledávají a jak to vyhledávají, je Taxonomie webového vyhledávání [20].

Nejkvalitnější informace o tom, co a jak uživatelé vyhledávají, mají pochopitelně opět komerční vyhledávače. Google tyto trendy ve vyhledávání zveřejňuje v rámci projektu Trends [85] a [86]. Výsledky těchto nástrojů mohou být využity jak pro potřeby teoretických analýz při vyhledávání informací, stejně tak jako určitá forma průzkumu veřejného mínění a průzkum aktuálních světových témat. Jako příklad může posloužit obrázek 3.3, který ukazuje nejvyhledávanější výrazy 16.1.2009 – v den, kdy došlo k nouzovému přistání Airbusu A320 v New Yorku.

Velká pozornost byla a je věnována také řešení problému č. 2, tedy zjišťování uživatelských preferencí. Společným přístupem většiny prací, je zpracování dat, která je možné získat na serveru sledování chování jednotlivých uživatelů. Jednou z metod sledování tohoto chování, je evidence *klikací historie* – tedy vyhodnocování uživatelské interakce se stránkou pomocí myši. Tímto

způsobem navrhuje implementovat automatickou detekci uživatelských preferencí práce [57]. Získané informace jsou pak použity v kombinaci s algoritmem TSPR viz. kap. 3.7.4 nebo [33]. Je-li splněna podmínka udržet velikost uživatelské historie ve zpracovatelných mezích (cca 100 záznamů), dává tato metoda velmi dobré výsledky. Stejný přístup, tedy analýzu „klikání", použili také autoři [65]. Kombinace zadávaných dotazů a následně zvolených výsledků je následně analyzována pomocí SVD rozkladu a výstupem z této analýzy je relace mezi chováním uživatelů, zadávanými dotazy a WWW stránkami. Omezením této práce je ovšem časová náročnost SVD rozkladu.

Další možností, jak zjišťovat uživatelské preference, je zachytávat historii jednotlivých návštěvníků pomocí Proxy serveru. Tento přístup je použit v sytému PROS [38]. Velice zajímavým přístupem je pak využití webových komunit jako zdroje pro získání podkladů pro filtrování výsledků [44].

Nyní si podrobněji popíšeme několik algoritmů zabývajících se řešením problému č.1, tedy automatickou detekcí tématu stránky a následného ovlivnění hodnotící funkce podle tohoto tématu. Společným rysem těchto prací je využití personalizačního vektoru PageRanku. Jde tedy ve všech případech o algoritmy, které k detekci tématu stránky využívají analýzy odkazů a tuto metodu případně doplňují o další.

První tři algoritmy: Topic Sensitive PageRank, Modular PageRank a BlockRank, mají původ na Stanfordské univerzitě. Tedy v prostředí, ze kterého vzešel i samotný Google, původně rovněž pouze akademický projekt. Také řada dalších klíčových výzkumů v oblasti získávání informací na webu má původ na této univerzitě a jde o výsledky projektu WebBase [89]. Označme tedy pro přehlednost tyto tři algoritmy jako WebBase algoritmy. Nejprve si uvedeme matematickou teorii, která je těmto algoritmům společná. Jejich autoři tak učinili v [32].

3.7.1 Matematická teorie WebBase algoritmů

Algoritmus výpočtu Pageranku umožňuje ovlivnit výsledky výpočtu pomocí personalizačního vektoru $\mathbf{v}$. Označme $\mathbf{x}(\mathbf{v})$ n-rozměrný personalizovaný vektor Pageranku, který koresponduje s n-rozměrným personalizačním vektorem $\mathbf{v}$. Vektor $\mathbf{x}(\mathbf{v})$ je opět řešen jako problém nalezení vlastních čísel. Ze vztahů:

$$A = [\alpha\mathbf{P} + (1-\alpha)\mathbf{E}]^T$$

$$\mathbf{x} = A\mathbf{x}$$

dostaneme složením následující:

$$\mathbf{x} = \alpha P^T\mathbf{x} + (1-\alpha)\mathbf{v}$$

$$\mathbf{x} - \alpha P^T \mathbf{x} = (1 - \alpha)\mathbf{v}$$

$$(I - \alpha P^T)\mathbf{x} = (1 - \alpha)\mathbf{v}$$

Kde I je jednotková matice. $(I - \alpha P)$ je diagonálně dominantní a tedy má inverzní matici. Z toho plyne, že také k $(I - \alpha P)^T = (I - \alpha P^T)$ můžeme najít inverzní matici. Díky tomu můžeme pokračovat vztahem

$$\mathbf{x} = (1 - \alpha)(I - \alpha P^T)^{-1}\mathbf{v}$$

Nechť $\mathbf{Q} = (1-\alpha)(I-\alpha P^T)^{-1}$. Pokud dosadíme $\mathbf{v} = \mathbf{e}_i$, kde $\mathbf{e}_i$ je i-tý jednotkový vektor, je patrné, že i-tý sloupec matice $\mathbf{Q}$ je $\mathbf{x}(\mathbf{e}_i)$ a personalizovaný PageRank vektor tedy odpovídá personalizačnímu vektoru $\mathbf{e}_i$.

Sloupce matice $\mathbf{Q}$ tvoří bázi pro personalizované PageRank vektory. Libovolný personalizovaný vektor může být vyjádřen jako konvexní kombinace sloupcových vektorů matice $\mathbf{Q}$. Pro libovolný personalizační vektor $\mathbf{v}$ lze personalizovaný vektor určit jako $\mathbf{Qv}$. To v zásadě odpovídá původní formulaci personalizace Pageranku[56].

Autoři se zde drželi původního návrhu odstranění dangling nodes a tak je matice $\mathbf{Q}$ poměrně hustá, což klade značné nároky na její uchování. Použití celé matice $\mathbf{Q}$ v reálném prostředí je z hlediska výpočetní náročnosti natolik neefektivní, že je prakticky nemožná reálná aplikace mimo prostředí teoretických experimentů. Tento problém autoři obešli redukcí matice $\mathbf{Q}$ a výpočtem tzv. low-rank aproximace. Označme jí $\mathbf{Q}_{red}$. Tato aproximace nám umožní využít personalizovaných vektorů za menších výpočetních nákladů. Místo kompletní báze zvolíme pouze redukovanou bázi, tedy pouze $k \leq n$ personalizovaných PageRank vektorů. Každý z těchto vektorů je sloupcem (nebo konvexní kombinací sloupců) matice $\mathbf{Q}$. Personalizaci libovolného PageRank vektoru tedy provedeme na základě redukované báze takto:

$$\mathbf{x}(\mathbf{w}) = \mathbf{Q}_{red}\mathbf{w}$$

Kde $\mathbf{w}$ je stochastický k-rozměrný vektor.

3.7.2 Modular PageRank

Algoritmus je aplikací teorie grafů a jeho autory jsou G. Jeh a J. Widom [40]. Principem personalizace je nalezení $n \times k$ rozměrné $\mathbf{Q}_{red}$ za použití k vektorů. Tyto vektory reprezentují množinu zdrojů H, kterou tvoří stránky s vysokým PageRankem. Přínosem algoritmu je, že díky navrženým metodám výpočtu aproximace personalizovaných vektorů dobře škáluje a poradí si i s poměrně velkou množinou H. Autoři sérií experimentů ověřili použitelnost algoritmu i pro $k \geq 10^4$.

Matematický model algoritmu je vystavěn na třech větách. První z nich je *věta o linearitě*, která říká, že každý personalizovaný vektor lze vyjádřit jako lineární kombinaci *základních vektorů*. Aplikace této věty pak umožňuje nepočítat personalizované vektory přímo, ale sestavovat je z tzv. *sdílených komponent*. Konstrukce těchto komponent se provádí aplikací *věty o zdrojích*, která říká, jak sestavit základní vektor pomocí parciálních vektorů a *kostry zdroje*. Poslední větou je *věta o rozkladu*, tou se zavádí lineární závislost mezi vektory báze, která je následně využita pro minimalizaci duplicit ve výpočtu. Důkazy těchto vět jsou přílohou práce [40].

Jednotlivé personalizované vektory jsou tedy sestavovány ze zdrojových vektorů, příslušných k určité stránce z množiny H. Zdrojové vektory $\mathbf{r}_p$ jsou tvořeny jako kombinace parciálních vektorů a kostry zdroje. Parciální vektory jsou definovány jako rozdíl mezi přesným vektorem $\mathbf{r}_p$ a jeho aproximací $\mathbf{r}_p^H$. Autoři využívají tranzitivnosti definovaných relací k přímé konstrukci personalizovaných vektorů pomocí parciálních vektorů a kostry zdroje. Právě to zrychluje výpočet, protože parciální vektory jsou pro každou stránku p unikátní a odpadá tak redundance ve výpočtech.

Pro volbu množiny zdrojových stránek H je doporučeno využít stránek s vysokým PageRankem. Aproximace $\mathbf{r}_p^H$ se u stránek s vysokým PageRankem blíží hodnotě přesného vektoru a díky tomu jsou parciální vektory poměrně malé. Což má opět pozitivní vliv na rychlost výpočtu. Proto se doporučuje použít několik stránek s vysokým PageRankem i pro zvláštní množiny H, které je samy o sobě neobsahují.

Algoritmus je velmi propracovaný a publikované experimenty potvrzují jeho efektivitu. Patří mezi nejcitovanější články o problematice personalizace PageRanku a dočkal se i reálné implementace.

3.7.3 BlockRank

Druhým algoritmem z prostředí Stanfordské WebBase je algoritmus Block Rank. Jeho hlavním autorem je S. D. Kamvar [42]. Jestliže předcházející algoritmus byl ukázkou aplikace matematické teorie na reálný problém, Block Rank demonstruje spíše inženýrský přístup k řešení problému. Pro urychlení výpočtu PageRanku i jeho možnou personalizaci totiž využívá část URL zkoumaných webových stránek: *hostname*, tedy jméno serveru.

Studiem jednotlivých web grafů a jejich incidenčních matic bylo zjištěno, že web je rozdělen do bloků neboli menších podgrafů, jejichž uzly jsou navzájem hustě propojeny. Tyto bloky mohou tvořit různé skupiny stránek, ale ne vždy je snadné identifikovat vlastnost společnou všem stránkám v bloku a tak identifikovat blok samotný. Může jít o stránky se stejným tématem, stránky spřátelených autorů apod. Jednou z možných vlastností, kterou lze

navíc velmi dobře identifikovat, je hostname, neboli jméno serveru, na kterém jsou stránky uloženy. A právě tuto vlastnost využili autoři algoritmu pro rozdělení webu na jednotlivé skupiny. Konkrétním příkladem takové skupiny tak mohou být například všechny stránky v doméně www.tul.cz, jejichž URL začíná http://www.tul.cz/.

Jak již bylo naznačeno, tento algoritmus lze využít pro urychlení výpočtu klasického PageRanku. To bylo také primárním cílem při jeho vývoji. Teprve další experimenty prokázaly, že BlockRank lze využít i pro potřeby personalizace. Klíčem k urychlení výpočtu je urychlení jeho konvergence díky přesnějšímu počátečnímu vektoru. Zatímco při běžném výpočtu jsou všechny položky počátečního vektoru stejné, viz kap. 4.1, Kamvar navrhuje provést nejprve aproximaci PageRanku pomocí jednotlivých bloků a tuto aproximaci pak použít jako startovací vektor pro další výpočet. Potřebný počet iterací, stejně jako doba výpočtu jednoho iteračního kroku tím znatelně klesá.

Aproximace PageRank vektoru se provádí ve třech krocích. Nejprve je graf W rozdělen na jednotlivé bloky J_i kde $i \in < 0, k >$ a k je počet unikátních hostname. Každý z bloků B_i lze reprezentovat grafem $G_J = (U, H)$. Množinu uzlů U tvoří všechny stránky s příslušným hostname, množinu hran H pak odkazy vzájemně propojující tyto stránky. Odkazy směřující na jiné bloky jsou v tomto kroku ignorovány, přijdou na řadu později. Pro každý z bloků J_i tedy můžeme vypočítat *lokální PageRank vektor*:

$$\mathbf{l}_J = PageRank(G_J, \mathbf{s}, \mathbf{v})$$

Kde počátetční vektor $\mathbf{s}$ je n_J rozměrný stochastický vektor a všechny jeho položky jsou rovny $1/n_J$, n_J je počet stránek v bloku J a $\mathbf{v}$ je n_J rozměrný personalizační vektor, jehož všechny položky jsou nulové, kromě položky reprezentující kořenovou stránku daného bloku.

Právě tyto lokální PageRanky jsou klíčem pro urychlení výpočtu, protože grafy jednotlivých bloků jsou, v porovnání s grafem celého webu, poměrně malé. Celý výpočet lokálních ranků tak probíhá v paměti počítače a většina grafů svou velikostí umožňuje i uchování iteračního kroku v cache procesoru. Kamvar také prokázal, že pro menší grafy výpočet rychleji konverguje a pro většinu hostname stačí méně než 12 iterací Mocninné metody.

Druhý krok dal celému algoritmu jméno. Po výpočtu lokálních PageRanků je totiž nutné určit důležitost jednotlivých bloků a propojit celý web opět dohromady. Odtud tedy název algoritmu Block Rank – hodnocení bloků. K tomu je opět použit algoritmu PageRank, pro který musíme nejprve sestavit graf bloků G_{blok} a jeho přechodovou matici. Množina uzlů má k prvků a tvoří jí jednotlivé bloky. Všechny odkazy, které propojují stránky ve dvou různých blocích I a J jsou pak reprezentovány jednou ohodnocenou hranou.

Pro ohodnocení této hrany se využije lokální PageRank zdrojové stránky. Jestliže $\mathbf{M}$ je původní matice sousednosti web grafu W, l_i je lokální PageRank stránky $i \in I$, pak hranu $h_{I,J}$ ohodnotíme dle vztahu:

$$h_{I,J} = \sum_{i \in I, j \in J} m_{ij}.l_i$$

Jde o $n \times k$ rozměrnou matici, jejíž sloupec j odpovídá vektoru lokálního PageRanku pro blok J. Pro výpočet hodnocení jednotlivých bloků potřebujeme sestavit přechodovou matici B. Ta je definována jako:

$$\mathbf{B} = \mathbf{L}^T\mathbf{MS}$$

Přičemž $\mathbf{S}$ je $n \times k$ rozměrná matice se stejnou strukturou jako $\mathbf{L}$, ale všechny její nenulové prvky jsou rovny jedné. Matice $\mathbf{B}$ má tedy rozměry $k \times k$.

Nyní již můžeme provést vlastní výpočet Block Rank vektoru $\mathbf{b}$ jako PageRank vektoru matice $\mathbf{B}$. Jako startovací i personalizační vektor, použijeme pravděpodobnostní vektor $\mathbf{v}_k$, jehož všechny položky jsou rovny $1/k$.

Zbývá poslední, třetí krok algoritmu – zkombinovat jednotlivá lokální hodnocení l_J a hodnocení jednotlivých bloků b. Přibližnou hodnotu globálního PageRanku stránky $j \in J$ můžeme vyjádřit jako součin:

$$x_j^{(0)} = l_j.b_j$$

Takto sestavený vektor $x^{(0)}$ je hledaným startovacím vektorem pro efektivní výpočet PageRank vektoru.

Zbývá popsat, jak lze algoritmus BlockRank využít pro personalizaci výsledků a zdůvodnit tak jeho zařazení do této kapitoly. Klíčem k personalizaci je jednoduchá restrikce modelu náhodného chování uživatele. Namísto náhodného skoku (teleportace) na libovolnou stránku je povolen pouze skok na libovolný server.

Tedy uživatel v tomto modelu může do prohlížeče zadat například adresu http://www.tul.cz/, ale už ne adresu http://www.tul.cz/studenti/dokument-1003.html. Touto restrikcí zmenšíme web graf z velikosti n na velikost k tedy na počet jednotlivých unikátních hostname.

Tato restrikce se nijak nedotýká již vypočtených lokálních PageRanků jednotlivých bloků, a tedy ani matice $\mathbf{B}$. Personalizační vektor $\mathbf{v}_k$ je pouze k rozměrný a personalizace probíhá na úrovni druhého kroku algoritmu. Můžeme tedy zvednout preference jednotlivých serverů a díky relativně malému k vypočítat daleko větší množství personalizovaných PageRank vektorů než při klasickém výpočtu.

Do jednotné terminologie WebBase algoritmů zapadá Block Rank následujícím zůsobem. Matice $\mathbf{Q}_{red}$ je stejně jako v předchozím případě $n \times k$ rozměrná a jde o matici jejíž sloupec $j = \mathbf{x}(\mathbf{v}_j)$, přičemž vektor $\mathbf{v}_j$ reprezentuje lokální PageRank daného bloku. Jde tedy o výše popsanou matici $\mathbf{L}$.

Nezodpovězenou otázkou je ovšem efektivita takovéto personalizace. Autoři experimentálně ověřili efektivitu personalizovaného výpočtu a rychlost jeho konvergence. Příliš se ale nezabývají problémem provázanosti hostname s určitým tématem. Z praxe ovšem známe řadu případů, kdy je na jednom serveru umístěna řada stránek, jejichž zaměření je tématicky zcela rozdílné. Dobrým příkladem mohou být například zpravodajské servery – pod hostname domaci.ihned.cz najdeme řadu stránek, které ale mají jako téma společné pouze to, že se jejich obsah nějak dotýká událostí v České republice. Může jít ale o zprávy ekonomické, kriminální, politické a další. Pokud by například uživatel preferoval ekonomická témata, nebude rozdělení webu na bloky podle hostname dostatečné. Block Rank nám umožňuje pouze zvednout preference například tak, aby články v Hospodářských novinách měly větší důležitost než články v MF Dnes, nikoliv to, aby ekonomické články na obou serverech měly větší důležitost než například zprávy o dopravních nehodách.

Block Rank nicméně naznačil cestu, kterou se později vydal výzkum v oblasti shlukování dokumentů dle tématu a bezesporu patří mezi základní algoritmy pro řazení výsledků vyhledávání.

3.7.4 Topic sensitive Pagerank

Posledním ze skupiny WebBase algoritmů popsaných v této práci, je Topic Sensitive PageRank (TSPR). Tento algoritmus propojuje PageRank s kontextem uživatelského dotazu a stránky za pomoci několika speciálně zaměřených Rank vektorů. Jeho autorem je T. Haveliwalla [33].

V terminologii WebBase algoritmů můžeme princip TSPR shrnout následovně. Základním krokem je nalezení $n \times k$ rozměrné $\mathbf{Q}_{red}$ za použití k vektorů generovaných určitým tématem. Počet vektorů k záleží na počtu zvolených základních témat a každý vektor určuje, do jaké míry patří daná stránka právě k určitému základnímu tématu. Pro každou stránku se tedy určuje hned k ranků - kde k je počet základních témat.

Nyní si detailně popíšeme, jak a odkud se volí velikost k a jak se provádí samotné zaměřování vektorů. Pro volbu počtu témat je rozhodujícím faktorem počet stránek, pro které chceme hodnocení pomocí TSPR použít. Pro každé z témat totiž musíme vypočítat kompletní PageRank vektor, zaměřený jiným personalizačním vektorem v_k. Výpočet je tedy k-krát náročnější než

standardní výpočet PR. Vzhledem k nezávislosti jednotlivých témat lze ale výpočet poměrně dobře paralelizovat, neboť jde o přirozeně paralelní problém. Přesto se v případě celého webu doporučuje držet počet zvolených témat v řádu desítek.

Krom výpočetní náročnosti je totiž nutné při volbě k vzít v úvahu také existenci dostatečného množství stránek, u kterých známe jejich téma a které použijeme pro sestavení tématického klasifikátoru T_j, kde $j \in < 1, k >$. Tématický klasifikátor T_j je množina stránek příslušejících tématu j a sestavením všech k množin rozdělíme web nebo jeho část do k tématických bloků. Personalizační vektor $\mathbf{v}_j$ určíme dle následujícího vzorce:

$$v_{ji} = \begin{cases} \frac{1}{T_j} \text{ pokud } i \in T_j \\ 0 \text{ v opačném případě} \end{cases}$$

Dělení skupiny dokumentů na jednotlivé bloky pomocí analýzy jejich obsahu, je netriviální a výpočetně náročná operace. Proto byla pro algoritmus TSPR využita již připravená kolekce několika dokumentů s přesně určenými tématy. Tyto základní dokumenty jsou díky personalizačnímu vektoru na začátku výpočtu PageRank vektoru zvýhodněny a předpokládáme, že odkazují na další dokumenty související s příslušným tématem. Proporcionální předávání PageRanku pak zajistí, že i tyto propojené dokumenty budou ohodnoceny lépe než dokumenty s tématem nesouvisející a tudíž neodkazované. Tím je zajištěno zaměření PageRank vektoru dle zvoleného tématu. Kolekce základních dokumentů musí být dostatečně velká, aby obsáhla maximální možnou množinu propojených stránek. V praxi lze předpokládat, že určité množství odkazů povede i na stránky s jiným nepříbuzným tématem, ale dle publikovaných výsledků TSPR se zdá, že chyba způsobená těmito netématickými odkazy není pro kvalitu hodnocení kritická.

Kde tedy lze najít dostatečně velkou množinu stránek seřazených dle témat? Jednou z efektivních možností je využití odkazového katalogu. Haveliwalla pro zaměření použil Open Directory project (ODP) [78]. Použití ODP určilo, že $k = 16$, což je počet základních tématických skupin. ODP je katalog stránek vytvářený a udržovaný komunitou uživatelů – podobně jako například otevřená encyklopedie Wikipedia. Otevřenost katalogu dává dle autora algoritmu záruku, že jednotlivé skupiny stránek nebudou ovlivněny komerčním zájmem autora katalogu.

Sestavení matice $\mathbf{Q}_{red}$ je prvním krokem algoritmu TSPR. Protože jde o výpočet několika PageRank vektorů, je možné ho pochopitelně provést offline při zpracování jednotlivých stránek. Větší výpočetní náročnost oproti normálnímu PageRanku tak není nepřekonatelnou překážkou pro nasazení TSPR v praxi.

Druhý krok algoritmu tedy musí logicky probíhat online, při zpracování uživatelského dotazu. Tento fakt je společný všem algoritmům, které pracují s kontextem stránky a uživatelského dotazu. A právě výpočetní náročnost tohoto kroku je rozhodující pro určení použitelnosti konkrétního algoritmu v praxi.

TSPR využívá pro zpracování dotazu Q v jeho případném kontextu Q' pravděpodobnostního modelu. Kontext dotazu Q' vzniká v případě, že dotaz je položen tak, že uživatel označí část textu na stránce u. Pak kontext dotazu tvoří slova, která stránka u obsahuje. Tento přístup zadávání dotazů se ovšem v praxi prakticky nevyužívá, takže reálnějším případem bude zadání dotazu Q bez kontextu. V takovém případě platí, že $Q' = Q$.

Každé z k témat je reprezentováno vlastní třídou c_j, kde $j \in < 1, k >$. Pomocí multinomického Bayesovského klasifikátoru vypočteme pro každé ze slov $q_i \in Q'$ pravděpodobnost jeho příslušnosti k třídě c_j podle vztahu:

$$P(c_j|Q') = \frac{P(c_j).P(Q'|c_j)}{P(Q')} \propto P(c_j) \prod_i P(q_i|c_j) \tag{3.6}$$

Pravděpodobnost $P(q_i'|c_j)$ vypočítáme za pomoci klasifikačního vektoru $\mathbf{D}_j$. Tyto vektory sestavíme během prvního kroku algoritmu offline a to tak, že položky příslušného vektoru tvoří frekvence výskytu všech slov, která obsahují stránky tvořící téma j. Velikost vektorů $\mathbf{D}_j$ je tedy pro jednotlivá témata různá a závisí na počtu slov v příslušných stránkách. Pravděpodobnost $P(c_j)$ se volí jako uniformní, ale Haveliwalla naznačuje možnost další personalizace pro jednotlivé uživatele právě pomocí této pravděpodobnosti.

Jestliže označíme množinu všech dokumentů, které vyhovují dotazu q jako Z, pak ohodnocení jednotlivých dokumentů $z \in Z$ provedeme takto:

$$s_{qz} = \sum_j P(c_j|q').r_{jd} \tag{3.7}$$

Ověření lokalizovatelnosti tohoto algoritmu pro české prostředí je jedním z cílů této disertační práce. Zhodnocení použitelnosti algoritmu pro reálné prostředí bude popsáno jako součást výsledků lokalizace.

3.7.5 Intelligent Surfer

Tento algoritmus nevznikl v prostředí WebBase, ale opět vychází ze základního Pageranku a z principu náhodné procházky webovým grafem. Kromě toho byla data ze Stanfordského projektu použita pro testování tohoto algoritmu. Jeho autory jsou M. Richardson a P. Domingos[60]. Namísto náhodného chování předpokládají autoři u uživatele určitou inteligenci - tedy

sledování odkazů, které jsou relevantní vzhledem k původnímu dotazu, případně náhodný skok na stránku, která ale opět je v relaci k původnímu dotazu. Odtud tedy název celého algoritmu - Intelligent Surfer (IS).

Základní myšlenkou tohoto algoritmu je nahrazení modelu zcela náhodně jednajícího uživatele modelem uživatele, který přemýšlí – tedy chová se inteligentně. Reálný uživatel nepřechází mezi stránkami zcela náhodně, tak jak to předpokládá Markovovský model náhodné procházky. Vybírá si stránky podle jejich obsahu a podle klíčových slov, která vyhledává. Pravděpodobnost, že skončí na stránce j, označme jako P_j. Tato pravděpodobnost určíme tak, že pravděpodobnost náhodného skoku sečteme s pravděpodobností přechodu na další stránku pomocí některého z odkazů. Na rozdíl od klasického výpočtu PageRanku ale tuto pravděpodobnost vyjádříme v návaznosti na dotaz uživatele označený jako q.

$$P_q(j) = (1-\alpha)P'_q(j) + \alpha \sum_{i \in B_i} P_q(i) P_q(i \rightarrow j) \tag{3.8}$$

Kde B_i je množina příchozích odkazů na stránku i, $P_q(i \rightarrow j)$ pravěpodobnost přechodu na stránku j využitím odkazu, α je teleporační koeficient (stejně jako u PageRanku) a konečně $P'_q(j)$ je pravděpodobnost přechodu na jinou stránku přímým zadáním adresy (tedy teleportace).

Principem algoritmu je rozdělení celého původního grafu W na velký počet menších podgrafů. Klíčem pro rozdělení grafu je množina slov S – slovník všech slov. Pro každé slovo $s_i \in S$ tvoří podgraf původního grafu W ty uzly, které obsahují v textu slovo s_i. Pro takto vygenerovaný podgraf provedeme výpočet PageRanku obvyklým způsobem.

Při sestavování matice G_{si} pro podgraf generovaný slovem s_i stanovíme jak pravděpodobnost přechodu pomocí odkazu, tak pravděpodobnost teleportace v závislosti na funkci $R_q(j)$. Což je funkce určující důležitost slova q ve stránce j. Pravděpodobnosti vyjádříme takto:

$$P'_q(j) = \frac{R_q(j)}{\sum_{n \in W} R_q(n)} \qquad P_q(i \rightarrow j) = \frac{R_q(j)}{\sum_{n \in F_i} R_q(n)} \tag{3.9}$$

Kde $n \in W$ je některý z uzlů webgrafu a F_i je množina odchozích odkazů ze stránky i.

Jako funkci $R_q(j)$ můžeme použít libovolný ze vzorců obvykle využívaných pro výpočet váhy slova v dokumentu. Může jít o prostou frekvenci slov, inverzní frekvenci TFIDF a další. Základní jednoduchou variantou je binární funkce $R_q(j)$, použitá v originálním návrhu algoritmu.

$$R_q(j) = \begin{cases} 1 \text{ pokud } q \in j \\ 0 \text{ v opačném případě} \end{cases}$$

Výsledný QD-rank pro dotaz Q tvořený množinou slov $Q = q_1, q_2, .., q_n$ je pak lineární kombinací vektorů jednotlivých slov. Algoritmus omezuje výsledky pouze na ty dokumenty, které vyhovují všem slovům z dotazu Q.

Z hlediska algoritmu jsou tedy důležité ty stránky, které obsahují klíčová slova originálního dotazu. Pro každé z klíčových slov, která se na webu mohou vyskytovat, je proveden výpočet Pageranku, a to ze stránek, které klíčové slovo obsahují. Tento přístup ale přináší hned několik problémů.

První problém představují nepříliš frekventované výrazy. Aby se výraz mohl do výpočtu zahrnout, musí být jak součástí určité stránky, tak stránek, které na ní odkazují. Pro nepříliš frekventované výrazy je pak výpočet postaven na příliš malé množině a může být snáze ovlivnitelný spamem.

Druhým problémem je škálovatelnost algoritmu. Pro celkovou dobu výpočtu je rozhodujícím prvkem počet slov ve slovníku S. Jestliže označíme počet dokumentů obsahujících slovo q jako d_q, pak sečtením d_q pro všechna slova ze slovníku dostaneme U – počet unikátních dvojic dokument–slovo. Poměr mezi celkovým počtem těchto dvojic a celkovým počtem stránek (tedy počtem uzlů v grafu W) zároveň určuje, o kolik bude výpočet IS náročnější oproti klasickému PageRanku, stejně jako diskové nároky pro uložení vypočtených QD-ranků.

Autoři algoritmu IS uvádějí tento poměr roven 165, když pracovali s množinou 2,3 mil. unikátních slov. Diskové nároky algoritmu jsou tedy podle nich 165x větší, výpočetní náročnost pak 124x větší. Lepší poměr u výpočtu je zdůvodněn empirickým pozorováním rychlejší konvergence menších grafů, což vede k redukci poměru pro výpočet na 0,75*U/N.

Teoretický poměr mezi náročností výpočtu algoritmů PageRank a Intelligent Surfer je tedy výrazně menší než první hrubý odhad, který nás napadne při vyslovení věty o tom, že pro každé ze slov se provádí výpočet PageRank vektoru, protože obvyklá velikost slovníku S je několik stovek tisíc slov. V době publikace tohoto algoritmu se ale výpočet jednoho PageRank vektoru odehrával v řádu hodin, což nasazení algoritmu IS v reálném prostředí prakticky znemožňovalo.

Ověření možnosti lokalizace algoritmu IS a jeho urychlení tak, aby byla možná jeho reálná implementace je dalším z cílů této disertační práce.

Kapitola 4

Výpočet PageRank vektoru

Většina algoritmů popsaných v předchozí kapitole je odvozena od algoritmu PageRank. Jejich společným požadavkem je tedy nutnost efektivního a rychlého výpočtu PageRank vektoru, který obvykle probíhá vícekrát. Následující kapitola popisuje tradiční postup výpočtu PageRank vektoru pomocí Mocninné metody, alternativní výpočet řešením řídkého lineárního systému a přehled dalších dosud publikovaných prací zabývajících se problematikou urychlení výpočtu PageRank vektoru.

4.1 Vlastní vektor a mocninná metoda

Základním přístupem k výpočtu PageRank vektoru, je určit ho jako vlastní vektor matice $\mathbf{G}$ Mocninnou metodou. Ta je základní numerickou metodou pro nalezení vlastních čísel. Mocninná metoda není považována za příliš efektivní, ale je základem některých pokročilejších metod. V případě PageRank problému dává překvapivě dobré výsledky a zejména je velmi efektivní z hlediska paměťových nároků výpočtu. Za dobu existence algoritmu PageRank sice výkon a paměťová kapacita počítačů lineárně rostly, ovšem stejně tak rostl i počet WWW stránek a tím dimenze matice $\mathbf{G}$.

Řešíme tedy problém nalezení vlastního čísla:

$$\mathbf{x}^T\mathbf{G} = \mathbf{x}^T$$

Iterační krok Mocninné metody má pak podobu

$$\mathbf{x}^{(k)T} = \mathbf{x}^{(k-1)T}\mathbf{G}$$

Od tohoto vztahu s maticí $\mathbf{G}$ můžeme ale zpětným dosazením dojít až k následujícímu vztahu:

$$\mathbf{x}^{(k)T} = \alpha\mathbf{x}^{(k-1)T}\mathbf{M} + (\alpha\mathbf{x}^{(k-1)T}\mathbf{a}) + (1-\alpha)\mathbf{v}^T$$

Kde $\mathbf{v}^T$ je personalizační vektor a $\mathbf{a}$ je vektor reprezentující dangling nodes. Platí, že $a_i = 1$, pokud je stránka i dangling node (tedy nemá odchozí linky), jinak je $a_i = 0$.

Jednotlivé prvky výsledného vektoru $\mathbf{x^T}(i)$ pak představují výsledný $\mathbf{Rank}$(i).

Právě díky tomuto zpětnému dosazení je Mocninná metoda tak paměťově efektivní. Není totiž nutné uchovávat celou matici $\mathbf{G}$, ale pouze nenulové prvky extrémně řídké matice $\mathbf{M}$: m_{ij}. Největší paměťové nároky tak představuje uchování dvou iteračních kroků vektoru $\mathbf{x}$.

Rychlost konvergence Mocninné metody závisí na velikosti α. Čím blíže je α jedné, tím je konvergence pomalejší, ale výsledný $\mathbf{Rank}(i)$ více odráží hyperlinkovou strukturu webu. Menší hodnoty α naopak dávají větší důraz na teleportaci (náhodné zvolení stránky) a vedou také k rychlejší konvergenci výpočtu. Jako optimální se zdá hodnota $\alpha = 0,85$ [48]. Zároveň tato hodnota říká, že v přibližně 1/6 případů dojde k teleportaci a ve zbývajících případech využije uživatel některého odchozího odkazu.

Jako kritérium přesnosti konvergence se obvykle využívá L norma definovaná jako:

$$\delta_k = ||\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}||_p$$

Jde o L_p normu vektoru rezidua pro $p = 1$ či $p = 2$. Požadovaná přesnost pak přímo ovlivňuje počet iterací Mocninné metody. Pro $\delta = 10^{-5}$ jde o přibližně 40 iterací [43], pro $\delta = 10^{-10}$ je to pak přibližně 100 iterací [48]. V práci [43] je věnována pozornost také dalším metrikám pro přesnost konvergence založeným na Kendallově τ-rank metrice. Závěr experimentu však je, že v tomto případě je L_1 norma dostatečně přesná.

4.2 Homogenní lineární systém

V [48] rozvíjí Langville a Meyer krom jiného také alternativní přístup k výpočtu. Z problému nalezení vlastního čísla lze odvodit následující lineární systém:

$$\mathbf{x}^T(\mathbf{I} - \alpha\mathbf{S}) = (1-\alpha)\mathbf{v}^T$$

Pro normalizaci tohoto systému použijeme vztah $\mathbf{x}^T\mathbf{e} = 1$. Matice $(\mathbf{I} - \alpha\mathbf{S})$ má řadu zajímavých vlastností. Uveďme alespoň část:

- Jde o M-matici [14] – tedy matici, která má na hlavní diagonále nezáporné prvky a mimo ní čísla záporná nebo nulu.

- Jde o regulární matici - tedy lze najít matici inverzní.
- Tato inverzní matice je nezáporná.
- Součet řádku matice je roven $(1 - \alpha)$.

Zpětnou substitucí lze opět dojít až k lineárnímu systému využívajícímu původní extrémně řídkou matici $\mathbf{M}$. O výhodách použití této matice byla již řeč v předchozí kapitole. Platí-li, že $\mathbf{S} = \mathbf{M} + \mathbf{a}\mathbf{v}^T$, můžeme lineární systém přepsat do následujícího tvaru:

$$\mathbf{x}^T(\mathbf{I} - \alpha\mathbf{M} - \alpha\mathbf{a}\mathbf{v}^T) = (1 - \alpha)\mathbf{v}^T$$

dále označme $\mathbf{x}^T\mathbf{a} = \gamma$, lineární systém pak lze upravit do tvaru:

$$\mathbf{x}^T(\mathbf{I} - \alpha\mathbf{M}) = (1 - \alpha + \alpha\gamma)\mathbf{v}^T$$

Skalární součin γ je hodnotou součtu x_i pro i dangling nodes. Pokud v závěru budeme výpočet normalizovat podle rovnice $\mathbf{x}^T\mathbf{e} = 1$, můžeme zvolit γ libovolně. Meyer a Langvile volí $\gamma = 1$ a lineární systém formulují jako:

$$\mathbf{x}^T(\mathbf{I} - \alpha\mathbf{M}) = \mathbf{v}^T \text{ přičemž } \mathbf{x}^T\mathbf{e} = 1$$

Matice $(\mathbf{I} - \alpha\mathbf{M})$ má řadu stejných vlastností jako matice $(\mathbf{I} - \alpha\mathbf{S})$, ale i některé další:

- $(\mathbf{I} - \alpha\mathbf{M})$ je také regulární M-matice
- Součet řádků $(\mathbf{I} - \alpha\mathbf{M})$ je roven 1, pokud řádek reprezentuje dangling node (dále DN), případně $1 - \alpha$ pokud jde o normální stránku.
- Inverzní matice $(\mathbf{I} - \alpha\mathbf{M})^{-1}$ je nezáporná.
- Součet řádků $(\mathbf{I} - \alpha\mathbf{M})^{-1}$ je roven 1, pokud řádek reprezentuje DN, případně $\frac{1}{1-\alpha}$ pokud jde o normální stránku.
- Řádek matice $(\mathbf{I} - \alpha\mathbf{M})^{-1}$ reprezentující DN i je $\mathbf{e}_i^T$, kde $\mathbf{e}_i$ je i-tý sloupec matice $\mathbf{I}$.

Poslední vlastnost umožňuje rapidní zjednodušení výpočtu Pageranku. Matici $\mathbf{M}$ je možné uspořádat tak, aby uzly reprezentující DN byly na spodních řádcích matice až za uzly reprezentujícími normální stránky.

$$\mathbf{M} = \begin{pmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \tag{4.1}$$

Kde submatice $\mathbf{M}_{11}$ reprezentuje normální uzly a $\mathbf{M}_{12}$ DN, které jsou cílem odkazů. Dolní řádky pak reprezentují DN, které pochopitelně nikam neodkazují. Matice $(\mathbf{I} - \alpha\mathbf{M})$ pak bude vypadat následovně:

$$(\mathbf{I} - \alpha\mathbf{M}) = \begin{pmatrix} \mathbf{I} - \alpha\mathbf{M}_{11} & -\alpha\mathbf{M}_{12} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}$$

a její inverze:

$$(\mathbf{I} - \alpha\mathbf{M})^{-1} = \begin{pmatrix} (\mathbf{I} - \alpha\mathbf{M}_{11})^{-1} & \alpha(\mathbf{I} - \alpha\mathbf{M}_{11})^{-1}\mathbf{M}_{12} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}$$

Pagerank vektor $\mathbf{x}^T = \mathbf{v}^T(\mathbf{I} - \alpha\mathbf{M})^{-1}$ můžeme rozepsat jako:

$$\mathbf{x}^T = \left(\mathbf{v}_1^T(\mathbf{I} - \alpha\mathbf{M}_{11})^{-1} \;\middle|\; \alpha\mathbf{v}_1^T(\mathbf{I} - \alpha\mathbf{M}_{11})^{-1}\mathbf{M}_{12} + \mathbf{v}_2^T\right)$$

Personalizační vektor byl rozdělen podle DN stejně jako předtím matice. Algoritmus výpočtu vektoru $\mathbf{x}^T$ je následující:

Algoritmus LangMey

1. uspořádáme stavy Markovova řetězce tak, aby vzniklá matice měla strukturu (4.1)
2. vypočteme $\mathbf{x}_1^T$ ze vztahu $\mathbf{x}_1^T = \mathbf{v}_1^T(\mathbf{I} - \alpha\mathbf{M}_{11})^{-1}$
3. vypočteme $\mathbf{x}_2^T = \alpha\mathbf{v}_1^T\mathbf{M}_{12} + \mathbf{v}_2^T$
4. normalizujeme výsledný vektor $\mathbf{x}^T = \frac{[\mathbf{x}_1^T\,\mathbf{x}_2^T]}{\|[\mathbf{x}_1^T\,\mathbf{x}_2^T]\|_1}$

V závislosti na počtu DN v množině stránek může být redukce výpočtu velice výrazná. Pro řešení lineárního systému spojeného s přeuspořádáním matice je možné využít řadu dalších iteračních metod (nejen Mocninnou) nebo dokonce některou přímou metodu (v závislosti na velikosti množiny stránek). Urychlení výpočtu je nutnou podmínkou pro efektivní nasazení personalizovaných verzí základního algoritmu PageRank.

4.3 Urychlení výpočtu PageRank vektoru

První práce zabývající se problematikou efektivity výpočtu PageRank vektoru a možnostmi jeho urychlení se začaly objevovat záhy po publikaci samotného algoritmu. Počet publikací na toto téma jde do desítek. Motivace pro tento rozsáhlý výzkum má dva základní důvody:

- Web graf je velmi rozsáhlý a velmi často se mění. Pro vyhledávače je tak nutné prakticky neustále sledovat jeho změny. Výpočet PageRank vektoru je tak nutné často opakovat, protože vypočtený vektor se rychle stává neaktuálním.
- Pro potřeby personalizovaných algoritmů, nebo algoritmů detekujících link-spam je potřeba počítat desítky až stovky PageRank vektorů.

Publikace zabývající se urychlením výpočtu lze rozdělit do několika skupin, podle toho jakou metodiku pro urychlení výpočtu používají. Přístupy jsou různé od modifikace matematického aparátu, přes komprimaci web grafu až po paralelní výpočty. Následující přehled představuje nejzásadnější práce, rozdělené do pěti skupin:

- První skupinu tvoří práce [34, 36] řešící efektivní výpočet PageRank vektoru s využitím hlavní paměti (RAM) počítače. Zpracování grafu W je realizováno pomocí opakovaných průchodů grafem.
- Zvolením vhodných komprimačních postupů lze v paměti počítače uchovat celý graf W, což má pozitivní vliv na rychlost výpočtu. Komprimačními technikami pro rozsáhlé grafy se zabývají práce [17, 58].
- Nejvíce prací se zabývá možností použít jiné numerické metody pro nalezení PageRank vektoru. Dle publikovaných výsledků v článcích [8, 21, 23, 48, 41, 42] se zdá, že právě efektivní uplatnění numerických metod je nejlepší cesta k rychlejšímu výpočtu.
- Paralelizací výpočtu jako další metodou urychlení se zabývají práce [28, 61, 71, 72]. Obvyklé problémy představují rozklad a vyvažování zátěže, škálovatelnost řešení a jeho efektivita.
- Poslední skupinu prací, představují různé přístupy k aproximaci řešení [37, 22, 49].

Nyní se podrobněji podíváme na některé z uvedených prací. Začneme metodami pro komprimaci web grafů a naznačením paměťových nároků celého výpočtu.

4.3.1 Komprimace Web grafu

Zdrojová data reprezentující web graf jsou obvykle uložena ve formě dvou souborů. První z nich je soubor obsahující seznam jednotlivých URL, který může být pro lepší efektivitu abecedně uspořádaný. Druhý ze souborů obsahuje informace o vzájemných propojeních jednotlivých URL. Jde opět o jednoduchý soubor, každý z jeho řádků obsahuje dvojici čísel (zdroj,cíl). Zdroj i cíl odpovídají příslušnému řádku v souboru s URL.

Ucelený přehled většiny použitelných komprimačních technik pro redukci velikosti těchto zdrojových dat představuje práce A. Boldiho [17]. Boldi a spol. jsou autory frameworku nazvaného příznačně *WebGraf* a tato práce představuje přehled komprimačních technik, které jsou v tomto prostředku k dispozici.

Většina komprimačních technik se zabývá redukcí datového prostoru potřebného pro uchování jednoho URL či odkazu. Chen v [36] odhaduje, že pro uchování jednoho odkazu je v komprimovaném souboru potřeba 3,5 bitu. Randalem [17, 58] publikovaná technika redukuje tuto velikost z 32bitového čísla na 6 bitů. Chen odhaduje, že pro uchování celého web grafu je potřeba asi 8GB paměti. Je pochopitelné, že po devíti letech jde o značně zastaralý odhad. Pro určení aktuálních paměťových nároků vyjdeme z odhadu velikosti, uvedeném v kap. 2.3.2, a odhadu o průměrném počtu odkazů ze stránky publikovaného v [46].

Paměťové nároky celého web grafu o velikosti $25,5 \times 10^9$, při průměrně $7,2$ odkazech ze stránky a $3,5$ bitu potřebných pro uchování jednoho odkazu jsou přibližně 75 GB. Je tedy zřejmé, že sebelepší komprimační technika nám na současném běžně dostupném hardware neumožní udržet celý graf W v paměti po dobu výpočtu. Paměťové nároky grafu českého webu jsou ale již v hranicích současné techniky - pro graf o velikosti 110 mil. stránek potřebujeme přibližně 330 Mb.

4.3.2 Efektivní využívání paměti

Při výpočtu PageRank vektoru Mocninnou metodou musíme v paměti uchovat minimálně dvě po sobě jdoucí iterace a řídkou matici **M**. Jestliže kapacita interní paměti počítače není dostatečná pro uchování těchto dat, musíme využít externí paměť – zpravidla pevný disk. Nízká rychlost (v porovnání s RAM) I/O operací z této externí paměti výpočet pochopitelně zpomaluje. Havellivala v [34] navrhuje rozdělit výsledný PageRank vektor na d bloků D_i. Počet bloků se určuje tak, aby se každý blok vešel do interní paměti počítače. Výpočet pak probíhá po jednotlivých blocích. Protože sestavování bloků vyžaduje opakované procházení souboru s odkazy, které je náročné na

čas, je součástí řešení také preprocessing souboru s odkazy a jeho rozdělení na d souborů L_i. Každý ze souborů L_i obsahuje pouze odkazy směřující na stránky v bloku D_i výsledného vektoru. Odkazy jsou řazeny dle zdrojového uzlu. V každém iteračním kroku je zpracován soubor zdrojových uzlů S a jeden L_i soubor. Výsledkem výpočtu je blok D_i, kde $i \in (0, d)$. Pro škálovatelnost tohoto algoritmu je určující velikost dostupné paměti. Pro každý výpočet je soubor S nutno načíst celkem d-krát. Pro zpracování rozsáhlých dat je tedy nutné držet d na nízké úrovni.

V již zmíněné práci [36] navrhuje Chen dva algoritmy pro efektivní výpočet. První z nich je založen na modifikaci algoritmu merge-sort a celý výpočet je rozdělen do několika opakovaných třídících kroků. V každém iteračním kroku dojde k předání hodnoty Rank() ze zdrojové stránky na cílovou, pro každý z odkazů. Předávání se uskutečňuje pomocí osmibajtových paketů. Každý z odkazů je reprezentován jedním paketem, který obsahuje kód cíle (číslo řádku v souboru S) a příslušnou hodnotu Rank(). Směrování těchto paketů se uskutečňuje pomocí třídění podle příslušného cíle. Hodnocení cílového uzlu se pak získává z kombinace paketů, které jsou na něj směrovány.

Ve druhém z algoritmů Chen kombinuje svůj předchozí návrh s několika postupy z Havellivalova algoritmu [34]. Opět je využito rozdělení souboru s odkazy L na několik bloků L_i. Zdrojový vektor, reprezentující data v souboru S, je rozdělen na d bloků S_i. Pro uchování Rank() hodnoty příslušného uzlu jsou potřeba 4 bajty – počet bloků d se tedy určuje tak, aby se každý z nich vešel do paměti. Každý ze souborů L_i obsahuje informaci o všech odkazech, jejichž zdrojem je stránka spadající do bloku S_i. Blok L_i je setříděn dle cílových stránek. Na rozdíl od Havellivaly neukládá Chen do bloku L_i informaci o počtu odchozích odkazů pro každý zdroj z bloku S_i. Tato informace označená jako *out-degree* je uložena do speciálního souboru označeného O_i. Pro uchování paketů s Rank() hodnotou, jejichž cílem je stránka v bloku D_i, slouží soubor P_i. Ve všech případech je $i \in (0, d)$.

V každém iteračním kroku Mocninné metody se provede výpočet pro všech d bloků. Nejprve je inicializován blok S_i – počáteční hodnota se nastaví v závislosti na startovacím vektoru celého iteračního procesu. Ten je obvykle nastaven jako uniformní pravděpodobnostní vektor. Jednotlivé položky vektoru S_i jsou inicializovány jako $\frac{(1-\alpha)R(0)}{n}$, kde $R(0)$ je součet počátečního Rank() vektoru, tedy obvykle 1. Dalším krokem algoritmu je načtení paketů směřujících do bloku D_i. Pakety se načítají ze souboru P_i a po jeho zpracování jsou načtena data ze souboru O_i. Každá z Rank() hodnot v souboru S_i je vydělena příslušným out-degree. Následně přichází na řadu zpracování souboru L_i. Pro každý jeho řádek, obsahující několik různých zdrojů a cílů (podle počtu propojených stránek) je sestaven jeden paket. Ten obsahuje informaci o cílové stránce a součet přenášeného hodnocení Rank(). Pakety

jsou ukládány do souboru, ze kterého je vytvořen nový soubor P_i pro další iterační krok.

4.3.3 Aplikace numerických metod

Hned několik autorů navrhuje převést výpočet PageRank vektoru Mocninnou metodou na řešení řídkého lineárního systému. Ten pak otevírá možnost použití různých numerických metod pro výpočet PageRank vektoru. Kromě řešení, které navrhují Langville a Meyer v [48] a které jsme si podrobně popsali v kap. 4.2, navrhují řešení pomocí lineárního systému také Arasu v [8] a del Corso [23]. I když k definici lineárního systému docházejí autoři různou cestou, ve výsledku je díky tomu vždy možné vypočítat PageRank vektor rychleji, než při použití tradiční Mocninné metody.

Hned tři práce postupně publikoval na téma urychlení výpočtu PageRanku prof. S. Kamvar s kolektivem spoluautorů jako výsledky výzkumu v projektu WebBase. Krom již zmíněného algoritmu Block Rank [42] navrhuje v práci [43] urychlit výpočet pomocí extrapolačních metod, které slouží pro opakované odhadování vedlejších vlastních vektorů v jednotlivých iteračních krocích Mocninné metody.

Poslední navrženou metodu využívající různé rychlosti konvergence jednotlivých částí PageRank vektoru, představuje práce [41]. Tuto metodu si popíšeme poněkud detailněji, protože její součástí je algoritmus *MAPR*, který byl součástí testů provedených v rámci této disertační práce. Podle publikovaných výsledků dosáhnou některé části PageRank vektoru stanovené přesnosti konvergence po několika málo iteracích a že je tedy zbytečné je dále znovu a znovu přepočítávat. Práce představuje návrh několika algoritmů, které do dalších iteračních kroků předávají pouze ty části PageRank vektoru, které ještě nedosáhly požadované přesnosti konvergence.

Základem všech těchto algoritmů je přeuspořádání matice $\mathbf{G}$ a k–té iterace vektoru řešení $\mathbf{x}^{(k)}$ takto:

$$\mathbf{x}^{(k)} = \begin{pmatrix} \mathbf{x}_N^{(k)} \\ \mathbf{x}_C^{(k)} \end{pmatrix}$$

a

$$\mathbf{G} = \begin{pmatrix} \mathbf{G}_N \\ \mathbf{G}_C \end{pmatrix}$$

Indexem N budeme nadále označovat ty části řešení, které ještě nedosáhly požadované přesnosti konvergence. Indexem C pak ty, které již do dalšího výpočtu nezarhneme, protože již konvergují dostatečně přesně. Iterační krok Mocninné metody tak můžeme přepsat následovně:

$$\begin{pmatrix} \mathbf{x}_N^{(k+1)} \\ \mathbf{x}_C^{(k+1)} \end{pmatrix} = \begin{pmatrix} \mathbf{G}_N \\ \mathbf{G}_C \end{pmatrix} \cdot \begin{pmatrix} \mathbf{x}_N^{(k)} \\ \mathbf{x}_C^{(k)} \end{pmatrix}$$

Protože $\mathbf{x}_C^{(k)}$ již konverguje, není nutné ho znovu přepočítavát, proto můžeme iterační krok zjednodušit na:

$$\begin{aligned} \mathbf{x}_N^{(k+1)} &= \mathbf{G}_N \mathbf{x}_N^{(k)} \\ \mathbf{x}_C^{(k+1)} &= \mathbf{x}_C^{(k)} \end{aligned} \tag{4.2}$$

Cílem dalšího vývoje algoritmu byla metoda pro periodickou identifikaci již konverujících stránek tak, aby nebylo nutné pokaždé znovu uspořádat matici $\mathbf{G}$. Toho lze dosáhnout dalším přeuspořádáním matice do tvaru:

$$\mathbf{G} = \begin{pmatrix} \mathbf{G}_{NN} & \mathbf{G}_{NC} \\ \mathbf{G}_{CN} & \mathbf{G}_{CC} \end{pmatrix}$$

Kde G_{NN} představuje odkazy mezi stránkami, které ještě nekonvergují, G_{CN} odkazy ze stránek, které již konvergují na stránky, které prozatím nekonvergují atd.

Rovnici (4.2) tak můžeme přepsat do tvaru:

$$\mathbf{x}_N^{(k+1)} = \mathbf{G}_{NN} \mathbf{x}_N^{(k)} + \mathbf{G}_{CN} \mathbf{x}_C^{(k)}$$

Protože $\mathbf{x}_C$ se již během dalších iterací nemění, nemění se dále ani $\mathbf{G}_{CN}\mathbf{x}_C^{(k)}$. Proto je nutné $\mathbf{G}_{CN}\mathbf{x}_C^{(k)}$ přepočítat pouze tehdy, dojde–li ke přeuspořádání matice $\mathbf{G}$. Nyní můžeme zformulovat modifikovaný adaptivní PageRank algorimtus:

Algoritmus MAPR

1. vypočteme $\mathbf{x}_N^{(k+1)} = \mathbf{G}_{NN}\mathbf{x}_N^{(k)} + \mathbf{y}$
2. nastavíme $\mathbf{x}_C^{(k+1)} = \mathbf{x}_C^{(k)}$
3. provedeme detekci konvergence $[N, C] = najdiKonvergenci(\mathbf{x}^{(k)}, \mathbf{x}^{(k+1)}, \epsilon)$
4. vypočteme vektor $\mathbf{y} = \mathbf{G}_{CN}\mathbf{x}_C^{(k)}$
5. vypočteme $\delta = ||\mathbf{G}\mathbf{x}^{(k)} - \mathbf{x}^{(k)}||_1$
6. dokud $\delta > \epsilon$ opakujeme předcházející kroky

7. $\mathbf{x}^{(k+1)}$ je hledaným výsledkem

Kde ϵ představuje požadovanou přesnost konvergence a funkce *najdiKonvergenci* určuje pro každou stránky $i \in W$, zda konverguje podle vztahu:

$$\frac{|x_i^{k+1} - x_i^k|}{|x_i|^k} < \epsilon$$

Pokud je tato podmínka splněna říkáme, že stránka i konverguje.

Takto modifikovaný výpočet PageRank vektoru je dle autorů rychlejší než neupravená Mocninná metoda o 20,3% v případě, že $\epsilon = 10^{-3}$ a o 21,6% v případě, že $\epsilon = 10^{-4}$. Adaptivní metoda potřebuje pro dosažení konvergence všech částí více iteračních kroků, ale výpočetní náročnost jednotlivých iterací je menší.

4.3.4 Paralelní výpočet PageRanku

První prací popisující paralelní řešení lineárního systému publikovali Gleich a Zhukov [28]. V rámci výzkumného projektu společnosti Yahoo implementovali paralelní řešení za pomoci knihovny PETSc. Otestovali řadu numerických metod, mimo jiné Jacobiho metodu a některé metody Krylovových podprostorů. Jako nejefektivnější se dle publikovaných výsledků jeví metoda GMRES. Hlavním problémem, jehož řešení práce příliš nenaznačuje, bylo podle autorů vyvažování zátěže a rozklad dat.

Paralelizaci s využitím iterační agregačně/deagregační (IAD) metody prezentuje práce [72]. Autoři navrhují algoritmus typu rozděl a panuj, ve kterém nejprve dojde k vypočtení lokálních PageRanků pro jednotlivé bloky a z těchto lokálních PageRanků je následně sestaven výsledný vektor. Bloky v tomto algoritmu představují jednotlivé stránky a jejich odkazová propojení. Publikovaná metoda konverguje dle autorů 3-5x rychleji než obvykle používaná metoda Mocninná, což dokládají i výsledky experimentů na třech různých datových kolekcích.

Další paralelní implementaci za využití knihovny MPI představili autoři [61] již v roce 2004. Práce je orientována na návrh paralelního řešiče, který využívá jednoduchý komunikační model pro synchronizaci PageRanků na jednotlivých výpočetních uzlech.

4.3.5 Aproximace výpočtu

Logickou otázkou je, do jaké míry je možné výpočet PageRank vektoru urychlit pomocí aproximace výsledku. Autoři práce [37] navrhují zahájit výpočet na určitém menším podgrafu a vypočtené hodnoty použít jako počáteční

hodnotu PageRanku pro hraniční uzly tohoto podgrafu při klasickém výpočtu Mocninnou metodou. Aproximací za využití grafové agregační metody ze zabývá práce [22]. Jednotlivé uzly grafu jsou rozděleny do bloků podle hostname a následně je hledáno stacionární rozložení Markovova řetězce jako kombinace lokálních stacionárních rozložení (pro jednotlivé bloky) a globálního stacionárního rozložení (vazby mezi jednotlivými bloky). Jde tedy o podobný princip jako u algorimtu Block Rank, rozdíl je ovšem v definici možného přecházení mezi jednotlivými bloky. V tomto návrhu se přechod realizuje pomocí náhodně zvolené stránky ve stejném bloku, který teprve následuje klasický přechod, jak je definován u PageRanku. Dle publikovaných výsledků jsou výpočetní nároky poloviční oproti klasickému výpočtu PageRanku, zatímco odchylka obou vektorů (klasického i aproximovaného) není velká. Využitím vlastností Markovových řetězců se zabývá také další z prací Langville a Meyera [51].

Kapitola 5

Cíle disertační práce

Po prostudování teorie vyhledávání informací na webu a řazení výsledků tohoto vyhledávání pomocí algoritmů odkazové analýzy byly zformulovány tyto cíle disertační práce.

Základní otázkou, jejíž řešení dosud nebylo publikováno, je zda jsou některé z algoritmů spojujících dohromady odkazovou analýzu a obsah dokumentů použitelné univerzálně. Tedy i pro další jazyky, nikoliv pouze pro angličtinu pro níž byly navrženy a testovány. Cílem práce je zjistit, do jaké míry je možné využít tyto algoritmy pro český jazykový model.

Z algoritmů popsaných v teoretické části byly na základě výsledků rešerše zvoleny dva algoritmy, které jsou nejvíce navázány na jazykový model, mají velmi dobré výsledky pro anglický jazyk a zároveň nabízí krom lokalizace i další otevřené problémy k dořešení. Jsou to algoritmy Topic Sensitive PageRank (kapitola 3.7.4.) a Intelligent Surfer (3.7.5). Primárním cílem práce je ověřit lokalizovatelnost těchto algoritmů.

Pro vlastní ověřování lokalizace, je nezbytné navrhnout a implementovat experimentální systém, ve kterém bude možné lokalizované algoritmy testovat a experimentálně ověřit jejich funkčnost. Tento systém by měl být plnohodnotným fulltextovým vyhledávačem. Při jeho vývoji jsou ale možná určitá zjednodušení. A to v oblastech, které nejsou pro řešení primárního cíle práce klíčové, jako například rychlost získání a zpracování základních dat, či rychlost datového úložiště a dalších.

Experimentální systém by měl být schopen přijímat dotazy od uživatele, najít v testovací datové kolekci odpovídající dokumenty a ty setřídit pomocí zvolené hodnotící funkce. Mělo by být také možné jednoduchým způsobem tyto hodnotící funkce měnit, případně kombinovat. Vytvoření experimentálního systému je prvním ze sekundárních cílů této práce.

Princip personalizace je v obou algoritmech stejný v tom, že jde o opakované počítání PageRank vektoru s různým nastavením personalizačního

vektoru. Publikované metody urychlení výpočtu PageRank vektoru jsou popsány v kapitole 4. Přestože pro lokalizaci jako takovou, není podstatné jakou metodou a jak rychle byl PageRank vektor spočítán, je celková rychlost výpočtu lokalizovaného algoritmu klíčovým faktorem určujícím, zda jde o řešení reálně použitelné, nebo řešení které zůstane pouze teoretickou studií. Proto je druhým z vedlejších cílů práce, najít před samotnou lokalizací dostatečně efektivní metodu pro výpočet PageRank vektoru.

Lokalizace každého z vybraných algoritmů pak přináší své vlastní problémy. U algoritmu TSPR je hlavním problémem nalezení dostatečně reprezentativní báze pro zaměřování jednotlivých tématických PageRanků. Stránky této báze jsou využity také pro vytvoření tématických slovníků, které jsou nutné ve druhé fázi algoritmu – identifikaci tématu uživatelského dotazu za pomoci naivního Bayesovského klasifikátoru.

U algoritmu IS je pak hlavním problémem lokalizace rozdílná velikost slovníků slov vyskytujících se na WWW stránkách v češtině a v angličtině. Obecným problémem algoritmu IS je příliš dlouhá doba výpočtu personalizovaných vektorů, což v době publikace tohoto algoritmu znemožňovalo jeho reálné nasazení. Krom urychlení výpočtu metodou nalezenou v rámci sekundárního cíle práce, by mělo být možné urychlit algoritmus i vhodnou aplikací jazykového modelu. Krom procesu lokalizace je tedy cílem práce pokusit se tento algoritmus zefektivnit tak, aby bylo možné jeho reálné nasazení.

Kapitola 6

Experimentální systém

Následující kapitola popisuje experimentální systém vytvořený pro potřeby disertační práce. Jde o fulltextový vyhledávač vytvořený jako kombinace vlastních a open-source řešení.

Vytvoření tohoto experimentálního systému bylo základním krokem k dalším experimentům. Pro testování algoritmů řadících výsledky vyhledávání, získání a zpracování dat, vytvoření web grafů a jejich incidenčních matic a pro řadu dalších operací je nutné mít k dispozici sadu nástrojů, které tyto operace usnadní, nebo dokonce vůbec umožní.

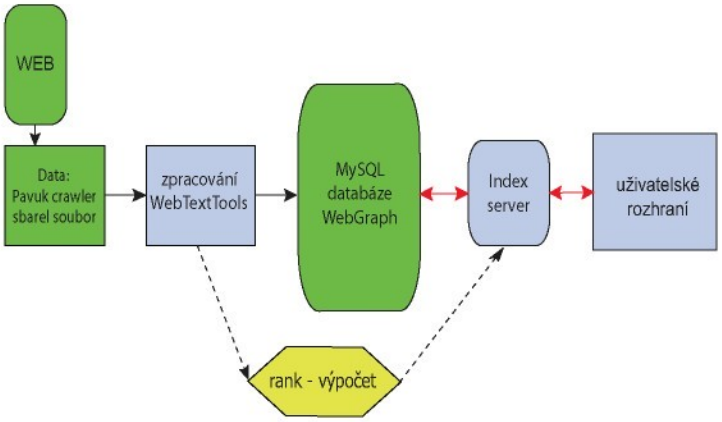
Základní požadavky na experimentální systémy byly následující:

- Efektivní získání a zpracování dat.
- Snadné vytvoření a uložení web grafu.
- Možnost modifikace řadících a indexovacích algoritmů.
- Jednoduché uživatelské rozhraní.

Logickým krokem před zahájením vývoje rozsáhlého programátorského projektu je rešerše dostupných open source řešení. Lze vyslovit předpoklad, že podstatnou většinu problémů, které programátor řeší, už řešil i někdo jiný před ním.

Fulltextové vyhledávání v nejrůznějších kolekcích dokumentů patří mezi hojně využívané aplikace a k dispozici je tak několik různých projektů. Některé z nich jsou velmi kvalitní, jiné se nacházejí v různém stádiu rozpracovanosti. Ve stručnosti si uvedeme alespoň dva nejzásadnější projekty.

Nutch+Lucene – je projekt, vyvíjený pod záštitou Apache software foundation. Je kompletně napsán v jazyce Java, stejně jako vyhledávací technologie Lucene, kterou Nutch rozšiřuje o komponenty potřebné pro ZIW – crawler, HTML parser, nástroje pro analýzu odkazů a další. Lucene je patrně



Obrázek 6.1: Blokové schema experimentálního systému.

nejlepší open source vyhledávacím strojem, který využívá celá řada projektů. (http://wiki.apache.org/lucene-java/PoweredBy)

Projekt **mnoGoSearch** – je zajímavý zejména z hlediska vícejazyčného vyhledávání, protože jeho nástroje podporují i zpracování jiných jazyků, nejen anglického. Podporuje například stemming za pomoci programu Ispell a další.

Po provedené rešerši bylo zřejmé, že existuje několik velmi kvalitních projektů, za jejichž vývojem stojí rozsáhlé týmy. Tyto projekty jsou ovšem díky své propracovanosti poměrně náročné na implementaci, a to zejména ve třetím bodě výše uvedených požadavků, kdy je potřeba modifikace zdrojových kódů podmíněná studiem rozsáhlé dokumentace.

Na opačném konci spektra vývojářových možností je 100% vlastní aplikace. Základní výhodou tohoto přístupu je plná kontrola nad kódem aplikace, kdy přesně víme, co která část programu dělá a nemusíme to složitě zjišťovat jako u kódu cizího. Podstatnou nevýhodou tohoto přístupu je ovšem velmi nízká produktivita práce, zejména jde-li o větší projekt a malý vývojový tým.

Na základě úvah shrnutých do předchozích dvou odstavců bylo rozhodnuto realizovat vývoj experimentálního prostředí jako kombinaci open source a vlastních řešení. Pro realizaci klíčových částí, které plní základní požadavky na aplikaci bylo použito vlastní řešení a pro ostatní části systému pak osvědčené open source produkty.

Obecná architektura vyhledávače je popsána v kapitole 2. Architektura experimentálního systému je pak schematicky zobrazena na obrázku 6.1. Podívejme se nyní postupně na na jeho jednotlivé části.

6.1 Data pro experimenty

6.1.1 Získávání dat - Crawler

Hlavním kritériem pro výběr Crawleru byla jeho snadná implementace a srozumitelný výstupní formát dat. Od počátku bylo plánováno že bude provedeno pouze několik spuštění tohto programu, s cílem získat experimentální data. Proto byl v případě crawleru jasnou volbou nějaký open source program.

OS programů pro sběr dat na webu je k dispozici mnohem více než kompletních fulltextových vyhledávacích strojů. Základním nástrojem je program Wget, který je šířen pod GNU licencí a je součástí většiny Linuxových distribucí. Velmi komplexním programem je pak například Heritrix crawler, který je využíván v projektu internet archive. Po otestování dvou výše uvedených a několika dalších crawlerů byl nakonec pro účely získávání dat pro experimenty použit program Pavuk, dostupný z [77]. Již z názvu je patrné, že jeho prvním autorem je Slovák Štefan Odrejička. Program je šířen pod licencí GNU a v současné době se o jeho vývoj stará menší komunita vývojářů.

K řízení procesu má Pavuk k dispozici širokou škálu voleb a přepínačů. Jeho konfigurace je ale poměrně snadná a to díky přítomnosti grafického rozhraní využívajícího knihovny GTK+. Data jsou ukládána na disk ve formě jednotlivých souborů. Více k formátu ukládaných dat viz dále.

6.1.2 Struktura a příprava dat

Data získaná pomocí crawleru je nutné zpracovat a uložit do připravené databáze (viz níže). K tomu účelu bylo nutné vytvořit sadu nástrojů, která zajišťuje načtení dat z pevného disku, jejich analýzu, případnou úpravu a konečně uložení zpracovaných dat do databáze. Tyto nástroje byly vytvořeny pomocí jazyka Python. Většina z nich byla realizována jako přímo spouštěné skripty s případnou možností konfigurace z příkazové řádky.

Pro experimenty byla k dispozici jednak data získaná pomocí crawleru Pavuk a dále pak data poskytnutá společností Seznam ve formátu Sbarel. V prvním případě jde o kopii adresářové struktury jednotlivých WWW serverů, v případě Sbarelu už jde o data zpracovaná a uložená ve strukturovaném formátu. Pro každý typ dat bylo tedy nutné zvolit jiný přístup.

Pavuk

Data jsou uložena v systému souborů tak, že ve zvoleném adresáři je postupně vytvářen obraz jednotlivých WWW serverů. Základem struktury je

vytvoření adresáře ve tvaru *jménodomény_port*. Do tohoto adresáře jsou pak ukládána data z konkrétní domény včetně replikace adresářové struktury. Například url http://www.tul.cz/karty/checkcard.php je uloženo do adresáře /data/www.tul.cz_80/karty/ jako soubor checkcard.php. Adresář /data/ představuje kořenový adresář pro ukládání.

Pro zpracování dat je potřeba procházet rekurzivně stromovou strukturou adresáře *data* a vytvořit seznam dvojic url+soubor. Tento úkol zajišťuje skript s názvem *lister.py*, který realizuje rekurzivní procházení pomocí Python modulu *os*. Výstupem skriptu jsou dvojice url, soubor ve formátu csv, oddělené středníkem a doplněné pořadovým číslem. Skript posílá data na standardní výstup.

Dalším krokem zpracování už je procházení seznamu souborů vygenerovaného programem lister a postupné zpracování jednotlivých dokumentů. To je úkolem programu *filer.py*. Tento skript vygeneruje pro každý soubor instanci objektu *WebStranka*. Definice třídy *WebStranka*, stejně jako jejích metod, je součástí modulu *Stranka.py*, který je importován při startu skriptu *filer*. Pro každou stránku program postupně zapíše do databáze tyto hodnoty – hostname, url, informace o stránce a kompletní text dané stránky zbavený značek jazyka html, javascriptu a zkonvertovaný do kódování UTF-8. Jako zdrojové kódování pro konverzi je použito kódování uvedené v html hlavičce konkrétní stránky.

Dále skript filer při zpracování odstraňuje některé chyby validity ve zpracovávaných datech. Jde hlavně o problémy, které mohou následně vést k chybné indexaci, nebo indexaci úplně znemožnit. Zejména: chybějící titulek stránky, nesprávné použití znaků vyhrazených pro řídící struktury SGML jazyků jako &, <, >, a další.

Sbarel

Sbarel je strukturovaný soubor využívající značek jazyka SGML. Bohužel se nejedná o well-formed XML dokument, což práci s ním poněkud komplikuje, stejně jako to, že průměrná velikost barelu je 40 GB a každý barel obsahuje kolem 9 milionů dokumentů. Seznam používá barely jako datová úložiště a pro práci s nimi má vyvinutou knihovnu nástrojů. Tato knihovna, stejně jako její dokumentace, je ovšem součástí firemního know-how, které bohužel nemůže být poskytnuto třetí osobě. Přes tyto problémy jsou poskytnutá data velmi hodnotnou kolekcí dat pro řadu experimentů, neboť dohromady představují jazykový korpus o velikosti 0,5 TB.

Každý z dokumentů je v barelu reprezentován následující strukturou:

```
<document id="id dokumentu">
  <url>kompletní url včetně protokolu</url>
  <title>titulek stránky</title>
  <contents>
    obsah stránky
  </contents>
</document>
```

Problém při zpracování barelu je zejména text stránek, tedy obsah elementu *contents*. Text originální stránky je sice během přípravy barelu zbaven značek html, ale jsou v něm ponechány některé znaky představující problém z hlediska XML syntaxe. Pro zpracování barelu není z důvodu jeho velikosti možné použít některý z parserů, který dokáže tyto chyby řešit, jako například BeautifulSoap, protože tyto programy se snaží načíst celý dokument do paměti. Sekvenční zpracování barelu vyžadovalo naprogramování vlastního parseru a ošetření všech výjimek, které obsah stránek může přinést. Tento parser je součástí skriptu *barel.py*, který kromě parsování kódu zajišťuje také jeho uložení do databáze.

6.1.3 Datové kolekce použité pro experimenty

Pro experimenty popsané ve zbytku této práce byly použity celkem čtyři různé datové kolekce. Pro větší přehlednost a možnost jejich srovnání jsou jejich parametry popsány v této kapitole. Základní přehled těchto datových kolekcí přináší tabulka 6.1.

Datová kolekce *California* byla vytvořena Jonem Kleinbergem jako výsledek vyhledávání stránek obsahujících slovo „California". Jde o poměrně malou kolekci, která navíc nebyla k dispozici ve formě fulltextu, ale jen jako data pro vytvoření web grafu. Kolekce byla do experimentů zahrnuta zejména jako srovnávací data, protože je volně dostupná na stránce [90] a několik autorů, mimo jiné Langville a Meyer [48], publikovalo výsledky výpočtů nad touto kolekcí.

Datová kolekce s názvem *tul* vznikla za pomoci crawleru Pavuk s následujícím nastavením: jako počáteční URL bylo zadáno http://www.tul.cz/. Následně crawler začal postupně zpracovávat stránky propojené s výchozím URL. Ukládání stránek bylo omezeno pouze na stránky ve formátu html, negenerované dynamicky a s preferencí na stránky v českém jazyce.

Jako rozlišení stránky generované dynamicky slouží v tomto případě znak „?" v URL. Takže například stránky, které jsou generovány skriptem, ale využívají také mod_rewrite crawler stáhne. Z velké většiny serverů byly ale staženy pouze soubory index z jednotlivých adresářů. Toto nastavení se ukázalo

název kolekce	*stránek*	*odkazů*	*velikost slovníku*
California	9664	16773	–
tul	45531	150789	315498
CZdat	998037	3822430	1805572
CZdat2	4980930	34998687	5598268

Tabulka 6.1: Datové kolekce použité pro experimenty.

jako vhodnější pro hledání spojité komponenty web grafu.

Třetí datovou kolekcí je kolekce s názvem *CZdat*. Ta vznikla z části dat poskytnutých společností Seznam.cz ve formátu SBarell. Původně byla plánována jako nejrozsáhlejší kolekce pro závěrečné testy poté, co budou jednotlivé experimenty odladěny na menších datech. Při experimentech s algoritmem TSPR se ale tato kolekce ukázala jako nedostatečná a došlo k vytvoření ještě rozsáhlejší kolekce CZdat2.

Základem pro vytvoření kolekce CZdat2 bylo 113565 adres z odkazového katalogu Seznam a 25959 adres z české části katalogu ODP. Po odstranění duplicitních stránek a vyřazení stránek, které se nepodařilo otevřít, vznikla množina 112443 URL, která byla použita jako startovací množina pro crawler. Nastavení crawleru bylo opět omezeno na stránky obsahující text ve formátu html s preferencí stránek v českém jazyce. Dále byl omezen počet vnoření (následovaných odkazů) na maximálně 3 kroky od zadané startovací adresy. Crawler s tímto nastavením ze startovních 112 tisíc adres vytvořil množinu čítající necelých 5 milionů stránek.

Protože byla tato datová kolekce vytvořena až poté, co se při experimentech s algoritmem TSPR, ukázalo že kolekce CZdat je pro ně nevyhovující, nebyla zahrnuta do ostatních experimentů. Experimenty s alogritmem TSPRL probíhaly jako poslední.

Vytvoření této poslední množiny trvalo díky několika výpadkům crawleru necelý měsíc, protože průměrně crawler za hodinu zpracoval jen asi 6000 URL. Zde je poměrně velký prostor pro budoucí zlepšení experimentálního systému.

6.2 Úložiště dat

Data získaná pomocí crawleru Pavuk, či ve formátu Sbarell, bylo dále potřeba někam uložit. Dalším krokem v návrhu experimentálního systému byl tedy návrh datového úložiště.

V praxi používají WWW vyhledávače obvykle proprietární databázové

formáty primárně zaměřené na rychlost navrácení informace a snadné aktualizování dat. Jedná se o efektivnější přístup než použití relačního databázového serveru. Tyto komplexní aplikace totiž obvykle nabízí daleko více možností, než je pro potřeby WWW vyhledávače potřeba. Široké možnosti plnohodnotného databázového serveru by zůstaly nevyužity a byly by zbytečně zaplaceny ztrátou rychlosti operací.

Přesto byla pro experimentální prostředí použita relační databáze MySQL, proč? Hlavní důvodem byla rychlá a snadná implementace tohoto řešení. Vývoj speciální databáze, třebaže relativně jednoduché, by zabral značné množství času a odvedl pozornost z pohledu cílů práce nepodstatným směrem. Další možností datového úložiště by byla přímá indexace zdrojových dat. Tato zpočátku uvažovaná varianta byla nakonec opuštěna poté, co se ukázalo, že zdrojová data budou pocházet z více zdrojů. To by vyžadovalo řešit přímou indexaci hned několikrát, pro každá data zvlášť. Jednotné úložiště, do kterého jsou data z různých zdrojů zpracována, je tak pro další práci efektivnější.

Pro nasazení MySQL databáze jako datového úložiště experimentálního systému jsou následující argumenty:

- rychlá implementace jednotného datového úložiště pro různá data
- rozsáhlá a dostupná dokumentace + podpora v řadě programovacích jazyků
- možnost nasazení distribuované databáze v případě potřeby
- předchozí dobré zkušenosti s produktem.

6.2.1 Schéma použité databáze

Obecně je pro návrh databázového schématu nutné zodpovědět dvě základní otázky: jaká data budeme do databáze ukládat a jaké informace budeme z databáze nejčastěji čerpat?

Ukládaná data:

- WWW stránky získané crawlerem. Zde je nutné umožnit opakované získání téže stránky ⇒ je nutné uchovávat identifikátor konkrétního crawlu.
- pro každou stránku je nutné uložit informace o jejím URL, čas uložení, datový typ (MIME)

- pro každou stránku vytvořit seznam odchozích linek
- pro každou stránku uložit upravený zdrojový text

Typické dotazy:

- kompletní informace o určité stránce na základě URL či jiného ID
- seznam odchozích / příchozích odkazů na určitou stránku
- seznam stránek s určitou vlastností: mime-typ, jazyk apod.
- seznam stránek z určité domény (1.-3. řádu)
- výpis stránek v uspořádání dle domény
- možnost sledování verze stránky
- vytvoření inverzního indexu term x dokument
- vytvoření incidenční matice web grafu

Na základě této analýzy bylo navrženo schéma databáze, zobrazené na obrázku 6.2 (včetně relací mezi tabulkami). Nyní si popíšeme jednotlivé tabulky reprezentující databázové relace a jejich klíčové atributy.

Crawl (CrawlID, Popis)

Tabulka sloužící pro rozlišení verzí zpracovávaných dat.

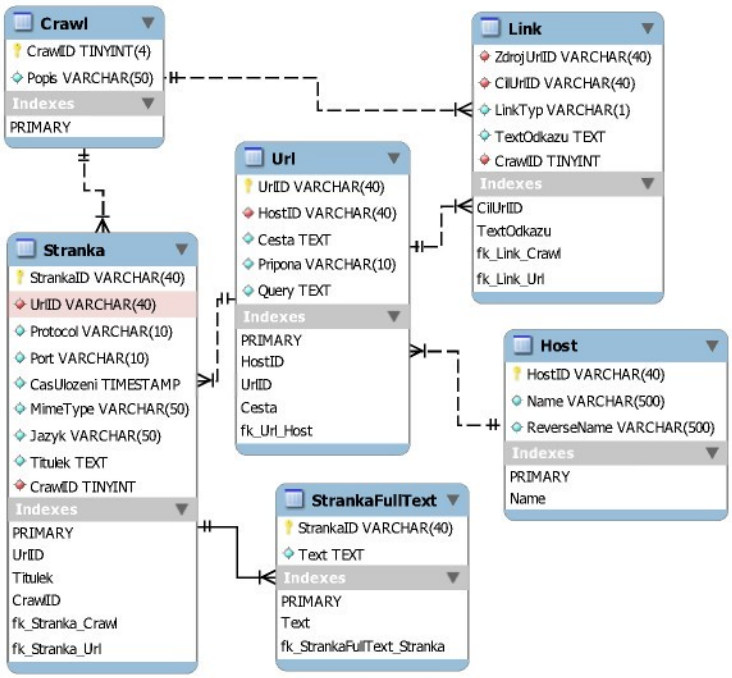
CrawlID – tinyint : identifikátor crawlu, referenční integrita pro tabulku Stranka, umožňuje uchování 255 různých crawlů, což je pro aktuální potřeby experimentálního systému dostatečné.

Stranka (StrankaID, UrlID, Protocol, Port, CasUlozeni, MimeType, Jazyk, Titulek, CrawlID)

Hlavní tabulka pro reprezentaci stránky. Umožňuje výpis stránek dle jazyka, MimeTypu, protokolu.

StrankaID – hash SHA1 vypočítaný z kompletního URL stránky včetně protokolu a portu, primární klíč.

UrlID – cizí klíč, referenční integrita.



Obrázek 6.2: Schema databáze WebIndex

Url (UrlID, HostID, Cesta, Pripona, Query)

Tabulka pro ukládání jednotlivých URL rozdělených na části. Umožňuje další filtraci souborů dle přípony, rozlišení url na statická, dynamická, přepisovaná, absolutní a relativní.

UrlID – hash SHA1 vypočítaný z URL bez portu a protokolu.

HostID – cizí klíč, referenční integrita.

Host (HostID, Name, ReverseName)

Tabulka pro ukládání jednotlivých hostname. Umožňuje uspořádat indexy stránek do bloků dle řádu domény.

HostID – hash SHA1 vypočítaný z hostname.

Link (ZdrojUrlID, CilUrlID, LinkTyp, TextOdkazu, CrawlID)

Tabulka pro reprezentaci odkazů. Slouží pro sestavování grafů a matic reprezentujících WWW.

Zdroj a Cil UrlID – cizí klíče, odkazující a odkazované URL.

LinkTyp – rozlišení zda jde o vnější či vnitřní odkaz, možné doplnit o další typy.

StrankaFullText(StrankaID,Text)

Tabulka pro uložení textu příslušné stránky.

Jednotlivé klíče vypočítané jako SHA1 hash, jsou uloženy jako datový typ varchar(40). Pro větší efektivitu a menší paměťové nároky by je bylo možné uložit také v binárním tvaru, což by přineslo cca 50% úsporu pro každý klíč. Binární reprezentace SHA1 hash má 20B, zatímco textová 40 znaků, varchar tedy zabere max. 41B. Hlavním důvodem pro textovou reprezentaci je snadnější vizuální rozlišení jednotlivých klíčů.

6.3 Využití dat z úložiště

Data uložená v databázi by za určitých podmínek šla již přimo využívat v klientském rozhraní. Ovšem většina experimentů popsaných v této práci dále se zabývala dalším zpracování těchto dat před tím, než jsou poskytnuta klientské aplikaci a tedy koncovému uživateli. Důležitou součástí experimentálního systému jsou tedy skripty, které dokáží z dat v úložišti sestavit incidenční matici web grafu, případně vytvářet různé externí (mimo SQL databázi) indexy.

6.3.1 Web graf a incidenční matice

Pro výpočet PageRank vektoru, stejně jako pro další experimentální výpočty, je nutné z uložených dat získat informace o grafu, který tvoří uložené stránky a jejich vzájemná propojení pomocí odkazů. Pro většinu výpočtů je pak nutná zejména incidenční matice tohoto grafu. To je možné pomocí skriptu *vytvormatici.py*, který za pomoci modulu *Matice.py* dokáže tuto matici vytvořit pro celou uloženou strukturu, nebo pro určitý podgraf, tvořený například pouze stránkami obsahujícími určité slovo.

Modul *Matice.py* umožňuje uložit matice ve formátech CSR a MPIAIJ. Tyto formáty jsou popsány v kapitole 7.2.

6.3.2 Indexace dat

Základem rychlého vyhledávání je indexace prohledávaných dat. Základní indexaci provádí databázový server MySQL již při ukládání dat do úložiště. Mimo indexů poskytovaných MySQL jsou v experimentálním systému použity také externí indexy, realizované pomocí slovníků v jazyce Python. Jde

zejména o dvě varianty inverzního indexu dokumentů, dále indexy sloužící při vytváření incidenčních matic a indexy ohodnocení jednotlivých stránek.

Základem každého indexu je třída *Index*, definovaná v modulu *Index.py*. Tato třída definuje strukturu indexu a metody pro I/O operace s indexem. Z této základní třídy jsou odvozeny třídy definující jednotlivé specifické indexy.

WordIndex je jednoduchý inverzní index, jehož klíče tvoří slova: prvky slovníku S, množiny všech indexovaných slov. Jednotlivé položky indexu jsou pak seznamy id stránek obsahujících příslušný klíč.

WordIndexTF používá stejnou množinu klíčů, ale položky indexu tvoří seznamy dvojic (strankaID,TF), kde TF je frekvence daného slova v dokumentu vypočtená ze vzorce (2.1). Instance těchto objektů jsou základem pro fulltextové vyhledávání, ke kterému slouží IndexServer, popsaný níže.

Třídy *UrlIndex* a *SubMatIndex* definují jednoduché indexy, které k id konkrétního URL přiřazují číslo sloupce/řádku, pod kterým je toto URL uloženo v incidenční matici. Pro základní výpočet PageRanku by bylo možné vytvořit takový index jako jeden ze sloupců tabulky *URL* či *Stranka*, ale použité řešení s externím indexem umožňuje snadnější a přehlednější vytvoření jinak uspořádaných matic (například host ordered apod.) a také matic reprezentujících podgrafy.

6.3.3 Index Server

Velikost uloženého indexu závisí na velikosti slovníku S a počtu indexovaných stránek N. Konkrétní hodnoty pro čtyři různé slovníky ukazuje tabulka 6.1 na konci této kapitoly. Pro rychlost vyhledávání je nezbytné, aby prohledávaný index byl trvale uložen v paměti počítače, protože doba načtení indexu do paměti je několik sekund. Proto byl za pomoci Python modulu socket vytvořen jednoduchý server *IndexServer.py*, který udržuje v paměti inverzní index a zpracovává dotazy od klienta – vyhledávacího rozhraní. Kromě inverzního indexu udržuje v paměti také jeden či více zvolených rank vektorů, které slouží pro ohodnocení nalezených výsledků.

Komunikace probíhá pomocí protokolu TCP na portu 50007. Algoritmus práce index serveru je následující:

1. přijmout dotaz $Q = q_1, ..., q_n$ od klienta. Kde q_1 až q_n jsou jednotlivá slova v dotazu Q.

2. pokud je použit index se stemy slov - převést q_i pro $i \in < 1, n >$ na stem.

3. postupně vytvořit množiny výsledků R_i pro každé q_i, kde $i = 1 \ldots n$.

4. výsledek $R = \bigcap_{i=1}^{n} R_i$.

5. uspořádat výsledek dle zvoleného algoritmu, odeslat klientovi.

6. počkat na další dotaz.

Toto chování serveru, kdy dotazu vyhoví pouze ty dokumenty, které obsahují všechna slova z dotazu, nazýváme absolutní shoda.

6.4 Webové rozhraní – klient

Jako většina klientských aplikací slouží i tento program jako prostředník pro komunikaci mezi uživatelem a serverem. Přijímá uživatelské dotazy, odesílá je serveru a z přijatých dat sestavuje výsledek. Skript je pojmenován *index.py* a ke své činnosti potřebuje web server Apache, protože je realizován pomocí modulu mod_python. Tento modul Apache serveru umožňuje vytvářet WWW stránky pomocí Pythonu.

Pro zpracování dat zaslaných od serveru slouží parser založený na modulu BeautifulSoup. Server posílá data jako proud znaků, rozdělených do TCP datagramů. Pro jejich pozdější zpracování je nutné tato data zasílat ve strukturované podobě. Použitý aplikační jazyk je popsán níže.

6.4.1 Komunikace klient/server

Jako aplikační komunikační jazyk mezi serverem a klientem byl navržen strukturovaný jazyk využívající XML syntaxe. Základním úkolem tohoto jazyka je umožnit rozlišení jednotlivých výsledků, jejich řazení a identifikaci řadící funkce. Výhodou nasazení aplikace jazyka XML je jeho rozšířenost, sebevalidace a dostatek nástrojů pro efektivní zpracování jeho aplikací.

Struktura přenášených dat je následující:

```
<results>
  <result>
    <url>url výsledku</url>
    <titulek>titulek výsledné stránky</titulek>
    <pr>ohodnoceni stránky</pr>
  </result>
...
</results>
```

Definice komunikačního jazyka pomocí XSD:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="results">
  <xs:complexType>
    <xs:sequence>
       <xs:element name="result" maxOccurs="unbounded">
          <xs:complexType>
          <xs:sequence>
            <xs:element name="url" type="xs:string"/>
            <xs:element name="titulek" type="xs:string"/>
            <xs:element name="pr" type="xs:decimal" minOccurs="0" />
            <xs:element name="tsprl" type="xs:decimal" minOccurs="0"/>
            <xs:element name="tfidf" type="xs:decimal" minOccurs="0"/>
            <xs:element name="czdis" type="xs:decimal" minOccurs="0"/>
          </xs:sequence>
          </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

6.5 Použitý hardware

Popis experimentálního systému zakončíme parametry použitého hardware.

Pro paralelní výpočty byl využit výpočetní cluster FM – hydra.nti.tul.cz. Jedná se o PC cluster s operačním systém RocksClusters 5 složený ze 17 uzlů. Každý z uzlů je osazen dvěma procesory AMD Opteron 252 o frekvenci 2,6 GHz, 4 GB RAM. To představuje jeden řídící počítač a 32 výpočetních procesorů s celkovou kapacitou sdílené paměti 64GB RAM. Jako propojovací síť je použit Gigabitový Ethernet. Všech 17 uzlů je přímo propojeno se síťovým přepínačem. Cluster bude v dalším testu uváděn jako *Hydra* nebo *cluster*.

Pro sekvenční výpočty, zpracování dat i jako testovací vyhledávací server bylo použito běžné PC s procesorem Intel Core^{tm}2 E8400, 8GB paměti RAM a 1,5 TB prostoru na pevném disku. Jako operační systém byla nainstalován 64-bit Fedora Linux. Tento počítač bude v dalším textu odkazován jako *Lexis*.

Kapitola 7

Efektivní metoda pro výpočet PageRank vektoru

Základním principem většiny personalizovaných algoritmů vycházejících z algoritmu PageRank, je opakovaný výpočet PageRank vektoru. Výjimkou nejsou ani algoritmy IS a TSPR, jejichž lokalizace byla hlavním cílem této práce. Proto bylo před zahájením samotného lokalizačního procesu, nutné najít dostatečně efektivní metodu pro výpočet PageRank vektoru.

Většina autorů prací zabývajících se urychlením základního výpočtu, popsaných v kapitole 4, se spokojila s tím, že svou metodu porovnala s Mocninnou metodou, která se pro výpočet PageRank vektoru tradičně používá. Vzájemné porovnání jednotlivých návrhů na zrychlení výpočtu ovšem v dostupných publikacích chybí.

Následující kapitola přináší výsledky dvou experimentů zaměřených na nalezení nejrychlejší výpočetní metody pro testovací data. První experiment byl zaměřen na porovnání rychlosti výpočtu několika vybraných metod. Druhý vycházel z výsledků prvního, ve kterém nejlepších výsledků dosáhlo řešení lineárního systému. Cílem tohoto druhého experimentu bylo srovnání různých metod pro řešení řídkého lineárního systému. V úvodu si nejprve popíšeme některé detaily implementace výpočtů, společné pro oba experimenty.

7.1 Implementace výpočtů

Veškeré výpočty byly implementovány v programovacím jazyce Python [73] za využití modulů SciPy [74] a Petsc4py [75]. Tyto nástroje byly použity i při dalších implementacích v rámci této práce, proto si zde stručně popíšeme jejich základní charakteristiky.

Python je interpretovaný objektově orientovaný programovací jazyk. Patří

mezi moderní jazyky – jeho první verzi vytvořil Guido van Rossum až v roce 1990. Jde o open source projekt vytvořený v jazyce C. V tomto jazyce je také psána většina knihoven – modulů, díky čemuž je běh programů velmi rychlý. Python využívají přední světové společnosti, například Google, a je v něm vytvořena řada významných aplikací, za všechny jmenujme populární server www.youtube.com, zaměřený na sdílení videa. Pro svou jednouchou a přehlednou syntaxi bývá také doporučován jako první programovací jazyk pro začátečníky.

Většina metod byla implementována pomocí modulu Scientific Python (SciPy), což je základní nástroj pro vědecké výpočty v Pythonu. Je složen z celé řady modulů podporujících numerické výpočty, operace s řídkými maticemi, vizualizaci výstupů, paralelní programování a další operace, které jsou spojeny s pojmem vědecké výpočty. Stejně jako jazyk Python je i tento modul open source projektem.

Pro implementaci výpočtů byl využit modul Scipy.sparse, který definuje speciální datový typ pro práci s n-rozměrným řídkým polem. Matice byly uloženy ve formátu CSR (viz. dále). Většina grafů v této práci pak byla vytvořena pomocí modulu MatPlotLib, který je také součástí SciPy balíčku.

Pro řešení lineárního systému byla využita numerická knihovna PETSc [76], která obsahuje jak paralelní tak sekvenční implementace mnoha lineárních řešičů. Protože je tato knihovna naprogramována primárně pro jazyky Fortran a C, byl použit také modul Petsc4py, což je Python API k této knihovně. Petsc4py umožňuje využívat většinu funkcí originální knihovny, ale převádí její ovládání do Python syntaxe, která je obvykle mnohem přehlednější než originál. Nevýhodou tohoto modulu je zatím poměrně slabá dokumentace. Řadu funkcí je nutné studovat přímo ze zdrojového kódu. Petsc4py posloužil také jako nástroj pro konverzi dat z formátu CSR (Scipy.sparse) do formátu SEQAIJ(Petsc).

7.2 Formáty CSR a AIJ

Scipy a PETSc umožňují ukládání řídké matice ve stejném formátu. Jde o formát Compressed Sparse Row, též označovaný jako Yale sparse matrix format. Implementace tohoto formátu je ale v obou knihovnách různá, proto není možné používat data ve formátu Scipy v knihovně PETSc a naopak. Díky tomu, že jde o stejný formát, je ale konverze dat mezi oběma knihovnami poměrně snadno řešitelná a rychlá. Kromě formátu CSR, se pro ukládání řídkých matic využívá také formátu CSC, který se odlišuje tím, že používá sloupce namísto řádků. Je podporován oběma knihovnami, stejně jako další formáty například *linked list*.

Formát CSR, který byl během výpočtu nejčastěji využíván, je jedním ze základních formátů pro ukládání řídkých matic. Jde o poměrně obecný formát, ve kterém je možné uložit jakoukoliv matici – řídkost matice se nijak nekontroluje. Nejde o nejefektivnější formát, protože například během násobení matice vektorem je nutné v každém kroku výpočtu načítat data přes nepřímé adresy. Jde ale o formát snadno implementovatelný a hojně využívaný.

Jako datové struktury využívá tři pole – jedno pro ukládání hodnot s požadovanou numerickou přesností, druhé pro ukládání indexů sloupců a třetí pro ukazatele na řádky. Některé implementace, jako například knihovna BLAS, používají ještě čtvrté pole ukazatelů, které slouží pro propojení mezi polem hodnot a prvním nenulovým prvkem na určitém řádku matice. Nenulové prvky matice jsou zpracovány a uloženy do paměti ve spojitých blocích, což vede k výrazné úspoře. Pole pro hodnoty obvykle bývá inicializováno jako float32 či 64, pro zbylá dvě pole si vystačíme s typem integer.

Jestliže si počet prvků ukládané matice označíme jako n a počet nenulových prvků této matice jako nnz, pak lze paměťové nároky pro uložení matice vyjádřit jako $2nnz + n + 1$, zatímco při uložení běžným způsobem, bychom potřebovali n^2 místa. Podrobněji tento formát popisuje například práce [62].

7.3 Efektivita jednotlivých metod - experiment 1.

Z dostupných metod pro nalezení PageRank vektoru popsaných v kapitole 4 je pro další experimenty nutno vybrat tu, která pro testovací data (viz kap. 6.1.3) určí hledaný vektor v nejkratším čase.

Pro testování byly vybrány zejména metody zabývající se aplikací numerických postupů. Kritériem výběru byly publikované výsledky konkrétní metody, ale také náročnost její případné implementace. Vynechány proto byly práce řešící PageRank vektor pomocí aproximace i práce zabývající se implementací vlastních datových typů pro lepší využívání paměti.

Celkem bylo v rámci prvního experimentu testováno následujících šest metod a algoritmů:

1. Mocninná metoda jako nejtradičnější způsob výpočtu a srovnávací metrika (kap. 4.1). Dále budne označována jako: *Standard*.

2. Mocninná metoda urychlená pomocí metody BlockRank (kap. 3.7.3). Označení: *Blockrank*.

3. Mocninná metoda urychlená pomocí algoritmu MAPR (kap. 4.3.3). Označení: *Mapr.*

4. Lineární systém řešený pomocí řešiče *spsolve*, který je součástí knihovny SciPY.linsolve (kap. 4.2). Označení: *Scipy.*

5. Lineární systém řešený metodou GMRES za využití knihovny PETSc. Označení: *Petsc.*

6. Algoritmus LangMey (kap. 4.2). Pro jednotlivé kroky algorimtu byla opět použita metoda GMRES. Označení: *LangMey.*

Kritériem konvergence Mocninné metody a metod z ní vycházejících byla velikost vektoru rezidua $\delta < 10^{-4}$. Hodnota koeficientu $\alpha = 0.85$. Řešiče spsolve (SciPy) i KSP (PETSc) byly ponechány ve výchozím nastavení příslušné knihovny.

Teoreticky by, podle předchozích publikovaných výsledků mělo být řešení lineárního systému libovolnou použitelnou metodou rychlejší než Mocninná metoda. Dále by dle výsledků publikovaných v [52] mělo dojít k dalšímu urychlení výpočtu lineárního systému, pokud bude řešen pomocí algoritmu LangMey (metoda 6.).

7.4 Výsledky prvního experimentu

Časy výpočtu jednotlivých metod pro všechny tři datové kolekce ukazuje tabulka 7.1. Jde o čistý čas výpočtu měřený jako rozdíl dvou časových razítek procesoru, pořízených bezprostředně před spuštěním řešiče a okamžitě po jeho ukončení. Uvedené výsledky nezahrnují časy potřebné pro sestavení matic a vektorů, ani pro jejich inicializaci. Tyto vedlejší náklady jsou pro všechny metody stejné a proto by jejich vliv na výsledek výpočtů měl být minimální. Každá metoda byla spuštěna pětkrát a v tabulce 7.1 je uveden aritmetický průměr dosažených časů. Grafické srovnání výsledků pak představuje obrázek 7.1.

V souladu s teoretickým předpokladem se řešení lineárního systému ukázalo jako rychlejší metoda než metoda Mocninná. U algoritmu Blockrank se pro obě menší datové kolekce nepotvrdil předpoklad urychlení Mocninné metody, ovšem pro největší datovou kolekci CZdat již ke zrychlení došlo. V poměru k velikosti datové kolekce není urychlení pro menší data dostačující k pokrytí větší režie algoritmu Blockrank. U algoritmu Mapr se potvrdily teoretické předpoklady a oproti Mocninné metodě došlo k urychlení v průměru o 19,27%.

metoda/data	*California*	*Tul*	*CZdat*
Standard	1,3099	8,4320	27,8223
Scipy	0,0978	0,6861	2,6854
Petsc	0,0135	0,0774	1,4919
Mapr	1,0706	6,7226	22,4606
Langmey	0,2798	1,7407	4,0409
Blockrank	3,7494	11,6551	21,8839

Tabulka 7.1: Časy výpočtu zvolených metod pro data California, Tul a CZdat. Časy jsou uvedeny v sekundách.

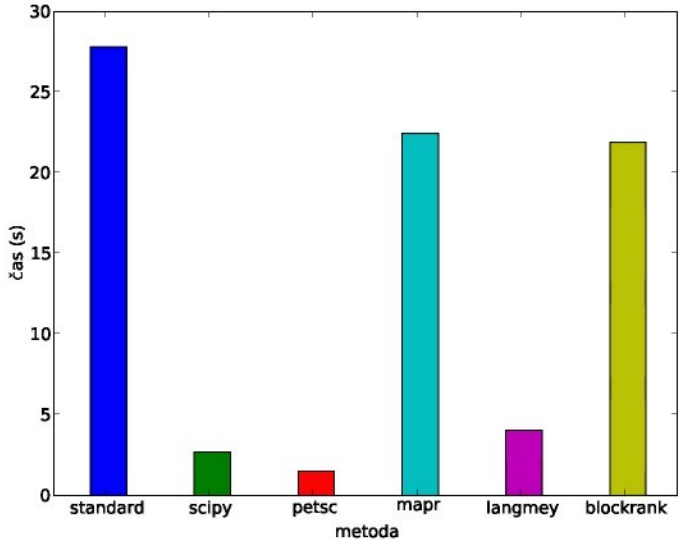
Ze tří různých variant řešení lineárního systému dosáhla nejlepších výsledků implementace s využitím knihovny PETSc. Lineární řešič KSP z této knihovny dosáhl ve své základní konfiguraci (GMRES s ILU předpodmíněním) 18x rychlejšího času oproti Mocninné metodě. Nepotvrdily se teoretické předpoklady s větším urychlením výpočtu pomocí LangMey algoritmu. Důvodem tohoto faktu, může být nízký počet koncových uzlů (dangling nodes) v testovacích datech a také režie algoritmu pro uspořádání matice. Protože cílem experimentu nebyla konkrétně implementace LangMey algoritmu a protože se výsledky metody *Petsc* jevily použitelné pro další experimenty, nebylo pokračováno v optimalizaci implementace LangMey algoritmu.

7.5 Hledání optimální iterační metody - experiment 2.

Na základě výsledků prvního experimentu byla zvolena knihovna Petsc, jako hlavní prostředí pro další výpočty PageRank vektorů pomocí lineárního systému. Otázkou tedy bylo, zda nelze změnou nastavení řešiče KSP dosáhnout ještě lepšího času výpočtu. Druhým experimentem v rámci hledání efektivní metody pro výpočet bylo testování různých nastavení řešiče KSP.

Základní nastavení tohoto řešiče je metoda GMRESm s restartem po 30 iteracích, využívající klasickou (nemodifikovanou) Gram-Schmidtovu ortogonalizaci. Dále je použito levé předpodmínění metodou ILU.

Druhý experiment by měl otestovat, zda některá jiná metoda, či jiné předpodmínění, nepovede k rychlejšímu výpočtu. Pro test byly vybrány následující iterační metody: Generalized minimal residual method (GMRES), Conjugate gradient (CG), Biconjugate gradient (BiCG), Stabilized Bi-Conjugate Gradient (BiCGstab) a Čebyševova metoda (chebychev). Pro každou metodu postupně uplatníme následující metody předpomínění: LU, ILU, Jacobi.



Obrázek 7.1: Grafické srovnání časů výpočtu jednotlivých metod pro data CZdat

Ostatní nastavení výpočtu zůstanou stejná jako v případě prvního experimentu.

7.6 Výsledky druhého experimentu

Časy výpočtu jednotlivých metod a předpodmínění ukazuje pro všechny tři datové kolekce tabulka 7.2. Stejně jako u prvního experimentu jde o čistý čas výpočtu měřený jako rozdíl dvou časových razítek procesoru a uvedené výsledky nezahrnují časy potřebné pro sestavení matic a vektorů, ani pro jejich inicializaci. Časy v tabulce opět představují aritmetický průměr časů z pěti měření. Grafické srovnání nejlepších výsledků pro vybrané metody, pak představuje obrázek 7.2. Do grafického srovnání nebyly zahrnuty metody CG a Čebyševova, které se pro kolekci CZdat ukázaly jako neefektivní.

Druhý experiment ukázal, že účinnost iterační metody velmi závisí na konkrétních datech. Příkladem je velmi rozdílná efektivita metody CG na datech California, kde v kombinaci s Jacobi předpodmíněním dosáhla nejlepšího času a na datech TUL, či CZdat, kde byla naopak nejpomalejší metodou.

Nejdůležitější byla účinnost jednotlivých metod na kolekci CZdat. Na těchto datech dosáhla nejlepšího výsledku metoda BiCGStab s Jacobi předpodmíněním, ovšem rozdíl oproti metodě GMRES / ILU je pouze 0,0051 s po

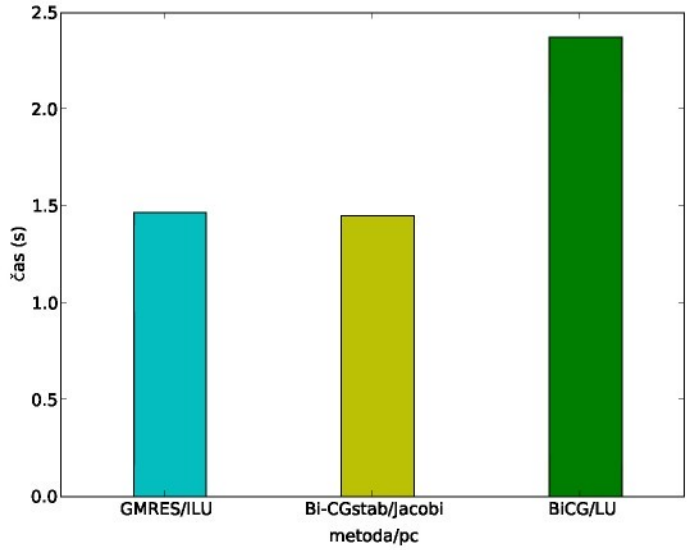
California	*LU*	*ILU*	*Jacobi*
gmres	0,0135	0,0135	0,0255
cg	1,5613	1,6152	0,0056
bicg	0,0236	0,0237	0,0395
bcgsl	0,0131	0,0131	0,0126
chebychev	0,8415	0,8586	0,2162
Tul	*LU*	*ILU*	*Jacobi*
gmres	0,0774	0,0750	0,0949
cg	38,6157	39,3060	21,7743
bicg	0,1055	0,1128	0,1041
bcgsl	0,0781	0,0790	0,0778
chebychev	3,5223	3,6128	1,0380
CZdat	*LU*	*ILU*	*Jacobi*
gmres	1,4919	1,4658	2,2753
cg	487,6151	497,6223	231,5244
bicg	2,2908	2,3747	3,0676
bcgsl	1,4787	1,4904	1,4517
chebychev	79,1825	80,9965	21,7570

Tabulka 7.2: Časy výpočtu lineárního systému pomocí iteračních metod a knihovny PETSc. Časy jednotlivých iteračních metod a předpodmínění pro grafy California, Tul a Czdat, jsou uvedeny v sekundách.

zaokrouhlení. Obě metody lze tedy pro testovací datové kolekce považovat za rovnocenné.

7.7 Závěr

Podařilo se prokázat, že pro testovací data je nejefektivnější metodou nalezení PageRank vektoru řešení lineárního systému pomocí knihovny PETSc. Výsledky druhého experimentu pak potvrdily, že základní nastavení řešiče KSP, který je součástí této knihovny, je pak pro výpočet lineárního systému na testovací kolekci CZdat dostatečně efektivní metodou. Pro další výpočty v rámci této práce bude tedy použita metoda GMRES s ILU předpodmíněním.



Obrázek 7.2: Grafické srovnání nejlepších časů jednotlivých metod/předpodmínění pro data CZdat

Kapitola 8

CZDIS - lokalizovaný algoritmus Intelligent Surfer

Algoritmus IS přináší vylepšenou hodnotící funkci oproti standardnímu PageRanku, ale zároveň nabízí několik problémů k dopracování a dořešení, a to v oblasti škálovatelnosti a rychlosti výpočtu. Základním cílem bylo ověřit, zda je možné v tomto algoritmu použít také český jazykový model. Protože byl ale původní algoritmus kritizován pro přílišnou výpočetní náročnost, bylo potřeba zaměřit se i na zvýšení efektivity výpočtu hodnotících vektorů tak, aby případný lokalizovaný algoritmus bylo možné nasadit v reálném prostředí a nezůstal jen teoretickou studií.

Základní algoritmus jsme si detailně popsali v kapitole 3.7.5. Rozhodující pro rychlost a škálovatelnost výpočtu je velikost slovníku S – čím více slov do algoritmu zahrneme, tím více bude generovaných podgrafů, u kterých musíme najít PageRank vektor. Ten autoři originálního algoritmu počítali pro všechny podgrafy sekvenčním výpočtem Mocninné metody. Uvědomíme–li si tato omezení algoritmu IS, můžeme se pokusit odpovědět na otázku: jak dosáhnout zrychlení výpočtu, při zachování či dokonce zlepšení kvality hodnotící funkce?

Teoreticky se nabízejí následující možnosti:

1. Snížení velikosti slovníku S a tím poměru U/N aplikací jazykového modelu.

2. Výpočet PageRank vektoru pro konkrétní podgraf, jako elementární krok celého výpočtu, lze urychlit pomocí výpočtu lineárního systému metodou GMRES, jak bylo dokázáno v kapitole 7.

3. Jednotlivé podgrafy není nutné řešit sekvenčně, ale je možné využít toho, že jde o přirozený paralelizmus problému a provést tak výpočet

jednotlivých vektorů paralelně na více výpočetních uzlech.

Modifikovaný algoritmus byl nazván Czech Distributed Intelligent Surfer (CZDIS) a touto zkratkou je označován ve zbytku textu. Nejprve si postupně projdeme jednotlivé navržené možnosti a na jejich základě zformulované teoretické návrhy, které byly následně experimentálně ověřeny.

8.1 Snížení poměru U/N aplikací jazykového modelu a statistky

Protože s nárůstem počtu stránek N vzrůstá také počet slov a tím i počet unikátních dvojic dokument–slovo označený jako U, můžeme předpokládat, že poměr U/N se s rostoucím počtem stránek příliš nemění. Jedinou možností jak ovlivnit poměr U/N ve prospěch rychlosti výpočtu je tedy snížení počtu slov ve slovníku S. Toho lze dosáhnout postupnou aplikací pravidel jazykového modelu i využitím statistiky sledovaných textů. V případě této práce byl použit český jazykový model, který by ale mohl být snadno nahrazen modelem jiného jazyka.

Prvním krokem ke snížení velikosti slovníku S je vypuštění často opakovaných výrazů – stop slov. V oblasti získávání informací z textů jde o často využívanou operaci, používanou obvykle již během indexace. Jako stop slova označujeme výrazy, které se v konkrétním jazyce často opakují, ale nenesou žádnou významovou informaci – typickým příkladem pro češtinu jsou spojky (a, aby, ale, ani,...), zájmena (on, ona, my,...), předložky (nad, u, pro,...) nebo některá slovesa (být, mít,...).

Charakteristickým znakem stop slov je, že se vyskytují v prakticky každém dokumentu a proto jejich indexací pouze zvětšujeme velikost indexu, aniž bychom tak zkvalitnili vyhledávání. Vypuštění stop slov může ovšem způsobit také problémy s nalezením některých výrazů – typicky uváděným příkladem pro angličtinu jsou vlastní jména, jako například „The Who“.

Seznam českých stop slov má 257 položek a lze ho najít na [64]. Přestože se množina stop slov jeví jako poměrně malá, generuje obsáhlé podgrafy a její podíl na vytvoření U je nemalý (v poměru k její velikosti). To je způsobeno výše uvedeným charakterem stop slov. Odstranění stop slov je i součástí původního řešení IS.

Další možností, jak snížit počet slov ve slovníku, a přitom zachovat, či dokonce zlepšit kvalitu vyhledávání, je použít ve slovníku pouze základní tvary slov – stemmy a lemmata. Lemma příslušného slova je jeho základní slovníková jednotka. V případě češtiny a podstatných či přídavných jmen jde obvykle o první pád jednotného čísla. Stem je pak kmenem či kořenem

slova – zpravidla vzniká odstraněním přípony či předpony. Samotný stem pak nemusí být smysluplné slovo, v tom je hlavní rozdíl mezi stemem a lemmatem. U některých slov se mohou oba tyto základní tvary shodovat, u jiných ne.

Pro český jazyk je bohužel častější druhý případ. Čeština má velmi složité tvarosloví, řadu výjimek a nepravidelností. Lemmatizace se tak obvykle realizuje za pomoci slovníkových dat, protože algoritmicky je velmi složité ošetřit všechny výjimky. Nejčastější problém představují mnohoznačná slova. Obvykle uváděným příkladem je slovo *tancích*, jehož lemmatem může být jak slovo *tank*, tak slovo *tanec*. Konkrétní lemma lze v tomto případě určit pouze z okolního kontextu.

Český stemming je naproti tomu realizovatelný i algoritmem, na základě sady pravidel. Jedním z takových algoritmů je práce [26]. Aplikace algoritmu CzechStemmer by měla vést k dalšímu snížení počtu slov ve slovníku S, a zároveň k zahuštění podsítí.

Posledním krokem, který povede ke snížení velikosti U, je omezení algoritmu pouze pro slova se střední frekvencí výskytu. Jde o opačný postup než v případě stop slov. Stop slova generují příliš rozsáhlé podgrafy a z hlediska uspořádání výsledek spíše zkreslují než naopak. Na opačném konci frekvenčního spektra stojí slova, která můžeme nazvat unikátní. To jsou slova, která se vyskytují jen ve zlomku dokumentů a generují na rozdíl od stop slov velmi malé podgrafy. Ty navíc velmi často tvoří pouze izolované body bez propojujících hran, protože jednotlivé stránky obsahující tato unikátní slova spolu nejsou propojeny. Tento fakt činí unikátní slova nezajímavými z hlediska analýzy odkazů a tedy z hlediska PageRanku.

Dalším důvodem pro vypuštění unikátních slov je samotná nízká frekvence jejich výskytu. Ta zajišťuje jejich dobré vyhledávání sama o sobě. Kvalitní hodnotící funkce je tak potřeba spíše pro často vyhledávaná slova s průměrným výskytem.

8.2 Výpočet lineárního systému

Nad zredukovanou množinou S nyní zformulujeme výpočet algoritmu IS pomocí lineárního systému. Výsledky uvedené v kapitole 7 dokazují, že pro testovací data je tento výpočet výrazně rychlejší než tradiční metoda – hledání vlastního vektoru Mocninnou metodou.

Abychom mohli provést výpočet příslušného rank vektoru, budeme nutně potřebovat personalizační vektor $\mathbf{v}_s$. Ten vygenerujeme z pravděpodobnosti P'. Před tím je ale nutné určit, jakou funkci $R_q(j)$ použijeme. Místo základní binární funkce byla pro návrh algoritmu CZDIS využita normalizovaná váha slova v dokumentu $w_{s,d}$, vypočtená ze vztahu:

$$w_{s,d} = fs_{s,d}. \log \frac{|D|}{|s \in D|} \tag{8.1}$$

Kde $fs_{s,d}$ je frekvence výskytu slova s v dokumentu d, $|D|$ je počet všech dokumentů a $|s \in D|$ počet dokumentů obsahujících slovo s.

Normalizovaná váha slova v dokumentu vyjadřuje důležitost konkrétního slova v dokumentu přesněji, než binární funkce. Lze tedy předpokládat, že i jako funkce $R_q(j)$ by měla mít pozitivní vliv na kvalitu výsledného hodnocení. Vzhledem k tomu, že testování kvality výsledné funkce je prováděno na základě subjektivního hodnocení testujících dobrovolníků, je ovšem velmi obtížné stanovit konkrétní hodnotu rozdílu v kvalitě hodnocení při použití $w_{s,d}$ nebo binární funkce.

Ze vztahu pro P' můžeme nyní za pomoci vztahu pro $w_{s,d}$ odvodit $\mathbf{v}$ pro příslušné slovo d. Platí, že $v_{si} = w_{s,d}(i)$. Počáteční nastavení vektoru řešení $\mathbf{x} : x_i = 1/|q|$, kde $|q|$ je velikost generované podsítě a $i \in < 1, |q| >$.

8.3 Formulace paralelního algoritmu

Řešení lineárního systému sice přináší zvýšení rychlosti výpočtu jednoho výpočetního kroku, a tím i celého výpočtu provedeného sekvenčně, ovšem algoritmus IS je ze své povahy přirozeně paralelní problém. Jde o klasický příklad SPMD - opakování stále stejného výpočtu na S různých datech. Paralelní výpočet několika podgrafů je tak logickým krokem k dalšímu zrychlení.

Výpočet je navržen jako master–worker algoritmus za použití knihovny openMPI. Ta zajišťuje nezbytné komunikační prostředky, stejně jako mechanizmus kontroly výpadku výpočetního uzlu. I když by bylo teoreticky možné implementovat i ošetření výpadku hlavního uzlu, součástí implementace tento mechanizmus není a výpadek hlavního uzlu je pro výpočet fatální.

Dekompozice problému je prováděna dynamicky, protože velikost jednotlivých podgrafů je velmi variabilní. Pro minimalizaci I/O operací jsou vygenerované datové struktury drženy v paměti a zasílány výpočetním uzlům. Vypočtené výsledky ukládají výpočetní uzly do sdíleného adresáře hlavního uzlu.

Nyní zbývá zformulovat oba algoritmy, jak řídící master proces, tak výpočetní worker proces.

Master algoritmus

1. nahrát inverzní index dokumentů příslušný slovníku S

2. pomocí indexu vygenerovat n úloh
3. všem n dostupným klientům přidělit úlohu
4. vygenerovat n úloh
5. dokud počet vygenerovaných úloh není roven počtu slov ve slovníku S, přidělovat výpočetním uzlům další úlohy, jakmile oznámí dokončení předchozí.
6. během tohoto cyklu stále udržovat n vygenerovaných úloh pro zajištění optimálního zatížení všech výpočetních uzlů
7. počkat na dokončení výpočtu na všech uzlech
8. ukončit program na všech uzlech

Worker algoritmus

1. dokud Master neukončí program provádí následující kroky
2. načíst data úlohy
3. řešit lineární systém
4. uložit příslušný vyřešený vektor

8.4 Implementace algoritmu

Pro vlastní implementaci algoritmu byl stejně jako v předchozím experimentu použit jazyk Python a moduly Scipy, Numpy, petsc4py. Pro řešení lineárního systému byla využita numerická knihovna PETSc a iterační metoda GMREs s LU předpomíněností, která se ukázala jako efektivní v předchozím experimentu. Koeficient α byl nastaven na 0.85 a kritériem konvergence byl rozdíl absolutních hodnot dvou po sobě jdoucích iterací 10^{-4}.

Základní operace pro sestavení úlohy – vytvoření incidenční matice a vektoru reprezentujícího dangling nodes – byly implementovány pomocí modulu Scipy, který zároveň poskytuje potřebné funkce pro MPI programování v Pythonu.

Pro výpočty pomocí knihovny PETSc pak byl využit modul petsc4py.

data	*S*	*U*	*N*	*poměr U/N*
TUL 2008/2	287989	3505887	22766	154
TUL 2008	315498	6853644	45531	150
Seznam B/3	1798903	76848849	499019	154
Seznam B/3	1805572	157689876	998037	158

Tabulka 8.1: Vliv počtu stránek na poměr U/N.

8.5 Výsledky experimentů

Experimenty a jejich výsledky jsou rozděleny do tří částí. Nejprve bylo provedeno ověření navržených teoretických postupů pro redukci počtu slov ve slovníku S. Druhý z experimentů se zaměřil na testování paralelní implementace – tedy na režii, zrychlení a efektivitu výpočtu. Posledním experimentem bylo uživatelské testování hodnotící funkce, jehož cílem bylo ověření kvality lokalizovaného hodnocení.

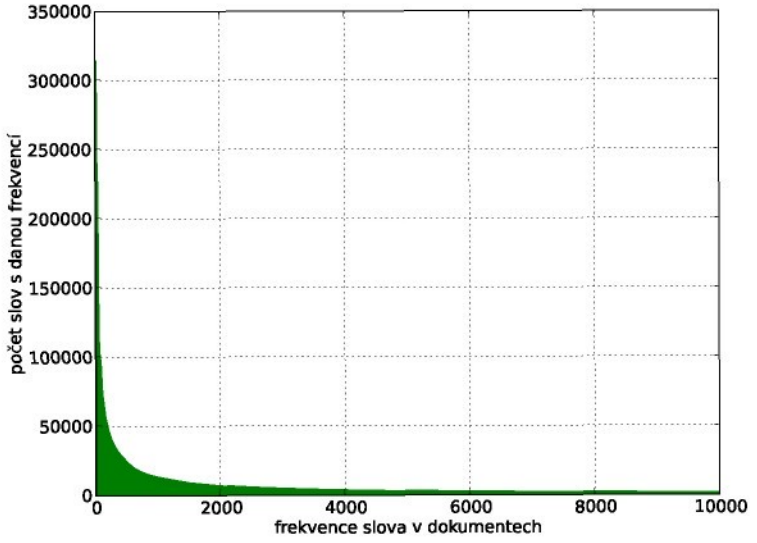
8.5.1 Vliv počtu stránek na poměr U/N

Na třech různých datových kolekcích byl sledován vliv rostoucího počtu stránek na poměr U/N. U každé datové kolekce byla velikost vygenerovaného slovníku S a počet dvojic dokument-slovo U změřena v polovině procesu zpracování dat a na jeho konci. Celkem tak byly k dispozici čtyři různé datové vzorky. Výsledky tohoto pozorování uvedené v tabulce 8.1 potvrzují teoretický předpoklad, že s rostoucím počtem stránek roste i U a S a poměr U/N tak zůstává přibližně stejný.

8.5.2 Redukce slovníku S o unikátní slova

Pro odstranění stop slov využíváme jejich seznam, někdy označovaný jako *negativní slovník*. Pro slova, která byla v teoretické části označena jako slova unikátní, takovýto slovník k dispozici nemáme. Pro jeho vygenerování je nejprve nutné stanovit si kritérium hodnocení, podle kterého rozhodneme, zda je slovo unikátní či nikoliv.

Jako kritérium unikátnosti byla v této práci zvolena frekvence výskytu slova v dokumentech. Jak můžeme vidět z obrázku 8.1 má graf počtu slov s určitou frekvencí výskytu podobu prudce klesající mocninné křivky. Tento graf byl vytvořen na největší z použitých datových kolekcí - CZdat. Slovník generovaný touto kolekcí, obsahuje celkem 1805572 slovních tvarů. Z nich pouze 23651, tedy 1,31%, se vyskytuje ve více než 500 dokumentech z této



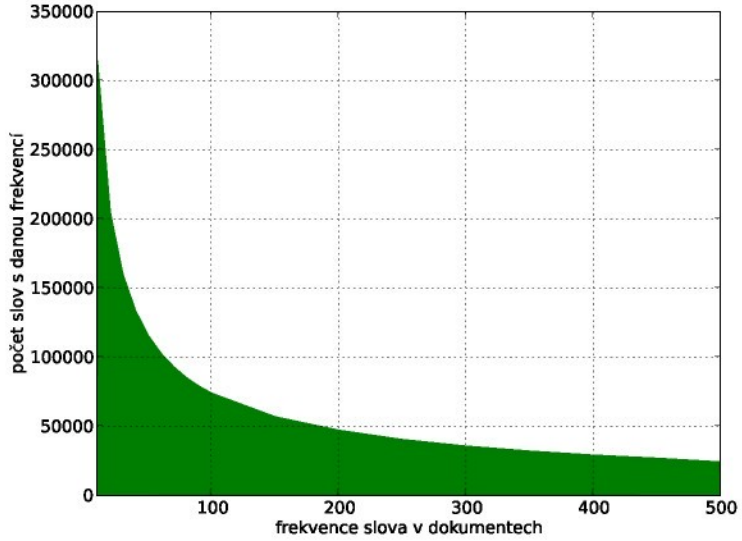
Obrázek 8.1: Rozložení slov vzhledem k frekvenci jejich výskytu v dokumentech.

kolekce, která celkem obsahuje 998037 dokumentů. Tato slova můžeme označit jako běžná.

Pro další hledání kritéria unikátnosti tedy zohledníme slova, která se vyskytují v méně než 500 dokumentech. Novou, detailnější křivku, zobrazuje graf na obrázku 8.2. Zde můžeme vidět, že prudký pokles přechází do mírnějšího přibližně v oblasti 100, dokumentů obsahujících dané slovo. Pokud na tuto křivku použijeme Paretův princip 80/20 [66] pro vyřazení 20% nejunikátnějších slov dostaneme se právě k číslu 100 jako kritériu pro unikátnost slova. Jako unikátní slova tedy budou vyřazena taková slova, která se vyskytují ve 100 a méně dokumentech.

Postupnou aplikací všech tří teoretických kroků - seznamu stop slov, stemmeru a vypuštění unikátních slov, dojde k postupné redukci slovníku S a to pro obě použité datové kolekce. K největší redukci vede krok třetí, tedy vypuštění unikátních slov. Výsledky aplikace jazykového modelu zobrazuje tabulka 8.2. Jednotlivé kroky jsou na sobě nezávislé a je možné je provést idnividuálně.

Pro urychlení celého procesu by bylo přínosnější nejprve provést redukci slovníku o unikátní slova a teprve po tomto kroku provádět stemming. Ten by se prováděl na několikrát menší množině slov. Převedení slov na stemy ovšem vede k tomu, že řada původně unikátních slov je nyní převedena na stejný tvar a generují tak svou podsíť, která je zajímavá pro další výpočet.



Obrázek 8.2: Detailní rozložení slov s výskytem v méně než 500 dokumentech.

Proto byl proces redukce sestaven v uvedeném pořadí.

8.5.3 Efektivita paralelního výpočtu

Výpočet byl proveden celkem na čtyřech testovacích slovnících. Vždy byl použit slovník úplný a poté slovník redukovaný. Pro porovnání byl nejprve celý algoritmus proveden jako sekvenční výpočet na jednom uzlu. Dále byl vypočten také klasický PageRank vektor a to jako sekvenční výpočet lineárního systému. Následně byl algoritmus CZDIS spuštěn paralelně, přičemž byl

datová kolekce	*velikost S size*	*U*	*poměr U/N*
tul neredukovaná	315498	6853644	150
tul bez stop slov	315268	6630472	145
tul pouze stemy slov	242371	6053120	133
tul_100 - bez unikátních sl.	8903	4789301	105
CZdat neredukovaná	1805572	157689876	158
CZdat bez stop slov	1805342	152555094	153
CZdat pouze stemy slov	1387907	136080806	137
CZdat_100 - bez unikátních sl.	100538	106512965	107

Tabulka 8.2: Redukce velikosti slovníku S aplikací jazykového modelu.

	tul_100		tul	
počet uzlů	zrychlení	čas (s)	zrychlení	čas (s)
1	1,00	4,688	1,00	81,789
2	1,97	2,380	1,89	43,275
4	3,82	1,227	3,19	25,639
8	7,23	0,648	6,87	11,905
16	14,28	0,328	13,23	6,180
32	24,89	0,188	22,35	3,659

Tabulka 8.3: Čas a zrychlení výpočtu pro slovník tul.

	CZdat_100		CZdat	
počet uzlů	zrychlení	čas (s)	zrychlení	čas (s)
1	1,00	56,131	1,00	939,666
2	1,91	29,388	1,85	507,928
4	3,55	15,811	3,04	309,101
8	7,11	7,895	6,32	148,681
16	13,94	4,027	12,85	73,126
32	23,92	2,347	20,48	45,882

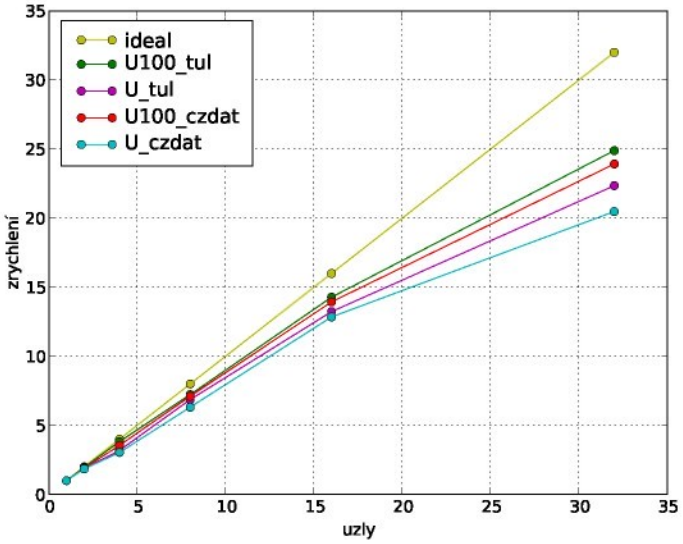
Tabulka 8.4: Čas a zrychlení výpočtu pro slovník CZdat.

po každém výpočtu zvýšen počet výpočetních uzlů na dvojnásobek. Celkem tedy paralelní algoritmus proběhl šestkrát nejprve na dvou a nakonec na 32 procesorech.

Časy jednotlivých výpočtů jsou uvedeny v tabulkách 8.3 a 8.4. Uvedený čas představuje celkový čas paralelního výpočtu včetně komunikace a sestavování dílčích úloh. Poměr zrychlení výpočtu, vypočtený jako podíl celkového času paralelního a sekvenčního výpočtu, je pak zobrazen v grafu na obrázku 8.3. Experimentální výsledky potvrzují teoretický předpoklad, že redukcí slovníku na slova generující větší podsítě, dojde ke snížení režie celého výpočtu.

Malé podsítě snižují efektivitu algoritmu, protože je jich příliš mnoho a náklady na vytvoření a komunikaci jsou výrazně větší než vlastní čas výpočtu. To se projevuje horší škálovatelností algoritmu – master nestíhá dostatečně rychle generovat příslušné dílčí úlohy a některé výpočetní uzly musejí čekat na přidělení úlohy. S rostoucím počtem výpočetních uzlů tak sice stále klesá výpočetní čas, ale bohužel i efektivita celého výpočtu.

Během vývoje algoritmu byl proveden také experiment s přípravou dílčích



Obrázek 8.3: Zrychlení (speedup) výpočtu.

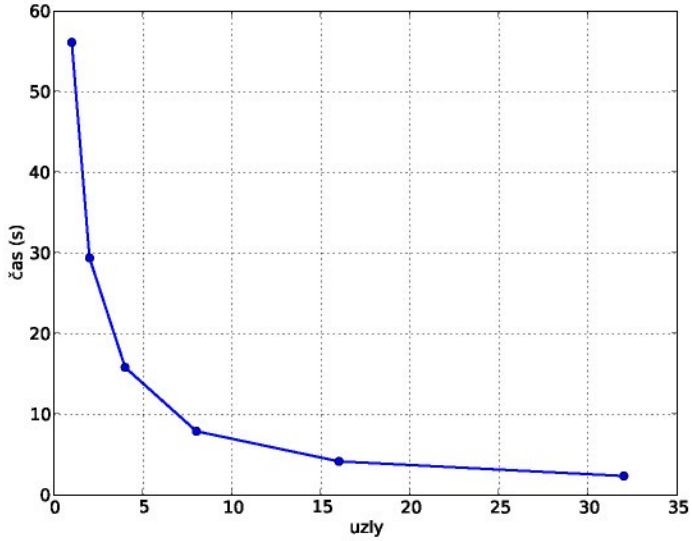
úloh před zahájením vlastního výpočtu. Ten sice prokázal lepší vyvažování zátěže na jednotlivých procesorech, ale celkový čas řešení v součtu přípravy a výpočtu byl horší než u zde prezentovaného řešení, při kterém master využívá volný strojový čas k přípravě úloh, proto byla tato vývojová větev ukončena.

U redukovaných slovníků je škálovatelnost lepší, nicméně i zde s rostoucím počtem výpočetních uzlů přestává zrychlení výpočtu kopírovat ideální křivku a začíná klesat. Čas a efektivita výpočtu nad redukovaným slovníkem U100_czdat jsou zobrazeny na obrázcích 8.4 a 8.5. Čas výpočtu zahrnuje celý čas běhu algoritmu, efektivita výpočtu byla vypočtena jako poměr mezi zrychlením výpočtu a počtem procesorů.

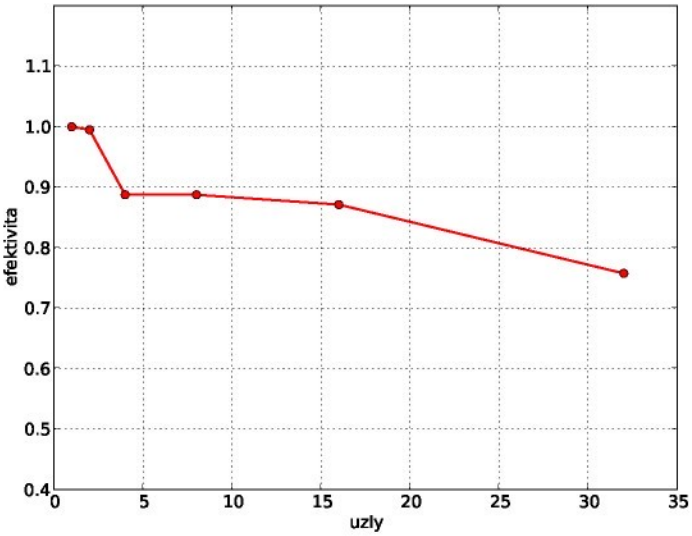
Jak můžeme vidět z výsledků v tabulce 8.5, teoretický poměr U/N a reálný poměr mezi časy výpočtu PageRank/CZDIS se poměrně výrazně liší. Teoretický poměr pro výpočet určený ze vztahu $0.75 * U/N$ [60] je v intervalu mezi 79 pro redukovaný slovník tul a 102 pro neredukovaný slovník CZdat. Reálný poměr je ve třech případech ze čtyř menší a výpočet tedy probíhá rychleji než teoretický odhad.

Ovšem pro neredukovaný slovník tul je reálný poměr pětkrát větší než teoretický. Logickou otázkou je, proč? Důvodem je opět redukce slovníku S a režie spojená se zpracováním malých sítí. Pro neredukovaný slovník je nutno vytvářet řadu malých podsítí, což vede ke značnému zpomalení celého procesu.

Pro srovnání byl vypočten také poměr mezi rychlostí výpočtu algoritmu



Obrázek 8.4: Čas výpočtu nad slovníkem S100_czdat.



Obrázek 8.5: Efektivita paralelního výpočtu nad slovníkem S100_CZdat.

	tul_100	*CZdat_100*	*tul*	*CZdat*
pagerank	0,162	14,820	0,162	14,820
sekvenční výpočet	4,688	56,131	81,789	939,67
teoretický poměr U/N	79	80	99	102
reálný poměr	28,936	3,787	504,870	63,41
čas výpočtu na 32 proc.	0,188	2,347	3,659	45,88
reálný poměr 32 proc.	1,163	0,158	22,589	3,10

Tabulka 8.5: Poměr PageRank/CZDIS.

CZDIS na 32 procesorech a sekvenčním výpočtem PageRanku. Z výsledků tohoto výpočtu je zřejmé, že časová náročnost paralelního výpočtu je přibližně třikrát větší, než pro klasický výpočet PageRanku. Ten je pro nasazení algoritmu nutné provést v každém případě, abychom získali hodnotící funkci pro unikátní slova. Celkový čas výpočtu hodnotící funkce algoritmu CZDIS je tak přibližně čtyřnásobkem času potřebného pro výpočet PageRanku. To dává dobré předpoklady pro reálné nasazení algoritmu CZDIS. Otázka, do jaké míry by se tento poměr upravil v případě paralelního výpočtu samotného PageRanku, bude předmětem dalšího zkoumání. Stejně jako celý PageRank vektor je totiž možné počítat paralelně také některé z rozsáhlejších generovaných podgrafů v algoritmu CZDIS.

8.5.4 Kvalita lokalizované hodnotící funkce

Pro testování kvality hodnotící funkce byla v případě algoritmu CZDIZ i ve zbytku této práce, použita stejná metodika jakou použili autoři algoritmu IS, s několika drobnými rozdíly. Počet testujících dobrovolníků byl zvýšen na 5 (v originále 3) a bodová škála byla rozšířena na pětistupňovou od 0, pro zcela nerelevantní výsledek, do 4, pro nejkvalitnější výsledek.

Do výsledků testování byla zahrunuta pouze množina stránek CZdat, protože se v průběhu testování ukázalo, že množina tul obsahuje příliš malé množství stránek a generované podsítě jsou tvořeny spíše izolovanými body a výsledná hodnotící funkce je zkreslená. Množina tul tedy posloužila zejména pro testování implementací a ověřování některých hypotéz. Pro hodnocení reálné kvality hodnotící funkce je nutné použít větší datovou kolekci, alespoň v rozsahu odpovídajícím kolekci CZdat.

Pro výpočet hodnotící funkce CZDIS byl využit redukovaný slovník S100 a k němu příslušející množina q-vektorů vypočtená již při experimentech s rychlostí výpočtu. Pro porovnání, a také jako řadící funkce pro slova mimo redukovaný slovník S100, byl vypočten také PageRank vektor příslušející

hledaný výraz	CZDIS	PageRank
aktuální akce	24,6	28,4
české články	18,2	24,8
financování	28,4	16,8
jazyk	20,5	10,2
dopis	28,4	16,2
pole	23,4	14,2
rozvrh	27,3	19,8
univerzitní ústav	23,2	29,8
aplikace	34,2	20,5
číslicový	23,4	23,8
průměrné hodnocení	25,16	20,45

Tabulka 8.6: Průměrná známka pro zadané dotazy

k množině CZdat.

Každý z dobrovolníků nejprve zvolil dva náhodné dotazy, čímž byla vytvořena množina deseti různých náhodně vybraných dotazů. Dále pak každý postupně zadal všech 10 dotazů a následně hodnotil navrácené výsledky. Výsledků měl k dispozici 20 – první polovina bylo 10 nejlepších výsledků CZDIS, druhá 10 nejlepších výsledků PageRanku. Těchto 20 výsledků bylo zobrazeno v náhodném pořadí, aby nedošlo k ovlivnění testujícího. Každý z výsledků byl ohodnocen známkou od 0 do 4 a z těchto známek se pak vypočetlo výsledné hodnocení.

Výpočet výsledků probíhal ve dvou krocích – nejprve byly sečteny výsledky každého z dobrovolníků pro každou hodnotící funkci. Maximální možné skore bylo 40 bodů. Druhým krokem pak byl výpočet průměrného bodového hodnocení, které je zobrazeno v tabulce 8.6.

Také pro český jazykový model byly zopakovány výsledky původního algoritmu IS a hodnotící funkce dosáhla lepších výsledků než PageRank. Lze tedy říci, že lokalizace algoritmu byla úspěšná a že algoritmus je při použití vhodného slovníku přenositelný i na jiné jazyky než angličtinu.

Jako kontrola činnosti dobrovolníků nepřímo posloužila slova s četností menší než 100 – konkrétně slovo „číslicový“. Výsledky pro toto slovo byly v obou případech řazeny dle algoritmu PageRank a jak je vidět z tabulky 8.6, je hodnocení algoritmu CZDIS přibližně stejné jako hodnocení algoritmu PageRank.

8.6 Zhodnocení úspěšnosti lokalizace

Úspěšná implementace českého jazykové modelu ukazuje, že algoritmus inteligent surfer je použitelný jako hodnotící funkce také pro některé jiné jazyky než angličtinu. To bylo hlavním cílem této práce. Protože algoritmus IS byl kritizován jako příliš pomalý pro reálnou aplikaci, bylo dalším cílem zrychlení procesu výpočtu. Potvrdil se teoretický předpoklad, že vhodná aplikace jazykového modelu vede i ke zrychlení celého výpočtu, aniž by tím utrpěla kvalita hodnotící funkce. Paralelní výpočet rank vektorů pak vede k dalšímu zrychlení výpočtu do řádu minut, což činí z algoritmu CZDIS použitelné řešení, pro řazení výsledků fulltextového vyhledávání založené na kontextu dotazu.

Stejně jako v případě původního algoritmu i v případě lokalizované verze dosahují víceslovné dotazy v průměru horšího hodnocení než PageRank, zatímco u jednoslovných dotazů je tomu naopak. Důvodem je zkreslení, ke kterému dojde v okamžiku kdy z vektorů hodnocení pro jednotlivá slova sestavujeme jeden hodnotící vektor pro víceslovný dotaz. V situaci kdy se tato dvě slova vyskytují jako ustálené slovní spojení, je zkreslení větší. Řešením tohoto problému spočívá v rozšíření základního unigramového slovníku i o nejpoužívanější bigramy se kterými pak bude dále z hlediska algoritmu nakládáno stejně jako s unigramy.

Pro další práci zůstává nedořešeno několik problémů. Krom již naznačeného rozšíření slovníku S o nejfrekventovanější bi či trigramy je to také implementace lemmatizace pomocí vhodného nástroje. Bude nutné ověřit, zda doplnění lemmatizace negativně neovlivní čas výpočtu. Lze předpokládat, že proces lemmatizace bude výpočetně o něco náročnější než stemming. Na druhou stranu by použití lemmat mohlo vést k větší redukci slovníku i k lepším výsledkům vyhledávání.

Aby mohl být algoritmus IS označen za skutečně univerzální řešení, bude nutné ověřit jeho použitelnost pro více jazyků. Lze předpokládat, že je použitelný pro jazyky západního typu. Existuje nicméně řada dalších jazyků, pro které obecně platí, že text IR je netriviální problém, jako arabština či jazyky východní Asie, a nelze se domnívat, že by algoritmu IS měl tvořit výjimku.

Kapitola 9

TSPRL – lokalizace algoritmu Topic Sensitive PageRank

Algoritmus TSPR jsme si podrobně popsali v kapitole 3.7.4. Ověření, do jaké míry je možné tento algoritmus lokalizovat – tedy použít i pro jiný jazyk než anglický, bylo jedním z cílů této disertační práce. Popis a výsledky lokalizačního procesu shrnuje následující kapitola.

Podle publikovaných výsledků se zdá být algoritmus TSPR poměrně použitelným řešením. V porovnání s klasickým PageRankem přináší lepší výsledky a přestože výpočetní náročnost personalizovaného algoritmu je k krát větší než u klasického algoritmu, je celý výpočet díky $k = 16$ realizovatelný v přijatelných časech i pro rozsáhlé datové kolekce. Určitou slabinou algoritmu je pouze volba kontextu dotazu, kde je použit teoretický model volby kontextu pomocí vyznačení slova v určité stránce.

V praxi je toto řešení nerealizovatelné, pokud se budeme držet pravidla, že ovládání vyhledávače má být maximimálně jednoduché. V tomto případě by bylo nutné, aby si uživatel instaloval speciální rozšíření do prohlížeče, které by obstaralo komunikaci si vyhledávačem. To je samozřejmě možné, ale vyžaduje to již jistou zkušenost uživatele a ochotu tento krok udělat. Což nelze očekávat od všech uživatelů a řešení by tak nebylo univerzální. Pokud navíc do modelu chování náhodného uživatele zahrneme jeho ochotu spolupracovat, je daleko snadnější nechat ho při vyhledávání zvolit preferovanou kategorii v dialogovém okně vyhledávače. A to jak po stránce implementační, tak po stránce ergonomie uživatelského rozhraní.

Lokalizace algoritmu má dva hlavní problémy. Tím prvním je nalezení dostatečně velké báze základních adres, potřebných pro sestavení tématických klasifikátorů T_k. Druhý problém je vytvoření frekvenčních slovníků potřebných pro trénování Bayesovského klasifikátoru. Při řešení druhého problému je ale již možné využít nalezené základní adresy k získání potřebných texto-

vých dat.

Přestože rychlost řešení algoritmu TSPR je například v porovnání s algoritmem Intelligent Surfer dostatečná, je teoreticky možné jeho další urychlení. Podobně jako u algoritmu IS, i zde jde o přirozeně paralelní problém. Opakovaně provádíme hledání vlastního vektoru matice $\mathbf{G}$, pouze pro její sestavení použijeme vždy jiný personalizační vektor $\mathbf{v}_k$. Můžeme tedy předpokládat, že i pro algoritmus TSPR by mělo dojít k urychlení výpočtu, pokud nahradíme sekvenční výpočet všech k vektorů Mocninnou metodou paralelním výpočtem lineárního systému na k uzlech. K dalšímu urychlení by mohlo dojít, pokud část výpočtu prováděného v originálním algoritmu během online zpracování dotazu, vypočítáme předem.

Nyní si projdeme jednotlivé problémy podrobněji a v rámci jejich řešení zformulujeme návrh lokalizovaného algoritmu označeného dále jako *TSPRL*.

9.1 Nalezení báze pro vytvoření tématických klasifikátorů

Haveliwalla využil pro sestavení tématických klasifikátorů Open Directory Project – což je katalog stránek spravovaný komunitou uživatelů podobně jako například známá encyklopedie Wikipedia. Jako hodnotu k zvolil 16, což je počet hlavních kategorií ODP. Z celkem 3 milionů adres, které ODP v roce 2002 obsahoval, použil 280 tisíc stránek, které byly součástí datové kolekce WebBase, kterou měl k dispozici pro testy.

Jestliže v roce 2002 byla odhadovaná velikost webu cca 2 miliardy stránek [80] a WebBase kolekce měla 120 milionů stránek, tvořila použitá báze 0,01% velikosti webu a 0,23% velikosti testovací kolekce.

V současné době obsahuje ODP 4,6 milionů stránek v různých jazycích. Českých stránek eviduje ovšem pouze 26 tisíc. Pokud vyjdeme z velikosti českého webu uvedené v kapitole 2.3.3, představuje báze tvořená stránkami v české části ODP opět 0,01% ze 300 milionů. Nejrozsáhlejším katalogem českých stránek ovšem není ODP, ale katalog Seznamu. Haveliwalla se komerčním katalogům vyhnul záměrně, protože se obával ovlivnění umístění do kategorií ze strany správce katalogu. Vzhledem k tomu, že Seznam má speciální katalog http://www.firmy.cz/ zaměřený na komerční sféru, lze katalog ostatních stránek umístěný na http://odkazy.seznam.cz/ považovat za dostatečně důvěryhodný zdroj dat pro vytvoření tématických klasifikátorů. Tento katalog obsahuje více než 110 tisíc adres, což představuje asi 0,03% webů v českém jazyce. Spojením těchto dvou zdrojů dohromady, dostaneme po odečtení duplicitních adres, 112 tisíc URL.

k	*výsledná kategorie*	*kategorie Seznam*	*kategorie ODP*
1	dům a bydlení	řemesla a služby, informační a inzertní servery, volný čas a zábava	domov
2	eshopy	obchody a obchůdky	eshopy
3	hry	hry	hry
4	kultura	kultura a umění	kultura
5	obchod	obchody a obchůdky, velkoobchod a výroba, vše pro firmy	obchod
6	počítače	počítače a internet	počítače
7	instituce	instituce a úřady, informační a inzertní servery	reference
8	společnost	lidé a společnost	společnost
9	sport	sport	sporty
10	cestování	cestování	státy
11	věda	věda a technika	věda
12	volný čas	volný čas a zábava, erotika	volný čas
13	zdraví	první pomoc a zdravotnictví, informační a inzertní servery	zdraví
14	zpravodajství	zpravodajství	zpravodajstvi

Tabulka 9.1: Výsledných 14 kategorií pro tvorbu tématických klasifikátorů T_k.

Při spojování obou zdrojů je nutné ještě provést sjednocení kategorií, protože oba katalogy používají vlastní dělení. Tabulka 9.1 zobrazuje jak byly kategorie obou katalogů sloučeny do výsledných čtrnácti kategorií. Jak je z tabulky vidět, některé kategorie Seznam katalogu byly při zařazování tomto slučování rozděleny na do více částí.

Nově vytvořené kategorie byly poté použity pro algoritmus TSPRL. Odkazy nejsou mezi jednotlivé kategorie rozděleny rovnoměrně, počty URL v jednotlivých sekcích ukazuje tabulka 9.2. Tyto počty jsou zároveň počtem nenulových prvků personalizačních vektorů $\mathbf{v}_k$.

Uvedených 112 tisíc URL posloužilo také jako startovací množina stránek pro vytvoření datové kolekce *CZdat2*, která sloužila jako testovací množina pro experimenty s lokalizovanou verzí TSPR. Její podrobná charakteristika je uvedena v kapitole 6.1.3.

Počet URL dostupných pro vytváření tématických klasifikátorů vznik-

k	*téma*	*počet URL*
1	dům a bydlení	2304
2	eshopy	873
3	hry	9621
4	kultura	19176
5	obchod	4713
6	počítače	2842
7	instituce	5075
8	společnost	27565
9	sport	10926
10	cestování	7089
11	věda	5537
12	volný čas	11012
13	zdraví	1782
14	zpravodajství	3928
	celkem	112443

Tabulka 9.2: Velikost jednotlivých tématických klasifikátorů T_k.

lých sloučením obou zdrojů je k celkovému počtu českých stránek v řádově stejném poměru, jako u originálního algoritmu. Protože testovací množina CZdat2 byla výrazně menší než WebBase, tvoří URL získané báze 2,18% všech testovacích dat, což je výrazně více než tomu bylo u Haveliwallova experimentu. Zaměřování pomocí této báze by tak mělo teoreticky přinést srovnatelné, nebo dokonce lepší výsledky.

9.2 Určení příslušnosti dotazu k určitému tématu

Abychom se mohli rozhodnout, kterou z kategorií použijeme pro zaměřování dotazu, musíme pochopitelně nejprve určit do jaké kategorie dotaz pravděpodobně patří. V originálním algoritmu je k tomu využito Naivního Bayesovského klasifikátoru.

Pro určení s jakou pravděpodobností spadá kontext dotazu Q do kategorie j reprezentované třídou c_j tedy využijeme vztahu (3.6) popsaného v kapitole 3.7.4.

$$P(c_j|Q') = \frac{P(c_j).P(Q'|c_j)}{P(Q')} \propto P(c_j) \prod_i P(q_i|c_j)$$

Jestliže stejně jako v originálním algoritmu nastavíme pravděpodobnost jednotlivých tříd uniformní na hodnotu $P(c_j) = 1/k$, zbývá nám určit pravděpodobnost, s jakou jednotlivá slova přísluší k určité třídě - $P(q_i|c_j)$. Ta se v originálním algoritmu počítá až v okamžiku zadání konkrétního dotazu. Výpočet této pravděpodobnosti je ovšem možný již v rámci přípravy dat – offline. Tento krok sice o něco zvýší délku trvání offline procesu, ale výrazně zredukuje online zátěž. Ta je z hlediska hodnotící funkce kritická, protože vyhledávací server ve špičce řeší desítky dotazů za vteřinu a přílišná délka trvání online procesu může odradit uživatele od dalšího využívání daného vyhledávače.

Pro určení pravděpodobností $P(q_i|c_j)$ offline, využijeme frekvenční slovníky vytvořené ze slov, která obsahují stránky tvořící bázi určité kategorie. Tyto slovníky poslouží jako základní prvky pro konstrukci unigramového jazykového modelu. Slovník S_j si definujeme jako množinu všech slov (termů) v bázi kategorie j. Každý slovník bude zároveň reprezentován vektorem $\mathbf{f}_j$. Tento vektor definujeme jako n-rozměrný, a n jako počet slov v kompletním slovníku báze, který můžeme definovat jako:

$$S = \bigcup_k S_j.$$

Pro jednotlivé prvky vektoru $\mathbf{f}_j$ pak platí, že:

$$f_{ji} = \begin{cases} \phi(i) \text{ pokud je slovo } i \in S_j \\ 0 \text{ v opačném případě} \end{cases}$$

Kde $\phi(i)$ je frekvence výskytu slova i ve slovníku S_j. Z takto sestavených vektorů následně sestrojíme matici $\mathbf{D}$, pro kterou platí, že její j-tý sloupec je tvořen vektorem $\mathbf{fs}_j^T$. Normalizací prvků matice $\mathbf{D}$ dle vztahu:

$$d_{m,n} = d_{m,n}/sum(d_m)$$

získáme matici $\mathbf{D}_{norm}$, jejíž jednotlivé prvky tvoří hledané pravděpodobnosti $P(q_i|c_j)$. Řádek matice pak tvoří pravděpodobnostní vektory pro jednotlivá slova $q \in S$.

V případě báze datové kolekce CZdat2 byla matice $\mathbf{D}_{norm}$ (n, k) rozměrná, přičemž $k = 14$ a $n = 838238$, což byl celkový počet slov ve slovníku báze definované celkem 112443 stránkami. Velikost jednotlivých slovníků pro konkrétní kategorie jsou zobrazeny v tabulce 9.3. Zároveň tato čísla představují počet nenulových prvků příslušného vektoru $\mathbf{f}_j$. Jejich součet jak pak počtem nenulových prvků matice $\mathbf{D}_{norm}$.

Při vytváření slovníků byly aplikovány první dva kroky redukce velikosti slovníku, popsané v kapitole 8.1 – vypuštění stop slov a převedení slov na

k	*téma*	*velikost S*
2	eshopy	52521
13	zdraví	74295
1	dům a bydlení	35194
6	počítače	78672
14	zpravodajství	112867
5	obchod	131294
7	instituce	136912
11	věda	133699
10	cestování	158376
3	hry	218056
9	sport	202077
12	volný čas	164620
4	kultura	235931
8	společnost	386758
	komplet báze	838238
	počet nnz matice $\mathbf{D}_{norm}$	2121272

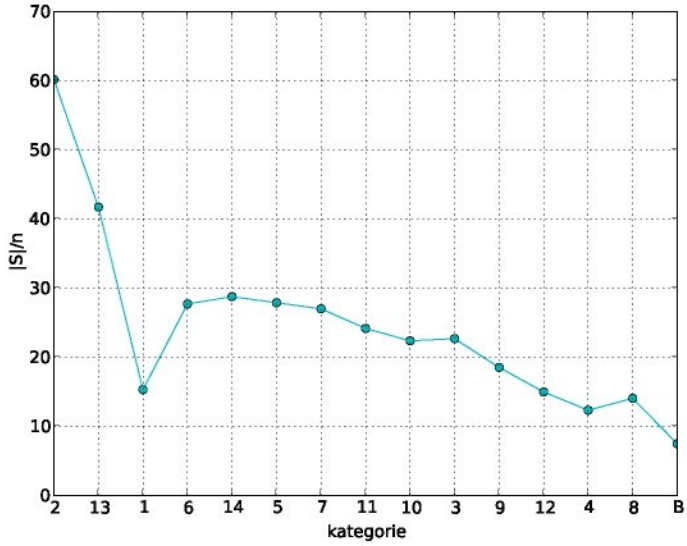
Tabulka 9.3: Počty slov v jednotlivých slovnících S_j.

stemmy. Vypuštění unikátních slov (3. krok) by v tomto případě nebylo přínosem, protože unikátní slovo typické například pouze pro jednu z kategorií, povede k zaměření výsledného hodnocení právě na tuto kategorii. Velikosti slovníků navíc není v případě TSPRL tak klíčovým faktorem jako u CZDIS. V tabulce 9.3 jsou uvedeny velikosti již redukovaných slovníků. Kategorie jsou seřazené dle velikosti báze.

V grafu na obrázku 9.1 je zobrazena křivka vyjadřující poměr mezi počtem stránek tvořících bázi příslušné kategorie a velikostí generovaného slovníku. Je zřejmé, že s rostoucím počtem zpracovávaných dokumentů, se počet stránek stále zvětšuje, ale přírůstek nových stránek klesá. Poměr mezi počtem stránek a počtem slov se tak neustále zmenšuje, ovšem jednodušše definovatelná matematická závislost mezi těmito veličinami není.

Dobrým příkladem podporujícím toto tvrzení je že početně málo obsazená kategorie eshopy generuje větší slovník než početnější kategorie bydlení, což je dáno velmi širokým počtem témat u elektronických obchodů.

Slovník generovaný kompletní kolekcí CZdat2 obsahuje dokonce po redukci přibližně stejně slov jako je stránek v kolekci. Teoreticky může velikost slovníku S růst spolu s rostoucím počtem stránek do nekonečna. Tento fakt je způsoben specifickým prostředím jazyka WWW stránek, ve kterých se vyskytuje řada slovních tvarů, která v běžném slovníku přirozeného ja-



Obrázek 9.1: Poměr mezi velikostí slovníku S a počtem stránek v bázi příslušné kategorie n. Kategorie uspořádány dle velikosti báze. Komplet báze je označená B.

zyka nenajdeme. Jde například o nejrůznější čísla, zdrojové kódy programů a řadu dalších. Většinu z těchto slov je ovšem potřeba indexovat. Podrobnější zkoumání složení jednotlivých slovníků a sledování, které typy slov je možné z dalšího zpracování vyřadit a které naopak zachovat, může být jedním ze směrů dalšího výzkumu.

9.3 Paralelní výpočet jednotlivých vektorů

Výpočet jednotlivých, tématicky zaměřených vektorů je přirozeně paralelním problémem, protože algoritmus TSPR jako celek lze snadno rozložit na k sekvenčních problémů. U těchto dílčích úloh navíc není nutná žádná synchronizace a komunikace během výpočtu. Pro tuto základní paralelizaci výpočtu tedy potřebujeme pouze mít k dispozici k výpočetních uzlů + sdílený adresář pro zdrojová a výsledná data.

Výpočet jednotlivých vektorů opět provedeme řešením lineárního systému, za pomoci knihovny PETSc a iterační metody GMRES s LU předpodmíněním. Jednotlivé personalizační vektory $\mathbf{v}_j$ pro $j = 1 \ldots k$ sestavíme za pomoci tématických klasifikátorů T_j takto:

$$v_{ji} = \begin{cases} \frac{1}{T_j} \text{ pokud } i \in T_j \\ 0 \text{ v opačném případě} \end{cases}$$

Jednotlivé personalizační vektory a matici $\mathbf{A} = (\mathbf{I} - \alpha\mathbf{P})$ uložíme do sdíleného adresáře clusteru s tím, že personalizační vektory předem přiřadíme k jednotlivým výpočetním uzlům. Toto přiřazení provedeme jednouše tak, že příslušný soubor s personalizačním vektorem pojmenujeme názvem konkrétního uzlu. Následně spustíme na všech k výpočetních uzlech následující algoritmus:

1. načti jméno uzlu
2. načti z hlavního uzlu příslušný vektor v a matici A
3. vyřeš lineární systém
4. výsledek ulož na hlavní uzel

Lze očekávat, že díky značnému objemu přenášených dat v bodě 2 nebude zrychlení výpočtu kopírovat ideální křivku a že režie síťových přenosů povede k určitému snížení efektivity výpočtu.

9.4 Algoritmus ohodnocení výsledků dotazu

Ohodnocení výsledků dotazu Q je online proces, kterému předchází sestavení několika různých indexů offline. URL všech stránek v kolekci CZdat2 (či obecně ve zpracovávaných datech) je nutné zaindexovovat do klasického inverzního indexu dokumentů a dále je potřeba vytvořit index bázových slov, který ke každému slovu přiřadí příslušný řádkový vektor matice $\mathbf{D}_{norm}$.

Posledním indexem potřebným pro vyhodnocení dotazu je index jednotlivých tématických ranků, který pro každý z dokumentů v kolekci CZdat2 přiřazuje buď 15 různých hodnot tématického hodnocení (14 kategorií + obecné), případně pouze hodnotu rankingu pro kategorii *obecné*, která je vypočtena jako normální (nezaměřený) PageRank.

Online zpracování dotazu Q pak bude probíhat dle následujícího algoritmu:

1. Dotaz Q rozdělit na jednotlivá slova q_i.

2. Otestovat existenci jednotlivých slov $q \in Q$ v indexu bázových slov. Pokud bylo slovo v indexu nalezeno, spárovat slovo a příslušný řádek matice $\mathbf{D}_{norm}$ obsahující pravděpodobnosti jeho výskytu v jednotlivých třídách. Pokud slovo v indexu bázových slov není, použít třídu *obecné* a pravděpodobnost příslušnosti k této třídě nastavit na 1.

3. Opakováním kroku 2. přiřadit pravděpodobnost $\forall q_i \in Q$.

4. Pomocí inverzního indexu dokumentů sestavit množinu výsledků V. Do této množiny zahrnout pouze stránky obsahující všechna slova z dotazu Q.

5. Ohodnotit jednotlivé výsledky podle vzorce (3.7): $s_{qv} = \sum_j P(c_j|q).r_{jd}$

9.5 Výsledky experimentů

V rámci testování lokalizovaného algoritmu byl nejprve proveden test efektivity paralelního výpočtu a poté proběhlo otestování samotné hodnotící funkce. Výsledky obou experimentů shrnuje následující část práce.

9.5.1 Efektivita paralelního výpočtu

Výpočet všech 14 vektorů byl nejprve proveden na jednom výpočetním uzlu sekvenčně. Měření času probíhalo stejně jako v předchozím případě pomocí časového razítka procesoru. Měřen byl tentokrát celkový čas běhu algoritmu, nikoliv samotný výpočet. Důvodem pro toto rozhodnutí byla rozdílná I/O režie při sekvenčním a paralelním výpočtu. Paralelní výpočet byl proveden na 14 uzlech clusteru Hydra. Opět byl měřen čas od okamžiku aktivace jednotlivých výpočetních skriptů příkazem cluster-fork, až do okamžiku uložení posledního vypočteného vektoru na disk hlavního uzlu clusteru.

Z naměřených časů uvedených v tabulce 9.4 vidíme, že režie I/O operací se negativně podepisuje na efektivitě paralelního výpočtu. Efektivita vypočtená jako podíl zrychlení a počtu procesorů je pouze 0,39. K největšímu zpoždění dochází před samotným zahájením výpočtu v době, kdy je soubor s maticí A (486 MB ve formátu PETSc.SEQAIJ) přenášen na jednotlivé výpočetní uzly. Řešením tohoto problému by mohlo být nakopírování matice na jednotlivé uzly pomocí NFS multicastu, bohužel v době testování paralelního výpočtu nebyla tato možnost na clusteru Hydra k dispozici.

I přes relativně nízkou efektivitu se paralelní výpočet provede 5,4 krát rychleji, takže při dostupné volné výpočetní kapacitě je vhodnou metodou

sekvenční výpočet	231,70 s
paralelní výpočet na 14 uzlech	42,63 s
zrychlení paralelního výpočtu	5,43
efektivita paralelního výpočtu	0,39

Tabulka 9.4: Efektivita výpočtu algoritmu TSPRL na množině CZdat2.

pro výpočet jednotlivých vektorů. Pro rozsáhlejší datové struktury, než je CZdat2, by bylo nutné věnovat optimalizaci přenosů více pozornosti.

9.5.2 Kvalita lokalizované hodnotící funkce

Pro testování kvality byla využita stejná metodika jako u algoritmu CZDIS, tedy skupina 5 testujících dobrovolníků, která hodnotila jednotlivé výsledky vyhledávání pětibodovou škálou (viz. kap. 8.5.4). Testování probíhalo nad množinou CZdat2, která byla sestavena právě pro tento účel.

Stejně jako v případě algoritmu CZDIS, i tentokrát každý z testujících dobrovolníků nejprve náhodně zvolil dva dotazy. Protože k testu této lokalizované funkce došlo o něco později než k testu CZDIS, a složení skupiny dobrovolníků bylo jiné, nebyla pro testování použita slova zvolená v předchozím experimentu a dobrovolníci dostali prostor pro navržení vlastních dotazů. Pokud by testovali slova z předem určené množiny, mohl by u nich vzniknout nežádoucí dojem, že experimentální systém je pro tato slova určitým způsobem „vyladěn".

Celkem tedy bylo opět k dispozici 10 náhodně zvolených dotazů. Pro každý dotaz vrátil experimentální systém 10 nejlépe hodnocených výsledků dle tématického ranku a 10 nejlépe hodnocených výsledků s nezaměřeným PageRankem.[1] Výsledky byly náhodně promíchány a každý z dobrovolníků je hodnotil na stupnici od 0 (nekvalitní výsledek) do 4 (nejkvalitnější). Pro každou z funkcí byl uložen součet udělených bodů pro testovaný dotaz. Nakonec byl z hodnocení od jednotlivých dobrovolníků vypočten aritmetický průměr, který je uveden v tabulce 9.5. Pro větší přehlednost byly výsledky vyneseny také do grafu, zobrazeného na obrázku 9.2.

U sedmi testovacích dotazů byla spokojenost dobrovolníků s uspořádáním pomocí funkce TSRPL lepší, než s uspořádáním pomocí funkce PageRank. Také celková průměrná známka pro TSPRL je lepší. Lze tedy říci, že i v případě TSPR algoritmu je jeho lokalizace do češtiny možná a že se podařilo

[1] Krom toho také 10 výsledků algoritmu CZDIS, ale výsledky srovnání TSPRL a CZDIS jsou popsány až v následující kapitole.

dotaz / algoritmus	TSPRL	PageRank
horské kolo	32,2	21,8
vážná hudba	25,2	18,7
český ráj	21,2	17,2
lyžování	24,1	12,1
předpověď počasí	28,2	22,7
dovolená	17,6	20,2
kanárské ostrovy	29,8	18,2
rýma	18,8	14,3
počítačová simulace	27,4	23,2
jazyk	15,2	20,1
průměrná známka	23,97	18,85

Tabulka 9.5: Průměrná známka pro jednotlivé testovací dotazy TSPRL.

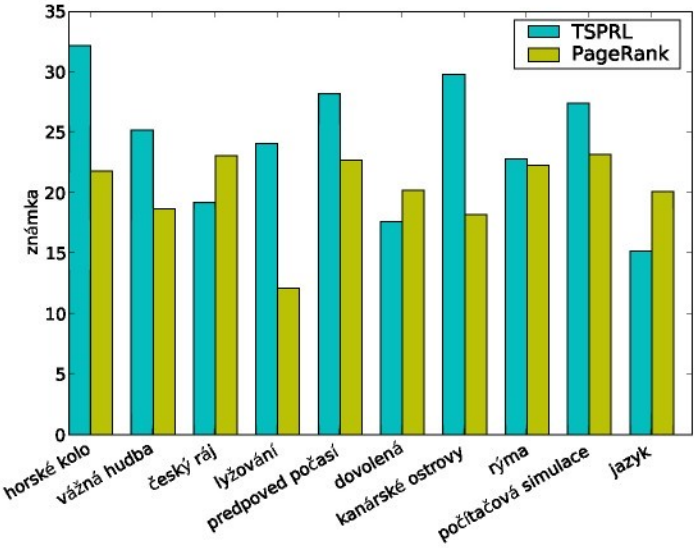
potvrdit teoretický předpoklad dobrých výsledků hodnotící funkce TSPRL na testovací množině CZdat2.

9.6 Zhodnocení úspěšnosti lokalizace a použitelnosti algoritmu

Podařilo se ověřit, že lokalizace algoritmu TSPR do češtiny je možná. V řadě případů bylo ovšem hodnocení obou porovnávaných algoritmů téměř stejné a přínos tématicky zaměřeného hodnocení nebyl tak velký, jak bylo očekáváno. Pokusme se nyní zodpovědět otázku proč. Odpověď je potřeba hledat v oblasti sestavení zaměřovaných vektorů – tedy v pravděpodobnosti přiřazení dotazu k jednotlivým třídám.

Výstup Bayesovského pravděpodobnostního modelu – pravděpodobnost s jakou dotaz Q patří do kategorie j, ukazuje pro jednotlivé testovací dotazy tabulka 9.6. Do jednotlivých tabulek byly zaznamenány pouze ty kategorie, pro které byla $P(c_j|q) >= 0,05$ Můžeme vidět, že zejména v případě jednoslovných dotazů nebyl pravděpodobnostní model příliš úspěšný. Kontext dotazu je v tomto případě příliš malý na to, aby mohl přesně určit téma. Dobře to demonstruje mnohovýznamové slovo „jazyk“.

Na algoritmu TSPR je dobře vidět rychlost vývoje v oblasti WWW. Za dobu, která uplynula od publikace originálního algoritmu TSPR, znalosti webmasterů v oblasti SEO hodně pokročily, a výsledek algoritmu TSPRL by byl poměrně hodně ovlivnitelný pomocí SEO technik ať již etických či nikoliv. I pokud vynecháme z hodnocení problematiku SEO Spamu, může



Obrázek 9.2: Průměrná známka pro jednotlivé testovací dotazy.

prostý obsah stránek hodně ovlivnit hodnocení pomocí TSPR(L). Například u dotazu „horské kolo“ se na stránce elektronického obchodu zaměřeného na prodej kol bude tento výraz vyskytovat u každé z položek v katalogu, zatímco u sportovního článku, který by mohl být z hlediska uživatele hodnocen jako kvalitnější zdroj, se hledané slovní spojení může vyskytnout jednou či dvakrát.

Tento fakt je dobře patrný na výsledcích uvedených v tabulce 9.6. Lze vypozorovat, že téměř pro všechny testovací dotazy je velmi pravděpodobné zaměření na kategorie obchod či eshopy. Důvodem může být to, že při tvorbě komerčních projektů je obvykle věnována daleko větší pozornost SEO technikám. Klíčová slova se v textu vyskytují tak často, jak je to možné, stránky jsou často odkazovány v rámci různých výměnných sítí apod. Navíc je tato kategorie poměrně široká, tak jako je široké spektrum různých obchodů. Je zřejmé, že sestavení tématických klasifikátorů podle zařazení stránek do kategorií katalogů, není ideální volbou, pokud ponecháme volbu zaměřovacího vektoru na pravděpodobnostním modelu. Z hlediska vyhodnocování stránek u kterých byla provedena optimalizace zohledňující hypertextové odkazy tedy tento algoritmu více méně kopíruje chování standardního PageRanku a optimalizované stránky dosahují lepšího hodnocení. Stejně jako PageRank by ani algoritmus TSPR nebylo možné použít jako samostatnou hodnotící funkci, pouze jako část kompozitní hodnotící funkce.

Pokud bychom použili naznačené řešení – ponechat uživatele, ať si zvolí

konkrétní téma či témata sám, mohl by naopak tohoto faktu uživatel využít k odfiltrování obchodních stránek, tím že kategorii obchod z výsledků vyřadí předem. Pokud se budeme držet principu, že průměrný uživatel nechce složité ovládací rozhraní, nechce se učit složitou syntaxi a dokonce si ani nechce příliš komplikovat práci zpřesňováním svých dotazů pro vyhledávač, ale přes všechnu tuto neochotu vyžaduje pokud možno ideálně řazené výsledky, bude nutné pro tvorbu jednotlivých tématických klasifikátorů využít jinou metodu než katalogy WWW stránek. Kromě ne vždy jasných kategorií, mají katalogy další podstatnou nevýhodu, a tou je postupné ukončování jejich provozu. Zatímco v roce 2000 byl katalog běžnou formou vyhledávání na Internetu, o deset let později majorita uživatelů vyhledává pomocí fulltextů a nově přicházející generace uživatelů již ani nemusí tušit, že se v minulosti nějaké katalogy používaly. Nové stránky se v katalozích neobjevují, stačí si porovnat velikost katalogu ODP v době publikace TSPR a dnes. Jinou metodou použitelnou pro tvorbu tématických klasifikátorů by se tak mohly stát algoritmy používané pro shlukování dokumentů či pro automatickou detekci tématu. Prověření, zda je efektivita těchto algoritmů dostatečná pro aplikaci na rozsáhlé datové kolekce bude předmětem dalších výzkumů.

horské kolo	
sport	0,42
obchod	0,19
cestování	0,18
eshopy	0,16

vážná hudba	
kultura	0,48
zpravodajství	0,21
eshopy	0,13
obchod	0,08

český ráj	
cestování	0,35
obchod	0,29
sport	0,11
zpravodajství	0,08
instituce	0,05

lyžování	
eshopy	0,36
obchod	0,22
sport	0,19
zpravodajství	0,12
cestování	0,09

předpověď počasí	
zpravodajství	0,68
cestování	0,11
věda	0,08
instituce	0,03

dovolená	
obchod	0,78
cestování	0,12
zpravodajství	0,06

kanárské ostrovy	
obchod	0,56
cestování	0,34

rýma	
zdraví	0,54
zpravodajství	0,25
obchod	0,12

počítačová simulace	
hry	0,38
počítače	0,18
eshopy	0,12
věda	0,09

jazyk	
počítače	0,19
zpravodajství	0,19
kultura	0,18
věda	0,16
zdraví	0,12
instituce	0,07

Tabulka 9.6: Pravděpodobnosti $P(c_j|q)$ pro jednotlivé testovací dotazy. V tabulkách jsou uvedeny pouze kategorie s $P >= 0,05$.

Kapitola 10

Srovnání obou lokalizovaných algoritmů

Porovnání obou lokalizací personalizovaných algoritmů je posledním experimentem představeným v této práci. Toto srovnání proběhlo v rámci testování algoritmu TSPRL jak již bylo naznačeno v kapitole 9.5.2. Testujícím uživatelům bylo kromě výsledků TSPRL a PageRanku předloženo také 10 nejlepších výsledků uspořádaných dle algoritmu CZDIS.

Teoretický předpoklad výsledku srovnání lze vyslovit na základě testů obou algoritmů. Algoritmus CZDIS dosahoval lepšího hodnocení než PageRank zejména v případě jednoslovných dotazů, zatímco u dotazů složených z více slov, byl hodnocen buď hůře a nebo přibližně stejně jako PageRank. U algoritmu TSPRL tomu bylo naopak. Víceslovné dotazy vytvářejí širší kontext dotazu a hodnocení u víceslovných dotazů tak bylo obvykle lepší než PageRank, zatímco u jednoduchých dotazů TSPRL na PageRank ztrácel. Chování obou algoritmů by se tedy mělo projevit i při jejich vzájemném porovnání, TSPRL by měl mít lepší výsledky pro složené dotazy, CZDIS naopak pro jednoslovné.

10.1 Aplikace CZDIS na kolekci CZdat2

Na datovou kolekci CZdat2 byl aplikován algoritmus CZDIS, tak jak byl představen v kapitole 8. Vzhledem k technickým problémům, které se v průběhu přípravy závěrečného experimentu objevily na clusteru Hydra, byl výpočet proveden pouze jako sekvenční na jednom procesoru. Primárním cílem výpočtu nebylo tentokrát testování paralelní implementace na další datové kolekci, ale příprava hodnotících vektorů pro zamýšlené porovnání algoritmů. Z tohoto důvodu nejsou výsledky experimentů nad kolekcí CZdat2 součástí

dotaz / algoritmus	TSPRL	CZDIS	PageRank	Kombinace
horské kolo	32,2	20,2	21,8	32,2
vážná hudba	25,2	18,2	18,7	25,2
český ráj	21,2	17,8	17,2	21,2
lyžování	24,1	31,5	12,1	31,5
předpověď počasí	28,2	19,2	22,7	28,2
dovolená	17,6	25,2	20,2	25,2
kanárské ostrovy	29,8	22,1	18,2	29,8
rýma	18,8	25,6	14,3	25,6
počítačová simulace	27,4	19,3	23,2	27,4
jazyk	15,2	28,4	20,1	28,4
průměrná známka	23,97	22,75	18,85	27,47

Tabulka 10.1: Průměrná známka algoritmů TSPRL a CZDIS pro jednotlivé testovací dotazy.

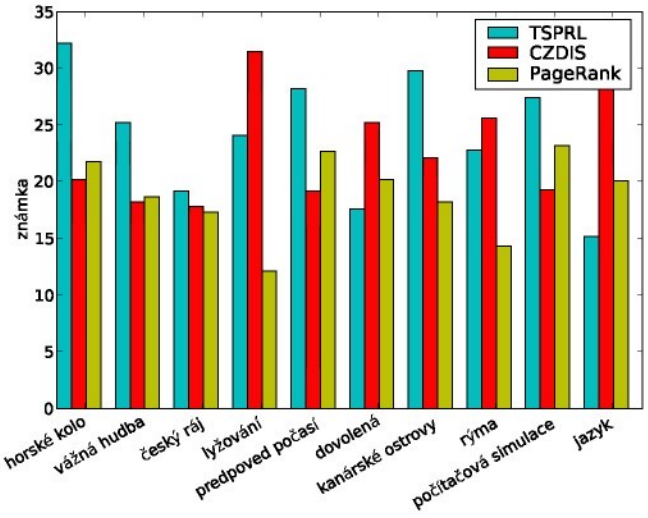
kapitoly 8, ale jsou popsány pouze v této části.

Aplikací jazykového modelu vznikl z originálního slovník S o velikosti 5,5 mil. slov redukovaný slovník S_100, který měl pro kolekci CZdat2 velikost 225057 slov. Sekvenční výpočet jednotlivých vektorů na počítači Lexis trval 198 sekund. Vektory byly zpracovány do inverzního indexu, který pak sloužil programu IndexServer pro uspořádání výsledků.

10.2 Výsledky uživatelského testování

Z výsledků uvedených v tabulce 10.1 je vidět, že i na množině CZdat2 se algoritmus CZDIS choval podobně jako v předchozím testu. V případě jednoslovných dotazů měl lepší hodnocení než oba zbývající algoritmy. Protože vzájemné výsledky algoritmů TSPRL a PageRank už jsme rozebrali v předcházející kapitole, lze konstatovat, že teoretický předpoklad byl správný. Algoritmus TSPRL dosáhl lepšího výsledku u 6 dotazů – všechny byly složeny ze dvou slov. CZDIS pak dosáhl lepšího hodnocení u 4 jednoslovných dotazů a to poměrně výrazně. Průměrná známka obou personalizovaných algoritmů je lepší než průměrná známka normálního PageRanku. Sloupec nazvaný *kombinace* zobrazuje teoretické výsledky kombinovaného algoritmu spoujicícho TSPRL a CZDIS (viz. dále).

Pro větší názornost byly hodnoty z tabulky opět vyneseny také do sloupcového grafu – obrázek 10.1.



Obrázek 10.1: Srovnání uživatelského hodnocení algoritmů CZDIS a TSPRL

10.3 Zhodnocení

Při testování na stejných datech byly oba algoritmy hodnoceny v průměru lépe než základní PageRank. Algoritmus TSPRL, se i přes nedostatky zmíněné v závěru kapitoly 9 zdá být vhodnou funkcí pro uspořádání víceslovných dotazů. Algoritmus CZDIS naopak dosahoval nejlepšího hodnocení u dotazů jednoslovných. Kombinací obou algoritmů by tak mohla vzniknout hodnotící funkce, která bude navázána na obsah stránky i uživatelské preference (personalizovaná) a zároveň bude univerzálně použitelná jak pro dotazy jednoduché, tak dotazy složené.

Základní možností kombinace obou algoritmů je výpočet hodnotících vektorů nezávisle a volba řadící funkce v okamžiku vyhodnocování dotazu, podle počtu jeho prvků. V porovnání s nepersonalizovaným PageRankem půjde sice o řešení náročnější na výpočetní výkon i prostorové nároky vytvořených indexů, ale výsledky uživatelských testů prokazují, že oba lokalizované algoritmy řadí výsledky k větší uživatelské spokojenosti. Teoretické hodnocení této kombinace obou algoritmů, je uvedeno v tabulce 10.1. Kombinovaný algoritmus by dosáhl průměrné známky 27,5.

Složitější kombinace – spojení obou algoritmů v jeden kompaktní celek může být dalším z cílů budoucího výzkumu.

Kapitola 11

Závěr

V této disertační práci byly popsány výsledky experimentálního ověření možnosti lokalizace algoritmů pro uspořádání výsledků fulltextového vyhledávače, založených na kontextu dotazu. Podařilo se prokázat, že anglický jazykový model, použitý pro návrh originálních algoritmů, je možné úspěšně nahradit modelem českého jazyka.

Konkrétně byla provedena lokalizace algoritmů Topic Sensitive PageRank (kapitola 9.6) a Intelligent Surfer (kapitola 8.6). Závěry obou kapitol podrobně rozebírají výsledky lokalizací. Obě lokalizované verze byly v rámci uživatelského testování hodnoceny skupinou pěti dobrovolníků. Návrh lokalizovaného algoritmu CZDIS byl přijat ke zveřejnění (duben 2009) na mezinárodní konferenci BIS 2009 [2].

Výsledky testů lokalizovaných verzí prokazují, že kvalita algoritmů nebyla lokalizací snížena. Hodnotící funkce, vzniklá kombinací obou lokalizovaných algoritmů by pak mohla být velmi kvalitním prostředkem pro uspořádání stránek a to jak pro jednoslovné, tak pro víceslovné dotazy (kapitola 10.3).

Dalším přínosem práce je urychlení výpočtu hodnotících vektorů lokalizovaných algoritmů. Toho bylo dosaženo nahrazením sekvenčního výpočtu vektoru paralelním řešením řídkého lineárního systému (kapitola 7.7). Tento krok spolu s aplikací jazykového modelu umožňuje nasadit lokalizované algoritmy i v reálném provozu. Výsledky experimentu s paralelním výpočtem více hodnotících vektorů, byly publikovány ve sborníku mezinárodní konference INTED 2009 [1].

Pro účely testování lokalizovaných algoritmů byl vyvinut a implementován experimentální systém, který lze, po dořešení některých nedostatků odhalených během jeho používání, využívat i pro další výzkumy v oblasti ZIW.

Shrnutí přínosů k rozvoji vědního oboru

V práci jsou navrženy dva lokalizované algoritmy pro řazení výsledků fulltextového vyhledávání na webu. Oba algoritmy jsou založené na kombinaci odkazové analýzy a kontextu dotazu. U algoritmů je popsána metoda urychlení výpočtu a proces lokalizace pomocí aplikace českého jazykového modelu.

Shrnutí přínosů pro praxi

Výsledky experimentů s lokalizovanými algoritmy potvrzují, že oba algoritmy jsou pro řazení výsledků kvalitnější funkcí, než nepersonalizovaný PageRank.

Také urychlení obou algoritmů aplikací numerických metod řešení a jazykového modelu, dává dobré předpoklady pro jejich reálné nasazení v praxi.

Další práce a experimenty

Dalším pokračováním v tématu práce by mělo být dopracování algoritmu kombinujícího TSPRL a CZDIS v jedno řešení, tak jak bylo naznačeno v závěru kapitoly 10. Kvalita tohoto kombinovaného algoritmu by pak měla být opět ověřena uživatelským testem. V tomto testu by zároveň mělo dojít k rozšíření metodiky volby dotazů. Náhodná volba testujícími uživateli, by měla být doplněna o dotazy získané analýzou dat z Google trends.

Budoucí výzkum v oblasti ZIW by měl být zaměřen na problematiku automatické detekce tématu stránek z jejich obsahu, shlukování dokumentů dle témat a využití těchto prostředků pro uspořádání výsledků fulltextového vyhledávání.

Další práce by také měla vést k dopracování experimentálního systému v transparentně použitelné prostředí, vypracování příslušné dokumentace a využití tohoto hotového prostředí pro výzkumnou činnost studentů FM v rámci závěrečných prací a projektů.

Literatura

Vlastní publikace

[1] Vraný J.: "CZDIS - distributed algorithm for personalized web pages ranking", INTED2009 Proceedings, Valencie Španělsko 2009, ISBN: 978-84-612-7578-6

[2] Vraný J.: "Parallel algorithm for query content based webpages ranking", přijato na konferenci BIS2009, Poznaň Polsko 2009

[3] Vraný J.: "Řazení výsledků WWW vyhledávače", Doktorandský den ústavu NTI, Liberec 2008

[4] Vraný J.: "Personalizace algoritmu PageRank", Doktorandský den ústavu NTI, Liberec 2007

[5] Vraný J.: "Experimentální prostředí pro WebIR", Seminární přednáška, Seminář ústavu NTI, Liberec 2008

[6] Vraný J.: "Úvod do teorie získávání informací na webu", Seminární přednáška, Seminář ústavu NTI, Liberec 2007

Použitá literatura

[7] Anosov D.V.: "Ergodic theory" Springer Encyclopaedia of Mathematics (2001)

[8] Arasu A., Novak J., Tomkins A., Tomlin J.: "Pagerank computation and the structure of the Web: Experiments and algorithms". In Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawai, U.S., 2002.

[9] Baeza-Yates R., Ribeiro-Neto B.: "Modern Information Retrieval", Addison-Wesley (1999).

[10] Bharat K., Henzinger M.: "Improved algorithms for topic distillation in a hyper-linked environment". In Research and Development in Information Retrieval, pages 104111, 1998.

[11] Bharat K., Broder A.Z.: "A technique for measuring the relative size and overlap of public Web search engines". In Proceedings of 7th International World Wide Web Conference, pages 379 – 388, Brisbane, Australia, 1998.

[12] Bharat K., Mihaila G.A.: "Hilltop: A Search Engine based on Expert Documents", http://www.cs.toronto.edu/ georgem/hilltop, 2001.

[13] Berberich K., Vazirgiannis M., Weikum G.: "T-rank: Time-aware authority ranking". In Proceedings of Third International Workshop on Algorithms and Models for the Web-Graph, WAW, pages 131142, Rome, Italy, 2004.

[14] Berman A., Plemmons R.J.: "Nonnegative Matrices in the Mathematical Sciences". Academic Press, Inc., 1979.

[15] Boldi P., Santini M., Vigna S.: "Pagerank as a function of the damping factor". In Proceedings of the 14th World Wide Web International Conference, pages 557566, New York, NY, USA, 2005.

[16] Boldi P.: "Totalrank: ranking without damping". In Special interest tracks and posters of the 14th World Wide Web International Conference, pages 898899, Chiba, Japan, 2005.

[17] Boldi P., Vigna S.: "WebGraph framework i: Compression techniques". In Proceedings of the 13th International World Wide Web Conference, pages 595602, New York, U.S., 2004.

[18] Bradley J., de Jager D., Knottenbelt W., Trifunovic A.: "Hypergraph partitioning for faster parallel Pagerank computation". In Proceedings of EPEW05, Proceedings of the 2nd European Performance Evaluation Workshop, Versailles, France, 2005.

[19] Brin S., Page L.: The anatomy of a large-scale hypertextual web search engine. In Proceedings of the Seventh International World Wide Web Conference, 1998.

[20] Broder A.Z.: "A taxonomy of Web search". In SIGIR Forum 36(2), pages 310, 2002.

[21] Broder A.Z., Kumar R., Maghoul F., Raghavan P., Rajagopalan S., Stata R., Tomkins A.,Wiener J.: "Graph structure in the Web". In Proceedings of the 9th International World Wide Web Conference, Amsterdam, The Netherlands, 2000.

[22] Broder A.Z., Lempel R., Maghoul F., Pedersen J. .: "Efficient Pagerank approximation via graph aggregation". In Proceedings of the 13th International World Wide Web Conference, pages 484485, New York, U.S., 2004.

[23] Del Corso G. M., Gulli A., Romani F.: "Fast Pagerank computation via a sparse linear system". Journal of Internet Mathematics, 2(3), 2005.

[24] Cohn D., Chang H.: "Learning to probabilistically identify authoritative documents". In Proceedings of the 17th International Conference on Machine Learning, pages 167174. Stanford University, 2000.

[25] Český národní korpus - SYN2005. Ústav Českého národního korpusu FF UK, Praha 2005. Dostupný z www: ¡http://ucnk.ff.cuni.cz¿.

[26] Dolamic L., Savoy J.: "Stemming Approaches for East European Languages". In Advances in Multilingual and Multimodal information Retrieval: 8th Workshop of the Cross-Language Evaluation Forum, CLEF 2007

[27] Fox E.: "Extending the Boolean and Vector Space Models of Information Retrieval with P-Norm Queries and Multiple Concept Types". Cornell University dissertation, Aug. 1983. Available from: University Microfilms International, Ann Arbor, Michigan.

[28] Gleich D., Zhukov L.: "Scalable computing for power law graphs: Experience with parallel Pagerank". Technical report, Yahoo! Research Labs, 2005.

[29] Gulli A., Signorini A.: "The indexable web is more than 11.5 billion pages". In Proceedings of 14th International World Wide Web Conference, pages 902903, Chiba, Japan, 2005.

[30] Gyöngyi Z., Garcia-Molina H., Pedersen J.: "Combating Web Spam with TrustRank" 2004

[31] Haveliwala T.H., Kamvar S.D.: "The second eigenvalue of the Google matrix". Technical Report 2003-20, Stanford University, 2003.

[32] Haveliwala T.H., Kamvar S.D., Jeh G.: "An Analytical Comparison of Approaches to Personalizing PageRank," Stanford University Technical Report, July 2003.

[33] Haveliwala T.H.: "Topic-sensitive PageRank". In Proceedings of the Eleventh International World Wide Web Conference, Honolulu, Hawaii, May 2002.

[34] T. Haveliwala.: "Efficient computation of PageRank". Technical report, Stanford University, 1999.

[35] Chakrabarti S.: "Mining the Web". Morgan Kaufmann Publishers 2003. ISBN 1-55860-754-4

[36] "I/o-efficient techniques for computing Pagerank". In Proceedings of ACM Conference on Information and Knowledge Engineering, McLean, Virginia, U.S., 2002.

[37] Chen Y., Gan Q., Suel T.: "Local methods for estimating Pagerank values". In Proceedings of International Conference on Information and Knowledge Management, pages 381389, Washington, D.C., U.S., 2004

[38] Chirita P.A., Olmedilla D., Nejdl W.: "PROS: A personalized ranking platform for Web search". In Proceedings of 3rd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, pages 3443, Eindhoven, Netherlands, 2004.

[39] Cho J., Garcia-Molina H. . "Parallel crawlers". In Proceedings of the eleventh international conference on World Wide Web (2002), pages 124135, Honolulu, Hawaii, USA. ACM Press.

[40] Jeh G., Widom J.: "Scaling Personalized Web Search", Technical Report, Computer Science Department, Stanford University, 2002

[41] Kamvar S.D, Haveliwala T.H., Golub G.H.: "Adaptive methods for the computation of Pagerank". In Proceedings of the International Conference on the Numerical Solution of Markov Chains, University of Illinois at Urbana-Champaign, 2003

[42] Kamvar S.D, Haveliwala T.H., Manning C.D., Golub G.H.: "Exploiting the block structure of the web for computing PageRank". Stanford University Technical Report, 2003.

[43] Kamvar S.D, Haveliwala T.H., Manning C.D., Golub G.H.: "Extrapolation methods for accelerating Pagerank computations". In Proceedings of the 12th International World Wide Web Conference, pages 261270, Budapest, Hungary, 2003.

[44] Kritikopoulos A., Sideri M.: "The compass filter: Search engine result personalization using Web communities". In In Proceedings of Intelligent Techniques for Web Personalization., Acapulco, Mexico, 2003.

[45] Krug. S.: "Web design Nenuťte uživatele přemýšlet". CPress 2003. ISBN: 80-7226-892-9

[46] Kumar R., Raghavan P., Rajagopalan S., Sivakumar D., Tomkins A., Upfal E.: "The Web as a graph". In In Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2000.

[47] Knuth D.: "The art of Computer Programming, Volume 3: Sorting and Searching." Second Edition (Reading, Massachusetts: Addison-Wesley, 1998),ISBN 0-201-89685-0

[48] Langville A., Meyer C.: "Deep inside Pagerank" Internet Mathmathematics, Vol. 1, No. 3, 2005, pp.335-400.

[49] Langville A., Meyer C.: "A survey of eigenvector methods for web information retrieval.", SIAM Review, Vol 47, No. 1, March 2005, pp. 135-161

[50] Langville A., Meyer C.: "The Use of Linear Algebra by Web Search Engines", Bulletin of the International Linear Algebra Society, No. 33, Dec. 2005, pp. 2-6.

[51] Langville A., Meyer C.: "Updating the Stationary Vector of an Irreducible Markov Chain with an Eye on Google's PageRank". SIAM Journal on Matrix Analysis, 27(4): 968-987

[52] Langville A.N., Meyer C.D.: "A Reordering for the PageRank Problem". SIAM J. Sci. Comput. 27, 6 (Dec. 2005)

[53] Lempel R., Moran S.: "The stochastic approach forlink-structure analysis (SALSA) and TKC effect"

[54] Ng A.Y., Zheng A.X., Jordan M.I.: "Stable algorithms for link analysis". In Proceedings 24th Annual Intl. ACM SIGIR Conference, pages 258266, New Orleans, Louisiana, U.S., 2001.

[55] Latent Semantic Analysis Website - http://lsa.colorado.edu/

[6] Očko, P.: "Rok 2000 a křižovatky informační ekonomiky". Ikaros [online]. 2005, roč. 9, č. 10 [cit. 2007-03-15]. http://www.ikaros.cz/node/2018. URN-NBN:cz-ik2018. ISSN 1212-5075.

[56] Page L., Brin S.,Motwani R., Winograd T.: "The Pagerank citation ranking: bringing order to the Web", 1999 Stanford University technical report.

[57] Qiu F., Cho J.: "Automatic Identification of User Interest for Personalized Search". In Proceedings of the World-Wide Web Conference (WWW), May 2006.

[58] Randall K., Stata R., Wickremesinghe R., Wiener J.: "The link database: Fast access to graphs of the Web". Technical report, Compaq Systems Research Center, 2001.

[59] Rafiei D., Mendelzon A.: "What is this page known for? computing Web page reputations". In Proceedings of the 9th International World Wide Web Conference, pages 823835, Amsterdam, The Netherlands., 2000.

[60] Richardson M., Domingos P.: "The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank". 2002 Cambridge

[61] Rungsawang A., Manaskasemsak B.: "Parallel Pagerank computation on a gigabit pc cluster". In Proceedings of Advanced Information Networking and Applications, pages 273277, Fukuoka, Japan, 2004.

[62] Saad Y.: "Iterative Methods for Sparse Linear Systems", PWS Publishing Company, 1996.

[63] Salton G., Wong A., Yang C.S.: "A vector space model for automatic intexing", Cornel University 1975

[64] Savoy, J.: Czech stop words 2008: http://www.unine.ch/info/clef/

[65] Sun J., Zeng H., Liu H.,Lu Y., Chen Z.: "Cubesvd: A novel approach to personalized Web search". In Proceedings of 14th International World Wide Web Conference, pages 382390, Chiba, Japan, 2005.

[66] Reed, W. J.: "The Pareto, Zipf and other power laws". 2001 Economics Letters 74 (1): 1519. doi:10.1016/S0165-1765(01)00524-9

[67] Teevan J., Alvarado C., Ackerman M.S., Karger D.R.: "The perfect search engine is not enough: A study of orienteering behavior in directed search". In In Proceedings of Computer-Human Interaction Conference, pages 415422, Vienna, Austria, 2004.

[68] Teevan J., Dumais S.T., Horvitz E.: "Beyond the commons: Investigating the value of personalizing Web search". In Workshop on New Technologies for Personalized Information Access, pages 8492, Edinburgh, Scotland, UK, 2005.

[69] Tomlin J.A.: "A new paradigm for ranking pages on the world wide Web". In Proceedings of the 12th International World Wide Wed Conference, pages 350355, Budapest, Hungary, 2003.

[70] Westbrook A., Greene R.: "Using Semantic Analysis to Classify Search Engine Spam", 2002 Stanford University technical report

[71] Wicks J. R., Greenwald A.: "More efficient parallel computation of pagerank". 2007 In Proceedings of the 30th Annual international ACM SIGIR Conference on Research and Development in information Retrieval

[72] Zhu Y., Ye S., and Li X.: "Distributed PageRank computation based on iterative aggregation-disaggregation methods". In Proceedings of the 14th ACM international Conference on information and Knowledge Management 2005.

[73] Python programing language, http://www.python.org/

[74] Scientific Python, http://www.scipy.org/

[75] PETSc for Python http://code.google.com/p/petsc4py/

[76] Portable, Extensible Toolkit for Scientific Computation http://www.mcs.anl.gov/petsc/petsc-as/

[77] Pavuk http://pavuk.sourceforge.net/

[78] Open Directory Project, http://www.dmoz.org/

[79] Search Engine Sizes 2005 http://searchenginewatch.com/2156481

[80] Search engine statistics 2002 http://searchengineshowdown.com/statistics/size.shtml

[81] http://www.worldwidewebsize.com/

[82] Google Personalized Search Leaves Google Labs http://searchenginewatch.com/3563036

[83] The Power Of Search Engines, http://www.internetnews.com/stats/article.php/1363881

[84] http://toolbar.google.com/

[85] http://trends.google.com/

[86] http://www.google.com/insights/search/

[87] http://mysearch.yahoo.com/

[88] http://www.ask.com/myjeeves/

[89] The Stanford WebBase Project http://diglib.stanford.edu:8091/ testbed/doc2/WebBase/

[90] Califorina.dat http://www.cs.cornell.edu/Courses/cs685/2002fa/data/gr0.California