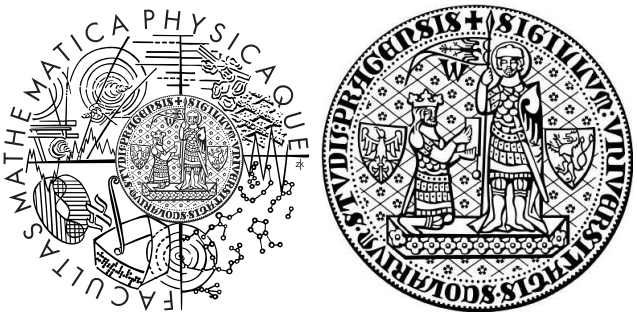
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Vojtěch Kulvait

Crawlování na Webu

Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Leo Galamboš, Ph.D.
Studijní program: Softwarové systémy

2007

Děkuju vedoucímu své diplomové práce, RNDr. Leo Galambošovi, Ph.D. za poskytnutí dat, pomoc a vedení. Děkuju také mamce, za podporu a pomoc v průběhu celého studia a za opravené gramatické chyby.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 3.8.2007 Vojtěch Kulvait

Obsah

Název práce: Crawlování na Webu
Autor: Vojtěch Kulvait
Katedra softwarového inženýrství
Vedoucí diplomové práce: RNDr. Leo Galamboš, Ph.D.
e-mail vedoucího: leo.galambos@mff.cuni.cz

Abstrakt: Cílem práce je porovnat stávající plánovací strategie ve webových robotech, navrhnout jejich případné vylepšení a implementovat odpovídající monitor (plánovací strategii) ve webovém robotu. Seznam odborné literatury
Klíčová slova: crawlování, Web, WWW, vyhledávač, relevance dat

Title: Crawling on the Web
Author: Vojtěch Kulvait
Department of Software Engineering
Supervisor: RNDr. Leo Galamboš, Ph.D.
Supervisor's e-mail address: leo.galambos@mff.cuni.cz

Abstract: Work focuses on studiing current implementations of crawling on the Web and try to create new and effective algorithms for crawling.
Keywords: crawling, Web, WWW, searcher, data relevance

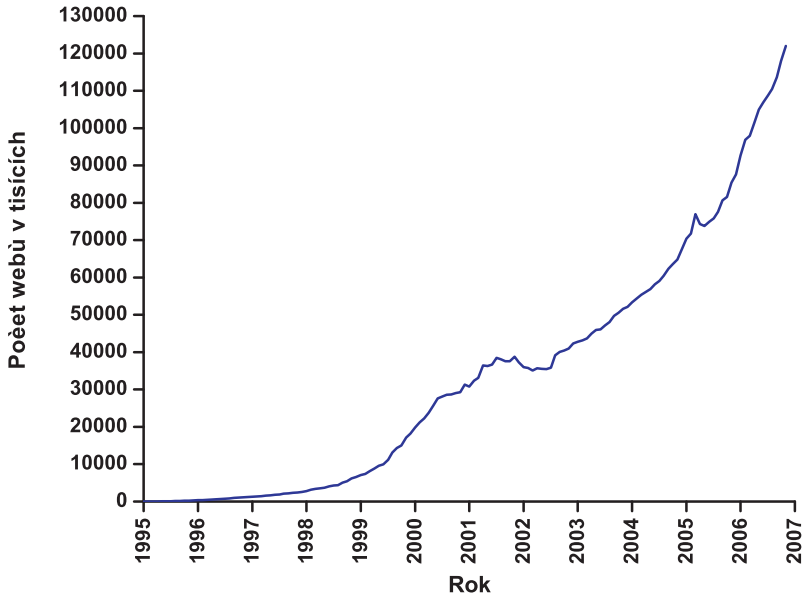
Kapitola 1

Úvod

1.1 Historie vyhledávání

Internet vyrostl, protože lidé chtěli sdílet informace jednodušeji a levněji. S nárůstem objemu informací na internetu rostla ale také poptávka po nástrojích, které by tyto informace sbíraly, katalogizovaly a umožňovaly jejich co nejsnazší nalezení. První katalogy byly spravovány lidmi. Ukazovalo se však, že centralizovaný seznam spravovaný člověkem zaostává svým tempem aktualizace za dynamikou obsahu na internetu. A protože byl index takových seznamů velmi malý v porovnání s obsahem celého internetu, mohlo se snadno stát, že na některé běžné dotazy nevracely tyto systémy žádné výsledky nebo výsledky, které vrátily, neodpovídaly dotazu. Tento nedostatek se snažily vyřešit fulltextové vyhledávače. Ty se začaly prosazovat v polovině devadesátých let dvacátého století. Aktivně procházely internet tak, že následovaly odkazy ze známých stránek a indexovaly všechen textový obsah nalezených webů. Nejznámějšími jejich zástupci byly AltaVista, Lycos a Excite.

Protože se Web rychle rozrůstal, viz Graf 1.1, roboty začaly stahovat stále více informací. Znamenalo to, že na jeden dotaz dostal uživatel typicky tisíce výsledků. Ty, buď nebyly setříděny vůbec, nebo podle nevyhovujících kritérií. Tím bylo například prosté sečtení počtu výskytů jednotlivých termů vyhledávané fráze v dokumentu. Často byl používán tzv. vektorový model, který měří blízkost obsahu stránky a dotazu ve vektorovém prostoru všech slov. Protože však chyběla dostatečně globální metrika pro řazení výsledků, byly na prvních místech často zobrazovány nedůležité výsledky a odkazy na balastní stránky.



Graf 1.1: Nárůst počtu webů na internetu, Zdroj: [34]

Moderní historie vyhledávání na internetu je do značné míry spojena s vyhledávací službou Google. Ta původně vznikla jako projekt dvou doktorandských studentů na univerzitě ve Standfordu. V roce 1998 byla publikována práce [9] popisující architekturu velkých vyhledávačů. Její autoři zdůrazňují nevhodnost řazení stránek ve výsledcích vyhledávání na základě rozšířeného vektorového modelu. Tento model má tendenci preferovat krátké stránky velmi podobné vyhledávané frázi a navíc často totálně selhává na víceslovných dotazech.

Bylo tedy zřejmé, že kvalitní vyhledávač musí mít dostatečně velký index, aby mohl vracet pouze dokumenty obsahující všechna slova dotazu. Dále bylo nutné najít globální míru důležitosti stránek na celém Webu. Autoři v [9] představují koncept PageRanku založený na pravděpodobnostním modelu chování uživatele internetu. Tento model, oproti předchozím řešením, využívá informaci o struktuře internetu, reprezentovaného orientovaným grafem, s vrcholy představujícími jednotlivé dokumenty Webu, propojené hranami v podobě hypertextových odkazů. Vyhledávací služba se tedy už nezaměřovala pouze na indexaci jednotlivých dokumentů, ale zkoumala také relace mezi nimi. Koncept PageRanku znamenal přelom ve vyhledávání

na internetu. Práce [9] dále nastiňuje architekturu vyhledávače s indexem, který může dosáhnout velikosti až 100 000 000 stránek. Původní akademická verze vyhledávače Google přitom měla ve svém indexu asi 30 000 000 stránek. Autoři se zmiňují o implementaci paralelního crawleru využívajícího najednou několik strojů a jsou zde také rozebrány některé problémy související s crawlováním. Původní crawler byl schopen při zapojení 4 strojů stáhnout 100 stránek za sekundu. Hlavním problémem byl DNS překlad internetových adres na IP adresy. Proto si každý stroj udržoval svou vlastní DNS databázi. Dalším problémem byla nehomogennost Webu, která klade velké nároky na předcházení a řešení chybových stavů webového robotu. Dále práce uvádí, že jedno z pomyslných úzkých hrdel systému představovala i rychlost I/O operací na síti. Autoři článek shrnují slovy: „So we are optimistic ... that there is a bright future for search“.

V červenci 2007 udává vyhledávač Google počet výsledků na dotaz "and" 5 660 000 000, přitom ještě v roce 2006 to bylo 12 890 000 000. Vyhledávač search.yahoo.com uvádí počet výsledků na dotaz "and" 8 890 000 000. V roce 2006 byly zprovozněny vyhledávací služby ask.com a vyhledávač společnosti Microsoft live.com. Tyto vyhledávače tvrdí, že na dotaz "and" našly 2 757 980 000 stránek (ask.com) a 2 448 335 855 (live.com). Největší vyhledávače tak pravděpodobně spravují kolekce, které čítají přes 20 000 000 000 stránek. Stahování takových databází stránek je výzva a jejich administrace určitě obnáší mnoho úsilí.

Jisté je, že o efektivní algoritmy pro vyhledávání na internetu je v současnosti zvýšený zájem. Souvisí to nejspíš s tím, že uživateli internetu se v posledních letech stali obyčejní lidé, kteří ho používají k nakupování nebo plánování dovolené. Proto je čím dál častěji skloňována perzonifikace, lokalizace, konvergence služeb, vyhledávání multimédií nebo tematické crawlování a vyhledávání. Existuje velká poptávka po algoritmech vylepšujících všechny části procesu vyhledávání a to, jak do relevance výsledků, tak do rychlosti jednotlivých částí vyhledávačů.

1.2 Současnost vyhledávání

Uživatelé obvykle navštíví jen ty stránky, které se objeví na prvních několika pozicích ve výsledcích vyhledávače.

Internetu dneška dominuje fulltextové vyhledávání se tříděním výsledků podle jejich váhy. Vyhledávače procházejí odkazy na internetu a postupně rekurzivně stahují jednotlivé dokumenty a následují další odkazy, které z

těchto dokumentů vedou. Právě popsaný proces procházení a indexace obsahu Webu se nazývá crawlování. Nepoužívá se ale pouze ve vyhledávačích. Crawlování používají mnohé instituce pro získávání e-mailových adres, kontaktů nebo zpravodajských informací.

Především pro vyhledávače je důležité, aby výsledky byly co nejvíce relevantní vzhledem k dotazu. Již ve fázi stahování, tedy crawlování, je třeba dbát na stažení takové podmnožiny internetu, která bude pro uživatele co nejzajímavější. Zaindexovat lze totiž jen malou podmnožinu celého Webu, viz [3]. Navíc je známo, že četnost zdrojů ve výsledcích vyhledávání je přímo úměrná počtu stránek, který je z daného zdroje stažen, viz [19].

Metrika důležitosti zavedená na indexovaném obsahu by měla co nejvíce odpovídat uživatelovým představám o relevanci jednotlivých stránek. V současnosti se nejvíc používá PageRank, řazení výsledků však ovlivňuje také vyhledávací fráze. Stránku může ve výsledcích zvýhodnit, pokud obsahuje slova vyhledávací fráze blízko u sebe nebo na významných pozicích jako je titulek nebo nadpis. Jak konkrétně text vyhledávací fráze modifikuje PageRank při řazení výsledků ale většina vyhledávačů tají.

1.3 Crawlování na Webu

Crawlování stojí na úplném začátku řetězce obstarávání dat pro indexaci a vyhledávání. I když můžeme být ohromeni velikostí indexů vyhledávačů, můžeme si také položit otázku, kolik na internetu existuje dokumentů obsahujících například zmíněnou frázi "and". Pokud bychom přijali tezi, že například Yahoo zaindexoval asi takový podíl dokumentů ze všech stránek s danou frází na internetu jako je podíl, který zaindexoval nováček live.com v porovnání s indexem Yahoo, dostaneme se na 32 000 000 000 dokumentů na internetu obsahujících frázi "and".

Jaké dokumenty vyhledávač zaindexuje, tomu bude odpovídat kvalita vyhledávání. Na souborech milionů stránek se ale ukazuje [8], že některé crawlovací techniky nezachovávají ve svém průběhu ani přibližně uspořádání vektoru PageRanku. Jsou to především techniky, které používají PageRank už v průběhu crawlování pro řazení dalších stránek ke stažení. Důležitost spočítaná na nacrawlovaném indexu pak může být odlišná od této míry na celém Webu. Pokud tento závěr přeneseme i na větší soubory stránek, stává se volba vhodné crawlovací techniky zásadní nejen pro aktuálnost, ale i pro pořadí zobrazovaných výsledků.

1.4 Účel práce

To, že konkrétní strategie crawlování ovlivňuje metriku důležitosti, je tedy obecně známý fakt. Jedním z cílů práce je spočítat, jak přesně se mění uspořádání metriky důležitosti v průběhu crawlování.

Práce zavádí novou metriku důležitosti na Webu, která by měla mít příznivější konvergenční vlastnosti než PageRank a v průběhu crawlování by měla být stabilnější. Cílem je změřit její vlastnosti a určit nakolik je vhodné její další nasazení a používání.

Dalším cílem práce je implementovat takové postupy pro počítání důležitosti, aby při měření vlastností crawlovacích strategií byly brány v potaz i listové vrcholy. Ostatní práce o crawlování totiž listové vrcholy často nezohledňují.

1.5 Struktura práce

V prvních částech se práce věnuje výsledkům dosavadních prací o crawlování a nastiňuje šíři celého oboru zabývajícího se sběrem dat z internetu. Důraz je přitom kladen na crawlovací strategie, dynamiku Webu a trendy ve vyhledávání na internetu.

V práci je podrobně popsáno několik existujících strategií crawlování a metoda ohodnocení indexovaných stránek podle PageRanku jak je nastíněna v [18]. Dál práce popisuje Kendallovo τ jako metriku pro porovnávání uspořádání dvou ohodnocení kolekce stránek podle důležitosti.

Crawlovací strategie a míry důležitosti jsou zkoumány na reálných datech o struktuře internetu, které mi poskytl vedoucí diplomové práce, RNDr. Leo Galamboš, Ph.D.

Práce se zaměřuje především na strategii BFS a rychlost s jakou tato strategie stahuje relevantní stránky hodnocené podle různých měr důležitosti.

V práci je představena nová metrika důležitosti, K-Rank. Práce zkoumá, jak rychle PageRank a K-Rank konvergují na menší podmnožině Webu k hodnotě dané metriky, vypočítané na celém referenčním souboru stránek.

Obě metriky jsou přitom upraveny tak, aby zohledňovaly i uzly, ze kterých nevedou žádné odkazy. Navržené řešení tak pracuje s veškerými informacemi, které jsou v dané fázi o grafu Webu k dispozici. Případné rekurzivní osekání listových vrcholů by totiž mohlo způsobit zkreslení výsledků, například při měření stability vektoru důležitosti v průběhu crawlování.

Kapitola 2

World Wide Web

2.1 Struktura Webu

Pro sestavení optimálních crawlovacích a vyhledávacích strategií je nezbytné seznámit se se samotnou strukturou internetu. Na základě informací o struktuře internetu je pak možné optimalizovat způsoby jak tuto strukturu procházet a indexovat.

2.1.1 Dynamika Webu

Na internetu stále vznikají a zanikají stránky. Pro navržení optimálních algoritmů je ale nutné tyto procesy kvantifikovat. K tomu je třeba znát rychlost obnovování stránek, životnost stránek, počet chybných odkazů a mechanizmy jak se nové stránky začleňují do struktury Webu.

Podle [3] je asi 15% odkazů na internetu nefunkčních.

Podle provedené analýzy [19] dokonce i univerzitní stránky, které jsou více statické, v porovnání s těmi z domény .com, mají značnou dynamiku. Při studiu asi 30 000 stránek z domény .edu se ukázalo, že 4.5% stránek po 45 dnech výzkumu zaniklo, 19% z nich byly alespoň jeden den z daného období nedostupné a přes 25% stránek se v průběhu výzkumu změnilo. U většiny z nich se jednalo o větší než malé změny.

Obnovování stránek

Vyhledávače se snaží udržet své indexy co nejčerstvější. Tedy aby obsah jejich indexu co nejlépe odpovídal skutečnému obsahu stránek. To ztěžuje

právě dynamika internetu a skutečnost, že stránky jsou často aktualizovány. Proto je nutné indexy vyhledávačů obnovovat a obsahy stránek stahovat znovu (recrawlování).

Tempo aktualizace není u všech stránek ani řádově stejné. Práce [19] například zmiňuje, že stránky v doméně .edu se mění, buď rychle v řádech dnů, nebo téměř vůbec. Proto je vhodné řadit pro potřeby crawlerů stránky podle časové periody s jakou je danou stránku třeba recrawlovat.

Dobu, za kterou bude obsah dané stránky aktualizován, nelze podle [41] a [38] aproximovat Poissonovým rozdělením. Článek [41] se zmiňuje o možné subexponenciální závislosti. Tedy, pokud $F(t)$ je funkce určující pravděpodobnost, že stránka je aktualizována po čase delším než t, pak platí:

$$F(t) = e^{-\lambda t^a} \tag{2.1}$$

Přitom $\lambda > 0$ a $a \in (0, 1)$.

Pro $a = 1$ by se jednalo o Poissonovo rozdělení.

Podobných modelů jako tento se využívá především při návrhu strategií recrawlování.

2.1.2 Dynamický obsah

Vyhledávače se na stránky dívají jako na statický obsah určený adresou stránky. Tuto představu však narušuje mnoho technologií jako php, asp nebo servlety. Řešení založená na použití takových technologií umožňují to, že aktualizaci stránek na internetu provádějí z velké části stroje a ne lidé. Proto se často mění obsah pouze fyzicky a ne informačně. To platí například pokud se na dynamické stránce změní každý den datum a případně styl podle preferencí uživatele. Uvedené informace však zůstávají beze změny.

Na internetu existuje mnoho blogů a diskuzních fór. Mnoho stránek si ukládá cookie pro své uživatele a identifikuje je podle obsahu cookie. Uživatel je pak například oslovován jménem nebo tematickou reklamou.

Na diskuzních fórech nebo v blozích je přirozené informace členit ne na stránky, ale na jednotlivé příspěvky. Podobně je tomu i u internetových obchodů. Web zde není primárním zdrojem informací, ale poskytuje pouze jakousi mezivrstvu mezi databází, kde je informace uložena a uživatelem, kterému je zobrazována.

Často mají internetové služby jako diskuzní fóra nebo blogy své vyhledávací nástroje. Bohužel většinou na dost primitivní úrovni, neumožňující

integraci s reálnými vyhledávacími službami. Taková integrace tedy v současnosti zcela chybí. Finanční náklady a technické překážky jsou hlavním důvodem proč takové řešení nelze v současnosti uskutečnit.

Podle [29] je na velkých internetových serverech, kde jsou umístěny weby velkých firem nebo akcí, přibližně 25% dynamicky generovaných stránek, 32% http dotazů je na dynamický obsah a průměrná velikost stránky je 10Kb. Studie je z roku 1999 a autoři za dynamicky generované považují jen ty stránky, které obsahují v URL symbol '?'. Je tedy pravděpodobné, že stránek s nějakými dynamickými prvky je více.

Pro vyhledávání na Webu je dynamický obsah velkou výzvou, která zatím přináší více otázek než odpovědí. Jednou z možných cest, o které se mluví v souvislosti s takzvaným "sémantickým Webem", je důsledné značkování logicky souvisejících oblastí obsahu a jeho následný parsing ve vyhledávačích. Faktem však zůstává, že kvůli chybám v zápisu kódu, je často problematické i samotné získání URL všech odkazů, které ze stránky vedou.

2.1.3 Hluboký Web

Další cestou je vyhledávání ve vrstvě databází, ze kterých se generuje dynamický obsah. Tato vrstva se nazývá "hluboký Web". Obvykle není možné k ní přistupovat přímo.

Problémem dynamických služeb totiž často je, že při jejich vývoji není příliš zohledňována možnost indexace obsahu ze strany třetích stran. Ani ze strany vyhledávačů není patrný velký zájem něco takového podporovat.

Jistou možností, jak v těchto případech postupovat, je takzvané metavyhledávání. Je ale téměř nemožné řešit tímto způsobem vyhledávání na stovkách nebo tisících serverů poskytujících obsah hlubokého Webu. Především z důvodů velkých nákladů na zpracování dotazu, kvůli nutnosti přenášet při každém dotazu data mezi vyhledávačem a všemi poskytovateli dat a možné dlouhé prodlevy mezi vznesením dotazu a zobrazením výsledků. Při takovém uspořádání je totiž nutné dotazovat se mnoha nezávislých vyhledávacích služeb, jejichž čas odezvy nemá primární vyhledávací služba pod kontrolou.

Metody indexování a vyhledávání v dynamickém obsahu jsou tedy zatím v plenkách a tak logickou jednotku informace na internetu někdy ne zcela přesně reprezentuje stále jedna konkrétní Webová stránka, identifikovaná svou URL.

2.2 Vyhledávání na internetu

Podle [33] žádný vyhledávač neindexuje víc než 16% obsahu internetu. Vyhledávače mají tendenci ignorovat nové stránky, které mají nízký PageRank (kvůli jejich čerstvosti).

Článek je z roku 1999. Vzhledem k růstu Webu však lze předpokládat, že mnohé jeho závěry jsou platné stále.

Říká, že již tehdy tvořila pornografie pouhé 1.5% internetu. Mnoho lidí se však stále mylně domnívá, že tento obsah internetu dominuje.

Dále autoři zmiňují, že vyhledávací služby častěji indexují komerční, než vzdělávací obsah a že často dlouho trvá, než se ve výsledcích vyhledávání projeví aktualizace obsahu stránek. Vyhledávače totiž ve svých indexech přednostně obnovují obsah z malé skupiny serverů často zobrazovaných ve výsledcích.

Průměrný počet neplatných odkazů ve vyhledávačích byl 5.3%.

Zaindexování velké většiny dokumentů na internetu není v současnosti proveditelné jedním vyhledávačem především kvůli nákladnosti a relativně nízké přidané hodnotě za rozšíření indexu. Východiskem mohou být tematicky zaměřené vyhledávací služby. Touto cestou se nevydávají jen malé firmy. Příkladem jsou služby jako scholar.google.com nebo news.google.com.

2.2.1 Metavyhledávače

Kromě klasického vyhledávání je možné použít odlišný přístup. Jedná se o takzvané metavyhledávání. Samotný vyhledávací proces probíhá tak, že po zadání není dotaz zpracováván nad interní databází nacrawlovaných stránek, ale posílá se dál do několika dalších vyhledávacích systémů, odkud jsou získávány výsledky. Představu o technikách metavyhledávání a jeho nákladech, ve srovnání s klasickým vyhledáváním, podává [19].

Metavyhledávání je alternativou ke klasickému vyhledávání zejména tam, kde nelze uplatnit crawlování, například při prohledávání webů nepřístupných crawlerům, které mají vlastní vyhledávací služby. To mohou být například vládní nebo medicínské zdroje. Dále je metavyhledávání vhodné v situacích, kdy je požadována absolutní čerstvost informací, typicky při prohledávání obsahu zpravodajských serverů.

Problémem je možná dlouhá odezva cílových serverů, kterou metavyhledávač nedokáže ovlivnit a také algoritmus slévání výsledků z jednotlivých vyhledávacích služeb. Na dokumentech totiž není zavedená jednotná met-

rika.

Při rostoucím zatížení vyhledávače roste i objem dat přenášených mezi metavyhledávačem a cílovými vyhledávači. To vede k tomu, že při velkém počtu dotazů, se začíná vyplácet klasické vyhledávání před metavyhledáváním. Metavyhledávání se tak vyplácí v podstatě jen v ojedinělých případech prohledávání několika málo dynamických informačních zdrojů a to pokud počet dotazů nepřesáhne několik set za týden (přibližná doba jednoho crawlovacího cyklu u vyhledávačů) viz [19].

2.2.2 Další přístupy k vyhledávání

Klasické crawlované vyhledávání a metavyhledávání lze kombinovat [19]. To může být užitečné v situacích, kdy vyhledáváme na několika rozsáhlých serverech s vlastními vyhledávacími službami a na dalších méně rozsáhlých bez vyhledávací služby. Autoři doporučují provádět crawlování významně méně často než je běžné, například jednou ročně. Vyhledávání v hybridním modelu probíhá pomocí metadotazování několika málo velkých serverů při současném prohledávání lokálního indexu. Snižuje se tak cena za crawlování a navíc je možné si podle crawlovaného indexu zavést alespoň přibližnou globální míru důležitosti nad výsledky z metavyhledávání různých serverů a optimalizovat podle ní slévání výsledků.

Další přístup, který by mohl řešit problematiku řazení výsledků při metavyhledávání, může být umístění software metavyhledávače přímo na servery, nad kterými metavyhledávač spouští své dotazy. Tento software vyhodnocuje jednotlivé dotazy a řadí je podle důležitosti. Výsledky předává metavyhledávači, viz [19]. Je nepravděpodobné, že by tento přístup mohl být uplatněn globálně. Vyžaduje totiž po každém zúčastněném serveru spuštění vyhledávacího software daného metavyhledávače.

2.3 Technologie na Webu

Porozumět struktuře internetu předpokládá také znalosti o tom, jakým konkrétním způsobem je na něm publikován obsah a jakými cestami se dostává ke koncovému uživateli. K tomu, že internet se stal masově používaným médiem a komunikačním prostředkem, významným způsobem přispívají právě technologie a standardy. Ty se dají rozdělit do dvou základních skupin. V první skupině jsou technologie, které slouží jako prostředek pro sdělování

informace. Sem patří jazyky pro psaní Webu (HTML, XHTML, ...), skriptovací jazyky, vkládané do kódu stránky a interpretované na straně uživatele (JavaScript, JScript, ...) a skriptovací jazyky interpretované serverem (PHP, ASP, ...). Druhou skupinu tvoří technologie pro přenos informací. Patří sem protokoly pro přenos dat (HTTP, ...) a programy, které nad těmito protokoly pracují. Na straně zdroje informací to jsou webové servery (Apache, IIS, ...).

Na pomezí obou skupin technologií na Webu stojí internetové klienty (Firefox, IE, ...). Ty jednak komunikují s webovými servery podle protokolů pro přenos dat a jednak přenesená data interpretují a zobrazují je uživateli. Za speciální skupinu internetových klientů lze považovat i crawlery.

2.3.1 Publikace obsahu

K vytváření internetových stránek se používá především jazyk HTML nebo XHTML. XHTML je postaven na jazyce XML a oproti HTML má přísnější pravidla pro zápis kódu. HTML nebo XHTML kód se často doplňuje použitím stylového jazyka CSS. Pro částečnou dynamiku stránek pomocí skriptů, interpretovaných na straně uživatele, se nejčastěji užívá JavaScript. Mnoho webů používá dynamicky generovaný obsah na straně serveru. Pro tento účel se používají technologie založené na PHP [23], ASP [16] nebo Java servletech [36].

Hotový web je pak uložen na webový server. Je to veřejně přístupný počítač s programem, který zpracovává požadavky na jednotlivé URL a vrací na ně odpovídající stránky. Pokud stránky používají dynamický obsah, interpretovaný na straně serveru, je úkolem serveru ještě navíc interpretovat kód daný dynamickým jazykem a vrátit uživateli stránku, například ve formátu XHTML. Webové servery musí zvládat zátěž od jednotek přístupů denně po statisíce uživatelů v jednom okamžiku.

V současnosti jsou podle [34] nejpoužívanější webové servery Apache [22] (podíl 53%) a Microsoft IIS [17] (podíl 33%). Díky službě blogspot.com pro uživatele, kteří chtějí publikovat vlastní blog, se začal prosazovat i server GFE společnosti Google s podílem kolem 4%.

2.3.2 Standardizace

Množství technologií na internetu vede k potřebě zavádět postupy a doporučení jakým způsobem tyto technologie používat. Nejdůležitější je standar-

dizovat ty, se kterými přichází do styku koncový uživatel. Jsou to protokoly pro přenos dat (HTTP, ...) a jazyky, které jsou interpretovány na straně uživatele (XHTML, JavaScript, CSS, ...).

Většinu webových standardů jako XHTML, HTTP nebo DOM spravuje konzorcium W3C [40].

U protokolů pro přenos dat na Webu, tedy především HTTP, již proběhl proces konvergence a odpovídající standardy jsou relativně stabilní. W3C uvádí, že u protokolu HTTP je specifikace již uzavřená.

Jazyky HTML a XHTML mají mnoho různých verzí a ke každé existují doporučení jakým způsobem je používat. Tyto jazyky se stále vyvíjí a mnoho lidí pracuje na dalších verzích. Existují ještě další technologie pro publikaci, obvykle založené na XML. Stejně tak se objevují stále nové verze stylových jazyků CSS. Množství nově vytvářených jazyků a standardů pro publikování vede k tomu, že ani výrobci prohlížečů nestíhají do svého software implementovat všechny novinky.

V zájmu získání konkurenční výhody se prohlížeče snaží být co nejtolerantnější k chybám v zápisu kódu stránek. Bohužel to způsobuje, že chybně zapsané stránky je obtížnější strojově zpracovávat. Standardy pro tvorbu obsahu na internetu jsou tedy nepochybně důležité, ale pokud neexistuje způsob jak vynutit jejich používání, stávají se nefunkční.

Proces konvergence standardů pro publikaci na internetu v současnosti ještě neskončil. Proto neexistuje jeden správný způsob jak psát Web. Existuje jen mnoho různých doporučení a je docela pravděpodobné, že pokud se autor bude alespoň přibližně držet některého z nich, obsah jeho stránek bude schopna zobrazit většina prohlížečů. Tento postup při tvorbě stránek však do značné míry znesnadňuje práci takových nástrojů jako jsou právě crawlery.

Kapitola 3

Crawlery

3.1 Popis Crawleru

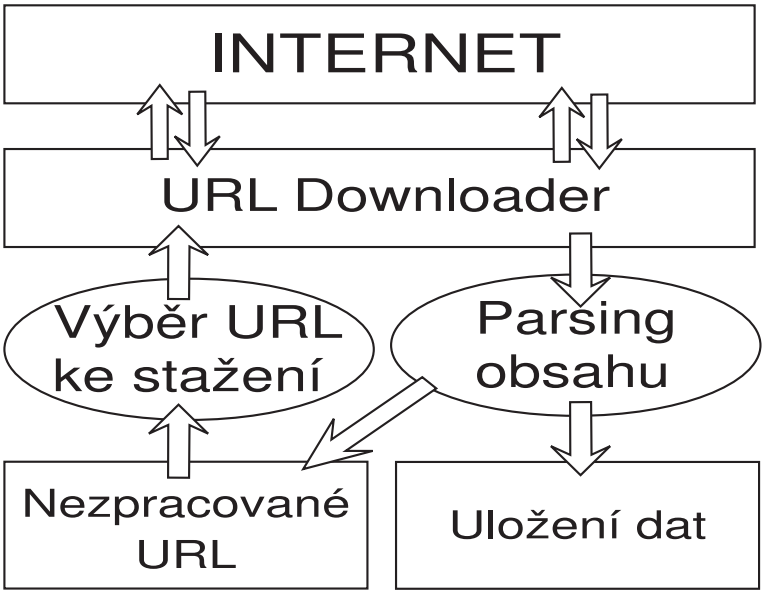
Crawler je program, který rekurzivně stahuje stránky z internetu. Má zadanou množinu známých stránek, odkud začíná stahování. Po stažení každou stránku analyzuje a extrahuje z ní odkazy na další stránky. Ověří, zda daný odkaz už nestáhnul a pokud ne, přidá ho do fronty URL ke stažení.

Crawler se skládá z několika částí, viz Obrázek 3.1.

Datová struktura udržující seznam dosud nestažených URL dokumentů bývá realizována jako fronta nebo n front, každá fronta pro jeden z aktuálně stahovaných webů. Tuto strukturu není vhodné celou uchovávat v hlavní paměti. Internet není možné stáhnout celý a její velikost stále roste jak se s postupujícím časem a hloubkou crawlování rozrůstá počet záznamů o nestažených URL. To, jak konkrétně je vhodné strukturu reprezentovat, je dáno především volbou crawlovací strategie.

Crawlovací strategie je algoritmus, který vybírá z dosud nestažených adres další URL ke stažení. Její volba významně ovlivní jaká data budou stažena. Základní strategií je BFS procházení stránek. S růstem internetu se rozmáhají i tematické crawlery. U těch musí crawlovací strategie vybírat ke stažení přednostně takové URL, které se nejspíše váží k danému tématu, viz [20].

Samotné stahování požadovaných URL adres provádí část crawleru, která se nazývá URL downloader. Tato jednotka se liší podle typu použitého crawleru. Může jít o nativní součást desktopové aplikace, ale může se jednat o sofistikované řešení v podobě stovek počítačů, rozmístěných v různých regionech světa. Ty vykonávají jednotlivé požadavky a stahují stránky. V



Obrázek 3.1: Schéma crawleru

případě distribuovaných řešení a paralelních crawlerů je nutné přizpůsobit tomu celý návrh. Paralelizací crawlerů a distribucí jejich práce mezi mnoho downloaderů se zabývají například práce [13], [6] a [7].

Další jednotka crawleru je zodpovědná za parsing dat. Extrahuje data z těla stažených stránek a získané informace ukládá do úložiště dat. Může pracovat i s dodatečnými informacemi o stránce jako je DNS záznam zdrojového serveru nebo HTTP hlavička. Zároveň extrahuje a zpracovává všechny odkazy vedoucí z dané stránky. Pokud se objeví nová URL, vloží ji do struktury uchovávající URL ke stažení.

Úložiště dat si lze představit jako strukturu, kam crawler uloží informace o stažených stránkách (datum stažení, ip adresa serveru, obsah, odkazy, klíčová slova, ...). Takto uložená data pak využijí vyhledávací služby, při vytváření svých struktur pro vyhledávání, různé analytické nástroje nebo sám crawler při recrawlování.

3.2 Rychlost crawlerů

Cílem každého výkonného crawleru je stáhnout co nejvíce relevantních dat. Mnoho tvůrců crawlerů tento úkol bere absolutně a zaměřují se především na počet stránek, který je možno stáhnout a indexovat za jednotku času. Takový přístup je opodstatněný a každý v praxi nasazovaný robot musí mít co největší propustnost.

Rychlost bývá závislá především na rychlosti, s jakou probíhá indexace dat, tedy jejich zápis do odpovídajících struktur. Naopak kapacita linky překvapivě není moc často úzkým hrdlem celého procesu. Je to dáno její relativně nižší cenou, oproti nákladům za vytváření rozsáhlých struktur nad velkými kolekcemi dat.

Kapacita linky je ovšem značně omezující v případě, že server, odkud crawler stahuje data, má pomalé připojení a velkou latenci. Nejedná se zde ovšem o kapacitu linky, kterou by si mohl vlastník crawleru koupit. Další nepříjemností, která značnou měrou ovlivňuje nejen rychlost crawlování ale i soubor stažených stránek, je nutnost čekat danou dobu (obvykle 5 - 30 sekund) mezi jednotlivými dotazy na jeden server.

Práce [2] uvádí rychlost crawleru 50-100 str./s. Původní práce autorů vyhledávače Google [9] mluví o 100 str./s. při použití několika crawlerů. Další práce [27], [5] nebo [39] hovoří o rychlostech jednotlivých crawlerů v řádech stovek stránek za sekundu. Při rychlosti sta stránek za sekundu tak lze za jeden den nacrawlovat kolekci 8 640 000 stránek.

Pro indexy velkých vyhledávačů taková rychlost není dostatečná. Index o velikosti několika miliard stránek je třeba obnovit v průběhu jednoho crawlovacího cyklu, který trvá přibližně týden. Významného zvýšení rychlosti crawlování tak lze dosáhnout použitím kvalitního distribuovaného řešení.

Podle [26] je crawler, použitý v projektu vyhledávače nabídek zboží become.com, schopen stáhnout kolekci asi 3 miliard stránek za 1 týden. To odpovídá rychlosti asi 5000 str./s. Podle návrhu je přitom do crawlovacího procesu zapojeno 30 distribuovaných strojů. Na jeden distribuovaný crawler tak připadá rychlost přibližně 170 str./s.

Pro měření rychlosti crawlerů neexistují žádné standardizované postupy. Bylo by nutné standardizovat nejen konfiguraci počítače, na kterém bude crawler spuštěn, ale také odezvu serverů, ze kterých stahuje data. To je ale vzhledem ke komplexnosti problému velice složité, viz [3].

3.3 Architektura současných crawlerů

Na internetu se běžně pohybují desítky crawlerů. Většinou se jedná o výrobky konkrétních firem a jejich architektura bývá skryta. Přesto však bylo publikováno několik článků o architekturách, ať již komerčních nebo akademických crawlerů. Především na akademické půdě vzniklo několik projektů, které mají volně dostupný zdrojový kód. Dobrý přehled publikací o architektuře crawlerů podává v sekci 2.1 práce [3].

3.4 Vylepšení procesu crawlování

Vyhledávací služby pravděpodobně využívají mnoho vylepšení, která urychlují crawlovací proces. Bohužel, mnohé tyto postupy jsou chráněná obchodní tajemství. Autoři robotů se také často bojí spamování a tak nechávají důležité implementační detaily raději neveřejné. Přesto i v tomto směru bylo publikováno několik vědeckých prací, viz [21].

Základní optimalizací crawlerů je detekce duplikátů celých webů. Pro jeden web často existuje mnoho zrcadel, která poskytují totožný obsah. Vyhledávání duplikátů se věnuje například práce [15].

Práce [19] zmiňuje možnost použití hlavičky při recrawlování. Pokud je v hlavičce poskytnut i kontrolní součet, je možné při recrawlování tento údaj využít a recrawlovat fyzická data jen pokud stránka byla změněna. Podobně lze rozhodnout o recrawlování na základě informace o čase poslední modifikace z hlavičky stránky. Problém je, že hlavička stránky nemusí uvádět pravdivé informace. Navíc tyto údaje poskytuje asi jen 46% serverů. Podle autorů [19] může přesto detekce duplikátů z hlavičky ušetřit až 31% nákladů na přenos.

Další možností vylepšení procesu crawlování, především ve fázi recrawlingu, je použití takzvaných inkrementálních crawlerů, viz [19] a [21]. Při recrawlování si nevytváří znovu strukturu úložiště dat, ale využívají a obnovují strukturu vytvořenou při prvním crawlování. I když toto řešení přináší problémy se synchronizací, inkrementální crawlery poskytují větší možnosti okamžitého využití nově stažených dat například při vyhledávání.

3.5 Recrawlování

Internet se vyvíjí. Stránky vznikají, zanikají a mění se jejich obsah. Crawlery v praxi mají již jednou staženou významnou část internetu a namísto toho, aby celý proces crawlování opakovaly od začátku, provádějí recrawlování. Tedy opětovné stahování již jednou stažených stránek s využitím informace o nich z předchozích fází crawlovacího procesu.

Tento přístup může být výhodný z mnoha důvodů. Jedním z nich je, že množina stahovaných URL je předem známa a nemusí se získávat analýzou odkazů v průběhu crawlování. To může rapidně urychlit celý proces. Je možné lépe plánovat jak budou rozděleny jednotlivé URL na crawlery. Obvykle se totiž používá několik crawlerů, které se vzájemně synchronizují. Dále je možné, na základě znalosti frekvence změn stránky, optimalizovat pro každou URL intervaly mezi jednotlivými staženími dané stránky. Méně dynamické stránky tak lze stahovat méně často, než například často se měnící stránky zpravodajských serverů.

Je třeba mít vždy na paměti konkrétní účel celé kolekce. Práce [12] uvádí, že chceme-li maximalizovat čerstvost kolekce jako celku, je výhodnější častěji stahovat stránky, které se mění málo často, než stránky s velkou rychlostí aktualizace. U velmi dynamických stránek totiž k zastarávání dochází průměrně mnohem rychleji než u ostatních. Stahujeme-li časově stabilnější stránky, znamená to, že pravděpodobnost změny těsně po aktualizaci bude menší a celá kolekce zůstane v průměru více čerstvá.

Podle velikosti PageRanku a obsahu stránky je možné odhadnout jak často bude ve vyhledávači stránka zobrazena ve výsledcích. Přitom stránky, které budou zobrazeny častěji, by měly být také častěji aktualizovány. Optimální strategie by tedy měla zohlednit, jak požadavek na čerstvost kolekce jako celku, tak větší význam častěji vyhledávaných výsledků.

Práce [41] se zabývá navržením algoritmů pro recrawlování. Navrhuje strategie, které minimalizují zastaralost indexů vyhledávačů. Jednak je možné minimalizovat průměrný počet zastaralých stránek v indexu nebo minimalizovat počet selhání vyhledávače. Přitom selháním se rozumí, že na dotaz uživatele vyhledávač vrátí odkaz, na který uživatel klikne a odkaz je buď poškozený nebo obsah stránky neodpovídá zadanému dotazu.

Celý model využívá teorii pravděpodobnosti, teorii alokace prostředků a konvexní analýzu. Navrhuje optimální intervaly mezi recrawlováním jednotlivých URL v rámci daného časového úseku. Dále představuje strategii, jak pomocí řešení dopravního problému z teorie optimalizace, lze přidělit poža-

davky na recrawlování jednotlivým synchronizovaným crawlerům a sestavit tak rozvrh recrawlování.

Předností recrawlování oproti plnému crawlovacímu procesu je především úspora zdrojů. Nevýhodou je, že tyto strategie málo využívají struktury Webu a tak, pokud se změní postavení dané stránky ve struktuře internetu, je pro recrawlovací strategie těžké tyto změny rozpoznat. Pokud například byly odstraněny některé odkazy na danou URL, stránka by se v řádném procesu crawlování do indexu vůbec nemusela dostat a naopak by se do něj mohly dostat jiné stránky. Strategie pro recrawlování také musí počítat s tím, že se mohou změnit odkazy ze stažených stránek a musí se nějak postavit ke stahování nově nalezených URL.

Kapitola 4

Strategie Crawlování

Crawlovací strategie je strategie procházení grafu Webu. Určuje jaký bude příští navštívený vrchol grafu. Na crawlovací strategii se tedy lze dívat jako na algoritmus, který vytváří očíslování na grafu Webu podle posloupnosti postupně stahovaných stránek.

V reálných crawlerech musí crawlovací strategie řešit i další úkoly dané nejen strukturou webového grafu, ale také kapacitou linky, dostupností jednotlivých serverů, místem na disku, řešením problémů s dynamicky generovanými internetovými stránkami a pravidly popsanými v souborech robots.txt.

Asi nejkomplexnější popis a srovnání crawlovacích strategií podává článek [3].

4.1 Porovnávání crawlovacích strategií

Obvyklý způsob jak porovnat crawlovací strategie je, spustit je na předem známých a nacrawlovaných datech a porovnávat, jak rychle je stažena většina důležitých stránek. Tedy zda daná strategie stahuje jako první stránky, které mají největší váhu a stránky s nejmenší vahou jako poslední. Obvyklou měrou důležitosti je pak PageRank.

4.2 Strategie BFS

Nové URL jsou řazeny na konec fronty stránek ke stažení. Celý proces stahování tak probíhá postupně po jednotlivých vrstvách. Nejprve je stažena

startovací stránka (nultý krok). V n-tém kroku jsou staženy všechny stránky vzdálené od nulté právě n kliknutí.

Překvapivě se tato strategie stahování ukazuje jako jedna z nejlepších, viz [37]. Nejenže jako první stahuje stránky s vysokým PageRankem, ale také méně než ostatní strategie deformuje parciální PageRank. Při stahování stránek touto strategií se PageRank spočítaný na dosud stažených stránkách více blíží PageRanku na celém internetu, než je tomu u strategií parciální důležitosti, viz [8].

4.3 Strategie DFS

Nově nalezené URL jsou dávány na zásobník. Prohledávání internetu tak probíhá podle schématu DFS. Překvapivě se tato strategie ukazuje jako jedna z nejlepších, jedná-li se o rychlost s jakou se blíží parciální důležitost v průběhu crawlování k důležitosti na celém souboru dat, viz [8]. Metodu, která kombinuje BFS a DFS s nastavitelnou hloubkou DFS procházení využívá například UbiCrawler [7].

4.4 Strategie parciální důležitosti

Tyto strategie provedou crawlování do určité hloubky. Poté co je v indexu dostatek stránek, spočítá strategie důležitost (PageRank) na dosud známých datech. Další stránky strategie stahuje v pořadí podle důležitosti na parciálním souboru. Poté, co počet stažených stránek překročí jistou mez(například se zdvojnásobí) provede strategie přepočítání důležitosti. Tyto strategie bývají, co do rychlosti s jakou stáhnou největší podíl PageRanku, nejefektivnější. Jsou zkoumány v [14] a [3]. Práce [8] ale ukazuje, že jejich parciální důležitost konverguje nejpomaleji ze zkoumaných strategií k důležitosti spočítané na celém souboru. Společně s faktem, že lze stáhnout jen malou podmnožinu internetu, je možné se domnívat, že kolekce stažené s použitím těchto strategií, mají špatně zavedenou míru důležitosti oproti odpovídající metrice důležitosti na internetu.

4.5 Strategie důležitosti známé z předchozího crawlování

Jsou-li k dispozici údaje z předchozích úplných crawlování, některé crawlery je používají k řazení stránek pro crawlování nové.

Tyto strategie jsou zajímavé, protože data z předchozích crawlování pravděpodobně využívají crawlery velkých vyhledávačů. Tento přístup může totiž významně urychlit indexaci stávajících stránek při recrawlingu.

Podrobná měření efektivity jednotlivých metod jsou relativně náročná. Je nutné mít dva soubory stránek z jisté části internetu stažené v různých časových obdobích a nad nimi provádět výzkum. Významné pro výsledky bude, jak moc dynamický obsah byl stažen a po jak dlouhé době recrawling probíhá. Kvůli velké složitosti takového výzkumu není v odborné literatuře mnoho článků na toto téma.

Strategie obvykle stahuje stránky v pořadí podle důležitosti spočítané na souboru z předchozího crawlování. V průběhu nového průchodu se však mohou objevit neznámé URL adresy, které je nutné zařadit na odpovídající místo ve frontě stránek ke stažení. Práce [3] uvádí několik možných přístupů, jak nově nalezené URL řadit.

4.5.1 Vševědoucí přístup

Důležitost nových URL je stanovena jako jejich konečná důležitost spočítaná na konci crawlování. Tato možnost není v praxi reálná. Je ji ale možné uplatnit, pokud celý soubor máme stažený a provádíme simulaci crawlování.

4.5.2 Náhodná distribuce důležitosti

PageRank nové URL je přidělen náhodně podle rozdělení PageRanku v původním souboru.

4.5.3 Přístup nulové důležitosti

Nové URL mají nulovou důležitost. To znamená, že na ně přijde řada až po stažení všech URL známých z předchozího crawlování.

4.5.4 Přístup důležitosti rodiče

Stránce se přidělí důležitost jejího rodiče dělená počtem stránek, na které rodič odkazuje. Tento podíl se nazývá skóre, viz definice 5.8 a hraje důležitou roli v definici míry důležitosti K-Ranku.

Podle [3] je nejméně vhodný přístup s nulovou důležitostí. Podobně dopadají přístupy s náhodnou distribucí důležitosti a důležitostí rodiče. Nejlépe dopadá vševědoucí přístup.

4.6 Strategie OPIC

Strategie OPIC(On-line page importance computation) [1] přiděluje každé stránce určitou sumu kreditů. Na začátku dostanou kredity pouze startovní stránky. V každém kroku je stažena ta stránka, která má na účtu nejvíc kreditů. Je-li stránka stažena, všechny její kredity se rovnoměrně rozdělí mezi stránky, na které odkazuje a které zatím nebyly staženy.

Ukazuje se, že tato strategie je, co do rychlosti konvergence jednou z nejefektivnějších. Podle [3] má dokonce lepší výsledky než strategie parciální důležitosti. Jediná strategie, která ji předběhla byla vševědoucí strategie popsaná dále. Navíc je poměrně snadné ji implementovat.

4.7 Další strategie

4.7.1 Strategie nejdřív velké weby

Tato strategie využívá toho, že mezi jednotlivými stahováními z jednoho zdroje musí být relativně dlouhá prodleva (5-30 sekund), aby nedošlo k přetížení serveru požadavky crawleru. Prioritně stahuje stránky, pro které je třída ekvivalence indukovaná "bytím z jednoho serveru" největší. Podle [3] si strategie vede hůře v první polovině stahování. Na konci se však stává vedoucí strategií a předstihuje dokonce i globálně vševědoucí strategii. Zůstává otázkou, zda chování v druhé polovině crawlování není u této strategie způsobeno tím, že v uspořádání stahovacího experimentu byla konečná množina stránek, které bylo třeba všechny stáhnout. Není tak jasné, zda by si tato strategie vedla stejně dobře i na internetu, kde je z jeho povahy pořád co stahovat.

4.7.2 Vševědoucí strategie

Globální vševědoucí strategie spočívá v tom, že stahuje jako první URL s nejvyšší důležitostí na celém souboru. To je možné jen pokud je kolekce předem stažená a příslušná míra důležitosti je na ní spočítaná.

Protože strategie musí dodržovat stromovou strukturu Webu, vede si hůře, než prostý součet největších důležitostí.

Podle [3] si tato strategie vede nejlépe co se týká rychlosti stahování nejdůležitějších stránek asi do 70% crawlování, kde jí předbíhá strategie nejdřív velké weby.

4.8 Vliv strategie crawlování na vyhledávání

Je možná trochu překvapivé, že strategie crawlování může významným způsobem ovlivnit výsledky vyhledávání na jednotlivé dotazy. Míry důležitosti jako PageRank totiž závisí na konkrétní podmnožině Webu, která byla stažena. Protože crawlovací strategie ovlivňuje podobu stažené struktury, závisí na použité strategii i výsledek počítání míry důležitosti. Kvantifikovat vliv crawlovací strategie na uspořádání vektoru důležitosti se snaží i tato práce.

Článek [19] zmiňuje, že existuje přímá úměrnost mezi počtem stránek, stažených z konkrétních zdrojů a zastoupením těchto zdrojů ve výsledcích vyhledávání. To je, vzhledem k existenci měr jako PageRank, poměrně zajímavá skutečnost. V této práci proto představuji míru, která by měla zrovnoměrnit distribuci důležitosti a předejít podobným efektům.

4.9 Vliv primární crawlovací strategie na výsledky

Měření vlastností crawlovacích strategií obvykle neprobíhá na internetu, ale pouze na jeho části, která byla předem stažena. Pokud tuto část v průběhu experimentu měřená strategie zcela nebo téměř zcela vyčerpá, budou výsledky experimentu téměř jistě zkresleny primární crawlovací strategií. Její vliv lze částečně eliminovat tak, že míra vyčerpání primární kolekce bude menší. Problém může nastat v tom, že pak každá strategie stáhne trochu jiné stránky a jejich porovnávání tak bude obtížnější. Navíc se pravděpodobně stejně nelze zcela vyhnout situaci, kdy se zkoumaná strategie dostane

někam "na okraj" primární kolekce, tedy k URL, kterou by tato strategie stáhla, ale v primární strategii zůstává nestažená. Mírou vlivu primární crawlovací strategie při zkoumání strategií se zabývám ve výpočetní části této práce.

Kapitola 5

Míry důležitosti

5.1 PageRank

PageRank byl představen v původní práci [9]. Je to obdoba citačního indexu, kdy dokument je tak důležitý, jak důležité jsou dokumenty, které na něj odkazují.

Aby bylo možné formálně definovat vektor PageRanku, bude vhodné uvést několik definic a pojmů především z lineární algebry.

Definice 5.1 (Matice Webu)**.** Buď M matice $n \times n$, kde n je počet stránek stažené kolekce. Každému dokumentu v kolekci je přitom přiřazeno číslo $ID \in (1..n)$. M nazveme Maticí Webu, pokud pro její prvek na místě (i, j) platí:

- $M_{i,j} = 1/c_j$ pokud dokument s $ID\ j$ odkazuje na dokument s $ID\ i$, c_j je celkový počet odkazů, které vedou z dokumentu s $ID\ j$
- $M_{i,j} = 0$ jinak

Definice 5.2 (Stochastická matice)**.** Buď M matice $n \times n$. M je stochastická, pokud má všechny prvky nezáporné a je-li součet všech jejích sloupců roven jedné.

Definice 5.3 (Nerozložitelná matice)**.** Buď M matice $n \times n$. M je nerozložitelná, pokud ji nelze permutací řádků a symetrickou permutací odpovídajících sloupců převést do tvaru:

$$\mathbf{M} = \begin{pmatrix} C_1 & A \\ O & C_2 \end{pmatrix}$$

Kde C_1 a C_2 jsou čtvercové matice, A je matice odpovídajícího typu a O je nulová matice.

Lze nahlédnout, viz [32], že pro nerozložitelné stochastické matice existuje vlastní vektor příslušný vlastnímu číslu 1 s nezápornými prvky a součtem prvků 1. Je to jediný vlastní vektor příslušný vlastnímu číslu 1 (až na násobení konstantou).

Dále lze nahlédnout, že jednoduchá mocninná iterační metoda bude konvergovat právě k tomuto vektoru.

Definice 5.4 (Jedničková matice)**.** Symbol J bude označovat čtvercovou matici $n \times n$, která má všechny své prvky rovny 1.

Definice 5.5 (PageRank ... klasická definice)**.** Buď W matice Webu taková, že neobsahuje žádné sloupce reprezentující stránky, ze kterých nevede žádný odkaz. Buď $e \in (0, 1)$. Pak vektor PageRanku je vlastní vektor příslušející vlastnímu číslu 1 pro stochastickou nerozložitelnou matici $\widetilde{W} = (1 - e)W + (e/n)J$. PageRank stránky odpovídá velikostí prvku vlastního vektoru na pozici dané jejím ID.

Číslu $d = (1 - e)$ se říká dumping faktor. Obvykle se volí $d = 0.75$, tedy $e = 0.25$. Práce [25] ukazuje souvislost d s velikostí druhého vlastního čísla matice $\widetilde{W}$ a tedy i velikostí spektrální mezery. Konkrétně $|\lambda_2| \leq d$. e tedy udává minimální spektrální mezeru, tedy vzdálenost absolutních hodnot dvou v absolutní hodnotě největších vlastních čísel. Velikost e souvisí s podmíněností matice $\widetilde{W}$. V teorii Markovových řetězců lze dokázat [28], že velikost spektrální mezery souvisí s rychlostí konvergence iteračních metod k vlastnímu vektoru. Čím větší tedy volíme e, tím větší stabilitu konvergence máme zajištěnu.

Volba nenulového e má i fundamentální opodstatnění. PageRank stránky odpovídá pravděpodobnosti, že člověk, který prochází internet, se po nekonečném počtu kliknutí bude nacházet na příslušné stránce. Surfer začíná z náhodně zvolené stránky na internetu. Nenulové e odpovídá situaci, že v každém kliknutí s pravděpodobností $1 - e$ bude následovat odkazy z dané stránky a s pravděpodobností e přejde na náhodnou stránku internetu. To docela odpovídá chování lidí, kteří při procházení internetu nejenže klikají na odkazy, ale někdy prostě jen zadají novou adresu, kam chtějí přejít.

Vzhledem k tomu, že z jedné stránky vede obvykle ne více než stovky odkazů, je matice Webu W, která typicky obsahuje desítky milionů řádků

velmi řídká. Naproti tomu $\widetilde{W}$ je velmi hustá. Proto byly vyvinuty algoritmy, jak při výpočtu PageRanku počítat stále s řídkou maticí a přitom dostat vlastní vektor husté matice $\widetilde{W}$ viz [7].

Algoritmus 5.1 Mocninná metoda pro výpočet PageRanku

1: N = MAXID ▷ Počet všech vrcholů
2: float[] Rank = new float[N] ▷ Pole pro PageRank všech vrcholů
3: **while** $||Rank|| > maxerr$ **do**
4: $Rank = (1 - e)W \times Rank + e[\frac{1}{N}]_{N \times 1}$
▷ $Rank = \widetilde{W} \times Rank$
5: **end while**
6: **return** $Rank$

Tato iterační metoda odpovídá klasické mocninné iterační metodě, pouze zde dochází k rozložení každého kroku na dvě fáze. Konvergence je tedy zaručena konvergencí mocninné metody.

Protože je zde patrná snaha stále zvětšovat indexy vyhledávačů, je nutné vylepšovat techniky výpočtu PageRanku tak, aby byly co nejefektivnější. Navíc pro perzonifikaci je často třeba, aby si uživatel spočítal osobní vektor PageRanku sám na svém počítači s omezenými zdroji. Akcelerací a vylepšováním iterativního výpočtu PageRanku se zabývá mnoho prací, například [11], [30], [24].

Asi největším problémem algoritmu výpočtu PageRanku je zacházení s vrcholy, ze kterých nevedou žádné odkazy. V klasickém přístupu je nutné tyto vrcholy rekurzivně odstranit. To ovšem vede ke značnému zkreslení celého grafu Webu. Navíc listové vrcholy nejsou vůbec ohodnoceny. To je nevhodné zvlášť pro crawlování, pro které je důležité znát právě váhy stránek, které ještě nebyly navštíveny a tvoří tak, v dané fázi crawlování, listové uzly grafu Webu.

Je překvapivé, že crawlovací strategie pro počítání parciálního PageRanku byly původně navrženy nad algoritmy, které neohodnocovaly listové vrcholy. Přitom podle pořadí důležitosti na listových vrcholech jsou u těchto strategií řazeny URL ke stahování. V původních návrzích to bylo řešeno tak, že listové vrcholy byly sice v průběhu počítání PageRanku z grafu rekurzivně odstraněny, v poslední fázi však byly do grafu vráceny a nad takto upraveným grafem proběhlo posledních několik iterací. Podle takto získaných hodnot vylo sestaveno pořadí vrcholů k dalšímu stahování. Práce [8] uvádí zacházení s vrcholy s nulovým počtem odkazů na prvním místě mož-

ných zdrojů chyb při porovnávání crawlovacích strategií.

V práci [18] je představen jiný koncept zacházení s listovými vrcholy. Každá stránka, která nemá žádné odkazy, nebo odkazy z ní vedoucí neznáme, je chápána tak, jakoby z ní vedly odkazy na všechny stránky známého Webu.

To vede k přirozené definici

Definice 5.6 (PageRank)**.** Buď W matice Webu. Buď $e \in (0, 1)$. Pak vektor PageRanku je vlastní vektor příslušející vlastnímu číslu 1 pro stochastickou nerozložitelnou matici

$$\widehat{W} = (1 - e)W + (1 - e)D + (e/n)J \tag{5.1}$$

D je přitom matice $D = vo^T$, kde $o_i = 1$, pokud z vrcholu i nevede žádný odkaz a $o_i = 0$ jinak. v je vektor $[\frac{1}{N}]_{N\times 1}$. PageRank stránky je dán velikostí prvku vlastního vektoru matice $\widehat{W}$ na pozici dané jejím ID. Pokud zavedeme vektor j tak, že $j_i = 1 \forall i$, pak PageRank je vektor z, pro který platí

$$\widehat{W}z = z \tag{5.2}$$

tedy

$$((1 - e)(W + vo^T) + evj^T)z = z \tag{5.3}$$

Často se při vyhledávání skloňuje perzonifikace. Chceme-li zavést hodnocení stránek zvlášť pro jednotlivé uživatele podle jejich preferencí, je možné zavést perzonifikační vektor p. Jeho součet je 1 a vektor musí mít všechny prvky kladné. Pak lze v definici PageRanku uniformní vektor v nahradit perzonifikačním vektorem p a jedničkovou matici J nahradit maticí pj^T, kde $j_i = 1 \forall i$. Pro účely crawlování není vhodné zvýhodňovat některé stránky na úkor jiných. Proto jsem se perzonifikací v této práci dále nezabýval.

Hlavní nevýhodou PageRanku zavedeného definicí 5.6 je, že matice $\widehat{W}$ je kvůli přítomnosti vrcholů s nulovým počtem odkazů hustá a nejde přímo použít podobný trik jako v algoritmu 5.1, kde součin řídké matice a vektoru lze jednoduše upravit tak, že výsledek odpovídá součinu pro hustou matici. Práce [18] však uvádí způsob, jak výpočet PageRanku, tedy vlastního vektoru pro matici $\widehat{W}$ transformovat na řešení lineární soustavy rovnic s řídkou maticí. Článkem [18] jsem se velmi inspiroval při psaní kódu pro

výpočet PageRanku. Autoři navíc ukazují, že při použití jistých transformací původní matice, lze při samotném výpočtu ušetřit až 89% času oproti mocninné metodě.

Implementoval jsem všechny postupy popsané v [18]. Skutečně jsem zaznamenal značnou úsporu času při běhu metody. Avšak přeuspořádání tvaru matice vyžadovalo velké časové nároky. Proto se jako nejefektivnější pro daný úkol ukázalo sestavit lineární systém rovnic a spočítat řešení klasickou Gauss-Seidelovou metodou.

Hlavní přínos [18] tak je v tom, že ohodnocení vrcholů, ze kterých nevede odkaz, je mnohem přirozenější než původní přístupy a že existují časově efektivní algoritmy na spočítání takto zavedeného PageRanku. Časová náročnost daných algoritmů přitom není větší než u mocninné metody. Jediné, co prodlužuje dobu výpočtu tak může být, že graf je větší o ty hrany, které byly v původním konceptu ořezány. To však vyvažuje fakt, že tyto hrany nesou velkou část informace o celém Webu.

5.2 K-Rank

Ačkoli je koncept PageRanku na internetu v současnosti široce používaný, existují jistá omezení původního algoritmu.

Asi největším problémem je možnost spamování crawlerů a potažmo výpočtu PageRanku. Proto, aby byla konkrétní URL zvýhodněna před ostatními, stačí na ni na různých místech internetu umístit odkazy. Často dochází ke spamování různých fór nebo diskuzí na internetu odkazy, které s daným tématem nemají nic společného. Roboty jsou pak nuceny takové odkazy následovat a zařadit je do modelu pro počítání PageRanku. Další sofistikovanější metodou jak uměle zvýšit PageRank, je vytvořit celou kolekci stránek a navzájem je prolinkovat. Tak se zvýší počet stránek odkazujících na cílovou stránku. Opět takové kolekce často obsahují nesmyslné informace a jejich jediným cílem je zvýšit pozici ve vyhledávačích pro konkrétní stránky.

To, že existují postupy jak PageRank ovlivnit, je také jedním z důvodů, proč velké vyhledávače nezveřejňují přesné algoritmy pro crawlování a vyhledávání.

Dalším problémem je, že algoritmus PageRanku preferuje staré stránky, na které bylo vytvořeno ohromné množství odkazů. Nové výsledky, například aktuální zprávy nebo dnešní předpověď počasí, pravděpodobně nenajdeme na prvních místech ve výsledcích současných vyhledávačů.

O pozici stránky ve vyhledávačích tak často nerozhoduje kvalita zdrojů, které na ni odkazují, ale pouhá kvantita odkazů (kterou lze téměř libovolně zvyšovat).

Proces crawlování by měl spíše, než preferovat existující již stažené stránky, vyhledávat nové informace. Problémem je, že velikost PageRanku pro nově vzniklé stránky je nízká a tak jsou, ať už při crawlování nebo vyhledávání, často opomíjeny. To způsobuje paradoxní situaci, kdy „bohatí bohatnou a chudí chudnou“.

Jedním z cílů této diplomové práce je modifikovat původní algoritmus PageRanku, aby lépe odpovídal filozofii crawlování, preferoval nové stránky s relevantními informacemi a mohl účinně vzdorovat spamu.

Někteří autoři [4] se snaží pomocí časových razítek pro jednotlivá crawlování zjišťovat, jak často jsou dané stránky aktualizovány nebo, jak moc je daná URL nová. U nových stránek pak odstraňují handicap způsobený jejich menším zapojením do struktury internetu a tedy nižším PageRankem. Článek jasně ukazuje, že PageRank se v průběhu existence dané URL mění a tento vývoj lze aproximovat jistou křivkou.

Protože jsem ale pro svou práci neměl k dispozici žádné údaje o vývoji jednotlivých URL v průběhu času, musel jsem se bez těchto informací při návrhu modifikace PageRanku obejít. Mým úkolem tedy bylo, pokud mám k dispozici kolekci stránek vzniklých jedním crawlováním, zavést nad nimi vhodnou metriku důležitosti, která by byla odolná proti spamu a neznevýhodňovala nové a kvalitní stránky, i když jsem neměl explicitní informaci o tom, které to jsou.

Definice 5.7 (K-Rank ... ideální řešení)**.** Buď $K \in 1, 2, ...$ přirozené číslo. K-Rank pak definujeme jako PageRank pro matici Webu W_K. Přitom celý Web ořežeme tak, že na každý uzel bude v nové struktuře odkazovat maximálně K stránek. Ořezání provedeme následujícím způsobem:

- Má-li uzel i K nebo méně příchozích odkazů, všechny odkazy v grafu Webu zachováme
- Vede-li do i více než K odkazů, necháme v grafu jen právě K odkazů na uzel i a to právě ty, které generují největší PageRank pro daný uzel ze všech K-tic.

W_K je pak matice tímto způsobem ořezaného grafu Webu.

Toto řešení může docela dobře vzdorovat známým formám spamu. Pro každý uzel povoluje pouze stanovený počet příchozích odkazů. Kvalitní stránky, na které vedou odkazy z kvalitních zdrojů, by neměly být metrikou nijak znevýhodněny. Pro nekvalitní stránky s uměle navyšovanou důležitostí však stanovení stropu pro počet příchozích odkazů znamená velkou penalizaci.

Další výhodou je, že K-Rank neznevýhodňuje nové a kvalitní stránky. Například zpravodajství. Pokud bude zpráva opravdu zásadní, snadno se na ni vytvoří, relativně rychle, řekněme stovka odkazů. Pokud tyto odkazy budou z kvalitních zdrojů, může stránka svou důležitostí téměř okamžitě, už v době svého vzniku, konkurovat již zaběhlým URL.

Pro posuzování čerstvosti přitom není třeba zanášet do algoritmu dodatečné mechanizmy jako jsou časová razítka.

Kvalitní stránky s mnoha odkazy na ně by si neměly významně pohoršit. Dá se totiž očekávat, že mezi mnoha odkazy bude mnoho významných odkazů.

Problém ideální definice však je, že v současné době neumím dokázat, že matice W_k vůbec existuje a pokud ano, její nalezení bude pravděpodobně NP-úplný problém. Problémem je, že konkrétní volba každé k-tice pro konkrétní i změní charakter celého grafu a mohla by lavinovitě způsobit nutnost přepočítat další k-tice.

Protože jsem dosud nevymyslel způsob, jak efektivně zkonstruovat matici W_k, navrhl jsem postup, kdy ji aproximuji pomocí původního vektoru PageRanku.

Definice 5.8 (Skóre odkazu)**.** Je-li W matice Webu a $(i- > j)$ představuje odkaz z i na j. Pak skóre odkazu je PageRank stránky i dělený počtem jejích odchozích odkazů.

Definice 5.9 (K-Rank)**.** Buď $K \in 1, 2, ...$ přirozené číslo. K-Rank pak definujeme jako PageRank pro matici Webu W_K. Přitom celý Web ořežeme tak, že každý uzel má maximálně K příchozích odkazů, následujícím způsobem:

- Má-li uzel i K nebo méně příchozích odkazů, všechny odkazy v grafu Webu zachováme
- Má-li uzel i více než K příchozích odkazů, necháme v grafu jen právě K odkazů na uzel i a to právě ty, které mají největší skóre vůči PageRanku původní matice Webu.

Matice Webu W_K je pak matice tímto způsobem ořezaného grafu Webu.

Matice W_K je dobře definovaná a proto K-Rank skutečně existuje. Jeho nevýhodou je, že algoritmus musí být proveden ve dvou krocích. Nejprve je nutné spočítat PageRank. Na základě tohoto výpočtu podle definice ořezat graf Webu a opět spustit výpočet PageRanku. Druhý výpočet bude už kratší, protože graf Webu bude řidší. Myslím, že vyšší časovou náročnost vynahradí vyšší kvalita výsledků a jsem přesvědčen, že K-Rank je dobrou měrou důležitosti na internetu. Výpočetní část práce zkoumá rychlost konvergence, stabilitu a další vlastnosti K-Ranku.

Kapitola 6

Kendallovo τ

Kendallovo τ se používá pro porovnání uspořádání dvou stejně dlouhých seznamů čísel. Je rovno jedné, jsou-li seznamy uspořádány stejně (vzhledem k relaci nerovnosti pro každou dvojici odpovídajících prvků) a -1 pro inverzní seznamy. Jsou-li seznamy na sobě nezávislé, blíží se Kendallovo τ 0.

Výpočetní část Kendallovo τ používá především ke sledování toho, jak moc se mění uspořádání vektoru důležitosti v průběhu crawlování pro danou strategii. Dále ho používám pro stanovení míry korelace měr důležitosti.

Definice 6.1 (Kendallovo τ [8])**.** Buď a_i, b_i posloupnosti celočíselného nebo reálného typu (short, int, long, float, double), $i \in \{1..n\}$. Pro dvojici (k, l) takovou, že $k \in \{1..n\}$, $l \in \{1..n\}$, $k \neq l$ řekneme, že

- (k, l) je kladná, pokud výrazy $a_k - a_l$ a $b_k - b_l$ jsou oba nenulové a mají stejné znaménko
- (k, l) je záporná, pokud výrazy $a_k - a_l$ a $b_k - b_l$ jsou oba nenulové a mají opačné znaménko
- (k, l) je nerozhodná vzhledem k a, pokud $a_k - a_l = 0$
- (k, l) je nerozhodná vzhledem k b, pokud $b_k - b_l = 0$
- (k, l) je oboustranně nulová, pokud $a_k - a_l = b_k - b_l = 0$

Buď K počet kladných dvojic, Z počet záporných dvojic, O_a počet dvojic nerozhodných vzhledem k a, O_b počet dvojic nerozhodných vzhledem k b a J počet oboustranně nulových dvojic. N buď počet všech dvojic v jednom seznamu.

$$N = \frac{n(n-1)}{2} \tag{6.1}$$

Kendallovo τ pro posloupnosti a a b je definováno jako

$$\tau = \frac{K - Z}{\sqrt{(N - O_a)(N - O_b)}} \tag{6.2}$$

Zřejmě platí

$$N = K + Z + O_a + O_b - J \tag{6.3}$$

Kendallovo τ není definováno, pokud je jmenovatel ve výrazu 6.2 nulový. To nastává, pokud má některý ze seznamů všechny prvky stejné. Protože tuto metriku budu používat pro porovnávání uspořádání dvou vektorů důležitosti není tato podmínka nijak omezující. Vektor důležitosti, který by měl všechny prvky stejné, by byl totiž k ničemu.

6.1 Knightův algoritmus

Pro výpočet Kendallova τ je možné použít Knightův algoritmus [31]. Tento algoritmus běží v čase $O(n \log n)$ a jeho varianta používající Mergesort je popsaná v [8].

Pro potřeby diplomové práce jsem napsal vlastní implementaci Knightova algoritmu v Javě. Přitom používám vnitřní a vnější stabilní Mergesort.

Definice 6.2 (Stabilní Mergesort)**.** Mergesort, při kterém je použito jako sekundárního klíče pořadí prvku v původním seznamu. Pro dva stejné prvky tedy nedojde k jejich prohození v průběhu třídění. Implementace viz Algoritmus 6.2.

Knightův Algoritmus 6.1 dvakrát projde celý seznam a provede dvě třídění. Má amortizovanou časovou složitost $O(n \log n)$ a I/O náročnost $O(n)$.

Pro Algoritmus 6.2 je vstupem funkce `StableMergeSort`, setříděný seznam, primárně podle prvního indexu a sekundárně podle druhého indexu. Funkce seznam setřídí podle druhé položky a vrátí počet přehození.

Důkaz korektnosti algoritmu spočívá v tom, že každý pár (k, l), který je záporný, bude určitě započten v okamžiku, kdy dojde ke skoku b_k přes b_l při druhém třídění. Z vlastností Mergesortu pak plyne, že každá záporná dvojice bude v průběhu algoritmu přeskočena nejvýše jednou a to ve fázi

Algoritmus 6.1 Knightův algoritmus pro výpočet Kendallova τ

1: **function** KENDALLTAU(Seznam1, Seznam2)
▷ Seznamy musí být stejně dlouhé
2: **var** Seznam = new Seznam[n][2]
3: **var** $N = \frac{n(n-1)}{2}$
4: **for all** i **do**
5: Seznam[i][1] = Seznam1[i]
6: Seznam[i][2] = Seznam2[i]
7: **end for**
8: SORT(Seznam, 1, 2) ▷

- Setřídí seznam primárně podle prvního a sekundárně podle druhého indexu
- Pro třídění jsem použil vnější Mergesort a vnitřní Quicksort
- Časová složitost třídění je $O(n \log n)$ (pro QS amortizovaná), I/O náročnost je $O(n)$

9: **var** $O_1 = 0$ ▷ počet nulových dvojic v prvním seznamu
10: **for all** (i, j): ISMAXIMAL(i, j, Seznam, 1) = `true` **do**
11: **var** $k = j - i + 1$
12: $O_1 = O_1 + \frac{k(k-1)}{2}$
13: **end for**
14: **var** $J = 0$ ▷ počet oboustranně nulových dvojic pro oba seznamy
15: **for all** (i, j): ISMAXIMAL$(i, j, Seznam)$ = `true` **do**
16: **var** $k = j - i + 1$
17: $J = J + \frac{k(k-1)}{2}$
18: **end for**
19: **var** Z = STABLEMERGESORT(Seznam, 1, n)
20: **var** $O_2 = 0$ ▷ počet nulových dvojic v druhém seznamu
21: **for all** (i, j): ISMAXIMAL(i, j, Seznam, 2) = `true` **do**
22: **var** $k = j - i + 1$
23: $O_2 = O_2 + \frac{k(k-1)}{2}$
24: **end for**
25: **var** $K = N - Z - O_a - O_b + J$ ▷ počet kladných dvojic
26: **return** $\frac{K-Z}{\sqrt{(N-O_1)(N-O_2)}}$
27: **end function**

Algoritmus 6.2 Stabilní Mergesort

```
1: var Pompole = new Pompole[n][2]
2: function StableMergeSort(Seznam, zac, kon)
3:     var stred = (zac + kon)/2
4:     var shifts = 0
5:     if zac ≤ stred then
6:         shifts = shifts + StableMergeSort(Seznam, zac, stred)
7:     end if
8:     if stred + 1 ≤ kon then
9:         shifts = shifts + StableMergeSort(Seznam, stred+1, kon)
10:    end if
11:    var i = zac
12:    var j = stred + 1
13:    var k = zac
14:    while (i ≤ stred) and (j ≤ kon) do
15:        if Seznam[i][2] ≤ Seznam[j][2] then
16:            Pompole[k][1] = Seznam[i][1]
17:            Pompole[k][2] = Seznam[i][2]
18:            i = i + 1
19:        else
20:            Pompole[k][1] = Seznam[j][1]
21:            Pompole[k][2] = Seznam[j][2]
22:            shifts = shifts + j - k
23:            j = j + 1
24:        end if
25:        k = k + 1
26:    end while
27:    while i ≤ stred do
28:        Pompole[k][1] = Seznam[i][1]
29:        Pompole[k][2] = Seznam[i][2]
30:        i = i + 1
31:        k = k + 1
32:    end while
```

Algoritmus 6.3 Stabilní Mergesort a pomocné funkce pro Algoritmus 6.1

```
33:     while j ≤ kon do
34:         Pompole[k][1] = Seznam[j][1]
35:         Pompole[k][2] = Seznam[j][2]
36:         j = j + 1
37:         k = k + 1
38:     end while
39:     for all k ∈ {zac...kon} do
40:         Seznam[k][1] = Pompole[k][1]
41:         Seznam[k][2] = Pompole[k][2]
42:     end for
43:     return shifts
44: end function
45: function IsMaximal(i, j, Seznam, e)
46:     if Seznam[i − 1][e] ≠ Seznam[i][e] = Seznam[i + 1][e] = ... =
    Seznam[j][e] ≠ Seznam[j + 1][e] then
47:         if i ≤ j then
48:             return true
49:         end if
50:     end if
51:     return false
52: end function
53: function IsMaximal(i, j, Seznam)
54:     if Seznam[i][1] = Seznam[i + 1][1] = ... = Seznam[j][1] then
55:         if Seznam[i][2] = Seznam[i + 1][2] = ... = Seznam[j][2] then
56:             if Seznam[i − 1][1] ≠ Seznam[i][1] or Seznam[i − 1][2] ≠
    Seznam[i][2] then
57:                 if Seznam[j][1] ≠ Seznam[j + 1][1] or Seznam[j][2] ≠
    Seznam[j + 1][2] then
58:                     if i ≤ j then
59:                         return true
60:                     end if
61:                 end if
62:             end if
63:         end if
64:     end if
65:     return false
66: end function
```

slévání, kdy indexy k a l leží v různých seznamech. Po slití budou oba prvky ležet ve stejném seznamu, a proto se již nemohou přeskočit.

Implementačně poněkud složitější byl vnější stabilní Mergesort, který vrací počet přeskoků. Bylo to proto, že při slévání, slévám více než dva soubory do jednoho. Slévané soubory tvoří haldu organizovanou podle velikosti jejich aktuálně čteného prvku. Každý soubor v haldě reprezentuje blok původního souboru, který byl setříděn ve vnitřní paměti. Údaj o počtu přehození při vnitřním třídění vrací funkce `StableMergeSort`, jak byla popsána v Algoritmu 6.2. K součtu všech přehození na jednotlivých blocích je třeba přičíst přehození při slévání bloků.

Pro každý záznam, přečtený z haldy při operaci `deleteMin`, je třeba znát kolik přesně bylo záznamů v blocích před blokem, ze kterého je záznam načítán a celkový počet operací `deleteMin`, provedených na tomto bloku. Dále má algoritmus čítač pro celkový počet všech provedených operací `deleteMin` na celé struktuře. Pokud je jejich celkový počet nižší než součet počtu záznamů v předchozích blocích s počtem operací `deleteMin` na aktuálním bloku, je třeba zvýšit počet přehození o tento rozdíl, při každé operaci `deleteMin`.

Kapitola 7

Navržené postupy, algoritmy a jejich implementace

Veškeré výpočty pro svou diplomovou práci jsem prováděl v počítačové laboratoři na Malé Straně. Nejvýkonnější stroje, které jsem měl k dispozici mají 2GB RAM a procesor Athlon dual core 4200+. Velikost paměti tak umožňuje uložit 536 870 912 položek typu int délky 4B.

Ke všem výpočtům jsem používal jazyk Java (verze VM 1.5). Systém mi pro potřeby provedení výpočtů dovolil pro Java VM alokovat maximálně 1624MB. To odpovídá 425 721 856 položkám typu int. Ve skutečnosti však samotné jádro virtuálního stroje a management paměti z tohoto množství ubraly další část. Reálně tak bylo možné alokovat pole velké přibližně 379 000 000 položek int.

Bylo jasné, že s takovouto konfigurací není myslitelné, aby údaje o struktuře Webu byly uloženy v hlavní paměti. Například struktura odkazů pro doménu cz čítala 3 000 000 000 položek typu int, to odpovídá asi 11GB. Pro práci s danými daty v hlavní paměti by tak bylo nutné mít k dispozici alespoň pětkrát větší kapacitu.

Právě uvedené omezení znamenalo, že musely být použity takové struktury, které umožňovaly co nejsnazší sekvenční čtení z disku a rychlou práci s daty mimo hlavní paměť. To často zvětšovalo celkovou velikost struktur. Bohužel jsem na počítačích v laboratoři neměl přidělené místo na disku a tak se často stávalo, že výsledky ukládané do adresářů tmp byly promazávány automatickými utilitami.

Přestože jsem neměl optimální podmínky co do hardware, podařilo se mi navrhnout takové algoritmy, které umožnily provést výpočty měr důleži-

tosti (PageRank, K-Rank), míry uspořádanosti, Kendallova τ a crawlovací strategie BFS v přijatelném čase.

7.1 Použité datové struktury a algoritmy

7.1.1 Reprezentace struktury Webu

Jedno z prvních rozhodnutí, které jsem při návrhu musel udělat, byla volba reprezentace struktury internetu na disku. Každá stránka je reprezentovaná svým unikátním ID. Struktura 7.1 zachycuje původní reprezentaci záznamu pro stránku s daným ID, ze které vede k odkazů na stránky ID_1, ..., ID_k.

Struktura 7.1 Původní struktura odkazů

$$ID\ k\ ID_1\ ...\ ID_k$$

Pro stránku s k odkazy tedy obsahoval původní soubor jeden záznam velikosti $k + 2$.

Po poradě s vedoucím práce jsme došli k závěru, že tato reprezentace není pro účely práce příliš vhodná. Proto jsem zvolil reprezentaci struktury odkazů v souboru dvojic. Struktura 7.2 zachycuje použitou reprezentaci záznamu pro stránku s daným ID, ze které vede k odkazů na stránky ID_1, ..., ID_k.

Struktura 7.2 Struktura odkazů používaná při výpočtech

$$\begin{array}{c} ID\ ID_1 \\ ID\ ID_2 \\ ... \\ ID\ ID_k \end{array}$$

Každý odkaz je ve Struktuře 7.2 reprezentovaný záznamem o ID zdroje a ID odkazu. Znamená to přibližně zdvojnásobení velikosti struktury reprezentující internet. Konkrétně se změnil počet čísel v záznamu pro stránku s k odkazy z $k + 2$ na $2k$.

I když se struktura přechodem od reprezentace 7.1 k 7.2 zvětšila, bylo jednodušší se strukturou pracovat a třídit ji.

Například, pro implementaci měr důležitosti, je nutné číst matici Webu z definice 5.1 po řádcích. To vyžaduje znát ke každému ID identifikátory

těch stránek, které na stránku s ID odkazují. Původní struktura je uložena tak, že v řádcích jsou pro dané ID zapsány odkazy, které ze stránky s ID vedou. Je tedy nutné najít algoritmus pro transformaci mezi těmito dvěma strukturami odpovídající transpozici matice incidence grafu Webu.

Transpozice grafu Webu by v původní struktuře byl obtížný proces. V nové struktuře stačilo setřídit záznamy podle druhé položky z dvojice a prohodit význam sloupců. Třídění probíhá v čase $O(n \log n)$ s I/O náročností $O(n)$.

7.1.2 Třídění

Třídění bylo základním algoritmem při počítání se strukturou internetu. Třídění se uplatňuje při počítání PageRanku, K-Ranku, Kendallova τ, uspořádání podle BFS schématu i při dalších manipulacích se strukturou internetu. Použité algoritmy třídění tedy měly zásadní vliv na rychlost téměř všech prováděných výpočtů. Třídit bylo obvykle nutné celou strukturu Webu, tedy typicky 1 500 000 000 dvojic typu int.

Vnější třídění

Vnější Mergesort se ukázal jako vhodný algoritmus vnějšího třídění. Soubor internetu byl tříděn po blocích, velikosti přibližně 100 000 000 dvojic, které se vešly do hlavní paměti. Tyto bloky byly setříděny ve vnitřní paměti a zapsány na disk. Vnější Mergesort byl tedy spuštěn nad přibližně 15ti soubory. Protože postupné přetřiďování jednotlivých dvojic souborů, jako u vnitřního Mergesortu, by znamenalo významný nárůst počtu I/O operací, použil jsem haldu a do výsledného souboru tak byly slévány všechny utříděné bloky najednou.

Vnitřní třídění

Pro vnitřní třídění jsem používal obvykle Quicksort. Pro výpočet Kendallova τ však Knightův algoritmus využívá Mergesort. Proto bylo nutné implementovat také vnitřní verzi tohoto algoritmu. Podrobnější vysvětlení je v kapitole 6.

7.2 Implementace crawlovacích strategií

Implementoval jsem strategie BFS, OPIC a vševědoucí strategii se znalostí vektoru maximální důležitosti. Kvůli omezením daným tím, že v laboratoři proces po 24 hodinách ztrácí právo přístupu na disk, ale nebylo možné na daných datech dokončit očíslování vrcholů podle OPIC a vševědoucí strategie. Pro strategii BFS jsem naproti tomu napsal algoritmus, který na testovaných datech do 24 hodin vytvoří BFS očíslování (trvalo to přibližně 5 hodin).

7.2.1 BFS

I když je očíslování grafu podle BFS jedním z nejzákladnějších grafových algoritmů, překvapivě neexistuje mnoho prací, které by se zabývaly vytvářením BFS stromu pro data uložená na disku.

Externím BFS se sice zabývá [35], jedná se však pouze o neorientované grafy. Pro obecný orientovaný graf zde není algoritmus uveden. Navržené algoritmy jsou navíc velmi implementačně náročné. Algoritmus nejprve hledá takové subgrafy původního grafu, ve kterých jsou malé vzdálenosti mezi vrcholy. V dalším kroku spočítá algoritmus BFS na lokálních subgrafech a v poslední fázi se spočítá BFS celého souboru.

Externím BFS na orientovaných grafech se zabývá [10]. Autoři představují novou datovou strukturu "Buffered repository tree", s jejíž pomocí je možné očíslovat vrcholy podle vrstev BFS s použitím $O((V+E/B)log(V/B)+sort(E))$ I/O operací. Přitom E je počet hran grafu, V počet vrcholů a B velikost bloku, který lze načíst při jednom přístupu na disk.

Pro neorientované grafy lze očíslování podle vrstev převést na úplné očíslování podle BFS s pomocí $O(sort(E))$ operací viz [10].

Toto tvrzení jsem konstruktivně dokázal dále v textu i pro orientované grafy. Proto je na grafu Webu možné v první fázi vytvořit očíslování podle BFS vrstev a ve druhé fázi s pomocí $O(sort(E))$ operací vytvořit úplný BFS strom.

Graf stažené struktury Webu má oproti obecným grafům svá specifika, která lze použít při návrhu algoritmů. Primární crawlovací strategie se podobala BFS a hloubka grafu (počet unikátních vrstev) byla relativně malá.

Očíslování grafu podle vrstev BFS

Měl jsem tedy informace, které mi umožnily navrhnout algoritmus, který spotřebuje maximálně $O(L * E)$ operací read, kde L je počet vrstev grafu.

Tento algoritmus očíslování grafu podle vrstev se vyplatí oproti [10], pokud $log(\frac{V}{B}) > L$. To graf Webu typicky splňuje (neobsahuje dlouhá chapadla).

Algoritmus 7.3 Očíslování vrcholů grafu Webu podle vrstev BFS

```
1: var Vrstvy = new Vrstvy[MaxId]
2: var Layer = 0
3: var Component = 0
4: function BFSLayers(WebTup)                ▷ Vstupem je Struktura 7.2
5:     Sort(WebTup, 1, 2)
6:     WebTup.MakeSeekAble()
7:     while (var v = searchForNextComponent())!= -1 do
8:         BFSComponent(WebTup, v)
9:     end while
10:    return Vrstvy
11: end function
12: var neprectenaPolozkaVrstvy = 1
13: function searchForNextComponent()
14:    for var i=neprectenaPolozkaVrstvy; i!=maxID + 1; i++ do
15:        if layers[i] == 0 then
16:            neprectenaPolozkaVrstvy = i + 1
17:            return i
18:        end if
19:    end for
20:    return −1
21: end function
22: function BFSComponent(WebTup, v)
23:    Component++
24:    Vrstvy[v] = Layer + 1
25:    Layer = ComponentLayers(WebTup, v, v)
26: end function
```

Algoritmus 7.3 dostává jako vstupní data strukturu dvojic 7.2 odpovídající danému grafu. Tato struktura se setřídí podle první položky jako primárního a druhé položky jako sekundárního klíče. Funkce `WebTup.MakeSeekAble` vytvoří soubor offsetů na začátek záznamů o odkazech ID v setříděné struktuře 7.2. To umožňuje provádění operace `seek` v souboru dvojic. Každá operace `seek` však vyžaduje dvě čtení. Při operaci `seek` je nejdřív nutné přečíst informaci v indexovém souboru, kde v souboru dvojic začínají záznamy pro

Algoritmus 7.4 Očíslování vrcholů grafu Webu podle vrstev BFS

```
27: function ComponentLayers(WebTup, minVertex, maxVertex)
28:     Layer++
29:     var changes = 0
30:     maxLay = Layer
31:     minNext = MAXINT
32:     maxNext = MININT
33:     var tuple = newtuple[2] ▷ Uchovává záznam o položce struktury 7.2
34:     WebTup.seek(minVertex)
35:     while (tuple = WebTup.readNextTuple()) != null do
36:         if tuple[1] > maxVertex then
37:             break
38:         end if
39:         if layers[tuple[1]] < layer then
40:             WebTup.lazySeek(tuple[1], maxVertex)
41:             continue
42:         end if
43:         if Vrstvy[tuple[2]] == 0 or Vrstvy[tuple[2]] > Vrstvy[tuple[1]]+
    1 then
44:             Vrstvy[tuple[2]] = Vrstvy[tuple[1]] + 1
45:             changes++
46:             minNext = Min(minNext, tuple[2])
47:             maxNext = Max(maxNext, tuple[2])
48:         end if
49:         maxLay = Max(maxLay, Vrstvy[tuple[1]], Vrstvy[tuple[2]])
50:     end while
51:     if changes == 0 then
52:         Layer = maxLay
53:         return Layer
54:     else
55:         return ComponentLayers(WebTup, minNext, maxNext)
56:     end if
57: end function
```

Algoritmus 7.5 Očíslování vrcholů grafu Webu podle vrstev BFS

```
58: function WebTup.lazySeek(from, maxVertex)
59:     var seekTo = maxVertex
60:     for var i = from; i! = maxVertex; i++ do
61:         if layers[i] >= layer then
62:             seekTo = i
63:             break
64:         end if
65:     end for
66:     if (seekTo − actualID) > 9 then
67:         WebTup.seek(seekTo)
68:     end if
69: end function
```

dané ID a pak je nutné k daným záznamům přistoupit. To zvedá cenu za provedení operace `seek`, před klasickým sekvenčním čtením souboru dvojic. Z testů na reálných datech jsem zjistil, že operace `seek` se vyplatí, pokud přeskakuje více než 9 ID v souboru dvojic. Proto funkce `lazySeek` provede operaci `seek`, jen pokud je vzdálenost záznamů dostatečně velká.

Funkce `ComponentLayers` zajišťuje samotné vyhledání vrstev pro jednu komponentu. Jejím vstupem je i informace o minimálním a maximálním ID, mezi kterými má vyhledávat. Meze jsou zde proto, aby nemusely probíhat I/O operace v místech, kde jsou vrcholy buď finalizované nebo kde jsou vrcholy, které nepatří do stejné BFS komponenty grafu.

Funkce `ComponentLayers` prochází všechny vrcholy v zadaných mezích, jejichž vrstva v poli Vrstvy je alespoň Layer. Všechny vrcholy s vrstvou menší než Layer jsou totiž již finalizované. Funkce kontroluje, zda z vrstvy Layer a vyšších vedou nějaké odkazy na dosud neznámé vrcholy nebo zda lze snížit vrstvu u vrcholů, na které odkazují vrcholy nižších vrstev.

Pokud při průchodu nebyl objeven žádný nový vrchol a nebylo provedené jediné snížení vrstvy, je komponenta očíslovaná podle BFS vrstev. Díky tomu lze v praxi redukovat počet průchodů až o polovinu oproti situaci, kdy bychom vyžadovali jeden průchod pro každou BFS vrstvu. Nastavení mezí zajišťuje, že tato heuristika je korektní.

Pro další komponentu proces opakujeme. Čítač *Layer* je přitom před zavoláním `ComponentLayers` nastaven na číslo nejvyšší známé vrstvy v předchozí komponentě nebo na 0, pokud se jedná o první volání funkce.

V poli $Vrstvy$, které vrací funkce `BFSLayers` je zapsáno očíslování vrcholů podle BFS vrstev.

Přechod od vrstev k očíslování

V práci [10] je dokázáno, že pro neorientovaný graf lze s použitím $O(sort(E))$ I/O operací, převést BFS čísla vrstev na BFS očíslování. Důkaz tohoto tvrzení pro orientované grafy je proveden dále v této práci.

Nejprve je nutné upravit soubor dvojic tak, aby v něm zůstaly jen ty hrany, které tvoří tranzitivní hrany BFS očíslování. Tedy ty, které vedou z vrstvy k do vrstvy $k+1$. Na základě znalostí očíslování vrstev lze toto provést v jednom průchodu strukturou internetu, tedy s použitím $O(E)$ I/O operací. Pro samotný algoritmus očíslování BFS zapíšeme údaje o tranzitivních hranách BFS do struktury 7.6.

Struktura 7.6 Struktura tranzitivních hran BFS pro algoritmus očíslování

ID_{zdroj} ID_{odkaz} VRSTVA$_{zdroj}$

Algoritmus 7.7 na základě informací ve struktuře 7.6 převádí vrstvy BFS na BFS očíslování. Nejprve setřídí strukturu 7.6 primárně podle vrstvy, sekundárně podle ID zdroje a ID odkazu. V každém kroku očíslujeme jednu vrstvu. V k-tém kroku očíslujeme vrstvu $k+1$. Očíslování BFS je pro vrstvu k již známo s výjimkou případu, kdy se jedná o začátek nové komponenty a tedy jednoprvkovou vrstvu. V tomto případě je prvku vrstvy přiřazeno nejnižší volné ID.

Očíslování vrstvy probíhá ve dvou krocích. Nejprve jsou načteny všechny dvojice odkazů jedné vrstvy. Dvojice jsou zapsány do dočasného souboru a to tak, že první položce je přiřazeno její známé BFS číslo. Druhá položka má původní ID. Vrstva je následně setříděná podle první položky. Z toho, že první položky jsou očíslovány podle BFS lze odvodit, že první výskyty druhých položek v setříděných záznamech odpovídají jejich BFS pořadí. Proto stačí namapovat postupně těmto položkám nejnižší nepoužitá ID.

Tímto způsobem lze z údajů o vrstvách BFS očíslovat celý graf pomocí BFS s použitím $O(sort(E))$ I/O operací. Z podstaty vnějšího Mergesortu $O(sort(E)) = O(E)$ I/O operací.

V implementaci bylo nutné šetřit pamětí. Proto jsem používal stejné pole, jak pro třídění, tak pro přečíslovávání podle BFS. Trik je v tom, že údaje o

Algoritmus 7.7 Převod BFS očíslování podle vrstev na BFS očíslování

```
1: function BFS(Triples)
2:     Sort(Triples, 3, 1, 2)
3:     var triple[]                    ▷ Ukládá jeden záznam ze struktury 7.6
4:     var mapping[] = newmapping[MAXID]
5:     var newID = 0
6:     var BFSNumbersFile
7:     triple = Triples.ReadNextTriple()
8:     while triple != null do
9:         var Layer = triple[3]
10:        var FileToSort
11:        while Layer == triple[3] do
12:            FileToSort.Write(Mapping(triple[1]))
13:            FileToSort.Write(triple[2])
14:            triple = Triples.ReadNextTriple()
15:        end while
16:        Sort(FileToSort, 1, 2)
17:        while !FileToSort.EOF() do
18:            FileToSort.Read()
19:            Map(FileToSort.Read())        ▷ Namapuje druhou položku z
   dvojice.
20:        end while
21:    end while
22: end function
23: function Mapping(vrchol)
24:    if mapping[vrchol] == 0 then
25:        Map(vrchol)
26:    end if
27:    return mapping[vrchol]
28: end function
29: function Map(vrchol)
30:    if mapping[vrchol] == 0 then
31:        newID++
32:        mapping[vrchol] = newID
33:        BFSNumbersFile.Write(vrchol, newID)
34:    end if
35: end function
```

očíslování k-té vrstvy jsou třeba vždy před setříděním souboru položek. Po setřídění lze do pole použitého pro třídění zapisovat mapování $k+1$ vrstvy.

7.3 Implementace měr důležitosti

7.3.1 PageRank

Soustava rovnic pro PageRank

PageRank je podle definice 5.6 vlastní vektor matice $\widehat{W}$ příslušející vlastnímu číslu 1. Práce [18] dává návod, jak výpočet PageRanku transformovat na řešení soustavy rovnic s řídkou maticí.

Rovnost 5.3 v definici 5.6

$$((1-e)(W+vo^T)+evj^T)z = z \tag{7.1}$$

vede na systém rovnic

$$ev = (I-(1-e)(W+vo^T))z \tag{7.2}$$

přitom využíváme, že pro vektor z platí

$$\sum_i z_i = 1 \tag{7.3}$$

a tedy

$$j^T z = 1 \tag{7.4}$$

Problémem zůstává, že

$$S = (I-(1-e)(W+vo^T)) \tag{7.5}$$

je hustá matice. Soustavu

$$Sz = ev \tag{7.6}$$

je třeba transformovat tak, aby výsledný algoritmus počítal s řídkou maticí a jeho výsledek byl uspořádaný stejně jako vlastní vektor z matice $\widehat{W}$. Je zřejmé, že u vektoru PageRanku záleží spíš na uspořádání jednotlivých hodnot vůči sobě, než na konkrétních hodnotách.

Autoři [18] ukazují, že vektor z lze získat jako $z = y/||y||_1$, kde y řeší soustavu

$$Ry = v \tag{7.7}$$

kde

$$R = I - (1 - e)W \tag{7.8}$$

Důkaz využívá Sherman-Morissonovu formuli pro počítání inverzní matice k

$$S = R - (1 - e)(vo^T) \tag{7.9}$$

$$S^{-1} = (R - (1 - e)(vo^T))^{-1} = R^{-1} + \frac{R^{-1}vo^TR^{-1}}{1/(1 - e) - o^TR^{-1}v} \tag{7.10}$$

Vektor z nyní vyjádříme jako $z = eS^{-1}v$ a využijeme $y = R^{-1}v$. Pak

$$z = e(1 + \frac{yo^T}{1/(1 - e) - o^Ty})y = \gamma y \tag{7.11}$$

Našli jsme tedy konstantu γ, pro kterou platí $z = \gamma y$.

Je vidět, že matice R je řídká a vektor y je uspořádaný stejně jako vektor z.

Implementace PageRanku

Pro samotnou implementaci bylo nutné nejprve zjistit pro každé ID počet odkazů, které z vrcholu s ID vedou. K tomu stačí setřídit strukturu 7.2 podle první položky a jednou sekvenčně projít. Počet výskytů každého ID v prvním sloupci pak odpovídá počtu odkazů, které z vrcholu ID vedou.

Pro samotný výpočet soustavy 7.7 je ke každému ID třeba znát ID stránek, které na danou stránku odkazují. Proto bylo nutné setřídit strukturu dvojic reprezentující internet primárně podle druhé a sekundárně podle první položky. Dále pak bylo potřebné implementovat odpovídající reprezentaci matice R.

Pro výpočet soustavy jsem použil Gauss-Seidelovu metodu. Aby správně fungovala, je třeba matici R procházet po jednotlivých řádcích. Nulové položky bylo třeba přeskakovat, aby bylo využito toho, že matice je řídká. K tomu jsem použil dva objekty, jeden prováděl samotný výpočet Gauss-Seidelovy metody a druhý poskytoval informace o matici R. Objekt zastřešující matici R má funkci `moveAhead`. Tato funkce posune ukazatel na další nenulovou položku matice R. Další funkce jsou `Ai` a `Aj`, které určují jaký je aktuální prvek (i, j) matice a funkce `value`, určující hodnotu prvku R_{ij}. Tyto funkce volá solver Gauss-Seidelovy metody. Z podstaty metody bylo

zásadní, aby posloupnost hodnot funkce `Ai`, po zavolání funkce `moveAhead`, byla neklesající v průběhu celého výpočtu. To zaručovalo korektnost Gauss-Seidelovy metody. Bylo tedy nutné ošetřit situace, kdy má být hodnota rovna 1, kvůli jednotkové matici v rovnosti 7.8 určující R. Pro jeden průchod Gauss-Seidelovy metody stačí jednou sekvenčně přečíst soubor struktury Webu, setříděný podle druhé položky. Vektor v byl uniformně nastaven $v_i = 1 \forall i$. Místo $Ry = v$ tak byla ve skutečnosti řešena soustava $Ry^* = nv$. Snadno však lze nahlédnout, že $y^* = ny$.

PageRank na parciální množině vrcholů

Pro testy zachování uspořádanosti vektoru PageRanku v průběhu výpočtu bylo nutné mít možnost vypočítat PageRank na nějaké menší množině vrcholů (například na 75% vrcholů objevených jako první strategií BFS). Možným přístupem by bylo projít celou strukturu Webu a postupně z ní odstranit všechna neznámá ID stojící na zdrojové straně. Tento přístup by byl ale velmi I/O nákladný a znamenal by nutnost pro každý vektor PageRanku (dlouhý přibližně 200 000 000 čísel int) vytvořit mnohonásobně delší unikátní strukturu. Počítače v laboratoři disponují disky velkými asi 100GB. Byla by to tedy jejich další velká datová zátěž, která by mohla vyvolat spuštění nepopulárních uklízecích algoritmů.

Pro počítání PageRanku na parciální množině vrcholů jsem se proto rozhodnul použít původní strukturu Webu a dodatečnou informaci o tom, jaké vrcholy jsou známé.

Počty odkazů vedoucích z jednotlivých vrcholů se při parciálním výpočtu nemění. Počítám totiž i s neznámými listovými vrcholy. Při procházení transponované struktury Webu tak stačí objektu reprezentujícímu matici R předat bitové pole známých vrcholů.

Funkci `MoveAhead`, pro matici 7.8 zužující výpočet na pole známých vrcholů, popisuje Algoritmus 7.9. Po načtení dvojice (zdroj, odkaz), ze struktury setříděné podle položky odkaz, stačí ověřit, zda má zdroj známé ID. Pokud ano, odkazované ID je pro výpočet relevantní a v matici R se objeví i položka $R_{(ID,ID)} = 1$ kvůli jednotkové matici v rovnosti 7.8. Pokud není zdrojové ID v poli známých vrcholů, daná položka se zahazuje a záznam o odkazu se dál nezpracovává. ID je zcela neznámé a nevytvoří se pro něj ani položka $R_{(ID,ID)} = 1$, pokud na něj neexistuje žádný odkaz ze známého zdrojového ID. Vektor y z rovnice 7.7 je inicializovaný $y_i = 0 \forall i$ a pro zcela neznámá ID tak prvek tohoto vektoru na místě ID zůstane nulový. Gauss-

Algoritmus 7.8 Funkce MoveAhead pro zúženou matici 7.8

```
1: boolean outofbound = false
2: boolean valueinbuffer = false
3: var Tuples                                          ▷ Struktura internetu
4: var to, from, value          ▷ Hodnoty čte solver Gauss-Seidelovy metody
5: var lastto, lastfrom, dumpingFactor
6: function MoveAhead(zcelaZnameVrcholy)               ▷ Funkce dostane
   bitové pole známých vrcholů
7:     if outofbound then
8:         throw EOF
9:     end if
10:    if valueinbuffer then
11:        to = lastto
12:        from = lastfrom
13:        value = -dumpingFactor / delky[from]
14:        valueinbuffer = false
15:    else
16:        Try                                         ▷ Zachytí přečtení EOF
17:            lastto = to
18:            lastfrom = from
19:            while true do
20:                from = TUPLES.READ()
21:                to = TUPLES.READ()
22:                if ZCELAZNAMEVRCHOLY.GET(from) then
23:                    break
24:                end if
25:            end while
26:            value = -dumping / delky[from]
27:            if from!=lastfrom and lastfrom != null then
28:                valueinbuffer = true
29:                var pom
30:                pom = to
31:                to = lastto
32:                lastto = pom
33:                lastfrom = from
34:                from = to
35:                value = 1
36:            end if
```

Algoritmus 7.9 Funkce MoveAhead pro zúženou matici 7.8

```
37:         Catch tuples.EOF
38:             value = 1
39:             to = lastto
40:             from = lastto
41:             outofbound = true
42:         EndTry
43:     end if
44: end function
```

Seidelova metoda tedy spočítá vektor y pro všechna ID, která jsou v bitovém poli a pro ta listová ID, na která ID z bitového pole přímo odkazují.

7.3.2 K-Rank

Podle definice 5.9 je K-Rank PageRank spočítaný na struktuře, kde pro každý vrchol necháme K odkazů s nejvyšším skóre.

Nejprve je tedy třeba spočítat skóre pro každý vrchol. To odpovídá PageRanku dělenému počtem odkazů. Pak algoritmus vytvoří ze struktury Webu strukturu trojic 7.10.

Struktura 7.10 Struktura trojic pro sestavení matice Webu pro K-Rank

$$ID_{zdroj}\ ID_{odkaz}\ \text{SKÓRE}_{zdroj}$$

Tuto strukturu je pak nutné setřídit primárně podle ID odkazu a sekundárně podle skóre zdroje. Pro každé ID odkazu pak stačí vzít prvních K záznamů, pokud existují, a zapsat je do nové struktury.

Nad novou strukturou se spočítá PageRank podle algoritmu popsaného v předchozí sekci.

Parametr K pro K-Rank jsem volil jako 100. Řazení stránek při nastavení této hodnoty se zdálo být správné. Přesto posouzení toho, jaký vliv má K na stabilitu K-Ranku a jak uživatelsky kvalitní výsledky toto řazení poskytuje, by vyžadovalo další studie nebo nasazení této metriky v nějakém užívaném vyhledávači.

Kapitola 8

Výpočty a výsledky

8.1 Použitá data

Výpočty jsem prováděl na datech, které mi poskytl vedoucí diplomové práce RNDr. Leo Galamboš, Ph.D. Jedná se o tři soubory dat nacrawlované v letech 2006 a 2007.

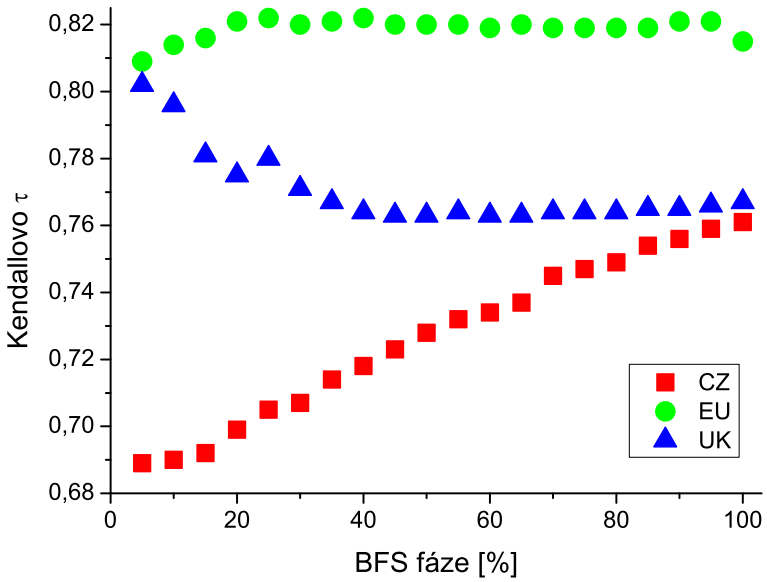
První soubor jsou data z domény .cz. Ve struktuře jsou záznamy o 195 681 681 unikátních URL. Ty jsou propojeny 2 901 617 078 odkazy. K tomuto souboru se dále budu odkazovat jako k souboru cz.

Druhý soubor představují data z domény .uk. Struktura obsahuje záznamy o 87 847 904 unikátních URL. Jsou propojeny 1 025 694 832 odkazy. K tomuto souboru se dále budu odkazovat jako k souboru uk.

Třetí soubor jsou data stažená ze serverů evropských univerzit. Obsahují 179 644 175 záznamů o unikátních URL a ty jsou propojeny 2 028 971 914 odkazy. Tato data budu označovat jako soubor eu.

Při výpočtech jsem se rozhodl orientovat na stabilitu metriky K-Ranku a na crawlovací strategii BFS. Při všech výpočtech zahrnujících počítání měr důležitosti jako PageRank nebo K-Rank byly použity algoritmy, vycházející z [18], které ohodnocují i listové vrcholy.

V některých tabulkách nebo grafech bylo nutné zkracovat názvy jednotlivých měr důležitosti. Pro K-Rank tedy používám zkratku KR a pro PageRank zkratku PR. Všechny výpočty, které jsem provedl a při kterých jsem počítal s K-Rankem, byly provedeny s nastavením $K = 100$. Tato volba se zdá být vhodná. Přesto stanovení optimálního K může být předmětem další diskuze.



Graf 8.1: Korelace PageRanku a K-Ranku v různých fázích BFS

8.2 Korelace PageRanku a K-Ranku

První výpočty byly zaměřeny na vzájemnou korelaci PageRanku a K-Ranku podle Kendallova τ v různých fázích crawlovacího cyklu. Protože K-Rank vychází z PageRanku, bylo by přirozené, kdyby mezi nimi korelace byla.

Pro crawlovací strategii BFS jsem postupně spočítal PageRank a K-Rank na 5-100% vrcholů setříděných podle toho, jak je postupně objevuje strategie BFS. Jak bylo zmíněno, při všech výpočtech měr důležitosti jsem bral v potaz i listové vrcholy. Důležitost počítaná na k% vrcholů znamená, že důležitost byla spočítaná na struktuře, kde na straně zdroje stojí pouze prvních k% vrcholů celé struktury Webu setříděné podle BFS. Na pozici ohodnocených listových vrcholů tak mohou být i ty vrcholy, na které vede z upravené struktury odkaz. Listové vrcholy v souboru k% mohou být i ty, které v originální struktuře odkazují na další stránky, ale tyto informace v dané fázi BFS strategie nezná.

V Tabulce 8.1 a Grafu 8.1 je uvedeno jak se mění Kendallovo τ v různých fázích crawlování.

Na první pohled je patrné, že obě metriky jsou korelované. Kendallovo τ se pohybuje v pásu od 0.68 do 0.82.

Fáze BFS	Kendallovo τ		
%	cz	eu	uk
5	0.689	0.809	0.802
10	0.690	0.814	0.796
15	0.692	0.816	0.781
20	0.699	0.821	0.775
25	0.705	0.822	0.780
30	0.707	0.820	0.771
35	0.714	0.821	0.767
40	0.718	0.822	0.764
45	0.723	0.820	0.763
50	0.728	0.820	0.763
55	0.732	0.820	0.764
60	0.734	0.819	0.763
65	0.737	0.820	0.763
70	0.745	0.819	0.764
75	0.747	0.819	0.764
80	0.749	0.819	0.764
85	0.754	0.819	0.765
90	0.756	0.821	0.765
95	0.759	0.821	0.766
100	0.761	0.815	0.767

Tabulka 8.1: Korelace PageRanku a K-Ranku v různých fázích BFS

Zajímavé je, že každý soubor má jiný trend vývoje Kendallova τ s rostoucím počtem vrcholů podle postupu BFS.

Nejlépe korelované jsou obě metriky na souboru dat evropských univerzit. Může to mít několik důvodů. K-Rank je počítán pro $K = 100$. Pokud většina stránek souboru tedy bude mít méně než 100 příchozích odkazů, budou K-Rank a PageRank konvergovat ke stejnému uspořádání. Akademické stránky mají rovnoměrněji rozložené odkazy. Navíc byla kolekce sbírána na webech několika institucí, v tomto případě univerzit. Každý web byl prolinkovaný sám o sobě a měl méně odkazů na ostatní weby. To vede právě k rovnoměrné distribuci odkazů, kde neexistuje mnoho stránek, které by byly velmi často odkazovány ze všech částí kolekce.

Oproti souboru eu je pro soubor uk korelace PageRanku a K-Ranku o něco menší. Díváme-li se pouze na část přibližně po 20 % strategie BFS zjistíme, že korelace je poměrně stabilní. Naproti tomu korelace pro doménu cz s postupujícím BFS významně roste. Oproti doméně cz je doména uk o mnoho větší. Navíc u domény uk bylo staženo o polovinu méně dat. U domény cz tak pravděpodobně s postupujícím crawlováním došlo k vyčerpání jejího obsahu. To mělo za následek stahování stránek, které nemají velký počet příchozích odkazů a větší korelaci obou metrik. U domény uk k podobnému vyčerpání nedošlo a proto byla pro soubor uk korelace PageRanku a K-Ranku stabilnější.

Pokud odhlédneme od trendů, zjistíme, že nejlépe korelované jsou stránky souboru eu následované souborem uk a stránky s nejhorší korelací jsou v doméně cz. Argumenty, proč je korelace větší u souboru eu, jsem uváděl v předchozích odstavcích a souvisí se speciálním charakterem této kolekce. Rozdíly v korelaci mezi doménami cz a uk mohou vypovídat o tom, že v doméně cz existují větší rozdíly mezi důležitými a nedůležitými stránkami seřazenými podle PageRanku než u domény uk. To potvrzuje i agregace měr důležitosti pro soubor uk, viz Graf 8.10. Ukazuje se totiž, že pro stránky z domény uk je distribuce PageRanku i K-Ranku nejrovnoměrnější ze všech zkoumaných kolekcí.

8.3 Uspořádání měr důležitosti v průběhu BFS

Strategie parciální důležitosti, viz kapitola 4.4, počítají opakovaně v průběhu crawlování důležitost na množině stažených stránek a podle té řadí stránky k dalšímu stahování. Vycházejí přitom z předpokladu, že uspořádání důležitosti se v průběhu crawlování příliš nemění.

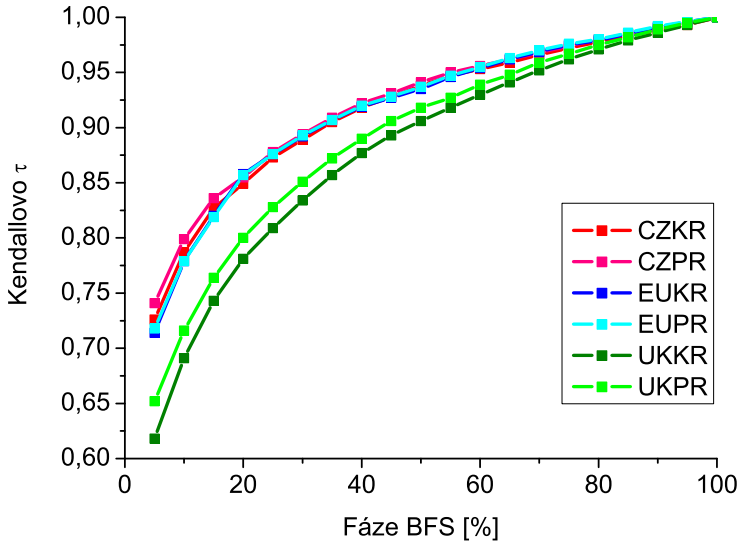
	Kendallovo τ					
Fáze BFS	CZ		EU		UK	
%	KR	PR	KR	PR	KR	PR
5	0.726	0.741	0.714	0.718	0.618	0.652
10	0.787	0.799	0.779	0.779	0.691	0.716
15	0.827	0.836	0.820	0.819	0.743	0.764
20	0.849	0.855	0.858	0.857	0.781	0.800
25	0.873	0.878	0.876	0.876	0.809	0.828
30	0.889	0.894	0.892	0.893	0.834	0.851
35	0.905	0.909	0.907	0.907	0.857	0.872
40	0.918	0.922	0.919	0.920	0.877	0.890
45	0.928	0.931	0.927	0.928	0.893	0.906
50	0.938	0.941	0.935	0.937	0.906	0.918
55	0.947	0.950	0.946	0.947	0.918	0.927
60	0.953	0.956	0.954	0.955	0.930	0.939
65	0.959	0.962	0.962	0.963	0.941	0.948
70	0.966	0.968	0.969	0.970	0.952	0.959
75	0.972	0.974	0.975	0.976	0.962	0.967
80	0.978	0.980	0.979	0.980	0.971	0.975
85	0.983	0.985	0.985	0.986	0.979	0.982
90	0.988	0.990	0.991	0.992	0.986	0.989
95	0.993	0.994	0.996	0.996	0.993	0.995
100	1.000	1.000	1.000	1.000	1.000	1.000

Tabulka 8.2: Stabilita měr důležitosti v různých fázích BFS

Vyhledávače vracejí výsledky setříděné podle míry důležitosti spočítané na kolekci stažených stránek. Tato kolekce je obrazem Webu a čeká se, že uspořádání stránek podle konkrétní metriky bude blízké uspořádání, které bychom získali spočítáním dané míry na všech stránkách celého internetu nebo té jeho části, která nás zajímá (například jedna doména).

Z těchto důvodů je důležité zkoumat, jak se liší uspořádání indukované příslušnou metrikou v průběhu crawlování. Další testy, které jsem provedl se týkaly toho, jak crawlovací strategie BFS udržuje uspořádání podle PageRanku a podle K-Ranku pro $K = 100$.

Z Tabulky 8.2 a Grafu 8.2 je vidět, že Kendallovo τ pro strategii BFS konverguje relativně rychle, jak pro PageRank, tak pro K-Rank.



Graf 8.2: Stabilita měr důležitosti v různých fázích BFS

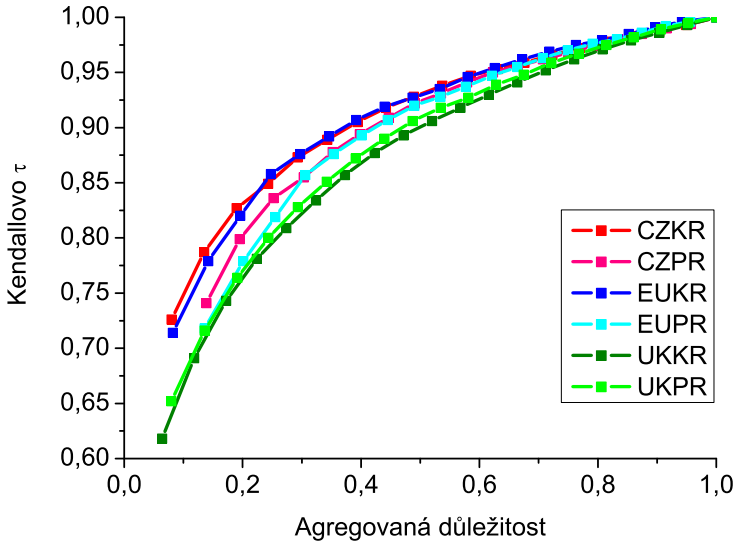
Přesto pro K-Rank se konvergence zdá být o něco pomalejší než pro PageRank. To se projevuje především pro soubor uk. Naopak pro univerzitní weby byla rychlost konvergence K-Ranku v některých fázích dokonce nepatrně rychlejší, než u PageRanku.

U univerzitních webů dochází k velmi zajímavému jevu, kdy rychlost konvergence je téměř stejná pro K-Rank i PageRank. Opět se tedy potvrzuje, uspořádání podle PageRanku a K-Ranku jsou si pro soubor eu velmi blízká.

Jak je patrné z dalších měření, pomalejší konvergence K-Ranku ve většině případů je dána tím, že K-Rank distribuuje důležitost rovnoměrněji přes všechny stránky. Její agregace je tak pro strategii BFS, ale i pro prosté sečtení nejvyšších důležitostí na celém souboru pomalejší pro K-Rank než pro PageRank. Menší podíl na stažené důležitosti pak způsobuje větší fluktuace uspořádání.

Graf 8.3 zobrazuje, jak konverguje uspořádání podle důležitosti měřené ne počtem stránek stažených pomocí BFS, ale podílem na celkové stažené důležitosti v průběhu BFS. Toto srovnání ukazuje asi největší naměřený klad nové metriky K-Rank, rychlou konvergenci uspořádání vzhledem k podílu stažené důležitosti, jejíž rychlost překonává PageRank.

Jedinou výjimkou je soubor uk. Pro tento soubor jsou rychlosti konvergence K-Ranku a PageRanku vzhledem k agregované důležitosti velmi blízké.



Graf 8.3: Konvergence uspořádání podle staženého podílu důležitosti

PageRank přitom K-Rank mírně překonává. Je to dáno tím, že na souboru uk je PageRank rozložen mnohem rovnoměrněji než na ostatních souborech a proto je rychlost agregace PageRanku i K-Ranku velmi podobná, viz Graf 8.10. K-Rank tu tedy ztrácí výhodu rovnoměrnějšího rozložení důležitosti.

Protože jsem výpočty o rychlosti konvergence uspořádání prováděl na datech o celých souborech, která mohla být zatížena chybou, vnesenou primární crawlovací strategií, rozhodl jsem se provést podobné měření na prvních 50% stránek podle BFS strategie a výsledky jsem zpracoval do Tabulky 8.3 a Grafu 8.4.

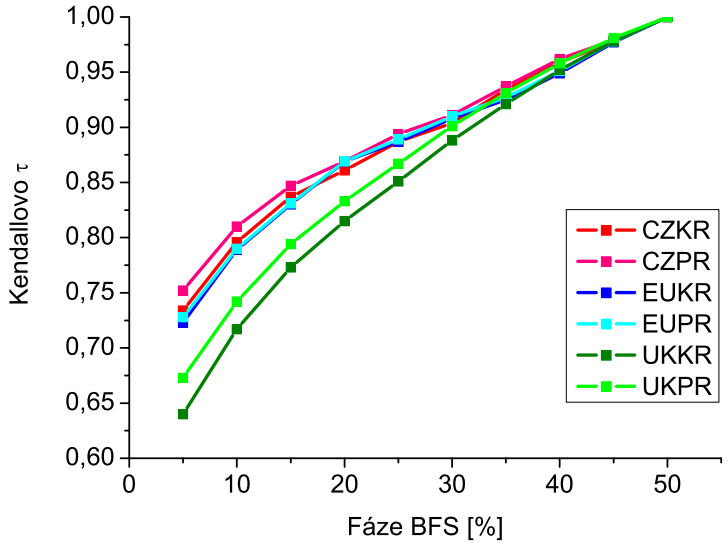
Opět pro soubor eu jsou konvergence uspořádání K-Ranku i PageRanku velmi podobné a podobně jako pro úplné BFS je u ostatních souborů konvergence K-Ranku vůči počtu stažených stránek o něco pomalejší.

Podobně jako v případě úplného BFS procházení jsem sestavil ještě graf rychlosti konvergence v závislosti na celkovém podílu stažené důležitosti 8.5.

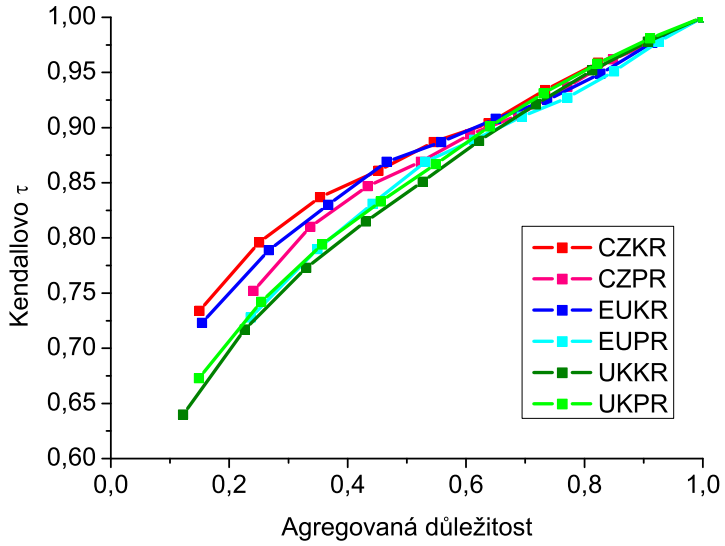
Je vidět, že výsledky jsou opět podobné jako pro úplné BFS procházení. K-Rank si u souborů cz a eu vede lépe než PageRank. Naopak u souboru uk PageRank konverguje rychleji než K-Rank. Rozdíly jsou ale minimální a obě míry se v případě souboru uk neliší o víc než o půl procenta. Je to dáno především tím, že rychlosti agregace K-Ranku a PageRanku jsou pro soubor uk velmi podobné.

	Kendallovo τ					
Fáze BFS	CZ		EU		UK	
%	KR	PR	KR	PR	KR	PR
5	0.734	0.752	0.723	0.728	0.640	0.673
10	0.796	0.810	0.789	0.790	0.717	0.742
15	0.837	0.847	0.830	0.831	0.773	0.794
20	0.861	0.869	0.869	0.869	0.815	0.833
25	0.887	0.894	0.887	0.889	0.851	0.867
30	0.904	0.911	0.908	0.910	0.888	0.901
35	0.934	0.937	0.925	0.927	0.921	0.931
40	0.959	0.962	0.949	0.951	0.952	0.958
45	0.978	0.979	0.977	0.978	0.978	0.981
50	1.000	1.000	1.000	1.000	1.000	1.000

Tabulka 8.3: Stabilita měr důležitosti v různých fázích pro 50 % BFS



Graf 8.4: Stabilita měr důležitosti v různých fázích pro 50 % BFS



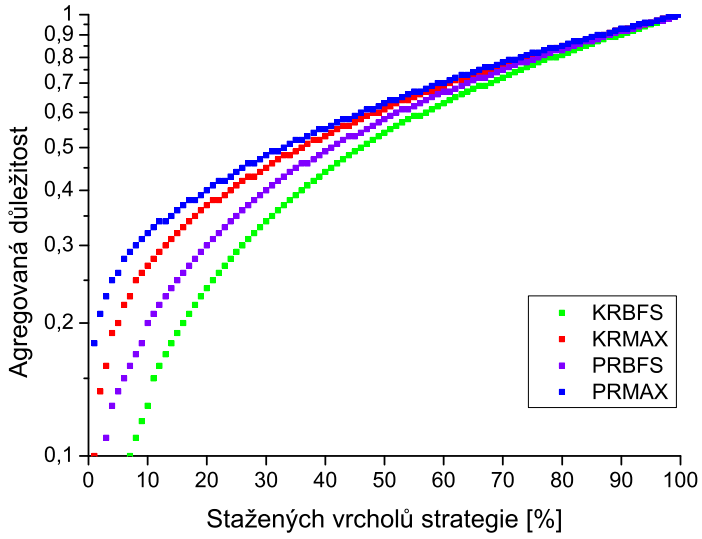
Graf 8.5: Konvergence uspořádání podle staženého podílu důležitosti pro 50 % BFS

I když všechny závěry, které platily pro soubor BFS 100 %, zůstaly v platnosti i pro soubor BFS 50 %, je vidět, že Graf 8.4 je vůči grafu 8.2 mírně deformovaný. Ke stejnému závěru dojdeme i porovnáním grafu 8.5 Kendallova τ v závislosti na agregované důležitosti pro BFS 50 % s grafem 8.3 pro BFS 100 %. Vliv primární crawlovací strategie tedy, i když je malý, není zanedbatelný.

8.4 Rychlost akumulace důležitosti pro BFS

Dále jsem se zaměřil na to, jak rychle sbírá BFS strategie PageRank a K-Rank. Porovnávám to s akumulací těchto měr důležitosti na celém Webu, která nerespektuje jeho grafovou strukturu a bere vždy vrchol s největší globální důležitostí.

Proto, abych odstranil případné vlivy primární crawlovací strategie na strategii sekundární, tedy BFS, jsem akumulaci důležitosti počítal pro K-Rank i PageRank, jak pro plně provedenou strategii BFS na analyzovaných datech, tak pro strategii BFS provedenou na polovině vrcholů z daného souboru. K této strategii se odkazuji jako k BFS 50 %.



Graf 8.6: Agregace strategie BFS a maximálních důležitostí pro 100% vrcholů cz

Pro vynášení výsledků jsem pro agregovanou důležitost použil logaritmickou škálu, protože jsou tak lépe patrné trendy vývoje grafu v prvních desítkách procent strategie.

8.4.1 Data cz

Data z domény cz byla zpracována do tabulek 8.4 , 8.5 a grafů 8.6, 8.7. Ukazuje se, že K-Rank má rovnoměrnější distribuci důležitosti než PageRank. Proto také BFS strategie sbírá rychleji celkový PageRank než K-Rank.

Na rozdíl od dříve publikovaných výsledků se pro ohodnocování důležitostí i na listových vrcholech nepotvrzuje trend extrémně rychlé konvergence PageRanku k vysokým hodnotám už pro první desítky procent nejlépe ohodnocených vrcholů. Míry důležitosti s ohodnocením listových vrcholů jsou tak distribuovány mnohem rovnoměrněji než bez něj a mohou proto poskytovat lepší výsledky.

Vliv primární strategie na výsledky není při měření agregace téměř vůbec postřehnutelný. Jak strategie BFS, tak prostá agregace obou důležitostí, mají téměř shodný vývoj na 100 % i na 50 % stránek.

Fáze	KR		PR	
%	BFS	MAX	BFS	MAX
0	0.00	0.00	0.00	0.00
1	0.02	0.10	0.07	0.18
2	0.04	0.14	0.09	0.21
3	0.06	0.16	0.11	0.23
4	0.07	0.19	0.13	0.25
5	0.08	0.20	0.14	0.26
6	0.09	0.22	0.15	0.28
7	0.10	0.23	0.16	0.29
8	0.11	0.25	0.17	0.30
9	0.12	0.26	0.18	0.31
10	0.13	0.27	0.20	0.32
11	0.15	0.28	0.21	0.33
12	0.16	0.29	0.22	0.34
13	0.17	0.30	0.23	0.34
14	0.18	0.31	0.24	0.35
15	0.19	0.32	0.25	0.36
16	0.20	0.33	0.26	0.37
17	0.21	0.34	0.27	0.38
18	0.22	0.35	0.28	0.38
19	0.23	0.36	0.29	0.39
20	0.24	0.37	0.30	0.40
21	0.25	0.38	0.31	0.41
22	0.26	0.38	0.32	0.42
23	0.27	0.39	0.33	0.42
24	0.28	0.40	0.34	0.43
25	0.29	0.41	0.35	0.44
26	0.30	0.42	0.36	0.45
27	0.31	0.43	0.37	0.46
28	0.32	0.43	0.38	0.46
29	0.33	0.44	0.39	0.47
30	0.34	0.45	0.40	0.48
31	0.35	0.46	0.41	0.49
32	0.36	0.47	0.42	0.49
33	0.37	0.48	0.43	0.50
34	0.38	0.48	0.44	0.51
35	0.39	0.49	0.45	0.52
36	0.40	0.50	0.46	0.52
37	0.41	0.51	0.46	0.53
38	0.42	0.52	0.47	0.54
39	0.43	0.52	0.48	0.55
40	0.44	0.53	0.49	0.55
41	0.45	0.54	0.50	0.56
42	0.46	0.55	0.51	0.57
43	0.47	0.56	0.52	0.58
44	0.48	0.56	0.53	0.58
45	0.49	0.57	0.53	0.59
46	0.50	0.58	0.54	0.60
47	0.51	0.59	0.55	0.61
48	0.52	0.60	0.56	0.61
49	0.53	0.60	0.57	0.62
50	0.54	0.61	0.58	0.63

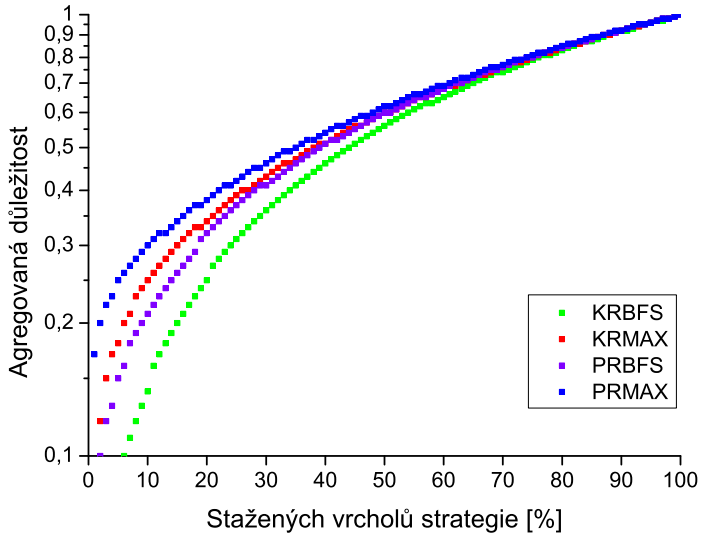
Fáze	KR		PR	
%	BFS	MAX	BFS	MAX
51	0.55	0.62	0.59	0.64
52	0.56	0.63	0.60	0.64
53	0.57	0.64	0.61	0.65
54	0.58	0.64	0.61	0.66
55	0.59	0.65	0.62	0.67
56	0.59	0.66	0.63	0.67
57	0.60	0.67	0.64	0.68
58	0.61	0.67	0.65	0.69
59	0.62	0.68	0.66	0.70
60	0.63	0.69	0.67	0.70
61	0.64	0.70	0.67	0.71
62	0.65	0.71	0.68	0.72
63	0.66	0.71	0.69	0.73
64	0.67	0.72	0.70	0.73
65	0.68	0.73	0.71	0.74
66	0.69	0.74	0.72	0.75
67	0.69	0.74	0.72	0.76
68	0.70	0.75	0.73	0.76
69	0.71	0.76	0.74	0.77
70	0.72	0.77	0.75	0.78
71	0.73	0.78	0.76	0.79
72	0.74	0.78	0.77	0.79
73	0.75	0.79	0.78	0.80
74	0.76	0.80	0.78	0.81
75	0.77	0.81	0.79	0.82
76	0.78	0.81	0.80	0.82
77	0.79	0.82	0.81	0.83
78	0.80	0.83	0.82	0.84
79	0.80	0.84	0.83	0.84
80	0.81	0.85	0.83	0.85
81	0.82	0.85	0.84	0.86
82	0.83	0.86	0.85	0.87
83	0.84	0.87	0.86	0.87
84	0.85	0.88	0.87	0.88
85	0.86	0.88	0.87	0.89
86	0.87	0.89	0.88	0.90
87	0.88	0.90	0.89	0.90
88	0.89	0.91	0.90	0.91
89	0.89	0.92	0.91	0.92
90	0.90	0.92	0.92	0.93
91	0.91	0.93	0.92	0.93
92	0.92	0.94	0.93	0.94
93	0.93	0.95	0.94	0.95
94	0.94	0.95	0.95	0.96
95	0.95	0.96	0.96	0.96
96	0.96	0.97	0.97	0.97
97	0.97	0.98	0.97	0.98
98	0.98	0.98	0.98	0.99
99	0.99	0.99	0.99	0.99
100	1.00	1.00	1.00	1.00

Tabulka 8.4: Agregace strategie BFS a maximálních důležitostí pro 100% vrcholů cz

Fáze	KR		PR	
%	BFS	MAX	BFS	MAX
0	0.00	0.00	0.00	0.00
1	0.03	0.09	0.07	0.17
2	0.04	0.12	0.10	0.20
3	0.06	0.15	0.12	0.22
4	0.07	0.17	0.13	0.23
5	0.09	0.18	0.15	0.25
6	0.10	0.20	0.16	0.26
7	0.11	0.21	0.18	0.27
8	0.12	0.23	0.19	0.28
9	0.13	0.24	0.20	0.29
10	0.14	0.25	0.21	0.30
11	0.16	0.26	0.22	0.31
12	0.17	0.27	0.23	0.32
13	0.18	0.28	0.24	0.32
14	0.19	0.29	0.25	0.33
15	0.20	0.30	0.26	0.34
16	0.21	0.31	0.27	0.35
17	0.22	0.32	0.28	0.36
18	0.23	0.33	0.29	0.37
19	0.24	0.33	0.31	0.37
20	0.25	0.34	0.32	0.38
21	0.27	0.35	0.33	0.39
22	0.28	0.36	0.34	0.40
23	0.29	0.37	0.35	0.41
24	0.30	0.38	0.36	0.41
25	0.31	0.39	0.37	0.42
26	0.32	0.40	0.38	0.43
27	0.33	0.40	0.39	0.44
28	0.34	0.41	0.40	0.45
29	0.35	0.42	0.41	0.45
30	0.36	0.43	0.41	0.46
31	0.37	0.44	0.42	0.47
32	0.38	0.45	0.43	0.48
33	0.39	0.46	0.44	0.49
34	0.40	0.46	0.45	0.49
35	0.41	0.47	0.46	0.50
36	0.42	0.48	0.47	0.51
37	0.43	0.49	0.48	0.52
38	0.44	0.50	0.49	0.52
39	0.45	0.51	0.50	0.53
40	0.46	0.51	0.51	0.54
41	0.47	0.52	0.52	0.55
42	0.48	0.53	0.52	0.56
43	0.49	0.54	0.53	0.56
44	0.50	0.55	0.54	0.57
45	0.51	0.56	0.55	0.58
46	0.52	0.56	0.56	0.59
47	0.53	0.57	0.57	0.59
48	0.54	0.58	0.58	0.60
49	0.55	0.59	0.59	0.61
50	0.56	0.60	0.60	0.62

Fáze	KR		PR	
%	BFS	MAX	BFS	MAX
51	0.57	0.60	0.60	0.62
52	0.58	0.61	0.61	0.63
53	0.59	0.62	0.62	0.64
54	0.60	0.63	0.63	0.65
55	0.61	0.64	0.64	0.66
56	0.62	0.65	0.65	0.66
57	0.63	0.65	0.65	0.67
58	0.63	0.66	0.66	0.68
59	0.64	0.67	0.67	0.69
60	0.65	0.68	0.68	0.69
61	0.66	0.69	0.69	0.70
62	0.67	0.69	0.70	0.71
63	0.68	0.70	0.70	0.72
64	0.69	0.71	0.71	0.72
65	0.70	0.72	0.72	0.73
66	0.71	0.73	0.73	0.74
67	0.72	0.73	0.74	0.75
68	0.73	0.74	0.75	0.76
69	0.74	0.75	0.75	0.76
70	0.74	0.76	0.76	0.77
71	0.75	0.77	0.77	0.78
72	0.76	0.78	0.78	0.79
73	0.77	0.78	0.79	0.79
74	0.78	0.79	0.80	0.80
75	0.79	0.80	0.80	0.81
76	0.80	0.81	0.81	0.82
77	0.81	0.82	0.82	0.82
78	0.81	0.82	0.83	0.83
79	0.82	0.83	0.84	0.84
80	0.83	0.84	0.84	0.85
81	0.84	0.85	0.85	0.86
82	0.85	0.86	0.86	0.86
83	0.86	0.86	0.87	0.87
84	0.87	0.87	0.87	0.88
85	0.87	0.88	0.88	0.89
86	0.88	0.89	0.89	0.89
87	0.89	0.90	0.90	0.90
88	0.90	0.90	0.91	0.91
89	0.91	0.91	0.91	0.92
90	0.92	0.92	0.92	0.92
91	0.92	0.93	0.93	0.93
92	0.93	0.94	0.94	0.94
93	0.94	0.94	0.95	0.95
94	0.95	0.95	0.95	0.95
95	0.96	0.96	0.96	0.96
96	0.97	0.97	0.97	0.97
97	0.97	0.98	0.98	0.98
98	0.98	0.98	0.98	0.98
99	0.99	0.99	0.99	0.99
100	1.00	1.00	1.00	1.00

Tabulka 8.5: Agregace strategie BFS a maximálních důležitostí pro 50% vrcholů cz



Graf 8.7: Agregace strategie BFS a maximálních důležitostí pro 50% vrcholů cz

8.4.2 Data eu

Data z domény eu byla zpracována do tabulek 8.6 , 8.7 a grafů 8.8, 8.9.

Data evropských univerzit byla specifická tím, že uspořádání vektorů PageRanku a K-Ranku se u nich v průběhu BFS nejméně ze všech souborů dat odlišovalo, viz Graf 8.1. I při zkoumání akumulace měr důležitosti je jasně vidět, jak křivka celkového agregovaného K-Ranku dobře kopíruje křivku agregovaného PageRanku. Stejně tak křivka agregované BFS důležitosti podle K-Ranku kopíruje odpovídající křivku podle PageRanku.

Ukazuje se rovněž, že vliv primární crawlovací strategie na sekundární je méně významný než by se dalo očekávat. Křivky pro 50% vrcholů jsou totiž v dobré shodě s křivkami pro celý soubor.

8.4.3 Data uk

Data ze souboru uk byla zpracována do tabulek 8.8, 8.9 a grafů 8.10, 8.11.

U domény uk dochází k zajímavému jevu, kdy data o konvergenci obou měr důležitosti, jak při celkové agregaci, tak při konvergenci BFS, jsou velmi blízko u sebe.

Fáze	KR		PR	
%	BFS	MAX	BFS	MAX
0	0.00	0.00	0.00	0.00
1	0.02	0.11	0.06	0.19
2	0.04	0.14	0.08	0.22
3	0.05	0.17	0.10	0.24
4	0.07	0.19	0.12	0.26
5	0.08	0.22	0.14	0.28
6	0.09	0.23	0.15	0.29
7	0.11	0.25	0.16	0.30
8	0.12	0.26	0.18	0.31
9	0.13	0.28	0.19	0.32
10	0.14	0.29	0.20	0.33
11	0.15	0.30	0.21	0.34
12	0.16	0.32	0.22	0.35
13	0.17	0.33	0.23	0.36
14	0.19	0.34	0.24	0.37
15	0.20	0.35	0.26	0.38
16	0.21	0.36	0.27	0.39
17	0.22	0.37	0.28	0.40
18	0.23	0.38	0.29	0.40
19	0.24	0.39	0.30	0.41
20	0.25	0.39	0.31	0.42
21	0.26	0.40	0.32	0.43
22	0.27	0.41	0.32	0.43
23	0.28	0.42	0.33	0.44
24	0.29	0.43	0.34	0.45
25	0.30	0.44	0.35	0.46
26	0.31	0.45	0.36	0.47
27	0.32	0.45	0.37	0.47
28	0.33	0.46	0.38	0.48
29	0.34	0.47	0.39	0.49
30	0.35	0.48	0.40	0.50
31	0.35	0.49	0.41	0.50
32	0.36	0.49	0.42	0.51
33	0.37	0.50	0.43	0.52
34	0.38	0.51	0.44	0.53
35	0.39	0.52	0.45	0.53
36	0.40	0.53	0.45	0.54
37	0.41	0.53	0.46	0.55
38	0.42	0.54	0.47	0.56
39	0.43	0.55	0.48	0.56
40	0.44	0.56	0.49	0.57
41	0.45	0.56	0.50	0.58
42	0.46	0.57	0.51	0.58
43	0.47	0.58	0.52	0.59
44	0.48	0.59	0.53	0.60
45	0.49	0.60	0.53	0.61
46	0.50	0.60	0.54	0.61
47	0.50	0.61	0.55	0.62
48	0.51	0.62	0.56	0.63
49	0.52	0.63	0.57	0.64
50	0.53	0.63	0.58	0.64

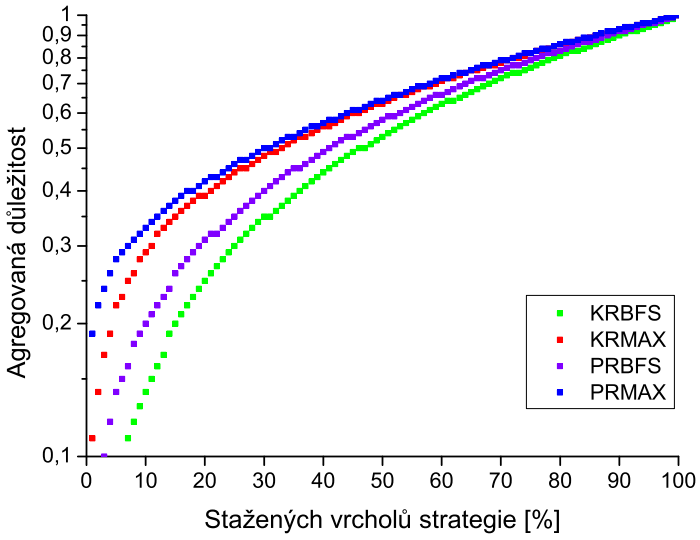
Fáze	KR		PR	
%	BFS	MAX	BFS	MAX
51	0.54	0.64	0.59	0.65
52	0.55	0.65	0.59	0.66
53	0.56	0.66	0.60	0.66
54	0.57	0.66	0.61	0.67
55	0.58	0.67	0.62	0.68
56	0.59	0.68	0.63	0.69
57	0.60	0.69	0.64	0.69
58	0.61	0.69	0.65	0.70
59	0.62	0.70	0.66	0.71
60	0.63	0.71	0.66	0.72
61	0.64	0.72	0.67	0.72
62	0.64	0.72	0.68	0.73
63	0.65	0.73	0.69	0.74
64	0.66	0.74	0.70	0.74
65	0.67	0.75	0.71	0.75
66	0.68	0.75	0.72	0.76
67	0.69	0.76	0.72	0.77
68	0.70	0.77	0.73	0.77
69	0.71	0.77	0.74	0.78
70	0.72	0.78	0.75	0.79
71	0.73	0.79	0.76	0.79
72	0.74	0.80	0.77	0.80
73	0.74	0.80	0.77	0.81
74	0.75	0.81	0.78	0.82
75	0.76	0.82	0.79	0.82
76	0.77	0.83	0.80	0.83
77	0.78	0.83	0.81	0.84
78	0.79	0.84	0.82	0.84
79	0.80	0.85	0.82	0.85
80	0.81	0.86	0.83	0.86
81	0.82	0.86	0.84	0.87
82	0.83	0.87	0.85	0.87
83	0.83	0.88	0.86	0.88
84	0.84	0.88	0.87	0.89
85	0.85	0.89	0.87	0.89
86	0.86	0.90	0.88	0.90
87	0.87	0.91	0.89	0.91
88	0.88	0.91	0.90	0.92
89	0.89	0.92	0.91	0.92
90	0.90	0.93	0.92	0.93
91	0.91	0.93	0.92	0.94
92	0.92	0.94	0.93	0.94
93	0.92	0.95	0.94	0.95
94	0.93	0.96	0.95	0.96
95	0.94	0.96	0.96	0.96
96	0.95	0.97	0.96	0.97
97	0.96	0.98	0.97	0.98
98	0.97	0.99	0.98	0.99
99	0.98	0.99	0.99	0.99
100	1.00	1.00	1.00	1.00

Tabulka 8.6: Agregace strategie BFS a maximálních důležitostí pro 100% vrcholů eu

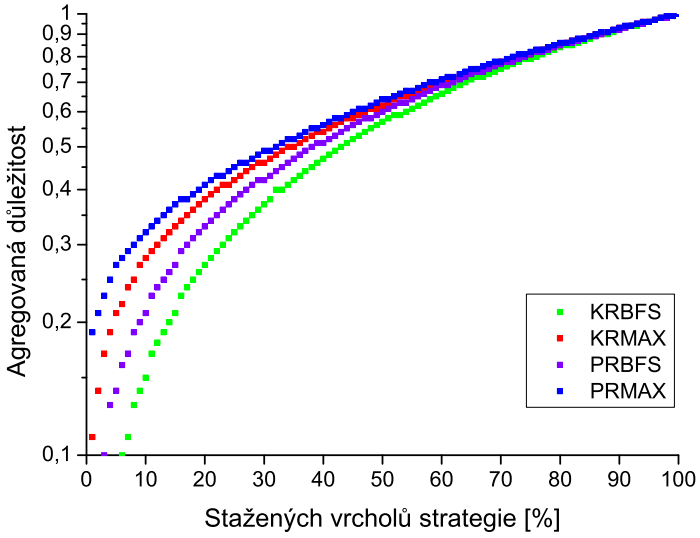
Fáze	KR		PR	
%	BFS	MAX	BFS	MAX
0	0.00	0.00	0.00	0.00
1	0.03	0.11	0.06	0.19
2	0.05	0.14	0.09	0.21
3	0.06	0.17	0.10	0.23
4	0.07	0.19	0.13	0.25
5	0.09	0.21	0.14	0.27
6	0.10	0.22	0.16	0.28
7	0.11	0.24	0.17	0.29
8	0.13	0.25	0.19	0.30
9	0.14	0.27	0.20	0.31
10	0.15	0.28	0.21	0.32
11	0.17	0.29	0.23	0.33
12	0.18	0.30	0.24	0.34
13	0.19	0.31	0.25	0.35
14	0.20	0.32	0.26	0.36
15	0.21	0.33	0.27	0.37
16	0.23	0.34	0.29	0.38
17	0.24	0.35	0.30	0.38
18	0.25	0.36	0.31	0.39
19	0.26	0.37	0.32	0.40
20	0.27	0.38	0.33	0.41
21	0.28	0.39	0.34	0.42
22	0.29	0.40	0.35	0.43
23	0.30	0.41	0.36	0.43
24	0.31	0.41	0.37	0.44
25	0.32	0.42	0.38	0.45
26	0.33	0.43	0.39	0.46
27	0.34	0.44	0.40	0.46
28	0.35	0.45	0.41	0.47
29	0.36	0.46	0.42	0.48
30	0.37	0.46	0.42	0.49
31	0.38	0.47	0.43	0.49
32	0.40	0.48	0.44	0.50
33	0.40	0.49	0.45	0.51
34	0.41	0.50	0.46	0.52
35	0.42	0.50	0.47	0.52
36	0.43	0.51	0.48	0.53
37	0.44	0.52	0.49	0.54
38	0.45	0.53	0.50	0.55
39	0.46	0.54	0.51	0.55
40	0.47	0.54	0.51	0.56
41	0.48	0.55	0.52	0.57
42	0.49	0.56	0.53	0.58
43	0.50	0.57	0.54	0.58
44	0.51	0.58	0.55	0.59
45	0.52	0.58	0.56	0.60
46	0.53	0.59	0.57	0.61
47	0.54	0.60	0.58	0.61
48	0.55	0.61	0.58	0.62
49	0.56	0.61	0.59	0.63
50	0.57	0.62	0.60	0.64

Fáze	KR		PR	
%	BFS	MAX	BFS	MAX
51	0.58	0.63	0.61	0.64
52	0.59	0.64	0.62	0.65
53	0.59	0.65	0.63	0.66
54	0.60	0.65	0.63	0.67
55	0.61	0.66	0.64	0.67
56	0.62	0.67	0.65	0.68
57	0.63	0.68	0.66	0.69
58	0.64	0.68	0.67	0.70
59	0.65	0.69	0.68	0.70
60	0.66	0.70	0.69	0.71
61	0.67	0.71	0.69	0.72
62	0.68	0.71	0.70	0.72
63	0.69	0.72	0.71	0.73
64	0.70	0.73	0.72	0.74
65	0.71	0.74	0.73	0.75
66	0.72	0.75	0.74	0.75
67	0.72	0.75	0.74	0.76
68	0.73	0.76	0.75	0.77
69	0.74	0.77	0.76	0.78
70	0.75	0.78	0.77	0.78
71	0.76	0.78	0.78	0.79
72	0.77	0.79	0.78	0.80
73	0.78	0.80	0.79	0.81
74	0.79	0.81	0.80	0.81
75	0.79	0.81	0.81	0.82
76	0.80	0.82	0.82	0.83
77	0.81	0.83	0.82	0.83
78	0.82	0.84	0.83	0.84
79	0.83	0.84	0.84	0.85
80	0.84	0.85	0.85	0.86
81	0.85	0.86	0.86	0.86
82	0.85	0.87	0.86	0.87
83	0.86	0.87	0.87	0.88
84	0.87	0.88	0.88	0.88
85	0.88	0.89	0.89	0.89
86	0.89	0.90	0.89	0.90
87	0.90	0.90	0.90	0.91
88	0.90	0.91	0.91	0.91
89	0.91	0.92	0.92	0.92
90	0.92	0.93	0.92	0.93
91	0.93	0.93	0.93	0.94
92	0.94	0.94	0.94	0.94
93	0.94	0.95	0.95	0.95
94	0.95	0.96	0.95	0.96
95	0.96	0.96	0.96	0.96
96	0.97	0.97	0.97	0.97
97	0.98	0.98	0.98	0.98
98	0.98	0.99	0.98	0.99
99	0.99	0.99	0.99	0.99
100	1.00	1.00	1.00	1.00

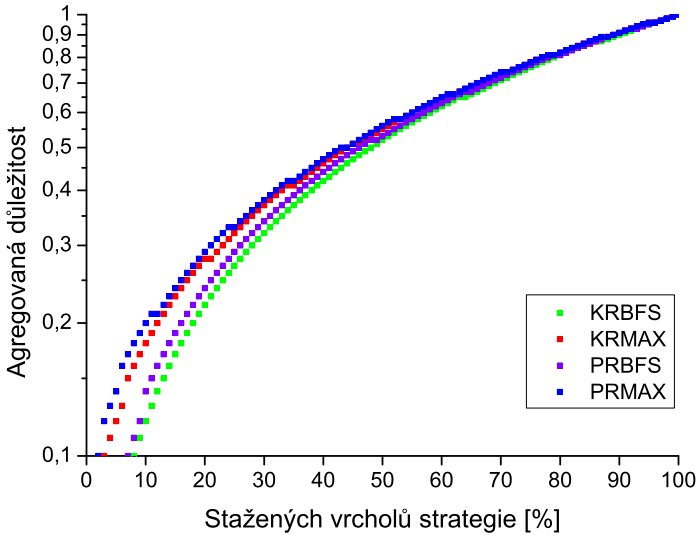
Tabulka 8.7: Agregace strategie BFS a maximálních důležitostí pro 50% vrcholů eu



Graf 8.8: Agregace strategie BFS a maximálních důležitostí pro 100% vrcholů eu



Graf 8.9: Agregace strategie BFS a maximálních důležitostí pro 50% vrcholů eu



Graf 8.10: Agregace strategie BFS a maximálních důležitostí pro 100% vrcholů uk

I na těchto datech je vidět, že data na 50% vrcholů se dobře shodují s daty naměřenými na 100%.

Distribuce PageRanku i K-Ranku jsou na datech z domény uk velmi rovnoměrně rozdělené. Ukazuje se tedy, že pokud bere výpočet PageRanku v úvahu i listové vrcholy, dochází k menšímu zvýhodňování vysoce postavených stránek než je tomu u verzí, které listové vrcholy v potaz neberou.

Fáze	KR		PR	
%	BFS	MAX	BFS	MAX
0	0	0	0	0
1	0.01	0.06	0.02	0.08
2	0.03	0.08	0.04	0.10
3	0.04	0.10	0.05	0.12
4	0.05	0.11	0.07	0.13
5	0.06	0.12	0.08	0.14
6	0.07	0.13	0.09	0.16
7	0.08	0.15	0.10	0.17
8	0.10	0.16	0.11	0.18
9	0.11	0.17	0.12	0.19
10	0.12	0.18	0.14	0.20
11	0.13	0.19	0.15	0.21
12	0.14	0.20	0.16	0.21
13	0.15	0.21	0.17	0.22
14	0.16	0.22	0.18	0.23
15	0.17	0.23	0.19	0.24
16	0.18	0.24	0.20	0.25
17	0.19	0.25	0.21	0.26
18	0.20	0.26	0.22	0.27
19	0.21	0.27	0.23	0.28
20	0.22	0.28	0.24	0.29
21	0.23	0.28	0.25	0.30
22	0.24	0.29	0.26	0.31
23	0.25	0.30	0.27	0.32
24	0.26	0.31	0.28	0.33
25	0.27	0.32	0.29	0.33
26	0.28	0.33	0.30	0.34
27	0.29	0.34	0.31	0.35
28	0.30	0.35	0.32	0.36
29	0.31	0.36	0.33	0.37
30	0.32	0.37	0.34	0.38
31	0.33	0.38	0.35	0.39
32	0.34	0.39	0.36	0.40
33	0.35	0.40	0.37	0.41
34	0.36	0.41	0.38	0.42
35	0.37	0.41	0.39	0.42
36	0.38	0.42	0.40	0.43
37	0.39	0.43	0.41	0.44
38	0.40	0.44	0.42	0.45
39	0.41	0.45	0.43	0.46
40	0.42	0.46	0.44	0.47
41	0.43	0.47	0.45	0.48
42	0.44	0.48	0.46	0.49
43	0.45	0.49	0.47	0.50
44	0.46	0.50	0.48	0.50
45	0.47	0.51	0.49	0.51
46	0.48	0.51	0.50	0.52
47	0.49	0.52	0.51	0.53
48	0.50	0.53	0.52	0.54
49	0.51	0.54	0.52	0.55
50	0.52	0.55	0.53	0.56

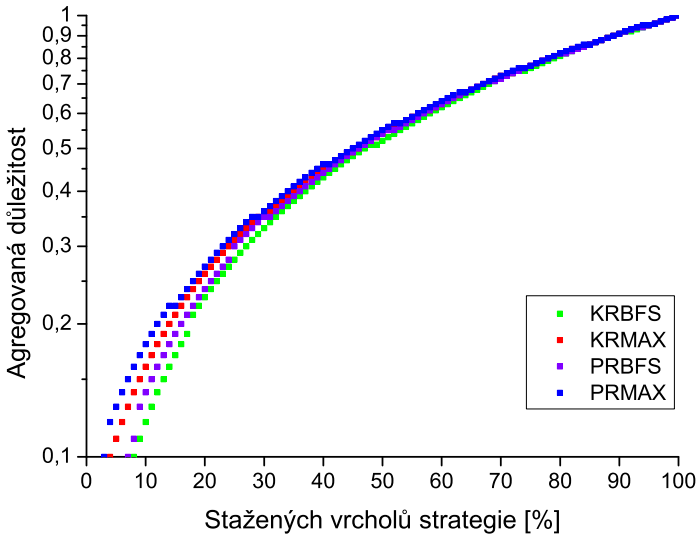
Fáze	KR		PR	
%	BFS	MAX	BFS	MAX
51	0.53	0.56	0.54	0.57
52	0.54	0.57	0.55	0.58
53	0.55	0.58	0.56	0.58
54	0.56	0.59	0.57	0.59
55	0.57	0.60	0.58	0.60
56	0.58	0.61	0.59	0.61
57	0.59	0.61	0.60	0.62
58	0.60	0.62	0.61	0.63
59	0.61	0.63	0.62	0.64
60	0.62	0.64	0.63	0.65
61	0.63	0.65	0.64	0.66
62	0.64	0.66	0.65	0.66
63	0.65	0.67	0.66	0.67
64	0.65	0.68	0.67	0.68
65	0.66	0.69	0.67	0.69
66	0.67	0.70	0.68	0.70
67	0.68	0.70	0.69	0.71
68	0.69	0.71	0.70	0.72
69	0.70	0.72	0.71	0.73
70	0.71	0.73	0.72	0.74
71	0.72	0.74	0.73	0.74
72	0.73	0.75	0.74	0.75
73	0.74	0.76	0.75	0.76
74	0.75	0.77	0.76	0.77
75	0.76	0.78	0.77	0.78
76	0.77	0.79	0.78	0.79
77	0.78	0.79	0.79	0.80
78	0.79	0.80	0.80	0.81
79	0.80	0.81	0.81	0.81
80	0.81	0.82	0.81	0.82
81	0.82	0.83	0.82	0.83
82	0.83	0.84	0.83	0.84
83	0.84	0.85	0.84	0.85
84	0.85	0.86	0.85	0.86
85	0.86	0.87	0.86	0.87
86	0.87	0.87	0.87	0.88
87	0.87	0.88	0.88	0.89
88	0.88	0.89	0.89	0.89
89	0.89	0.90	0.90	0.90
90	0.90	0.91	0.91	0.91
91	0.91	0.92	0.92	0.92
92	0.92	0.93	0.93	0.93
93	0.93	0.94	0.93	0.94
94	0.94	0.95	0.94	0.95
95	0.95	0.96	0.95	0.96
96	0.96	0.96	0.96	0.96
97	0.97	0.97	0.97	0.97
98	0.98	0.98	0.98	0.98
99	0.99	0.99	0.99	0.99
100	1.00	1.00	1.00	1.00

Tabulka 8.8: Agregace strategie BFS a maximálních důležitostí pro 100% vrcholů uk

Fáze	KR		PR	
%	BFS	MAX	BFS	MAX
0	0	0	0	0
1	0.01	0.05	0.02	0.07
2	0.03	0.07	0.03	0.09
3	0.04	0.08	0.05	0.10
4	0.05	0.10	0.06	0.12
5	0.06	0.11	0.08	0.13
6	0.08	0.12	0.09	0.14
7	0.09	0.13	0.10	0.15
8	0.10	0.14	0.11	0.16
9	0.11	0.15	0.13	0.17
10	0.12	0.16	0.14	0.18
11	0.13	0.17	0.15	0.19
12	0.14	0.18	0.16	0.20
13	0.15	0.19	0.17	0.21
14	0.16	0.20	0.18	0.22
15	0.17	0.21	0.19	0.22
16	0.18	0.22	0.20	0.23
17	0.19	0.23	0.21	0.24
18	0.21	0.24	0.22	0.25
19	0.22	0.25	0.23	0.26
20	0.23	0.26	0.24	0.27
21	0.24	0.27	0.25	0.28
22	0.25	0.28	0.26	0.29
23	0.26	0.29	0.27	0.30
24	0.27	0.30	0.28	0.31
25	0.28	0.31	0.30	0.32
26	0.29	0.32	0.31	0.33
27	0.30	0.33	0.32	0.34
28	0.31	0.34	0.33	0.35
29	0.32	0.35	0.34	0.35
30	0.33	0.35	0.35	0.36
31	0.34	0.36	0.35	0.37
32	0.35	0.37	0.36	0.38
33	0.36	0.38	0.37	0.39
34	0.37	0.39	0.38	0.40
35	0.38	0.40	0.39	0.41
36	0.39	0.41	0.40	0.42
37	0.40	0.42	0.41	0.43
38	0.41	0.43	0.42	0.44
39	0.42	0.44	0.43	0.45
40	0.43	0.45	0.44	0.46
41	0.44	0.46	0.45	0.46
42	0.45	0.47	0.46	0.47
43	0.46	0.48	0.47	0.48
44	0.47	0.49	0.48	0.49
45	0.48	0.49	0.49	0.50
46	0.49	0.50	0.50	0.51
47	0.50	0.51	0.51	0.52
48	0.51	0.52	0.52	0.53
49	0.51	0.53	0.53	0.54
50	0.52	0.54	0.54	0.55

Fáze	KR		PR	
%	BFS	MAX	BFS	MAX
51	0.53	0.55	0.55	0.56
52	0.54	0.56	0.55	0.57
53	0.55	0.57	0.56	0.57
54	0.56	0.58	0.57	0.58
55	0.57	0.59	0.58	0.59
56	0.58	0.60	0.59	0.60
57	0.59	0.61	0.60	0.61
58	0.60	0.61	0.61	0.62
59	0.61	0.62	0.62	0.63
60	0.62	0.63	0.63	0.64
61	0.63	0.64	0.64	0.65
62	0.64	0.65	0.65	0.66
63	0.65	0.66	0.66	0.67
64	0.66	0.67	0.67	0.67
65	0.67	0.68	0.68	0.68
66	0.68	0.69	0.69	0.69
67	0.69	0.70	0.70	0.70
68	0.70	0.71	0.71	0.71
69	0.71	0.72	0.71	0.72
70	0.72	0.73	0.72	0.73
71	0.73	0.73	0.73	0.74
72	0.74	0.74	0.74	0.75
73	0.75	0.75	0.75	0.76
74	0.75	0.76	0.76	0.76
75	0.76	0.77	0.77	0.77
76	0.77	0.78	0.78	0.78
77	0.78	0.79	0.79	0.79
78	0.79	0.80	0.80	0.80
79	0.80	0.81	0.81	0.81
80	0.81	0.82	0.82	0.82
81	0.82	0.83	0.82	0.83
82	0.83	0.84	0.83	0.84
83	0.84	0.84	0.84	0.85
84	0.85	0.85	0.85	0.86
85	0.86	0.86	0.86	0.86
86	0.87	0.87	0.87	0.87
87	0.88	0.88	0.88	0.88
88	0.89	0.89	0.89	0.89
89	0.90	0.90	0.90	0.90
90	0.91	0.91	0.91	0.91
91	0.92	0.92	0.92	0.92
92	0.92	0.93	0.93	0.93
93	0.93	0.94	0.94	0.94
94	0.94	0.95	0.94	0.95
95	0.95	0.95	0.95	0.95
96	0.96	0.96	0.96	0.96
97	0.97	0.97	0.97	0.97
98	0.98	0.98	0.98	0.98
99	0.99	0.99	0.99	0.99
100	1.00	1.00	1.00	1.00

Tabulka 8.9: Agregace strategie BFS a maximálních důležitostí pro 50% vrcholů uk



Graf 8.11: Agregace strategie BFS a maximálních důležitostí pro 50% vrcholů uk

Kapitola 9

Závěr a diskuze

9.1 Shrnutí výsledků práce

9.1.1 Ohodnocování listových vrcholů

Všechny výpočty byly provedeny s takovými metodami důležitosti, které ohodnocovaly i listové vrcholy. Použil jsem přitom metodu popsanou v článku [18] z roku 2004. Není mi známo, že by existovala jiná práce zabývající se crawlovacími strategiemi, která by listové vrcholy brala v potaz. Proto i výsledky, které uvádím se značně liší od výsledků jiných prací o crawlování.

Především je to dáno tím, že míry důležitosti jsou distribuovány mnohem rovnoměrněji, než tomu bylo u jiných přístupů neohodnocujících listové vrcholy. Rovnoměrnější rozložení důležitosti je způsobeno především tím, že listových vrcholů je v grafu někdy víc než polovina a ty mají všechny téměř stejnou důležitost.

Rovnoměrněji je distribuován nejen K-Rank, u kterého to je očekávané chování, ale i PageRank, který má u algoritmů bez ohodnocení listových vrcholů problémy právě s nerovnoměrností rozdělení.

9.1.2 BFS algoritmus

Ze znalostí typických vlastností grafu Webu se mi povedlo navrhnout a implementovat metodu na počítání BFS očíslování, která na grafech Webu potřebuje méně I/O operací, než nejlepší metody používané pro obecné orientované grafy. Navíc se mi použitím heuristik podařilo dosáhnout toho, že počet průchodů struktury Webu bývá menší, než je její celkový počet BFS

vrstev. Dále jsem navrhl a implementoval algoritmus pro převod BFS vrstev na BFS očíslování a dokázal jsem, že tento převod lze uskutečnit s použitím $O(sort(E))$ operací, kde E je počet odkazů ve struktuře Webu. Na datech, která jsem měl k dispozici, byla typická doba běhu BFS očíslování 5-6 hodin. Pro data ze souborů cz, eu a uk jsem ověřil, že výsledné očíslování je skutečně BFS.

9.1.3 Úspěšnost BFS v porovnání s totální důležitostí

Strategie BFS je velmi úspěšná co do rychlosti sběru důležitosti i v porovnání s distribucí měr důležitosti. Přitom nemůže existovat žádná strategie, která by byla lepší než distribuce důležitosti. Tento fakt byl potvrzen i v dalších pracích jako [37].

9.1.4 Kendallovo τ

Napsal jsem vlastní verzi velmi efektivního algoritmu pro výpočet Kendallova τ v Javě.

9.1.5 Struktura dvojic

Struktura 7.2 dvojic (zdroj, odkaz) mi umožnila rychlou práci s datovými strukturami reprezentujícími Web na disku. Díky této reprezentaci bylo možné se strukturou provádět mnoho transformací, které by jiné reprezentace neumožnily. Proto jsem i s relativně omezenými prostředky mohl počítat míry důležitosti a crawlovací strategie na velkých kolekcích dat.

9.1.6 Framework pro práci s daty Webu

V průběhu své práce jsem napsal mnoho algoritmů pro práci se strukturou Webu, od jednoduchých utilit na její procházení, až po složité implementace algoritmů důležitosti nebo očíslování podle crawlovacích strategií.

9.1.7 Několik souborů dat

Všechny výpočty jsem provedl na třech souborech dat Webu. Ukázalo se, že internet je různorodý a každá kolekce poskytuje odlišné výsledky. Proto každá studie, která se zabývá strategiemi crawlování, je do velké míry závislá právě na použitých kolekcích.

9.1.8 Vliv primární crawlovací strategie

Tato práce ukazuje, že vliv primární crawlovací strategie na sekundární není pro data, která jsem měl k dispozici, nijak zásadní. I přesto, že vliv primární crawlovací strategie není zanedbatelný, mnohem větší rozdíly se vyskytují u výsledků pro různé kolekce stránek než u výsledků počítaných na 50% dat v porovnání s výsledky na celém souboru pro jednu kolekci.

Měřitelné jsou rozdíly mezi daty, které získáme zkoumáním kolekce 50% a 100% stránek, především pokud jde o uspořádanost metrik důležitosti v průběhu crawlování. Pokud se jedná o agregaci celkové důležitosti, nebyly rozdíly téměř vůbec viditelné.

9.1.9 K-Rank

Cílem bylo navrhnout algoritmus pro hodnocení důležitosti na grafu Webu, který by více odpovídal požadavkům crawlování. Především by měl vyhledávat nové a slibné stránky a měl by mít přijatelné konvergenční vlastnosti. Jeho konvergence se však ukázala poněkud problematická, protože ve většině kritérií nepřekonával PageRank. Je ale třeba zmínit, že matice Webu, na které byl K-Rank počítán byla až o třetinu menší než původní matice. Přesto se jeho konvergenční vlastnosti držely těsně pod PageRankem.

Navíc je zřejmé, že K-Rank důležitost rozkládá rovnoměrněji oproti PageRanku. To, že jsou stránky rovnoměrněji distribuovány do jednotlivých skupin podle velikosti důležitosti, znamená lepší představu o kvalitě, kterou K-Rank poskytuje. PageRank oproti tomu má tendenci vytvářet skupinu stránek s obrovskými hodnotami důležitosti a zbytek, kde jsou hodnoty velmi nízké. K-Rank vytváří širší střední vrstvu.

Při měření stability uspořádání vzhledem k celkové agregované důležitosti se ukázalo, že K-Rank PageRank dokonce překonává. Stále věřím, že K-Rank vytváří kvalitní uspořádání a to především pokud se jedná o nově vzniklé stránky. Ke skutečnému ověření síly této metriky by ale bylo třeba její nasazení v nějaké vyhledávací službě.

9.2 Další možnosti pokračování

9.2.1 Jiné crawlovací strategie

Navrhl jsem sadu algoritmů pro hodnocení crawlovacích strategií v Javě a implementoval je. Pomocí této sady funkcí je tedy možné provést měření libovolné crawlovací strategie.

Mohlo by být zajímavé, jakým způsobem se chovají zejména strategie parciální důležitosti pro metriky s ohodnocením listových vrcholů.

9.2.2 Další zkoumání K-Ranku

V dalším výzkumu je možné zaměřit se na efektivitu K-Ranku. Tato metrika byla navržena a byly proměřeny její základní vlastnosti. Nebyla ale nasazena při vyhledávání, kde by se mohlo ukázat, jak uživatelsky přijatelné výsledky vrací.

9.2.3 Velké kolekce dat

Dále je možné zkoumat, jakým způsobem se budou výpočty chovat na opravdu extrémně rozsáhlých kolekcích dat v řádu miliard vrcholů. Zde je problém v tom, že nejen struktura Webu, ale ani samotný vektor důležitosti se nevejdou do paměti. To se ve velkých systémech řeší úměrným zvětšením paměti. 1 000 000 000 stránek zabere 3.7 GB paměti. 4B celočíselný datový typ dokáže však reprezentovat pouze 4 294 967 296 unikátních identifikátorů. Pokud překročí počet vrcholů tuto mez, je nutné zvětšit rozsah datového typu na 8B. Pole 20 000 000 000 položek pak zabere 148 GB paměti.

Vzhledem k omezeným prostředkům a velkým hardwarovým nárokům je jen málo prací, které by se zabývaly zkoumáním takto velkých kolekcí dat. Přesto jejich vlastnosti mohou být klíčem k pochopení fungování velkých vyhledávacích strojů i struktury Webu jako takové.

Literatura

[1] Serge Abiteboul, Mihai Preda, and Gregory Cobena. Adaptive on-line page importance computation. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 280–290, New York, NY, USA, 2003. ACM Press.

[2] Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, and Sriram Raghavan. Searching the web. *ACM Trans. Inter. Tech.*, 1(1):2–43, 2001.

[3] Ricardo Baeza-Yates, Carlos Castillo, Mauricio Marin, and Andrea Rodriguez. Crawling a country: better strategies than breadth-first for web page ordering. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 864–872, New York, NY, USA, 2005. ACM Press.

[4] Klaus Berberich, Michalis Vazirgiannis, and Gerhard Weikum. T-rank: Time-aware authority ranking. In *Algorithms and Models for the Web-Graph*, pages 131–142. 2004.

[5] Donna Bergmark, Carl Lagoze, and Alex Sbityakov. Focused crawls, tunneling, and digital libraries. In *ECDL '02: Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries*, pages 91–106, London, UK, 2002. Springer-Verlag.

[6] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Trovatore: Towards a highly scalable distributed web crawler. In *WWW Posters*, 2001.

[7] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: a scalable fully distributed web crawler. *Software: Practice and Experience*, 34(8):711–726, 2004.

[8] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. Do your worst to make the best: Paradoxical effects in pagerank incremental computations. In *Algorithms and Models for the Web-Graph*, pages 168–180. 2004.

[9] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, 1998.

[10] Adam L. Buchsbaum, Michael Goldwasser, Suresh Venkatasubramanian, and Jeffery R. Westbrook. On external memory graph traversal. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 859–860, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[11] Yen-Yu Chen, Qingqing Gan, and Torsten Suel. I/O-efficient techniques for computing pagerank. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 549–557, New York, NY, USA, 2002. ACM Press.

[12] Junghoo Cho and Hector Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 200–209, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[13] Junghoo Cho and Hector Garcia-Molina. Parallel crawlers. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 124–135, New York, NY, USA, 2002. ACM Press.

[14] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient crawling through url ordering. *Comput. Netw. ISDN Syst.*, 30(1-7):161–172, 1998.

[15] Junghoo Cho, Narayanan Shivakumar, and Hector Garcia-Molina. Finding replicated web collections. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 355–366, New York, NY, USA, 2000. ACM Press.

[16] Microsoft Corporation. Asp.net. Website. `http://www.asp.net/`.

[17] Microsoft Corporation. Internet information services. Website. `http://www.microsoft.com/WindowsServer2003/iis/default.mspx`.

[18] Gianna M. Del Corso, Antonio Gullí, and Francesco Romani. Fast pagerank computation via a sparse linear system (extended abstract). In *Algorithms and Models for the Web-Graph*, pages 118–130. 2004.

[19] Nick Craswell, Francis Crimmins, David Hawking, and Alistair Moffat. Performance and cost tradeoffs in web search. In *ADC '04: Proceedings of the 15th Australasian database conference*, pages 161–169, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[20] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using context graphs. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 527–534, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[21] Jenny Edwards, Kevin McCurley, and John Tomlin. An adaptive model for optimizing performance of an incremental web crawler. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 106–113, New York, NY, USA, 2001. ACM Press.

[22] The Apache Software Foundation. The apache http server project. Website. `http://httpd.apache.org/`.

[23] The PHP Group. Php: Hypertext preprocessor. Website. `http://cz.php.net/`.

[24] Taher H. Haveliwala. Efficient computation of pagerank. Technical Report 1999-31, 1999.

[25] Taher H. Haveliwala and Sepandar D. Kamvar. The second eigenvalue of the google matrix, 2003.

[26] Janice J. Heiss. Become.com's web crawler: A massively scaled java technology application. Website. `http://java.sun.com/developer/technicalArticles/WebServices/become/`.

[27] Allan Heydon and Marc Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.

[28] Ben Hough. The spectral gap. In *Lecture given at U.C. Berkeley for a course by Jim Pitman.*

[29] Arun K. Iyengar, Mark S. Squillante, and Li Zhang. Analysis and characterization of large-scale web server access patterns and performance. *World Wide Web*, 2(1-2):85–100, 1999.

[30] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation methods for accelerating pagerank computations. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 261–270, New York, NY, USA, 2003. ACM Press.

[31] William R. Knight. A computer method for calculating Kendall's tau with ungrouped data. *Journal of the American Statistical Association*, 61(314):436–439, 1966.

[32] Daniel Král'. Stochastické matice. In *Používáme linearní algebru*, pages 260–261, 2002.

[33] Steve Lawrence and C. Lee Giles. Accessibility of information on the web. *Intelligence*, 11(1):32–39, 2000.

[34] Netcraft Ltd. Netcraft web server survey. Website. `http://survey.netcraft.com/archive.html`.

[35] Kurt Mehlhorn and Ulrich Meyer. External-memory breadth-first search with sublinear i/o. In *ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms*, pages 723–735. Springer-Verlag, London, UK, 2002.

[36] Sun Microsystems. Java servlet technology. Website. `http://java.sun.com/products/servlet/`.

[37] Marc Najork and Janet L. Wiener. Breadth-first crawling yields high-quality pages. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 114–118, New York, NY, USA, 2001. ACM Press.

[38] Venkata N. Padmanabhan and Lili Qiu. The content and access dynamics of a busy web site: findings and implications. In *SIGCOMM*

'00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pages 111–123, New York, NY, USA, 2000. ACM Press.

[39] Vladislav Shkapenyuk and Torsten Suel. Design and implementation of a high-performance distributed web crawler. In *ICDE*, 2002.

[40] W3C. World wide web consortium. Website. `http://www.w3.org/`.

[41] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 136–147, New York, NY, USA, 2002. ACM Press.