Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Jaroslav Ciml

Personifikovaný metavyhledávač

Katedra softwarového inženýrství
Vedoucí diplomové práce: RNDr. Leo Galamboš, Ph.D.
Studijní program: Informatika

Rád bych na tomto místě poděkoval všem, kdo přispěli ke vzniku této práce. Zejména pak mému vedoucímu RNDr. Leovi Galambošovi, Ph.D. za množství cenných rad i připomínek a v neposlední řadě také za pomoc při realizaci zkušebního provozu metavyhledávače. Dále patří můj velký dík Ondřeji Vencálkovi za to, že byl nápomocen při řešení problémů z oblasti matematické statistiky.

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

V Praze dne Jaroslav Ciml

Obsah

Název práce: Personifikovaný metavyhledávač
Autor: Jaroslav Ciml
Katedra: Katedra softwarového inženýrství
Vedoucí diplomové práce: RNDr. Leo Galamboš, Ph.D.
e-mail vedoucího: Leo.Galambos@mff.cuni.cz

Abstrakt

Existuje mnoho vyhledávačů pro hledání na Webu. Různé vyhledávače někdy poskytují naprosto odlišné odpovědi pro tentýž dotaz, a proto může být užitečné prohlédnout si výsledkové listiny od několika z nich. Správná volba vyhledávačů se může navíc pro různé uživatele lišit. Metavyhledávač je nástroj, který pro dotaz uživatele vybere nejvhodnější vyhledávače a těm dotaz rozešle k vyhodnocení. Odpovědi vrácené jednotlivými vyhledávači poté zkombinuje do jediné výsledkové listiny, kterou prezentuje uživateli. Cílem této práce je navrhnout a implementovat metavyhledávač s podporou uživatelských profilů.

Klíčová slova: vyhledávání na Webu, metavyhledávač, uživatelský profil

Title: Personified Meta-Search Engine
Author: Jaroslav Ciml
Department: Department of Software Engineering
Supervisor: RNDr. Leo Galamboš, Ph.D.
Supervisor's e-mail address: Leo.Galambos@mff.cuni.cz

Abstract

There are many search engines designed to help find information stored on the Web. Different search engines often return completely different result lists for the same query. It can be useful to see the results from more of them. Furthermore the right choice of the search engines may vary from user to user. A meta-search engine is a tool that accpets a query from user, chooses the most suitable search engines and sends them the query. After the search engines return their responses, the meta-search engine combines these results into a single list that is displayed to the user. The objective of this work is to design and implement a meta-search engine with a user profile support.

Keywords: Web searching, meta-search engine, user profile

Kapitola 1

Úvod

Síť World Wide Web (dále jen Web) se stala rozsáhlým zdrojem informací v nejrůznějších oblastech. Stále roste množství zdrojů[1] na této síti i počet uživatelů, kteří ji využívají. Uživatel si často může být jist, že Web obsahuje odpověď na jeho otázku. Limitujícím faktorem pak není samotná existence informace, ale spíše schopnost ji najít. Nejoblíbenějšími prostředky, které výrazně usnadní navigaci v této záplavě dat, jsou webové vyhledávače. Z nejznámějších lze uvést například Google či Yahoo! Search. Mezi vyhledávači v češtině jsou to pak třeba Jyxo nebo Seznam.

Data na Webu mohou mít různou formu. Nejčastěji je to text nebo obrázek. V této práci se dále budeme zabývat pouze textovými dokumenty.

Různé vyhledávače mohou poskytovat více či méně kvalitní výsledky v závislosti na oblasti, které se dotaz týká, na způsobu formulace dotazu a samozřejmě i na jazyce, v jakém je dotaz položen. Uživatel tedy může podle vyhledávané informace zvolit určitý vyhledávač. Výběr může ovlivnit i doba odpovědi. I přesto výsledek jediného vyhledávače nemusí být dostačující. Lze se pak dotázat několika vyhledávačů a prohlédnout si odpověď každého z nich. Tento postup bude ovšem pro uživatele značně nepohodlný. Je potřeba si pamatovat, kde se jednotlivé vyhledávače nacházejí, a je třeba se vypořádat s odlišnostmi v jejich uživatelském rozhraní. Ve většině případů nakonec uživatel nebude pokládat svůj dotaz více než jednomu vyhledávači, protože ho odradí čas, který by tím strávil.

Právě metavyhledávač by měl vyplnit tento prostor. „Obyčejný“ vyhledávač použije pro zodpovězení dotazu tzv. index. Index je datová struktura, která pro jednotlivé termy obsahuje seznam dokumentů, které daný term obsahují (nebo obecněji seznam dokumentů, pro které je term významný). Na základě indexu vyhledávač určí, které dokumenty jsou pro dotaz relevantní. Oproti tomu metavyhledávač takovou datovou strukturu nemá. Stejně jako vyhledávač i metavyhledávač je program, který od uživatele dostane dotaz

[1] Celkový objem dat na Webu je těžko odhadnutelný. Pro přiblížení lze uvést, že webové vyhledávače uvádějí počet indexovaných stránek v jednotkách miliard.

a jeho úkolem je vrátit seznam dokumentů, které dotazu nejlépe odpovídají. Metavyhledávač ovšem není sám o sobě schopen rozhodnout, které dokumenty vrátí ve své výsledkové listině a které ne. Místo toho rozešle dotaz několika vyhledávačům a počká na jejich odpovědi. Výsledkové listiny jednotlivých vyhledávačů se pak určitým způsobem (různé metavyhledávače používají k tomuto účelu různé algoritmy) agregují do jediného seznamu. Tento seznam je pak prezentován uživateli jako odpověď na jeho dotaz.

Uživatel tedy používá jediné rozhraní a zadává dotaz pouze jednou, ačkoli tento dotaz je ve skutečnosti rozeslán několika vyhledávačům. Metavyhledávače jsou většinou schopny automaticky vybrat vyhledávače, kterým se dotaz rozešle. Tato volba je založena na sběru zpětné vazby. Sledovat lze množství vrácených dokumentů, rychlost odpovědi a v neposlední řadě dokumenty, které uživatel označil jako relevantní.

Předchozí úvahy už místy narážejí na problém, že relevanci dokumentu k danému dotazu lze těžko určit absolutně. Závisí totiž na oblastech zájmu uživatele a také na jeho zvyklostech a zkušenostech při formulaci dotazu. Nabízí se tedy myšlenka podporovat uživatelské profily. Ta je samozřejmě použitelná pro obyčejný vyhledávač i metavyhledávač.

Cílem této práce je podrobně navrhnout algoritmy komplexně pokrývající problémy, které musí metavyhledávač řešit. Dále pak implementovat takový metavyhledávač a experimentálně ověřit kvalitu výsledků, které poskytuje. Tento metavyhledávač by měl splňovat následující požadavky:

- snadná možnost přidat další vyhledávače, kterých se může metavyhledávač dotazovat,
- inteligentní algoritmus pro agregování odpovědí vyhledávačů na dotaz do jedné výsledkové listiny,
- podpora uživatelských profilů,
- schopnost pro daný dotaz a daného uživatele automaticky vybrat vyhledávače, kterým se dotaz rozešle.

Další text této práce je strukturován následovně. Kapitola 2 se zabývá teoretickými podklady, ze kterých návrh metavyhledávače vychází. Další kapitoly už popisují výsledky autora této práce. V kapitole 3 bude podrobně rozebrán návrh metavyhledávače. Popíšeme, jak ze zmíněných podkladů sestavíme komplexní návrh metavyhledávače, a rozebereme rozšíření a úpravy těchto výchozích algoritmů a technik. Kapitola 4 stručně pojednává o implementaci metavyhledávače (v jazyku Java), která byla použita pro získání některých experimentální výsledků. Více o těchto výsledcích uvedeme v kapitole 5. Kapitola 6 se zmiňuje o příbuzných pracích a dalších možnostech rozšíření. Závěrečné zhodnocení obsahuje kapitola 7.

Součástí této diplomové práce je také přiložené CD s implementací metavyhledávače. Instrukce k použití podává příloha A.

Kapitola 2

Podklady

Následující kapitola obsahuje podklady, které nejsou dílem autora této práce. Uvedené informace slouží jako teoretický podklad, ze kterého vychází návrh našeho metavyhledávače.

2.1 Vyhledávání dokumentů

Klíčovým problémem, který musí vyhledávače řešit, je nalézt v kolekci dokumentů ty nejrelevantnější vzhledem k danému dotazu.

Kvalitu konkrétní metody lze popsat pomocí dvou veličin - přesnosti a úplnosti. Dokumenty v kolekci mají pro uživatele různou míru relevance vzhledem k jeho dotazu. Dále však budeme předpokládat, že pro daný dotaz můžeme veškeré dokumenty v kolekci rozdělit na relevantní a nerelevantní. Mějme tedy dotaz q a kolekci D. Nechť D_q jsou dokumenty skutečně relevantní pro dotaz q a M_q dokumenty, které vrátila zkoumaná metoda. Potom *přesnost* (precision) definujeme jako

$$P_q = \frac{|M_q \cap D_q|}{|M_q|}$$

a *úplností* (recall) budeme rozumět

$$R_q = \frac{|M_q \cap D_q|}{|D_q|}.$$

Je zřejmé, že přesnost i úplnost nabývají hodnot z intervalu $\langle 0, 1 \rangle$. Zjevně bychom rádi dosáhli toho, aby metoda měla oba parametry vysoké, ideálně $P_q = R_q = 1$. Praxe však ukazuje, že tyto veličiny jsou na sobě nepřímo závislé.

Chceme-li popsat nějakou konkrétní metodu pro vyhledávání dokumentů, musíme provést následující rozhodnutí. Je třeba stanovit, jakým způsobem budeme reprezentovat dotaz, jak dokument a kdy řekneme, že dokument je pro jistý dotaz relevantní. Tento koncept pak nazýváme model.

V boolském modelu je dotazem buď jediný term nebo některý z výrazů $d1$ & $d2$, $d1 \vee d2$, $\neg d1$, kde d_1, d_2 jsou dotazy. Dokument je pak reprezentován množinou (případně posloupností) termů, které jsou pro tento dokument významné. Tato množina termů může být dokumentu přiřazena ručně nebo automaticky. V druhém případě se obvykle použijí veškerá slova obsažená v dokumentu vyjma nejběžnějších slov příslušného jazyka. Je-li dotaz tvořen jediným termem, pak tato metoda považuje za relevantní dokumenty právě ty, jež obsahují tento term. Pro dotazy tvaru t_1 & t_2 jsou relevantní dokumenty obsahující oba termy, pro $t_1 \vee t_2$ jsou to dokumenty s alespoň jedním z termů a pro $\neg t$ dokumenty, které tento term neobsahují. Je zřejmé, jak by vypadala sémantika složitějších dotazů.

Tento model můžeme dále zobecňovat. Krom termů lze v dotazech používat i fráze. Fráze můžeme chápat jako posloupnost termů. Zapíšeme ji tak, že termy uzavřeme do uvozovek. Dotaz obsahující frázi může vypadat například takto:

Queen & "Freddie Mercury".

Tomuto dotazu budou odpovídat dokumenty, které obsahují term *Queen* a zároveň term *Freddie* těsně následovaný termem *Mercury*.

Boolský model má celou řadu variant, které ho rozšiřují o další prvky. Sem patří zástupné znaky, proximitní omezení (tj. specifikování vzájemné polohy dvou termů v dokumentu, například požadavek, aby se dva termy vyskytovaly ve stejném odstavci), fuzzy logika, min-max model a další. Zájemci mohou najít více informací třeba v knize [3]. Tato rozšíření však nejsou pro naše účely příliš zajímavá. Nejsou totiž obvykle podporována a metavyhledávač by měl používat pouze takové dotazy, které budou pokud možno srozumitelné pro většinu existujících vyhledávačů.

Vektorový model chápe dotaz i dokument jako vektor $(w_1, \ldots, w_m)$. Číslo m představuje počet všech termů vyskytujících se v kolekci dokumentů. Hodnoty $w_1, \ldots, w_n$ jsou z intervalu $\langle 0, 1 \rangle$. Složka w_i vyjadřuje významnost i-tého termu pro dokument či dotaz. Nulová nebo nule blízká hodnota znamená term nevýznamný nebo málo významný. Složky blízké jedné vyjadřují důležité termy. Vzhledem k tomu, že dotaz i dokument jsou reprezentovány pomocí vektorů v prostoru $\langle 0, 1 \rangle^m$, intuitivně se nabízí měřit relevanci dokumentu pro daný dotaz jako podobnost příslušných dvou vektorů. Funkci, která dotazu a dokumentu přiřadí jejich podobnost, nazýváme podobnostní funkce. Nechť q_j je j-tá složka vektoru dotazu, d_i je i-tý dokument a $w_{i,j}$ je j-tá složka jeho vektoru (tj. významnost j-tého termu v i-tém dokumentu). Příklady podobnostních funkcí mohou být skalární součin

$$Sim(q, d_i) = \sum_{j=1}^{m} q_j \cdot w_{i,j}$$

nebo kosinová míra

$$Sim(q, d_i) = \frac{\sum_{j=1}^{m} q_j \cdot w_{i,j}}{\sqrt{\sum_{j=1}^{m} {q_j}^2} \cdot \sqrt{\sum_{j=1}^{m} {w_{i,j}}^2}}.$$

Uživatel obvykle formuluje dotaz jako množinu termů, přičemž počet termů je typicky velmi malé číslo[1]. Vektor dotazu má pak téměř všechny složky nulové, pouze pro uvedené termy jsou příslušné složky rovny jedné. Tento přístup můžeme snadno zobecnit tím, že pro každý term v dotazu dovolíme uvést váhu.

Vektory dokumentů v kolekci můžeme spočítat jako

$$w_{i,j} = TF_{i,j} \cdot IDF_j,$$

kde $w_{i,j}$ je opět j-tá složka vektoru i-tého dokumentu, $TF_{i,j}$ představuje frekvenci (počet výskytů) j-tého termu v i-tém dokumentu a IDF_j je tzv. *inverzní frekvence j-tého termu v dokumentech*. Inverzní frekvenci termu v dokumentech spočteme ze vztahu

$$IDF_j = \log \frac{n+1}{DF_j}.$$

Hodnota DF_j představuje počet dokumentů v kolekci, ve kterých se j-tý term vyskytuje, n je pak počet všech dokumentů. Smyslem inverzní frekvence termu v dokumentech je posílit vliv málo používaných termů. Dá se předpokládat, že právě tyto termy mohou dobře vystihnout charakter dokumentu.

Z vzorců podobnostních funkcí je vidět, že budou zvýhodněny delší vektory před krátkými, což není příliš žádoucí. Proto je vhodné aplikovat podobnostní funkci na znormované vektory.

Existuje opět řada možností, jak vektorový model modifikovat či dále rozšiřovat. Sem patří normalizace frekvencí termu v dokumentu, další podobnostní funkce nebo ošetření nejběžnějších (nevýznamných) slov daného jazyka. Pro podrobnosti je možné opět nahlédnout do knihy [3].

Nevýhodou boolského modelu je příliš hrubé dělení dokumentů. Rozlišuje se pouze mezi těmi, které dotazu vyhovují a které ne. Nedává žádný nástroj, jak dokumenty odpovídající dotazu uspořádat. Vektorový model nabízí řešení tohoto problému. Používá jemnější rozlišení míry relevance dokumentů, tj. přiřazení reálného čísla (obvykle z intervalu $\langle 0, 1 \rangle$). Vyhledávače na Webu stojí na pomezí těchto dvou modelů. Využívá se zde uspořádání z vektorového modelu (číslo vyjadřující relevanci dokumentu se v tomto kontextu

[1] V článku [9] se uvádí, že průměrný počet termů v dotazu pro webový vyhledávač je $2,21$.

často nazývá *rank*). Z boolského modelu je převzata formulace dotazu pomocí termů a logický spojek. V návrhu našeho metavyhledávače se navíc využije vektorový model při výběru vyhledávačů, kterým bude rozeslán dotaz uživatele. Klasický vyhledávač přiřazuje dotazu relevantní dokumenty. Podobně metavyhledávač hledá pro dotaz vhodné vyhledávače, kterým dotaz rozešle. Právě k řešení tohoto úkolu můžeme použít některé myšlenky vektorového modelu. Podrobně je tento problém popsán v kapitole 2.5.

2.2 Formát dotazu a výsledkové listiny

Prohlížeč Mozilla podporuje funkci vyhledávání na Webu. V postranní liště může uživatel zvolit vyhledávač a zadat dotaz. Jako výsledek se v téže liště zobrazí seznam výsledků vrácených vyhledávačem.

Po technické stránce tato operace proběhne tak, že podle zadaného dotazu se sestaví URI. Vyhledávači se odešle HTTP požadavek (HTTP request) s tímto URI a jako odpověď (HTTP response) vrátí vyhledávač HTML stránku se nalezenými výsledky. Tento proces tedy simuluje vyplnění formuláře na webovém rozhraní vyhledávače. Nepoužívá se žádný zvláštní přístup, jenž některé vyhledávače pro podobné účely poskytují (např. Google - více informací na adrese `http://www.google.com/apis`).

Pro každý podporovaný vyhledávač má Mozilla k dispozici jeden soubor popisující formát URI dotazu a HTML stránky s výsledky. Podobný problém musí řešit i metavyhledávač ve chvíli, kdy rozesílá dotaz uživatele vybraným vyhledávačům a poté potřebuje analyzovat jejich odpovědi. Obsah takového souboru může být následující:

```
<search
   name="Google"
   method="GET"
   action="http://www.google.com/search"
>

<input name="q" user>
<input name="sourceid" value="mozilla-search">
<inputnext name="start" factor="10">
<interpret
    browserResultType="result"
    resultListStart="<!--a-->"
    resultListEnd="<!--z-->"
    resultItemStart="<!--m-->"
    resultItemEnd="<!--n-->"
>

</search>
```

Tento jazyk pro popis formátu je rozšířením jazyka používaným vyhledávacími nástroji Sherlock v systému MAC OS. Syntakticky je podobný jazyku HTML. Obsahuje elementy, které mohou být hnízděné. Elementy mají název a atributy, kde atribut je dvojice řetězců (jméno a hodnota). Podobně jako v HTML jsou některé elementy nepárové, takže jazyk nevyhovuje syntaxi XML.

URI dotazu (tj. URI použité v HTTP požadavku odesílaném vyhledávači) se získá jako hodnota atributu `action` v elementu `search`, za kterou se připojí znak "?" následovaný parametry. URI pro dotaz *Java forum* a výše uvedený formát by vypadalo takto:

```
http://www.google.com/search?q=Java+forum
        &sourceid=mozilla-search
```

Parametry URI jsou definovány pomocí elementů `input`. Jméno parametru určuje atribut `name` a jeho hodnotu atribut `value`. V našem příkladu je takto specifikován parametr `source-id=mozilla-search`. Tento parametr se doporučuje uvádět v každém souboru s formátem. Některé vyhledávače na základě tohoto parametru doplňují do výsledku HTML komentáře, které usnadňují jeho další zpracování. Popis formátu musí obsahovat právě jeden element `input` s atributem `user`. Jedná se o speciální atribut, který nemá žádnou hodnotu. Tento element `input` nemá atribut `value`. V URI bude u příslušného parametru jako hodnota uveden dotaz. Jednotlivé termy se oddělují pomocí mezer. Mezera se pak v URI kóduje pomocí znaku "+". Vyhledávač chápe dotaz jako konjunkci uvedených termů. Je možné používat i fráze uzavřením příslušných termů do uvozovek. Uvozovky v URI je pak třeba zakódovat pomocí znaku "%" a hexadecimálního kódu. Pro dotaz *"Java programming"* by parametr v URI vypadal takto:

```
q=%22Java+programming%22
```

Element `interpret` popisuje strukturu odpovědi. Jeho atributy `resultItemStart` a `resultItemEnd` definují řetězce, které ohraničují položky výsledkové listiny. Položkou (říkejme jí dále hit) rozumíme informace o jednom nalezeném dokumentu. Typicky sem patří název dokumentu, jeho URI a úryvek textu (snippet, popisek), v němž jsou často zvýrazněny termy použité v dotazu. V uvedeném příkladu je za hit považován libovolný podřetězec v odpovědi vyhledávače vyskytující se mezi řetězci `<!--m-->` a `<!--n-->`. Jedná se o HTML komentáře, které vyhledávač Google umísťuje do výsledkové listiny právě jako podporu vyhledávání v Mozille. Element `interpret` navíc může mít nepovinné atributy `resultListStart` a `resultListEnd`. Hodnota `resultListStart` definuje podřetězec za nímž se budou v HTML stránce vrácené vyhledávačem hledat hity. Podobně `resultListEnd` určuje místo, od kterého se už odpověď neprohledává. Tyto atributy lze použít například pro odfiltrování sponzorovaných odkazů.

Jak je vidět z předchozího výkladu, formát odpovědi je popsán pomocí výskytu podřetězců. Její HTML struktura není nijak zohledněna. Dále stojí za povšimnutí, že hit je chápán jako úsek HTML kódu. Jazyk není schopen postihnout jeho vnitřní strukturu. Není možné popsat, jak se najde titulek hitu, jak URI atd.

Na stránkách `http://mycroft.mozdev.org` je možno najít podrobnější informace, jak Mozilla popisuje formát dotazu a odpovědi. Tato kapitola měla jen nastínit základní koncept. Jazyk použitý Mozillou má některé hezké vlastnosti vhodné i pro metavyhledávač, avšak v některých ohledech není dostačující. Úskalí tohoto jazyka a rozšíření doplněná pro potřeby metavyhledávače budou rozebrána později v kapitole 3.4.

2.3 Slévání výsledkových listin

Poté, co obdržíme odpověď na dotaz od jednotlivých vyhledávačů, je potřeba z nich vytvořit jednu výsledkovou listinu, kterou bude prezentovat metavyhledávač uživateli. Od každého vyhledávače dostáváme seznam hitů seřazený podle míry relevance k dotazu. Ta bývá obvykle vyjádřena nezáporným reálným číslem, kterému budeme říkat rank. (Přirozeně větší rank znamená relevantnější dokument.) Tento problém nyní popíšeme formálně a ukážeme algoritmus pro jeho řešení.

Mějme databázi obsahující konečné množství objektů. Počet těchto objektů označme N. Dále mějme pevně určené přirozené číslo m. S každým objektem R jsou asociovány hodnoty $x_1, \ldots, x_m$, kde $x_i \in \langle 0, 1 \rangle$ pro $i \in \{1, \ldots, m\}$. Hodnotou i-tého atributu objektu budeme rozumět právě číslo x_i. Databázi můžeme chápat jako m setříděných seznamů $L_1, \ldots, L_m$, které mají všechny délku N. Prvek seznamu L_i má tvar (R, x_i), kde x_i je hodnota i-tého atributu objektu R. Přirozeně každý seznam obsahuje pro každý objekt R právě jeden prvek (R, x_i). Prvky i-tého seznamu jsou setříděné v nerostoucí posloupnost podle hodnot x_i.

V řeči metavyhledávače představují objekty v databázi všechny dokumenty nalezené všemi vyhledávači, kterým byl položen jeden konkrétní dotaz. Číslo m odpovídá počtu vyhledávačů, kterým byl dotaz rozeslán. Prvek (R, x_i) znamená, že i-tý vyhledávač přiřadil dokumentu R rank x_i. V případě, že některý dokument se nevyskytuje na všech výsledkových listinách, budeme uvažovat, že vyhledávače, které tento dokument nevrátily, mu přiřadily rank 0.

Dále mějme m-ární agregační funkci t. Tato funkce hodnotám atributů $x_1, \ldots, x_m$ objektu R přiřadí celkovou významnost objektu $t(x_1, \ldots, x_m)$. Úkolem je pro dané číslo k nalézt k objektů s nejvyšší celkovou významností.

V obecné verzi problému je možné k datům přistupovat dvěma způsoby. Sekvenční přístup představuje zpracovávání hodnot i-tého atributu jednotlivých objektů v pořadí, jak jsou uvedeny v seznamu L_i. Má-li objekt R

l-tou nejvyšší hodnotu i-tého atributu, pak k nalezení hodnoty (R, x_i) potřebujeme l sekvenčních přístupů. Druhou možností je náhodný přístup. Pro objekt R a číslo i nalezneme hodnotu (R, x_i) jedním náhodným přístupem.

Naivní algoritmus, který řeší uvedenou úlohu, spočívá v přesném spočítání celkové významnosti pro každý objekt a následném výběru k objektů, které budou mít tuto hodnotu nejvyšší. Docílíme toho tím, že projdeme celou databázi a pro každý objekt tak zjistíme hodnoty jeho atributů $x_1, \ldots, x_m$, abychom mohli spočítat $t(x_1, \ldots, x_m)$. Ať už průchod databáze provedeme jakoukoli kombinací sekvenčních a náhodných přístupů, je tento postup časově příliš náročný. Naším cílem je samozřejmě efektivnější metoda. Uvedeme zde algoritmus mezní hodnoty, který dále značme TA (threshold algorithm). Autory algoritmu jsou Ronald Fagin, Amnon Lotem a Moni Naor[2].

Algoritmus TA popíšeme ve třech krocích.

1. Paralelně procházíme všechny seznamy $L_1, \ldots, L_m$. Jednotlivé prvky každého seznamu čteme sekvenčně. Jakmile pro objekt R poprvé nalezneme libovolnou položku (R, x_i), dohledáme pro tento objekt pomocí náhodného přístupu hodnoty všech dalších atributů, tj. hodnoty (R, x_j) pro $j \neq i$. Pak můžeme pro objekt R spočítat číslo $t(R) = t(x_1, \ldots, x_m)$. Průběžně uchováváme dvojice R a $t(R)$ pro k objektů s dosud nejvyšší nalezenou celkovou významností. Množinu těchto dvojic označme Y. Stačí tedy v paměti uchovávat pouze k objektů, v případě shodné celkové významnosti více objektů, zvolíme mezi objekty libovolně.

2. Pro každý seznam L_i definujme hodnotu $\underline{x}_i$ jako poslední hodnotu i-tého atributu, které byla v seznamu L_i přečtená sekvenčním přístupem. Pojmem *mezní hodnota* pak označíme číslo $\tau = t(\underline{x}_1, \ldots, \underline{x}_m)$.

3. Výpočet zastavíme ve chvíli, kdy máme nalezených k objektů s celkovou významností alespoň τ. Množina Y je nyní řešením naší úlohy.

Definice. O funkci t řekneme, že je monotónní, jestliže pro libovolné hodnoty $x_1, \ldots, x_m, x_1', \ldots, x_m'$ z intervalu $\langle 0, 1 \rangle$ platí

$$(\forall i \in \{1, \ldots, m\} : x_i \leq x_i') \Rightarrow t(x_1, \ldots, x_m) \leq t(x_1', \ldots, x_m').$$

Požadavek monotonie je vcelku přirozený. Říká jen, že snížením hodnot některých atributů objektu nelze zvýšit celkovou významnost. Tento předpoklad splňují všechny obvyklé agregační funkce, jako jsou minimum, maximum, suma nebo aritmetický průměr. Je zřejmé, že pro monotónní agregační funkci t je algoritmus TA korektní.

Kromě TA je samozřejmě řada jiných možností, jak vyřešit daný problém. Jednou z nich je například Faginův algoritmus popsaný v článku [7].

[2]Krom této trojice autorů prezentovaly ekvivalentní algoritmus další dvě skupiny. (více o této situaci např. v dokumentu [8])

TA je ovšem jistým způsobem optimální. Co přesně tato vágní formulace znamená popíšeme nyní formálně.

Mějme databázi D a algoritmus A řešící naši úlohu. Časovou složitost algoritmu A nad databází D můžeme vyjádřit vzorcem

$$cost(A, D) = c_s \cdot s + c_r \cdot r$$

kde s je počet sekvenčních přístupů, r je počet náhodných přístupů a c_s, c_r jsou nezáporné reálné konstanty vyjadřující cenu každého sekvenčního a každého náhodného přístupu.

Definice. Nechť $\mathcal{A}$ je třída algoritmů řešící popsaný problém a nechť $\mathcal{D}$ je třída databází. ($\mathcal{A}$ nemusí být nutně třída všech algoritmů, které korektně řeší danou úlohu. Stejně tak $\mathcal{D}$ nemusí být třída všech databází vyhovujících našemu zadání.) Algoritmus A je *instančně optimální nad* $\mathcal{A}$ *a* $\mathcal{D}$, jestliže $A \in \mathcal{A}$ a jestliže existují konstanty c, c' takové, že pro každé $B \in \mathcal{A}$ a každé $D \in \mathcal{D}$ platí

$$cost(A, D) \leq c \cdot cost(B, D) + c'.$$

Konstantu c nazýváme *koeficient optimality.*

Instanční optimalita je optimalita v poměrně silném významu. Vyžaduje, aby algoritmus byl až na multiplikativní konstantu nejrychlejší pro libovolnou instanci databáze. Je to tedy silnější požadavek než optimalita v nejhorším či průměrném případě.

Definice. O monotónní agregační funkci t řekneme, že je *striktní*, jestliže $t(x_1, \ldots, x_m) = 1$, právě když $x_1 = x_2 = \cdots = x_m = 1$.

Věta 1. *Nechť agregační funkce t je monotónní. Nechť $\mathcal{D}$ je třída všech databází vyhovujících zadání úlohy a $\mathcal{A}$ je třída všech algoritmů, které tuto úlohu korektně řeší a které nečtou nikdy položku (R, x_i) náhodným přístupem, pokud pro tento objekt R již dříve nepřečetly sekvenčně (R, x_j) pro nějaké j. Pak TA je instančně optimální nad $\mathcal{A}$ a $\mathcal{D}$. Když je funkce t navíc striktní, pak žádný instančně optimální algoritmus nad $\mathcal{A}$ a $\mathcal{D}$ nemá lepší koeficient optimality než TA. Pro TA má tento koeficient hodnotu $m + m(m-1)c_r/c_s$. Konstanty c_s a c_r vyjadřují po řadě dobu každého sekvenčního přístupu a dobu každého náhodného přístupu.*

Důkaz tohoto tvrzení zde uveden nebude. Lze jej najít v dokumentu [8]. Tam je velmi podrobná analýza algoritmu TA obsahující mimo jiné i další postačující podmínky pro instanční optimalitu.

Algoritmus TA má celou řadu modifikací. Zde si popíšeme dvě z nich, které jsou užitečné pro slévání výsledkových listin metavyhledávačem. Podmínka, kdy TA skončí, je příliš silná. K nalezení několika nejvýznamnějších objektů je potřeba často prohledat značnou část seznamů $L_1, \ldots, L_m$. Místo

toho je možné použít aproximační verzi algoritmu TA. Snížíme počet přístupů za cenu toho, že objekty vrácené k-tice budou mít vysokou celkovou významnost, leč ne nutně nejvyšší. Druhou nutnou modifikací je odstranění náhodných přístupů. Webové vyhledávače vracejí seznam výsledků setříděný sestupně dle ranku, ale nelze získat (jinak než sekvenčním prohledáváním tohoto seznamu) rank pro daný dokument. Ronald Fagin v článku [7] rozebírá ještě další varianty (například situaci, kdy náhodný přístup je možný, ale výrazně dražší než sekvenční, nebo naopak případ, kdy nelze použít přístup sekvenční), kterými se zde ale nebudeme zabývat, protože nemohou pro metavyhledávač dost dobře najít uplatnění.

Modifikace základní verze algoritmu TA na jeho aproximační variantu je vcelku triviální. Nejprve upravíme formální specifikaci naší úlohy. Mějme reálné číslo $\theta > 1$. Naším cílem bude nalézt θ-aproximaci nejvýznamnějších k objektů, tj. takovou množinu Y, která obsahuje dvojice $(R, t(R))$ a která splňuje následující vlastnost

$$\forall R, R' : ((R, t(R)) \in Y \ \& \ (R', t(R')) \notin Y) \Rightarrow \theta t(R) \geq t(R').$$

Zápis $t(R)$ přitom zkracuje $t(x_1, \ldots, x_m)$, kde $x_1, \ldots, x_m$ jsou hodnoty atributů pro objekt R. Aproximační verzi algoritmu TA označme TA_θ. Jediná změna, kterou musíme provést je změnit pravidlo, kdy ukončujeme výpočet. TA zastaví, jestliže máme k objektů, jejichž celková významnost je alespoň τ. Oproti tomu TA_θ zastaví ve chvíli, kdy nalezneme k objektů s celkovou významností aspoň τ/θ. I tady evidentně platí, že algoritmus TA_θ korektně nalezne θ-aproximaci nejvýznamnějších k objektů, pokud agregační funkce t je monotónní.

Dále si ukážeme variantu algoritmu TA, který používá pouze sekvenční přístup k prvkům seznamů. V základní verzi algoritmu, jakmile najdeme hodnotu jednoho atributu objektu, dohledáme pomocí náhodného přístupu okamžitě i hodnoty ostatních atributů. To nám umožňuje okamžitě spočítat celkovou významnost objektu. Máme-li k dispozici jen sekvenční přístup, zůstávají hodnoty některých atributů objektu neznámé. Tyto hodnoty se postupně doplňují, jak sekvenčně procházíme seznamy. I když pro daný objekt R hodnoty všech atributů neznáme, můžeme je odhadnout. Připomeňme, že hodnota každého atributu je z intervalu $\langle 0, 1 \rangle$. Dolním odhadem tedy bude 0. Předpokládejme dále, že pro objekt R ještě neznáme hodnotu i-tého atributu. Poslední přečtený prvek seznamu L_i nechť je $(R', {x_i}')$. Vzhledem k tomu, že seznamy jsou setříděné, můžeme ${x_i}'$ použít jako horní odhad i-tého atributu pro R. Jestliže agregační funkce t je monotónní, snadno odvodíme i dolní a horní odhad pro celkovou významnost objektu. Tato přibližná informace nám může dovolit určit k-tici nejvýznamnějších objektů v databázi, aniž bychom jejich celkovou významnost znali.

Před formálním popisem algoritmu zavedeme některá značení. Mějme objekt R. Množinu indexů atributů, jejichž hodnoty pro R jsou známé

označme $S(R)$, tj. $S(R) = \{i_1, i_2, \ldots, i_l\} \subseteq \{1, \ldots, m\}$. Definujme $W_S(R)$ jako nejnižší možnou hodnotu $t(R)$ a $B_S(R)$ bude nejvyšší možná hodnota. Je-li z kontextu zřejmé, jak vypadá množina $S(R)$, můžeme psát zkráceně $W(R)$ a $B(R)$. Dále označme $\underline{x}_i$ poslední hodnotu i-tého atributu přečtenou v seznamu L_i. Nechť například $S(R) = \{1, \ldots, l\}$. Jestliže je agregační funkce t monotónní, bude platit

$$\begin{aligned} W_S(R) &= t(x_1, x_2, \ldots, x_l, 0, \ldots, 0), \\ B_S(R) &= t(x_1, x_2, \ldots, x_l, \underline{x}_{l+1}, \ldots, \underline{x}_m). \end{aligned}$$

Upravený algoritmus TA, který používá pouze sekvenční přístup, pak může vypadat takto.

1. Paralelně čteme hodnoty prvků všech seznamů $L_1, \ldots, L_i$ pomocí sekvenčního přístupu. Pro každý objekt R průběžně počítáme hodnoty $W(R)$ a $B(R)$. Udržujeme si množinu T_k, která obsahuje k objektů, jenž mají v současné chvíli nejvyšší hodnotu $W(R)$. Má-li několik objektů stejnou hodnotu $W(R)$, dáváme do množiny T_k přednostně objekty s vyšší $B(R)$. Je-li i tato hodnota stejná, volíme mezi objekty libovolně. Pokud jsme zatím nalezli méně než k objektů, obsahuje množina T_k právě všechny dosud nalezené objekty.

2. Výpočet zastavíme ve chvíli, kdy jsou splněny následující podmínky.

   - Množina T_k obsahuje alespoň k prvků.
   - Pro každý objekt $R \in T_k$ a každý objekt $R' \notin T_k$ platí $W(R) \geq B(R')$. Jestliže jsme pro nějaký objekt R' zatím nenalezly žádnou dvojici $(R', x_i{}')$ při sekvenčním průchodu seznamů, pak předpokládáme $B(R') = t(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_m)$.

3. Na konci obsahuje množina T_k řešení úlohy. (Řešení tvoří pouze množina nejvýznamnějších objektů. Základní verze algoritmu oproti tomu dovolovalo snadno pro každý takový objekt vrátit i jeho celkovou významnost.)

Obě varianty lze samozřejmě zkombinovat. Máme-li k dispozici pouze sekvenční přístup a hledáme-li θ-aproximaci k nejvýznamnějších objektů, ukončíme výpočet, jakmile platí

$$\forall R \in T_k \forall R' \notin T_k : W(R) \geq B(R')/\theta.$$

Postačující podmínkou korektnosti je opět monotonie agregační funkce. Podobně jako u základní verze TA lze i pro uvedené modifikace algoritmu vyslovit analogická tvrzení o instanční optimalitě.

2.4 Zpětná vazba

Vyhledávání dokumentů je proces, který dotazu uživatele přiřazuje nejrelevantnější dokumenty. Je ovšem prakticky nemožné přesně popsat, jak takové dokumenty najít. Míra relevance dokumentu se navíc může lišit od uživatele k uživateli. Vyhledávání dokumentů je proto nutně nepřesné a snaží se pouze přiblížit ideálnímu chování. Pro tento záměr je samozřejmě důležité zvolit rozumný algoritmus, který vybere dokumenty pro daný dotaz. Je ovšem možné jít ještě dál. Když vyhledávač či metavyhledávač vrátí uživateli výsledkovou listinu, může zjistit, jak je uživatel s touto odpovědí spokojen. Z této reakce uživatele se může systém poučit a využít tuto informaci pro budoucí hledání. Tento proces nazýváme zpětná vazba.

Některé níže popsané algoritmy používají tzv. tezaurus, což je slovník obsahující vztahy mezi slovy pro určitý jazyk. Mezi takové vztahy patří nejčastěji synonymita nebo relace obecný - speciální výraz.

Je mnoho způsobů, jak zpětnou vazbu realizovat. Tato kapitola podává přehled o tom, jak můžeme zpětnou vazbu rozdělit na několik typů podle různých kritérií. Zmíněné metody (pokud nebude v textu řečeno jinak) jsou navrženy pro vyhledávače. Některé jsou snadno aplikovatelné i pro metavyhledávač, někdy je nutné provést úpravy a některé techniky použitelné pro vyhledávače jsou bohužel v rámci metavyhledávače z principu nerealizovatelné.

Jako kritérium pro klasifikaci zpětné vazby můžeme použít fakt, zda uživatel hodnotil výsledky vrácené vyhledávačem kladně či záporně.

Pozitivní zpětnou vazbou rozumíme kladné hodnocení některých vrácených dokumentů. Dostaneme-li kladné hodnocení jednoho konkrétního dokumentu, můžeme takový dokument považovat za relevantní vzhledem k položenému dotazu, možná i k některým podobným dotazům. Na tuto skutečnost reagujeme třeba tím, že pro daný dokument zvýšíme relevanci těch termů, které uživatel v dotazu použil. Další možností je modifikace dotazu. Lze například do dotazu přidat ty termy, které jsou nejrelevantnější pro pozitivně hodnocený dokument.

Negativní zpětná vazba analogicky znamená záporné hodnocení některého dokumentu nebo stavu, kdy vyhledávač žádný relevantní dokument pro zadaný dotaz najít nedokázal. Podobně můžeme i reagovat snížením významnosti termů dokumentu nebo změnou dotazu. V tomto případě je možné upravit dotaz nahrazením konkrétních pojmů obecnějšími za použití tezauru.

Kombinovaná zpětná vazba je současné použití obou předchozích technik. V praxi se používá obvykle zpětná vazba pozitivní nebo právě kombinovaná. Negativní zpětná vazba sama o sobě nedosahuje příliš kvalitních výsledků.

Dále je pro zpětnou vazbu důležité rozhodnout, jakým způsobem získáme od uživatele hodnocení vrácené odpovědi. Můžeme rozlišit dva přístupy.

Explicitní hodnocení uživatelem předpokládá aktivní spolupráci uživatele na sběru zpětné vazby. Očekává se od něj, že zadá vyhledávači své hodnocení toho, jak je s konkrétními dokumenty ve výsledkové listině spokojen či nespokojen. Tento přístup je jednoduchý a přímočarý, leč těžko můžeme předpokládat, že typický uživatel bude mít dostatek trpělivosti pro takovou spolupráci.

Analýza chování uživatele naopak už žádnou další aktivitu od uživatele nepožaduje. Tato metoda je obvykle založena na sledování toho, které dokumenty si uživatel vyžádal (což prakticky znamená, že v prohlížeči klikl na hypertextový odkaz na dokument). Vzhledem k tomu, že součástí hitu (tedy položky výsledkové listiny) je i popisek (snippet), je pravděpodobné, že si dokumenty neprohlíží náhodně a má už o dokumentu přibližnou představu dříve, než si vyžádá jeho kompletní obsah. Obvykle systém chápe takto vybrané dokumenty jako relevantní. Na druhou stranu, projde-li uživatel velké množství dokumentů, může to znamenat, že se mu nedaří žádný vhodný najít.

Klíčovým je rozhodnutí, jak dlouho si bude vyhledávač pamatovat hodnocení, které mu poskytl uživatel.

Iterativní dotazování je metoda, která takové hodnocení udržuje pouze po dobu jednoho sezení uživatele. Uživatel zadá dotaz, dostane odpověď a kladně či záporně ohodnotí (obvykle explicitně) dokumenty ve výsledkové listině. Na základě tohoto hodnocení vyhledávač modifikuje dotaz, vrátí odpověď na takto upravený dotaz a celý proces se opakuje, dokud si uživatel žádá další iterace. Některé techniky provádějí úpravy dotazu zcela automaticky, jiné jsou založené na tom, že se vygeneruje několik variant dotazu a uživatel si jednu z nich vybere. Tento proces nijak neovlivňuje ostatní uživatele, kteří vyhledávač používají. Navíc ve chvíli, kdy skončí poslední iterace, se zapomenou veškeré informace o tom, které dokumenty považoval uživatel za relevantní a které ne. I když bude vyhledávači v budoucnu položen stejný dotaz, začne celý proces opět od začátku.

Perzistence dat, která reprezentují hodnocení odpovědi uživatelem, může vyřešit problém předchozího přístupu. Jestliže budeme uvažovat vektorový model, pak jednoduchou realizací tohoto přístupu může být úprava vektoru, který reprezentuje uživatelem ohodnocený dokument. Veškerá data, která při sběru zpětné vazby uložíme, mohou mít globální charakter, tedy budou ovlivňovat budoucí vyhodnocení dotazů pro všechny uživatele. Druhou možností je sbírat taková data zvlášť pro různé uživatele, přesněji řečeno pro různé uživatelské profily. Tato

schopnost vyhledávače „učit se“ se jeví na první pohled jako výhoda. Někteří autoři však tvrdí, že za určitých okolností může být i na obtíž. Lze si totiž snadno představit situaci, kdy stejný uživatel formuluje při různých sezeních dvakrát tentýž dotaz, ovšem pokaždé bude hledat jinou informaci.

Posledním kritériem, které zde uvedeme, je rozdělení podle využití získaných dat. Nebudeme se tedy zabývat tím, jak hodnocení výsledků uživatelem získáme, ani jak ho uložíme, ale rozdělíme algoritmy podle toho, v jakém místě procesu vyhledávání získané hodnocení využívají.

Zpětná vazba jako pre-filtr zahrnuje metody, které použijí hodnocení relevance dokumentů od uživatele ještě před tím, než k danému dotazu vyhledávají vhodné dokumenty. Jinými slovy sem patří všechny metody, které modifikují dotaz. Modifikace dotazu může znamenat převážení jeho termů, přidání či záměnu termů nebo kombinaci obojího. Pokud nevystačíme pouze s převážením termů, použije se k přidávání a záměně termů tezaurus. Většina algoritmů upravujících dotaz je však založena na vektorovém modelu a právě na převážení termů. Příkladem může být jednoduchá technika, která je založená na myšlence, že i uživatelský profil lze reprezentovat m-rozměrným vektorem p, kde m je počet všech termů a kde jednotlivé složky vektoru p jsou z intervalu $\langle 0, 1\rangle$. Podobně jako pro vektor dotazu či dokumentu platí, že j-tá složka vektoru p představuje významnost j-tého termu pro uživatelský profil. Vysokou hodnotu budou mít právě ty složky vektoru p, které odpovídají termům z oblasti zájmu tohoto uživatele. Označíme-li q původní dotaz položený uživatelem, pak modifikovaný dotaz q' dostaneme jako bod na úsečce spojující p a q. Jinými slovy pro $j \in \{1, \ldots, m\}$ bude $q'_j = \alpha \cdot p_j + (1 - \alpha) \cdot q_j$, kde $\alpha \in \langle 0, 1\rangle$ je parametrem algoritmu. Variantou je nepřidávat do dotazu nové termy, tedy spočítat složku q'_j podle předchozího vzorce, pokud $q_j > 0$, a položit $q'_j = 0$, jestliže $q_j = 0$. Sofistikovanější algoritmy pro převážení termů v dotazu jsou popsány v knize [2].

Zpětná vazba jako ko-filtr obnáší použití získaných dat přímo ve vyhledávacím algoritmu, který přiřazuje k dotazu relevantní dokumenty. Jako ilustraci uvedeme metodu, která se opět opírá o vektorový model. I tady předpokládejme stejný vektor p reprezentující uživatelský profil. Klasický vektorový model odvozuje relevanci dokumentu ze vzdálenosti vektoru tohoto dokumentu od vektoru dotazu. Přidání zpětné vazby jako ko-filtru realizujeme tak, že spočteme nejen vzdálenost dokumentu a dotazu, ale také vzdálenost dokumentu a profilu. Pomocí agregační funkce získáme z těchto dvou vzdáleností jedinou hodnotu určující relevanci dokumentu. Agregační funkcí může být minimum, maximum, součet nebo součin. Před aplikací agregační funkce je možné

každou ze vzdáleností přenásobit pevně zvolenými koeficienty. Výpočet relevance ovlivňuje i metrika použitá pro měření vzdáleností. Více o těchto algoritmech je možné najít v knize [10]. Zde se jimi nebudeme více zabývat, protože metavyhledávač pouze rozesílá dotazy vyhledávačům a sám nemá nad procesem přiřazení dokumentů k dotazu kontrolu. Tato třída algoritmů proto není pro metavyhledávač použitelná.

Zpětná vazba jako post-filtr znamená využít získaná data až ve chvíli, kdy už máme k dispozici výsledkovou listinu. Její položky se přeuspořádají (případně některé úplně odstraní) a teprve takto upravenou listinu prezentuje vyhledávač uživateli.

Selekce vyhledávačů je kategorie zajímavá pouze pro metavyhledávač. Zpětnou vazbu tady použijeme v algoritmu, jenž rozhoduje, kterým vyhledávačům se daný dotaz pošle a kterým nikoli. Jedné takové metodě je věnována celá kapitola 2.5.

2.5 Selekce vyhledávačů

Úkol metavyhledávače vybrat pro dotaz vyhledávače, jimž bude rozeslán, je někdy nazýván *selekční problém* (*database selection problem*). V této kapitole popíšeme algoritmus řešící selekční problém, který byl použitý v metavyhledávači SavvySearch. SavvySearch byl vytvořen na Státní univerzitě Colorado a patří mezi první metavyhledávače (začal fungovat v roce 1995). Později ho pohltil metavyhledávač Search.com, jenž je k nalezení na adrese `http://www.search.com`. Článek [6] podává stručný přehled o algoritmech, které SavvySearch používá. Na rozdíl od většiny ostatních metavyhledávačů existují pro SavvySearch volně dostupné dokumenty popisující velice detailně některé použité algoritmy, architekturu metavyhledávače i některé experimentální výsledky. Navíc jsou zmíněné dokumenty zaměřené právě především na řešení selekčního problému.

SavvySearch při každém dotazu spočte skóre pro všechny podporované vyhledávače. Skóre je reálné číslo, čím vyšší, tím vhodnější je vyhledávač pro daný dotaz. Dotaz se poté rozesílá několika vyhledávačům s nejvyšším skóre.

V iniciálním stavu algoritmu jsou všechny vyhledávače rovnocenné. Jinými slovy pro libovolný dotaz boudou mít všechny vyhledávače skóre stejné. K rozlišení vyhledávačů v závislosti na dotazu dochází až sběrem zpětné vazby.

Klíčovou datovou strukturou pro zaznamenání reakce uživatelů na vracené výsledky a pro vypočtení skóre vyhledávačů je tzv. meta-index. Existuje pouze jedna instance této struktury společná pro všechny vyhledávače. To tedy znamená, že SavvySearch nepodporuje uživatelské profily. Meta-index M je matice typu $n \times m$, kde n je počet podporovaných vyhledávačů

a m je počet všech termů. V i-tém řádku a j-tém sloupci je reálné číslo M_{ij}, které určuje kvalitu odpovědí i-tého vyhledávače na dotazy obsahující j-tý term. Jestliže s je i-tý vyhledávač a t je j-tý term, budeme někdy místo M_{ij} psát pro jednoduchost M_{st}. Prvky matice mohou být vlivem negativní zpětné vazby i záporná čísla. Jejích rozsah ani není omezen žádným intervalem, k dodatečnému normování dochází až při výpočtu skóre.

SavvySearch nepodporuje explicitní hodnocení vrácených dokumentů uživatelem. Výsledková listina se prezentuje jako HTML stránka, kde pro každý hit je možné kliknout na hypertextový odkaz pro zobrazení celého dokumentu. Každé takové kliknutí se zpracuje jako pozitivní zpětná vazba. Negativní zpětná vazba pak spočívá v reakci na situaci, kdy některý z vyhledávačů vrátil prázdnou výsledkovou listinu.

Mějme dotaz q a označme $|q|$ počet různých termů v něm obsažených. Nyní je metavyhledávač v situaci, kdy předložil výsledkovou listinu pro dotaz q uživateli. Pokud si uživatel vyžádal obsah jistého dokumentu, pak se zvýší hodnoty M_{ij} o číslo $1/|q|$ pro všechny dvojice (i, j) takové, že i-tý vyhledávač vrátil ve své výsledkové listině zmíněný dokument a j-tý term je přítomen v dotazu q. Dále se pak zmenší prvek M_{ij} o hodnotu $1/|q|$ pro všechny dvojice (i, j) takové, že i-ty vyhledávač nenašel žádný relevantní dokument pro dotaz q a j-tý term je přítomen v q.

Jestliže byl metavyhledávači zadán dotaz q a je třeba vybrat vyhledávače, kterým bude rozeslán, pak se pro každý vyhledávač s spočte hodnota Q_{sq}, která poslouží jako základ pro výpočet skóre. Tuto hodnotu získáme z následujícího vztahu.

$$Q_{sq} = \sum_{t \in q} \frac{M_{st} \cdot I_t}{\sqrt{T_s}}$$

Suma probíhá přes všechny termy v dotazu. Autoři metavyhledávače SavvySearch modelují dotaz jako množinu termů (a chápou ho jako konjunkci všech jeho termů). Dodejme ještě, že s tímto pojetím nevystačíme, pokud se rozhodneme podporovat fráze. Číslo I_t nazýváme *inverzní frekvence termu ve vyhledávačích*. Tato hodnota má zvýhodnit termy, pro které dává kvalitní odpovědi jen několik málo vyhledávačů. Představuje analogii k inverzní frekvenci termu v dokumentech známé z indexace dokumentů ve vektorovém modelu. Označíme-li S množinu všech podporovaných vyhledávačů, pak se hodnota I_t spočte jako

$$I_t = \log \frac{|S|}{|\{s \in S; M_{st} > 0\}|}.$$

Když budeme řádek meta-indexu pro vyhledávač s chápat jako vektor, pak T_s označuje jeho velikost měřenou L_1 metrikou. Můžeme tedy psát

$$T_s = \sum_{t \in T} |M_{st}|.$$

T značí množinu všech termů. Použitím T_s ve vzorci pro výpočet Q_{sq} docílíme toho, že nebudou příliš zvýhodněné vyhledávače, pro které máme v meta-indexu velké množství informací.

Samotná hodnota Q_{sq} ještě není skóre vyhledávače. SavvySearch toto číslo ještě normuje a odečítá penalizace za nízký počet hitů ve výsledkové listině vrácené vyhledávačem a za dlouhou dobu odezvy. Tyto dva záporné příspěvky do skóre jsou vypočteny z několika posledních dotazů, odeslaných příslušnému vyhledávači, ale nejsou závislé na dotazu, pro který se selekce provádí.

Nechť h_s je průměrný počet položek ve výsledkové listině pro posledních l dotazů odeslaných vyhledávači s. Dále t_s označme průměrný čas odpovědi pro posledních l dotazů odeslaných vyhledávači s. Označme P_{sh} penalizaci vyhledávače s za nízký průměrný počet hitů ve výsledkové listině a dále označme P_{st} penalizaci za dobu odpovědi. Jestliže průměrný počet hitů ve výsledkové listině vyhledávače s pro posledních l dotazů dosáhl alespoň jisté mezní hodnoty h_{th} (threshold), pak položme $P_{sh} = 0$. Jinak dostaneme P_{sh} ze vztahu

$$P_{sh} = \left(\frac{h_{th} - h_s}{h_{th}} \right)^2.$$

Konstantou t_{to} označme maximální povolenou dobu odpovědi vyhledávače (timeout). Podobným způsobem definujeme i P_{st}. Pokud průměrná doba odpovědi vyhledávače s pro posledních l dotazů nepřesáhla hranici t_{th}, potom bude $P_{th} = 0$. Jinak položme

$$P_{st} = \left(\frac{t_s - t_{th}}{t_{to} - t_{th}} \right)^2.$$

Za t_s, t_{to} a t_{th} dosazujeme časové údaje v sekundách. Konstanty l, h_{th}, t_{to} a t_{th} mají neměnnou hodnotu po celou dobu běhu metavyhledávače. Autoři metavyhledávače SavvySearch volili hodnoty těchto konstant takto:

$$\begin{aligned} l &= 5, \\ h_{th} &= 1, \\ t_{to} &= 45, \\ t_{th} &= 15. \end{aligned}$$

Pro vyhledávač s a dotaz q už nyní můžeme spočítat skóre R_{sq} ze vztahu

$$R_{sq} = \frac{Q_{sq}}{\max_{s' \in S} Q_{s'q}} - P_{st} - P_{sh}.$$

Jak je ze vzorce vidět, hodnoty Q_{sq} jsou ještě normovány tak, aby největší z nich byla rovna jedné. Může se stát, že čísla Q_{sq} budou pro všechny vyhledávače záporná. Normování těchto hodnot podle uvedeného vzorce pak bude dávat trochu nepřirozené výsledky. Krom toho může být maximální Q_{sq}

rovno nule a hodnota zlomku by pak byla nedefinovaná. V dokumentech k metavyhledávači SavvySearch tato situace není ošetřena a lze předpokládat, že v praxi k ní téměř docházet nebude.

Kapitola 3

Návrh

Tato kapitola popisuje komplexní návrh metavyhledávače. V předchozím textu byly popsány algoritmy a techniky, které pokrývají všechny klíčové problémy, jenž musí návrh metavyhledávače řešit. V některých místech ale algoritmy či specifikace nepopisují všechny nezbytné detaily a samozřejmě je možné (a někdy nutné) přidat řadu rozšíření.

Obrázek 3.1 ukazuje celkový pohled na činnost metavyhledávače.

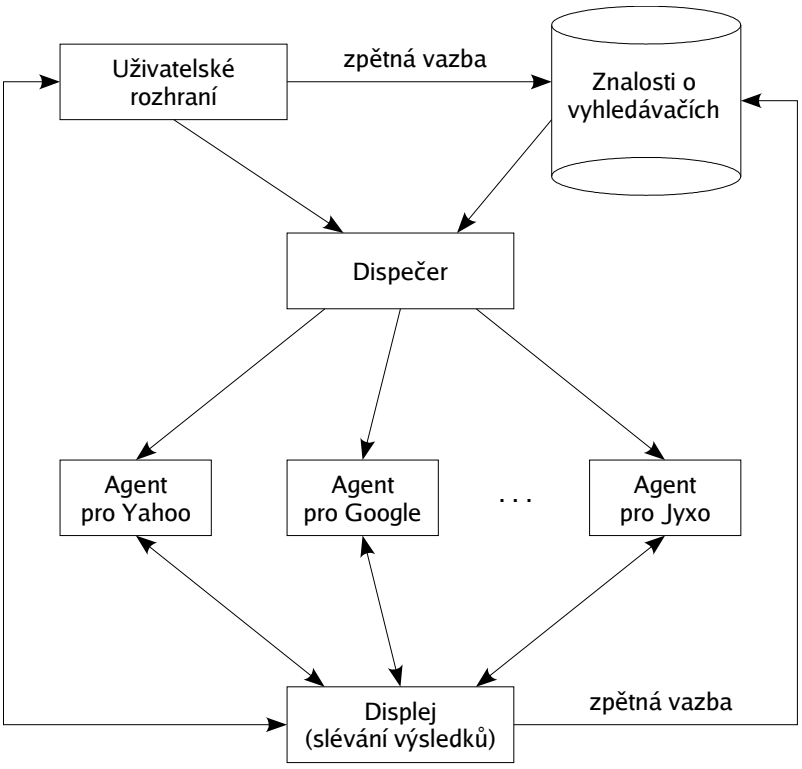
Uživatel zadá metavyhledávači dotaz. Ten jej poté rozparsuje na jednotlivé termy a fráze.

Dále přichází na řadu *dispečer*. Je to část metavyhledávače, která má k dispozici jednak dotaz a jednak informace o vyhledávačích získané sběrem zpětné vazby. Z těchto dvou vstupů provede dispečer selekci vyhledávačů.

Pro každý vyhledávač existuje *agent*, jehož úkolem je odstínit zbytek metavyhledávače od specifických vlastností formátu dotazu a formátu výsledkové listiny daného vyhledávače. Poté, co dispečer vybere vyhledávače, rozešlou těmto vyhledávačům příslušní agenti dotaz. Každý agent pak čeká na odpověď svého vyhledávače a ve chvíli, kdy dostane zpět první stránku výsledkové listiny, tuto stránku rozparsuje a převede položky listiny (hity) na společný formát. Jakmile jsou hity v tomto společném formátu, přechází zodpovědnost za jejich další zpracovaní na komponentu, které má na starost slévání výsledků do jediné listiny. Činnost všech agentů probíhá samozřejmě paralelně.

Již zmíněná část metavyhledávače, které slévá hity produkované několika agenty, se nazývá *displej*.[1] Jejím výstupem je celková výsledková listina, která už může být prezentována uživateli jako odpověď na jeho dotaz. Při slévání výsledků může dojít k situaci, že první stránka výsledků (jedná se o HTML stránku obsahující obvykle 10-20 hitů) od jednoho či více vyhledávačů neposkytuje dostatek informací pro sestavení finální listiny. V takovém případě displej zpětně vyžaduje od agentů, aby získali další stránky výsledků. Pro zvýšení efektivity je možné mezi agenty a displej umístit vy-

[1] tento termín byl použitý v metavyhledávači SavvySearch



Obrázek 3.1: Struktura metavyhledávače

rovnávací paměť. Agenti si tak mohou dopředu připravit několik stránek výsledkové listiny od svého vyhledávače dříve, než o ně bude žádat displej.

Podobně jako displej dostane nejprve od agenta jen začátek výsledkové listiny vyhledávače a její další části získává až na požádání, tak i uživatel má nejprve k dispozici jen několik prvních hitů z celkové listiny. Chce-li uživatel více dokumentů, doručí se tato žádost zpět displeji.

Selekce vyhledávačů je založena na sběru zpětné vazby. Ta zahrnuje reakce uživatele na předloženou výsledkovou listinu, ale i některé další informace, které jsou k dispozici ještě dříve, než je odpověď uživateli doručena. Displej může sledovat, kdy některý z vyhledávačů nenašel žádné relevantní dokumenty a realizovat tak negativní zpětnou vazbu. Každý agent může sbírat informace o době odpovědi vyhledávače (tato skutečnost už není na obrázku 3.1 pro přehlednost zachycena).

Místo, kde systém uchovává hodnocení výsledků uživatelem, je vhodné pro přidání uživatelského profilu. Místo jednoho úložiště sdíleného všemi uživateli lze sbírat zpětnou vazbu odděleně pro různé uživatelské profily.

3.1 Zpětná vazba a uživatelské profily

Použití zpětné vazby v metavyhledávači nyní rozeberme podrobněji. Jak již bylo řečeno, mezi zpětnou vazbou a uživatelskými profily je velice úzký vztah. Chceme-li podporovat rozdílné chování metavyhledávače pro různé uživatele, je přímočarým řešením sledovat spokojenost uživatele s výsledky a dynamicky měnit způsob vyhledávání. To, co jsme teď popsali, je ale právě zpětná vazba.

V metavyhledávači použijeme zpětnou vazbu (a tedy i uživatelský profil) pouze pro řešení selekčního problému. Tento přístup byl použitý i v metavyhledávači SavvySearch. Budeme sbírat informace o tom, které dokumenty si uživatel prohlížel a které nikoli. Dále nás bude zajímat rychlost odpovědi vyhledávačů a budeme reagovat také na situace, kdy vyhledávač není schopen k danému dotazu najít žádné relevantní dokumenty.

Ostatní techniky, jak využít zpětnou vazbu, jsou v kontextu metavyhledávače poměrně problematické. Tyto možnosti byly nastíněny v kapitole 2.4.

Metody, které používají zpětnou vazbu jako pre-filtr (tedy k modifikaci dotazu), jsou prakticky vždy založeny na převážení termů. Některé pouze mění váhy termů, jiné používají tuto techniku v kombinaci s přidáváním nových termů, často za pomoci tezauru. Metavyhledávač je ovšem nucen se při formulaci dotazů přizpůsobit rozhraním, která obvykle vyhledávače na Webu nabízejí, a tato rozhraní neumožňují termům v dotazu přiřazovat váhy.

Použití zpětné vazby jako ko-filtru je ze své podstaty úplně mimo možnosti metavyhledávače. Samotný proces, kdy se dotazu přiřazuje seznam relevantních dokumentů je v režii vyhledávačů, kterým je dotaz rozeslán, a metavyhledávač ho sám nemůže dost dobře ovlivnit.

Přeuspořádání hitů ve výsledkové listině (tj. zpětná vazba jako post-filtr) si naopak lze představit i v metavyhledávači. Tato technika byla použitá například v metavyhledávači MetaCrawler. Všechny dokumenty vyskytující se ve výsledkové listině jsou staženy a podrobeny textové analýze. Na základě této analýzy se pak upraví uspořádání hitů a nedostupné dokumenty jsou z výsledkové listiny odstraněny úplně. Celý tento proces probíhá on-line (tj. v době, kdy uživatel čeká na odpověď na svůj dotaz). Ačkoli uvedená metoda může zlepšit kvalitu výstupu, je zjevné, že tato schopnost je vykoupena zvýšením časové náročnosti. Návrh našeho metavyhledávače přeuspořádávání výsledkové listiny nezahrnuje.

Dále je klíčové rozhodnutí, jakým způsobem budeme zpětnou vazbu sbírat. Nebudeme po uživateli žádat, aby explicitně hodnotil kvalitu jednotlivých dokumentů vyskytujících se ve výsledkové listině. Lze předpokládat, že jen malá část uživatelů bude ochotna takovou informaci metavyhledávači poskytovat. Veškeré informace, které při sběru zpětné vazby získáme, budou uloženy a ovlivní vyhledávání dokumentů pro podobné dotazy položené v budoucnu. Alternativou k tomuto přístupu by bylo iterativní vyhledávání,

kde se hodnocení kvality výsledku neuchovává a místo toho metavyhledávač pomáhá uživateli ladit dotaz. Nevýhodou iterativního vyhledávání je opět netrpělivost uživatele, který obvykle nebude chtít provádět více než jednu nebo jen několik málo iterací. Krom toho společným jmenovatelem metod, které nějakým způsobem ladí dotaz, je neschopnost zlepšit přesnost odpovědi a místo toho se snaží pouze zvýšit úplnost. Tyto metody jsou založené na tom, že uživatel označí, které dokumenty v odpovědi metavyhledávače jsou pro něj skutečně relevantní. Další iterace se pak snaží nalézt podobné dokumenty. Typická situace je ale taková, že uživateli stačí libovolný dokument obsahující informaci, kterou hledá. Jinými slovy přesnost je obvykle před úplností upřednostňována. Metavyhledávač tedy nebude iterování dotazu podporovat.

3.2 Modelování dotazu a hitu

Dříve než přistoupíme k popisu jednotlivých komponent, je na místě formálně popsat, jak budeme modelovat vstupy a výstupy metavyhledávače.

Vyhledávače na Webu přijímají dotazy uživatelů ve formě termů a frází spojených logickými spojkami. Uživatelé používají výrazně nejčastěji dotazy, ve který se vyskytuje pouze konjunkce. Navrhovaný metavyhledávač bude podporovat právě jen dotazy tohoto typu. Omezení se pouze na konjunkce termů a frází může na první pohled vypadat příliš restriktivní, nicméně většina v praxi položených dotazů spadá právě do této kategorie. Krom toho některé vyhledávače (např. AOL Search), jimž metavyhledávač dotazy rozesílá, bohatší možnosti nenabízejí.

Definujme *dotaz* jako posloupnost elementů dotazu, kde *elementem dotazu* označujme buď term nebo frázi. Za relevantní dokumenty považujeme ty, které obsahují současně všechny termy a fráze dotazu. Povšimněme si, že elementy dotazu tvoří posloupnost, nikoli množinu. U některých vyhledávačů právě pořadí termů a frází v rámci dotazu hraje roli při uspořádávání dokumentů ve výsledkové listině. Pojem *fráze* označuje posloupnost termů délky alespoň 2. Uvnitř fráze nerozlišujeme velká a malá písmena a typy oddělovačů (mezeru, čárku, středník, pomlčku apod.) jejích termů. Vezměme jako příklad frázi *"meta-search"*. Pak za výskyt této fráze v dokumentu považujeme větu *„See the table of features of meta-search engines.“* stejně jako *„Meta search engines don't crawl the Web themselves.“* Tento přístup používají i vyhledávače na Webu, takže snaha metavyhledávače dbát na použité oddělovače mezi termy by byla zbytečná. *Termy* samozřejmě označují slova přirozeného jazyka, formálně neprázdné sekvence alfanumerických znaků.

Pojmem *hit* budeme rozumět informace o nalezeném dokumentu zahrnující čtyři složky. Těmi složkami jsou URI dokumentu, název, popisek (snippet) a rank. Trojice URI, název a popisek představuje opět skupinu nejzajímavějších atributů pro uživatele a zároveň je „rozumným průnikem“

poskytovaným všemi vyhledávači na Webu.

Rank dokumentu obvykle uživateli prezentován nebývá, ale je samozřejmě užitečný interně pro třídění hitů ve výsledkové listině. Na rozdíl od předchozích tří položek není možné od vyhledávačů rank jednotlivých dokumentů získat. Metavyhledávač proto použije vlastní funkci pro přiřazování ranku jednotlivým položkám výsledkové listiny. Konkrétní funkce a měření, které k její volbě vedlo, jsou popsány v kapitole 5.1. Tuto funkci použijeme jen pro odhad ranků ve výsledkových listinách jednotlivých vyhledávačů. Rank hitu ve výsledkové listině metavyhledávače se vypočte při procesu slévání.

Hity vrácené vyhledávačem jsou uspořádány v nerostoucí posloupnost dle ranku, který je vždy z intervalu $\langle 0, 1 \rangle$. Oproti tomu výsledková listina metavyhledávače tyto vlastnosti nesplňuje. Rank hitu zde může být větší než 1. Nemáme tu ani monotonii ranků. To je způsobeno tím, že algoritmus pro třídění hitů dle ranku (popsaný v kapitole 2.3) je z důvodu efektivity pouze aproximační.

3.3 Dispečer a sběr zpětné vazby

Úkolem dispečera je přijmout na vstupu dotaz uživatele a na základě informací získaných při sběru zpětné vazby rozhodnout, kterým z podporovaných vyhledávačům rozeslat dotaz a kterým nikoli. Jinými slovy v této kapitole podrobně rozebereme, jak bude metavyhledávač řešit selekční problém. Tento návrh vychází z algoritmu použitého v metavyhledávači SavvySearch (popsaném v kapitole 2.5).

SavvySearch se při selekci vyhledávačů opíral o datovou strukturu pojmenovanou meta-index. Meta-index je matice obsahující pro každou dvojici (term, vyhledávač) reálné číslo, které udává vhodnost vyhledávače pro vyhodnocení dotazu obsahujícího daný term. Meta-index je velmi příhodná struktura pro přímočaré přidání podpory uživatelských profilů. V našem případě je však meta-index souborem mnoha matic. Pro každého uživatele existuje zvláštní matice a navíc meta-index obsahuje jednu globální matici, která je pro všechny uživatele společná. (Lze říci, že zmíněná globální matice odpovídá původní podobě meta-indexu z metavyhledávače SavvySearch.) Globální matici je vhodné používat v případech, kdy uživatel nemá ve své profilu nasbíraný dostatek informací pro výběr vyhledávačů.

Dokumentace k metavyhledávači SavvySearch hovoří o meta-indexu jako o matici, ale už se nezabývá jeho reprezentací v paměti počítače. Vícerozměrné pole je samozřejmě nevhodné, protože dopředu neznáme výčet všech termů a podobně můžeme chtít v průběhu činnosti metavyhledávače přidávat i podporu nových vyhledávačů. Matice (a to především matice odpovídající jednotlivým profilům) budou navíc poměrně řídké. Pro reprezentaci meta-indexu proto použijeme hašovací tabulku. Klíčem bude trojice (p, s, t),

kde p označuje uživatelský profil, s vyhledávač a t term. Všechny matice meta-indexu tedy uložíme do jediné hašovací tabulky. Prvky meta-indexu budeme označovat zápisem M_{pst}. Pokud klíč (p, s, t) není v hašovací tabulce přítomen, položíme $M_{pst} = 0$. Symbolem G označme globální profil.

Kdyby byl meta-index tvořen jedinou maticí, jako je tomu u metavyhledávače SavvySearch, lze si představit, že je tato matice celá uložena v operační paměti. Jestliže použijeme speciální matici pro každý uživatelský profil, jsou paměťové nároky na uložení meta-indexu příliš velké na to, aby byl v operační paměti uchováván celý. Proto použijeme variantu hašování známou jako externí hašování (někdy též označovanou jako rozšiřitelné či Faginovo hašování). Její popis je mimo rámec tohoto textu a lze ho nalézt například v knize [13]. Zde se zmíníme jen o konstrukci hašovací funkce. Jejím vstupem je trojice (p, s, t), která se zobrazí na posloupnost 64 bitů. Prvních 32 bitů je odvozeno z profilu p, dalších 24 bitů z termu t a posledních 8 bitů z vyhledávače s. Při výběru vyhledávačů pro vyhodnocení konkrétního dotazu je třeba přečíst z meta-indexu několik hodnot. Pro jeden dotaz se tyto hodnoty vážou vždy k jedinému profilu a jen k několika málo termům (právě k těm termům, jenž se vyskytují v dotazu). Klíče pro uvažované hodnoty se tedy zahašují na řetězce bitů, které budou mít společný prefix. Tato vlastnost vede k snížení počtu přístupů na disk. Podobně se chová i zápis do meta-indexu při sběru zpětné vazby.

Předpokládejme, že metavyhledávač vrátil uživateli s profilem p výsledkovou listinu na jeho dotaz q, a předpokládejme, že tato listina obsahuje dokument d. Jestliže si uživatel vyžádá celý obsah dokumentu d, pak se pro každý vyhledávač s, který při vyhodnocování dotazu q vrátil dokument d ve své výsledkové listině, a pro každý term t obsažený v dotazu q zvýší každý z prvků meta-indexu M_{Gst} a M_{pst} o hodnotu $r_{sd}/|q|$. Přitom $|q|$ označuje počet termů v dotazu q a r_{sd} je rank, který dokumentu d přiřadil[2] vyhledávač s při vyhodnocování dotazu q. Jestliže vyhledávač s nenašel k dotazu q žádné relevantní dokumenty, pak se pro každý term t obsažený v q prvek M_{Gst} sníží o $1/|q|$.

Uvedené úpravy meta-indexu jsou analogií k úpravám, jenž používal SavvySearch. Přesto jsou tu rozdíly. Vzhledem k tomu, že chceme podporovat uživatelské profily, měníme jako součást pozitivní zpětné vazby nejen hodnotu M_{Gst}, ale také M_{pst}. Naopak u negativní zpětné vazby nijak nemodifikujeme prvky meta-indexu vztahující se ke konkrétnímu profilu. Zřejmě hodnocení relevance jednotlivých dokumentů ve výsledkové listině je subjektivní názor konkrétního uživatele. Oproti tomu zjištění, že vyhledávač pro jistý dotaz není schopen vrátit žádné dokumenty, se subjektivitou pohledu uživatele nijak nesouvisí. Dále je rozdíl v tom, že u pozitivní zpětné

[2] Ve skutečnosti se metavyhledávač od jednotlivých vyhledávačů ranky dokumentů nedozví, takže by bylo přesnější mluvit o ranku, který metavyhledávač odhadne tak, jak je popsáno v kapitole 3.2.

vazby navýšíme hodnoty prvků meta-indexu o $r_{sd}/|q|$, a nikoli o $1/|q|$. SavvySearch pouze rozlišuje mezi vyhledávači, které daný dokument ve své výsledkové listině uvedli a které ne. Nicméně je poměrně přirozené zvýhodnit ty vyhledávače, jenž dokument zařadily blízko začátku listiny, tedy daly mu vysoký rank. Proto se prvky meta-indexu zvětšují o číslo úměrné právě ranku. Dodejme ještě, že i tato metoda se dopouští jistých nepřesností. Předběžně lze říci, že při slévání výsledkových listin jednotlivých vyhledávačů je použito aproximační verze algoritmu TA (viz kapitola 2.3). Na výsledky pomalejších vyhledávačů se proto za určitých okolností vůbec nečeká. I kdyby obsahovaly dokumenty, jenž uživatel označí za relevantní, příslušné změny v meta-indexu se samozřejmě neprovedou.

Stejně jako SavvySearch budeme provádět selekci na základě skóre, které spočteme pro každý vyhledávač, a stejně jako v metavyhledávači SavvySearch bude hlavní složka tohoto skóre vypočtena z hodnot obsažených v meta-indexu. Předpokládejme, že uživatel s profilem p zadal metavyhledávači dotaz q. Zmíněnou složku skóre pro vyhledávač s spočteme ze vztahu

$$Q'_{psq} = \sum_{t \in q} \left(\alpha_{pt} \cdot \frac{M_{pst} \cdot I_{pt}}{\sqrt{T_{ps}}} + (1 - \alpha_{pt}) \cdot \frac{M_{Gst} \cdot I_{Gt}}{\sqrt{T_{Gs}}} \right),$$

kde

$$\begin{aligned} I_{pt} &= \log \frac{|S|}{\max\{\,|\{s \in S; M_{pst} > 0\}|\,, 1\}}, \\ T_{ps} &= \sum_{t \in T} |M_{pst}|. \end{aligned}$$

Přitom S označuje množinu všech podporovaných vyhledávačů a T množinu všech termů. Zápis $t \in q$ byl použit pro jednoduchost, ačkoli není formálně zcela korektní. Značíme tak, že suma probíhá přes všechny termy, ať už se vyskytují v dotazu q samostatně nebo jako součást fráze. Pro úplnost ještě dodejme, že v situaci, kdy $T_{ps} = 0$ nebo $T_{Gs} = 0$, nahradíme ve vzorci pro Q'_{psq} příslušný zlomek nulou.

Analogické vzorce byly použity v metavyhledávači SavvySearch. Rozdíl spočívá pouze v tom, že zde tvoří každý člen sumy ve vzorci pro výpočet Q'_{psq} dvě složky. První složka odpovídá informacím specifickým pro uživatelský profil p, druhá je odvozena z hodnot globální matice meta-indexu. Příspěvek pro každý term je vyjádřen jako vážený součet obou složek, kde váhy určuje číslo $\alpha_{pt} \in \langle 0, 1 \rangle$. Koeficienty α_{pt} jsou závislé na konkrétním termu a uživatelském profilu. Je vhodné, aby profilová složka hrála pokud možno velkou roli a odrážela tak subjektivní názor uživatele. Globální složka představuje především východisko ze situace, kdy profilová část obsahuje příliš málo informací na to, aby věrohodně vystihovala kvalitu jednotlivých vyhledávačů pro daný term. Na této úvaze je založen vzorec pro výpočet α_{pt}.

Označme nejprve

$$\beta_{pt} = |\{s \in S; M_{pst} \neq 0\}|.$$

Potom definujeme

$$\alpha_{pt} = \begin{cases} \alpha_{max} \cdot (\beta_{pt}/\beta_{Gt}) & \text{pro } \beta_{pt} \neq 0 \ \& \ \beta_{Gt} \neq 0 \ \& \ \beta_{pt}/\beta_{Gt} < 1, \\ \alpha_{max} & \text{pro } \beta_{pt} \neq 0 \ \& \ \beta_{Gt} \neq 0 \ \& \ \beta_{pt}/\beta_{Gt} \geq 1, \\ 1 & \text{pro } \beta_{pt} \neq 0 \ \& \ \beta_{Gt} = 0, \\ 0 & \text{pro } \beta_{pt} = 0. \end{cases}$$

Definice α_{pt} v případě, kdy $\beta_{pt} = 0$ (resp. $\beta_{Gt} = 0$), triviálně řeší situaci, kdy hodnoty meta-indexu vztahující se k profilu p (resp. hodnoty obsažené v globální matici meta-indexu) nemají vůbec žádnou schopnost rozlišit mezi vyhledávači pro daný term. V ostatních případech nabývá α_{pt} hodnot z intervalu $\langle 0, \alpha_{max} \rangle$. V nejjednodušším případě lze položit $\alpha_{max} = 1$. Použijeme-li jinou hodnotu z intervalu $\langle 0, 1 \rangle$, stanovíme maximální váhu profilové složky. Z definice α_{pt} a Q'_{psq} snadno nahlédneme, že zavedení minimální hodnoty α_{pt} by nedávalo dost dobrý smysl. Autorem této práce byla zvolena konstanta $\alpha_{max} = 0,85$. Je bohužel poměrně obtížné navrhnout a realizovat měření, podle kterého bychom našli její optimální hodnotu.

Předpokládejme, že již máme pro dotaz q a uživatelský profil p spočtená čísla Q'_{psq}. Tyto hodnoty ještě dále upravíme. Definujme nejprve Q'_{max} jako

$$Q'_{max} = \max_{s \in S} Q'_{psq}.$$

Nyní můžeme spočítat hodnoty Q_{psq} podle následujícího vztahu.

$$Q_{psq} = \begin{cases} Q'_{psq}/Q'_{max} & \text{pro } Q'_{psq} > 0 \\ Q'_{psq} & \text{pro } Q'_{psq} \leq 0 \end{cases}$$

Podobné normování provádí i metavyhledávač SavvySearch. Rozdíl je pouze v tom, že v našem případě ponecháme beze změny záporné hodnoty Q'_{psq}. K situaci, kdy $Q'_{psq} < 0$, nedochází v praxi příliš často. Pokud už k ní dojde, znamená to, že příslušný vyhledávač vracel v minulosti na podobné dotazy velmi špatné odpovědi, často dokonce prázdné výsledkové listiny. V takovém případě hodnotu Q'_{psq} číslem Q'_{max} nedělíme a ponecháme $Q_{psq} = Q'_{psq}$, abychom nevhodnost tohoto vyhledávače pro vyhodnocování dotazu ještě zvýraznili.

Při výpočtu Q'_{psq} potřebujeme hodnoty T_{ps}, $|\{s \in S; M_{pst} > 0\}|$ a $|\{s \in S; M_{pst} \neq 0\}|$, k jejichž získání dle definice je třeba projít celý řádek či sloupec jisté matice meta-indexu. Pro efektivní výpočet je vhodné tyto hodnoty udržovat pod speciálními klíči v hašovací tabulce reprezentující meta-index.

Kromě Q_{psq} spočteme ještě pro každý vyhledávač penalizaci P_s za délku doby odpovědi. Označme t_s průměrnou dobu, která uplynula od odeslání HTTP požadavku vyhledávači s a přijetím HTTP odpovědi, za posledních

l požadavků, jenž byly vyhledávači s odeslány. Nepřekročí-li t_s jistou mezní hodnotu t_{th}, položíme $P_s = 0$. V ostatních případech získáme P_s ze vztahu

$$P_s = \left(\frac{t_s - t_{th}}{t_{to} - t_{th}} \right)^2.$$

Přitom t_{to} značí maximální dobu (timeout), která může uplynout od odeslání HTTP požadavku do přijetí příslušné HTTP odpovědi. Hranice t_{th} hraje roli minimální odezvy, tj. času, který potřebují k odpovědi i nejrychlejší vyhledávače a který se tudíž do penalizace nezapočítává. Jedná se o stejný vzorec, který byl použit v metavyhledávači SavvySearch. Rozdílná je ovšem volba hodnot jednotlivých konstant. Časy t_{to} a t_{th} jsou uvedeny v sekundách.

$$\begin{aligned} l &= 15, \\ t_{to} &= 5, \\ t_{th} &= 0,1. \end{aligned}$$

Tyto hodnoty vycházejí z měření popsaného v kapitole 5.4.

Poznamenejme, že měření penalizace P_s na základě rychlosti odezvy na HTTP požadavek v sobě skrývá jistou nepřesnost. Jeden HTTP dotaz odpovídá jedné stránce výsledkové listiny a jednotlivé vyhledávače se samozřejmě liší v počtu hitů na stránce. Na druhou stranu vydělíme-li dobu odezvy jednoduše počtem hitů na stránce, dopouštíme se jen jiného typu nepřesnosti. Při procesu slévání výsledkových listin potřebujme často několik dalších hitů od jistého vyhledávače, ne však nutně celou stránku. Krom toho není zcela zřejmé, které hity můžeme označit za nepotřebné, protože mohou být užitečné později, vyžádá-li si uživatel dodatečně od metavyhledávače další část finální výsledkové listiny. Zůstaňme tedy u výpočtu P_s na základě rychlosti odezvy na HTTP požadavek pro jeho jednoduchost a také kvůli skutečnosti, že většina vyhledávačů vrací 10 hitů na stránku.[3]

Metavyhledávač SavvySearch ještě navíc počítal další penalizaci P_{sh} v případech, kdy vyhledávač s vrátil ve výsledkových listinách na několik posledních dotazů v průměru méně než jeden hit. (viz kapitola 2.5) Podobnou zápornou složku skóre však vynecháváme, protože toto znevýhodnění některých vyhledávačů se jeví jako duplicitní. Jestliže vyhledávač příliš často nevrací žádné hity, je to buď proto, že není skutečně schopen žádný relevantní dokument k dotazu uživatele najít, nebo to může také znamenat, že se vyskytly potíže v síťové komunikaci s tímto vyhledávačem. První případ je již ošetřen snížením jistých hodnot v meta-indexu. Tento typ negativní zpětné vazby je navíc vhodnější tím, že se váže konkrétně k termům, které se vyskytovaly v dotazu. V druhém případě obvykle vyprší časový limit pro odpověď vyhledávače, což se projeví vysokou hodnotou P_s.

[3] Výjimkou je například Jyxo či Morfeo.

Celkové skóre vyhledávače s pro uživatelský profil p a dotaz q dostaneme jednoduše jako

$$R'_{psq} = Q_{psq} - P_s.$$

SavvySearch v tomto okamžiku dle hodnoty R'_{psq} (přesněji řečeno dle hodnoty, která byla analogií k R'_{psq}) všechny vyhledávače setřídil v nerostoucí posloupnost. Poté posloupnost v určitém místě rozdělil na dvě části. Vyhledávačům na začátku posloupnosti byl dotaz rozeslán, vyhledávačům na konci nikoli. Nabízí se ovšem možnost u vyhledávačů, kterým dotaz odeslán bude, navíc použít hodnotu R'_{psq} jako váhu, jenž rozhodne, jak moc jejich příspěvek ovlivní celkovou výsledkovou listinu. Hodnoty R'_{psq} je tedy vhodné ještě jistým způsobem znormovat, aby mohly sloužit jako váhy. Pro každý vyhledávač proto spočteme

$$R_{psq} = \frac{R'_{psq} - \min_{s' \in S} R'_{ps'q}}{\max_{s' \in S} R'_{ps'q} - \min_{s' \in S} R'_{ps'q}} \cdot (1 - R_{min}) + R_{min}.$$

V případě, že si jsou hodnoty R'_{psq} rovny pro všechny vyhledávače (a zlomek v předchozím vzorci tudíž není definován), položme $R_{psq} = 1$ pro všechna s. Čísla R_{psq} jsou z intervalu $\langle R_{min}, 1 \rangle$. Hodnota konstanty určující váhu vyhledávače s nejnižším skórem R'_{psq} byla zvolena $R_{min} = 0,7$. Využití takto získaných vah R_{psq} bude ještě rozebráno při výkladu slévání výsledkových listin (tj. v kapitole 3.5). Volbou hodnoty R_{min} se zabývá kapitola 5.2.

Je zřejmé, že hodnoty R_{psq} i R'_{psq} dávají stejné uspořádání vyhledávačů. Jakmile máme toto uspořádání, vybere dispečer několik prvních vyhledávačů a těm dotaz rozešle. Počet vyhledávačů, který vzejde z této selekce není pevně dán žádnou konstantou. Lze umožnit uživateli, aby sám podle potřeby určil, kolik vyhledávačů bude vybráno. Tento přístup je použitý metavyhledávačem SavvySearch.

Kvalita odpovědí jednotlivých vyhledávačů se může v průběhu času měnit. Je proto vhodné podporovat stárnutí meta-indexu, tj. posílit význam informací získaných při sběru zpětné vazby v nedávné době oproti informacím starším. Jednoduchý způsob, jak toho dosáhnout, je přenásobit vždy po uplynutí doby t_{aging} koeficientem c_{aging} všechny hodnoty meta-indexu. Je obtížné navrhnout a realizovat měření, které by našlo ideální hodnoty těchto konstant. Dle úsudku autora této práce byla jako interval stárnutí t_{aging} zvolena doba 30 dní a volba koeficientu stárnutí byla $c_{aging} = 0,95$. Závěrem poznamenejme, že SavvySearch stárnutí meta-indexu neimplementuje.

3.4 Agent pro vyhledávač

V metavyhledávači existuje jeden agent pro každý podporovaný vyhledávač. Úkolem této komponenty je pro dotaz uživatele získat od příslušného vyhledávače výsledkovou listinu. Ta se skládá z hitů, tedy čtveřic (URI, název, popisek, rank). Rank vyhledávače neposkytují, takže ho metavyhledá-

vač jednotlivým dokumentům přiřazuje sám, jak bylo popsáno v kapitole 3.2. Dále se tedy už touto složkou nebudeme zabývat.

Agent komunikuje s vyhledávačem pomocí HTTP protokolu. (HTTP protokol je specifikován v RFC 2616 [14].) Simuluje tak chování uživatele, který pokládá vyhledávači dotaz přes standardní webové rozhraní. Jedná se o stejný přístup, jaký používá pro vyhledávání prohlížeč Mozilla (více o řešení použitém v Mozille v kapitole 2.2). Pro podobné situace existují ještě speciální rozhraní, která umožňují pohodlnější komunikaci s vyhledávačem. Tuto možnost nabízí například Google, ale bohužel se jedná spíše o výjimku. Zvolený postup, tedy použití webového rozhraní, je oproti tomu použitelný pro libovolný vyhledávač na Webu.

Pro zadaný dotaz pošle agent vyhledávači požadavek (HTTP request). V tomto požadavku nás bude zajímat pouze URI, ostatní složky nejsou pro naše účely podstatné. Od vyhledávače dostane agent odpověď (HTTP response). V této odpovědi se můžeme víceméně omezit jen na tělo zprávy[4] (message body). To obsahuje jednu stránku výsledkové listiny v HTML kódu. Můžeme se tedy omezit na to, jak k dotazu zkonstruovat zmíněné URI a jak rozparsovat HTML stránku v odpovědi na posloupnost hitů. Bude-li se dále mluvit o formátu vyhledávače, máme na mysli právě strukturu tohoto URI a této HTML stránky.

V zásadě existuje několik možností, jak pro daný vyhledávač zkonstruovat k dotazu URI a jak zpracovat vrácený HTML kód.

- Každý agent může být modul metavyhledávače. Přímo v programovacím jazyku implementuje vždy zvláštní komponentu pro každý vyhledávač. Toto řešení není ale příliš flexibilní. Vyžaduje příliš mnoho práce a znalost použitého programovacího jazyka ve chvíli, kdy chceme přidat podporu pro další vyhledávač nebo jsme nuceni aktualizovat již existujícího agenta kvůli změně formátu na straně vyhledávače.

- Můžeme mít zvláštní jazyk pro popis formátu jednotlivých vyhledávačů. Oproti předchozímu řešení se tím jistě flexibilita zlepší. Je ovšem třeba navrhnout dostatečně silný jazyk, který je schopen zachytit rozmanitost formátů různých vyhledávačů.

- Nejcennějším řešením by bylo implementovat pouze jednoho univerzálního agenta. Tato komponenta by měla schopnost automaticky analyzovat webové rozhraní vyhledávače. Problémy s aktualizací by tím odpadly a přidání podpory nového vyhledávače by znamenalo pouze zadání adresy, kde se rozhraní tohoto vyhledávače nachází.

V návrhu metavyhledávače použijeme variantu se zvláštním jazykem pro specifikaci formátu vyhledávače. Implementace každého agenta přímo v pro-

[4] Výjimkou z tohoto zjednodušení je informace o použité znakové sadě, která je specifikována v hlavičce `Content-Type`.

gramovacím jazyku by byla příliš náročná na údržbu. Ke změnám formátu vyhledávače dochází v praxi bohužel relativně často a z toho vyplývá i nutnost časté aktualizace agentů. Vytvořit agenta schopného analyzovat rozhraní vyhledávače je extrémně náročný úkol. Vzhledem k velkému množství odlišností ve formátu jednotlivých vyhledávačů není ani návrh jazyka pro popis tohoto formátu zcela triviální.

3.4.1 Struktura specifikace formátu

Pro popis formátu URI a HTML stránky s výsledkovou listinou vyjdeme z jazyka používaného prohlížečem Mozilla (kapitola 2.2). Připomeňme, že tento zápis připomíná HTML dokument. Je tvořen elementy, které lze do sebe hnízdit a tvořit tak stromovou strukturu. Každý element má své jméno a seznam atributů, kde atribut je dvojice řetězců - klíč a hodnota. Jazyk Mozilly je velmi jednoduchý a některé jeho obraty jsou právě pro svou jednoduchost velmi elegantní. Metavyhledávač má však na rozdíl od prohlížeče Mozilla větší nároky na porozumění struktuře HTML dokumentu s výsledkovou listinou, a proto bylo nutné doplnit některá rozšíření. Nebylo třeba vytvářet nové elementy či jinak výrazně měnit strukturu, stačilo se omezit na přidání několika atributů k již existujícím elementům. V příloze B je jako příklad uvedena specifikace formátu vyhledávače Jyxo.

Jména elementů, jejich počet a stromová struktura, kterou vytvářejí, jsou pevně dány, podobně jako je tomu u jazyka HTML. Celý popis formátu je uzavřen v jediném kořenovém elementu `search`. Ten obsahuje několik elementů `input`, dále jeden element `inputnext` a jeden element `interpret`. Jedná se tedy o velice prostý strom s jediným vnitřním uzlem (který je zároveň kořenem) a třemi typy listů. Dále už tuto stromovou strukturu nebudeme zmiňovat a elementy označujme pouze jejich jménem.

Jméno a hodnota atributu jsou odděleny rovnítkem, hodnota je navíc uzavřena v uvozovkách (např. `name="Jyxo"`). Některé atributy musí být v elementu vždy uvedeny, některé jsou nepovinné. Hodnota atributu je neprázdný řetězec. (Nepovinný atribut lze uvést s prázdnou hodnotou, což je ekvivalentní zápisu, kde tento atribut není vůbec uveden.) Někdy je potřeba, aby hodnota atributu obsahovala znak uvozovky. Mozilla kupodivu použití tohoto znaku neumožňuje. Hodnotou atributu je často krátký úsek HTML kódu a jen velmi obtížně se lze bez této možnosti obejít. Například pro vyhledávač All the Web je vhodné nastavit atribut `itemSnippetStart` (význam bude vysvětlen později) na hodnotu `<span class="resTeaser">`. V metavyhledávači umožníme pomocí sekvence `&#c;` zápis libovolného znaku, kde `c` je kód tohoto znaku (ve znakové sadě UTF). Kromě uvozovek můžeme takto pro přehlednost zapsat i některé bílé znaky jako jsou tabulátor nebo odřádkování. Zápis takové hodnoty atributu pak může vypadat například takto:

```
itemSnippetStart="<span class=&#34;resTeaser&#34;>"
```

Element `search` obsahuje atribut `name` s názvem vyhledávače a nepovinný atribut `description`, jenž může obsahovat krátký popis vyhledávače. Zbytek specifikace formátu můžeme rozdělit na dvě části. Ostatní atributy elementu `search` a elementy `input`, `inputnext` popisují konstrukci HTTP požadavku odesílaného vyhledávači.[5] Element `interpret` definuje způsob parsování odpovědi.

3.4.2 URI dotazu

Bylo řečeno, že při tvorbě HTTP požadavku, který se posílá vyhledávači, se stačí omezit na URI. Mozilla definuje v elementu `search` atribut `method`, jehož hodnotou je metoda uvedená v HTTP požadavku. Idea byla umožnit použití metod `GET` a `POST`. Praxe je taková, že samotná Mozilla implementuje pouze metodu `GET`. Rovněž metavyhledávač s metodou `GET` naprosto vystačí.

Popis, jak vytvořit URI pro daný dotaz, je téměř beze změny převzatý z vyhledávače Mozilla. Toto URI definujeme hodnotou atributu `action` v elementu `search`, za kterou se připojí otazník a parametry URI. Parametry jsou definovány pomocí elementů `input` včetně parametru, ve kterém se předává dotaz uživatele. Tato konstrukce byla podrobněji popsána v kapitole 2.2.

Zbývá ještě vyřešit, jak informujeme vyhledávač o tom, kterou stránku výsledkové listiny požadujeme. I tento problém je v Mozille jistým způsobem řešen, bohužel ne zcela šťastně. Číslo stránky, kterou od vyhledávače požadujeme, se zadá jako jeden z parametrů URI, např. `page=3`. Jméno parametru se samozřejmě liší od vyhledávače k vyhledávači. Krom toho hodnota parametru může být buď pořadové číslo stránky (např. u vyhledávače Ask Jeeves) nebo pořadové číslo prvního hitu na požadované stránce (např. Morfeo). Chceme-li druhou stránku výsledkové listiny, bude v prvním případě součástí URI například parametr `page=2`, v druhém případě může mít tento parametr tvar `from=11` (za předpokladu, že jedna stránka obsahuje 10 hitů). Mozilla tento parametr URI popisuje elementem `inputnext`, který vypadá například takto:

```
<inputnext name="from" factor="10">
```

Atribut `name` určuje jméno parametru URI, atribut `factor` definuje rozdíl hodnot parametru pro dvě po sobě jdoucí stránky. Neřeší se ovšem už od jakého čísla indexování začíná. Google je příkladem vyhledávače, který čísluje hity od nuly, Seznam indexuje hity od jedné. V Mozille je element `inputnext` nepovinný. Předpokládá se, že indexování začíná od nuly. Praxe je taková, že v popisu formátu vyhledávače, který indexuje od jedné, se element `inputnext` neuvádí. Mozilla pak parametr URI pro specifikaci stránky

[5] Chceme-li být naprosto přesní, je třeba zmínit, že element `search` může obsahovat nepovinný atribut `responseCharset`, který samozřejmě definuje znakovou sadu použitou v odpovědi.

nepoužívá a je schopna pracovat pouze s první stránkou výsledkové listiny. V metavyhledávači by byl podobný přístup velmi nevhodný, proto je element `inputnext` rozšířen o atribut `initial` určující číslo, od kterého indexace začíná. Element pak může mít tvar

```
<inputnext name="page" initial="1" factor="1">
```

Na závěr se ještě zmíníme, jak poznat konec výsledkové listiny. Předpokládejme, že si agent od vyhledávače vyžádal stránku s příliš vysokým pořadovým číslem. (Například chceme třetí stránku od vyhledávače, který vrací na každé stránce 10 hitů a pro daný dotaz nenašel více než 20 relevantních dokumentů.) Webové vyhledávače v tomto případě vykazují jeden ze tří typů chování.

- Vrátí stránku, která neobsahuje žádné hity (sem patří i případ, kdy vyhledávač k dotazu vůbec žádné relevantní dokumenty nenalezne).
- Vrátí stránku obsahující jediný hit, a to poslední hit výsledkové listiny.
- Vrátí poslední stránku výsledkové listiny (tj. i když bude agent zvyšovat pořadové číslo stránky, kterou po vyhledávači požaduje, bude se mu vracet stále stejná stránka).

3.4.3 Parsování výsledkové listiny

Struktura výsledkové listiny je popsána elementem `interpret`. Tento element obsahuje atribut `browserResultType`, který má v Mozille obvykle hodnotu `result` (může nabývat ještě hodnoty `category`). Popisuje charakter výsledků a z jeho hodnoty odvodí Mozilla způsob, jak budou výsledky graficky prezentovány. Je možné dokonce použít dva elementy `interpret` s různou hodnotou tohoto atributu. Metavyhledávač bude zpracovávat pouze takové elementy `interpret`, kde je jako `browserResultType` zadána hodnota `result`.

K nalezení jednotlivých hitů v odpovědi používá Mozilla atributy `resultListStart`, `resultListEnd`, `resultItemStart` a `resultItemEnd`. Nevyužívá se nijak faktu, že odpověď má podobu HTML kódu. Její zpracování je založeno pouze na hledání podřetězců. Tento způsob parsování je jednoduchý, rychlý a ukazuje se, že postačující. Atribut `resultItemStart` definuje řetězec, který uvozuje každý hit, hodnota atributu `resultItemEnd` specifikuje řetězec, kterým hity končí. Jakákoli sekvence znaků ohraničena těmito řetězci je považována za hit. (Řetězce uvedené v `resultItemStart` a `resultItemEnd` samy součástí hitu nejsou.) Atributy `resultListStart` a `resultListEnd` jsou nepovinné. Je-li zadán atribut `resultListStart`, nehledají se v odpovědi vyhledávače hity až do místa, kde se vyskytuje řetězec definovaný tímto atributem. Typické použití atributu `resultListStart` vypadá takto:

```
resultListStart="<h1>web results</h1>"
```

Možnost ignorovat začátek HTML stránky se většinou využije k odfiltrování sponzorovaných odkazů. Analogicky pomocí atributu `resultListEnd` určíme místo, od kterého dále se už hity nehledají.

Tyto čtyři atributy a jejich sémantiku převezmeme i pro náš metavyhledávač. Mozilla získá hit jako úsek HTML kódu a jeho vnitřní strukturou už se dále nezabývá. Oproti tomu metavyhledávač chápe znaky mezi `resultItemStart` a `resultItemEnd` jako řetězec, z něhož je třeba extrahovat URI dokumentu, titulek a popisek. K tomuto záměru už Mozilla žádnou podporu nenabízí, proto pro metavyhledávač přidáváme následující skupinu atributů:

```
itemURIStart, itemURIEnd, itemURISkip,
itemTitleStart, itemTitleEnd, itemTitleSkip,
itemSnippetStart, itemSnippetEnd, itemSnippetSkip.
```

S URI, titulkem i popiskem pracujeme analogicky. Pro nalezení URI byl navíc ještě přidán atribut `itemURIEncoding`, o kterém se zmíníme později. Začátek a konec každé ze tří položek ohraničují řetězce definované atributy s příponami `Start` a `End`. Jedná se o stejný princip jako u atributů `resultItemStart` a `resultItemEnd`.

Někdy může složka hitu (URI, titulek či popisek) obsahovat sekvenci znaků (nejčastěji jde o HTML značku), které se chceme zbavit. Podívejme se na následující úsek HTML kódu.

```
...
<font size=-1>
The greatest value of the Internet is the ability to find<br>
out information about anything at anytime anywhere you want.
</font>
...
```

Element `font` zde obsahuje popisek hitu. Uvnitř popisku se vyskytuje HTML značka pro řádkový zlom. Je vhodné, aby agent tuto značku odstranil. Atribut `itemSnippetSkip` udává řetězec, který není součástí popisku, ačkoli se vyskytuje mezi `itemSnippetStart` a `itemSnippetEnd`. Je-li uvnitř popisku více výskytů tohoto řetězce, odstraní agent všechny. Pro výše uvedený HTML kód, můžeme tedy definovat atributy následovně.

```
itemSnippetStart="<font size=-1>"
itemSnippetEnd="</font>"
itemSnippetSkip="<br>"
```

Analogicky se pracuje i s atributy `itemURISkip` a `itemTitleSkip`.

Ukazuje se však, že při hledání URI, titulku a popisku nevystačíme pouze s porovnáváním řetězců. Podívejme se na následující úsek HTML kódu.

```
...
<a class=l href="http://www.eclipse.org/">Eclipse home</a>
...
```

Uvedený element obsahuje titulek dokumentu. U vyhledávače vracející odpověď v podobném formátu nelze definovat atribut `itemTitleStart` lépe než `itemTitleStart=">"` nebo `itemTitleStart="&#34;>"`. Znaky předcházející uvozovkám a úhlové závorce už nemohou být součástí hodnoty atributu `itemTitleStart`, protože se u různých hitů liší. Řetězec `">"` má ale v HTML kódu velmi mnoho výskytů, takže se nám uvedeným způsobem nepodaří jednoznačně identifikovat místo, kde se titulek dokumentu nachází. Problém tohoto charakteru se vyskytuje při hledání URI, titulku či popisku velmi často.

Nabízí se otázka, zda nenastala chyba ve chvíli, kdy jsme se rozhodli nebrat zřetel na HTML strukturu odpovědi vyhledávače a parsovat tuto odpověď pomocí porovnávání řetězců. V předchozím příkladu bychom mohli chtít popsat umístění titulku takto: *„Titulek dokumentu nalezneme jako obsah elementu `a`, u kterého má atribut `class` hodnotu `l`.“* Zápis, který by umožňoval zaznamenat uvedený způsob navigace v HTML dokumentu, by ovšem sám o sobě nestačil a musel by být kombinován ještě s dalšími prostředky, jak ukazuje následující příklad.

```
...
<a class=yschttl href="http://rds.yahoo.com/_ylt=Av8GkHTs0A
aWYycBEPx3ku9XNyoA;_ylu=X3oDMTE1Ym80OHYxBGNvbG8DdwRsA1dTMQR
wb3MDNARzZWMDc3IEdnRpZANERlhIXzI-/SIG=11fu7eouh/EXP=1136922
541/**http%3a//www.chesscorner.com/">Chess Corner</a>
...
```

Zde se zaměříme na to, jak v uvedeném HTML kódu najdeme URI dokumentu. To je obsaženo v hodnotě atributu `href`. Tato hodnota je ale interní odkaz vyhledávače Yahoo a URI dokumentu, které hledáme, je až na jeho konci. I v případě, že by agent rozuměl struktuře HTML, musel by být stále schopen určitého vyhledávání vzorku v textu. Jazyk s požadovanými vlastnostmi by se už značně zkomplikoval.

V návrhu metavyhledávače byl zvolen jiný přístup. I nadále nebudeme brát ohled na to, že výsledková listina je vracena ve formě HTML kódu. Avšak místo toho, abychom požadované složky hitu hledali pomocí prostého porovnávání řetězců, umožníme zapsat sekvenci znaků, která se vyskytuje před a za URI, titulkem či popiskem, pomocí regulárního výrazu. Pro uvedené ukázky HTML kódu pak specifikujeme způsob parsování atributem

```
itemTitleStart="<a class=l href=&#34;[^&#34;]*&#34;>"
```

respektive

```
itemURIStart="<a class=yschttl href=&#34;[^*]*\*\*"
```

Přidání regulárních výrazů je relativně jednoduché rozšíření jazyka pro popis formátu vyhledávače. Navíc se snadno implementuje. Pro různé programovací jazyky jsou dostupné knihovny pro práci s regulárními výrazy.

Pomocí regulárních výrazů snadno zapíšeme, že se v odpovědi vyhledávače může alternativně vyskytovat několik řetězců. V HTML kódu jsou uvnitř titulku či popisku často přítomny značky pro zvýraznění slov, které byly součástí dotazu. Tyto značky jsou pro nás nežádoucí, ale snadno je odstraníme definováním atributu

```
itemSnippetSkip="(<br>)|(<b>)|(</b>)"
```

URI dokumentu, titulek a popisek tedy hledáme pomocí regulárních výrazů. U atributů `resultListStart`, `resultListEnd`, `resultItemStart` a `resultItemEnd` však zůstaneme u prostého porovnávání řetězců. Jde o kompromis mezi vyjadřovací silou jazyka a efektivitou zpracování odpovědi vyhledávače. Pro toto rozhodnutí existují tři důvody.

- Jakmile agent přijme začátek odpovědi od vyhledávače, může ji okamžitě začít parsovat. Není vhodné načítat najednou celou odpověď (která mívá někdy více než 100 kB). Místo toho můžeme udržovat vždy jen relativně malý úsek textu ve vyrovnávací paměti. Protože nemáme obecně žádný odhad pro maximální délku vzorku zadaného regulárním výrazem, nedalo by se rozumně rozhodnout, jakou část textu můžeme z vyrovnávací paměti již odstranit. Oproti tomu text ohraničený pomocí `resultItemStart` a `resultItemEnd` je poměrně krátký. Lze ho tedy držet celý v paměti a vyhledávat v něm jednotlivé složky hitu pomocí regulárních výrazů.

- Na atributy `resultListStart`, `resultListEnd`, `resultItemStart` a `resultItemEnd` nejsou kladeny tak vysoké nároky na přesnost. Atributem `itemURIStart` musíme například popsat posloupnost znaků, která se vyskytuje *těsně* před URI dokumentu. Podobně `itemURIEnd` definuje znaky *bezprostředně* za URI dokumentu. Na druhou stranu text vyskytující se například mezi `resultItemStart` a `resultItemEnd`, není pro metavyhledávač sám o sobě důležitý. Stačí najít jakýkoli úsek, ze kterého půjdou následně extrahovat URI, titulek a popisek.

- Samotné vyhledávače často podporují vyhledávání v Mozille tím, že do odpovědi umísťují speciální HTML komentáře pro snadné nalezení začátku a konce hitu.

3.4.4 Znakové sady

Znakové sady v nichž je zakódovaný dotaz odesílaný vyhledávači a vrácená odpověď jsou určeny atributy `queryCharset` a `responseCharset` v elementu `search`.

Dotaz, jak už bylo uvedeno, se pošle vyhledávači jako parametr URI v HTTP požadavku. Speciální znaky (především znaky s diakritikou) se kódují pomocí sekvence `%XX` (případně `%XX%XX` pro dvoubytové kódy), kde `XX` je kód příslušného znaku hexadecimálně. Předpokládejme dotaz uživatele tvořený pouze termem *žluťoučký*. Je-li definován atribut

```
queryCharset="UTF-8",
```

URI v HTTP požadavku může vypadat takto:

```
http://web.ask.com/web?q=%C5%BElu%C5%A5ou%C4%8Dk%C3%BD
```

Nepovinný atribut `responseCharset` specifikuje kódování použité v odpovědi vyhledávače. (Narozdíl od `queryCharset` podobný atribut v Mozille neexistuje.) Pokud není uveden, určí metavyhledávač použité kódování z hlavičky `Content-Type` v HTTP odpovědi. Explicitní zadávání znakové sady pomocí atributu `responseCharset` je vhodné jen v případě, že se v odpovědi hlavička `Content-Type` nenachází.

Na závěr se ještě zmíníme o atributu `itemURIEncoding`, který je součástí elementu `interpret`. V odpovědi mívají často hity v URI dokumentu zakódovány některé znaky pomocí sekvence začínající znakem `"%"`. Například v odpovědi Yahoo (ale patří sem také vyhledávače All the Web, AOL search a další) je potřeba URI dokumentu získat z následujícího HTML kódu:

```
...
<a class=yschttl href="http://rds.yahoo.com/_ylt=Av8GkHTs0AaWY
ycB...541/**http%3a//www.chesscorner.com/">Chess Corner</a>
...
```

Atributem `itemURIEncoding` určíme znakovou sadu, která se použije při dekódování těchto znaků (v našem příkladu jde pouze o sekvenci `"%3a"`). Atribut je nepovinný. Není-li uveden, metavyhledávač žádné dekódování neprovádí.

3.5 Displej

Displej je komponenta, jejíž úkolem je slévat výsledkové listiny produkované různými vyhledávači. Proces slévání realizujeme pomocí algoritmu TA popsaném v kapitole 2.3. Použijeme aproximační variantu s koeficientem $\theta = 1,7$ s výhradně sekvenčním přístupem. Měření vedoucí k volbě hodnoty θ je popsáno v kapitole 5.3.

Jako jednoznačný identifikátor dokumentu bude sloužit jeho URI. Jinými slovy budeme předpokládat, že hity vrácené dvěma různými vyhledávači se vztahují k témuž dokumentu, právě když se URI v těchto hitech shoduje. U některých vyhledávačů dochází k tomu, že ve své výsledkové listině uvedou dva (nebo dokonce více) hitů se shodným URI. Jedná se o poměrně řídký jev, který lze spíše považovat za chybu příslušných vyhledávačů. Přesto k němu někdy dochází, například v případě vyhledávačů Jyxo nebo Yahoo. V těchto situacích budeme ignorovat druhý a další výskyt hitu se stejným URI dokumentu v jedné výsledkové listině. Jestliže se objeví hity se stejným URI ve výsledkových listinách od různých vyhledávačů a tyto hity se liší titulkem dokumentu (resp. popiskem), ve výsledkové listině metavyhledávače se použije titulek (resp. popisek) vrácený vyhledávačem, který největším dílem přispěje k celkovému ranku dokumentu.

Na výpočet ranku dokumentu v celkové výsledkové listině, kterou již metavyhledávač prezentuje uživateli, se nyní zaměříme podrobněji. Je třeba zvolit agregační funkci, která rankům, jež dokument získal od jednotlivých vyhledávačů, přiřadí celkový rank. Metavyhledávač SavvySearch používal prostý součet ranků od jednotlivých vyhledávačů, tj. pro dokument d spočítal celkový rank $rank(d)$ jako

$$rank(d) = \sum_{s \in S} rank_s(d),$$

kde $rank_s(d)$ je rank, jenž přiřadil dokumentu d vyhledávač s, a S je množina vyhledávačů, které byly vybrány dispečerem pro vyhodnocení dotazu. [6]

Zobecněme tuto agregační funkci a přidejme k rankům $rank_s(d)$ váhy. Jako váhy vyhledávačů použijeme hodnoty R_{psq} vypočtené dispečerem při selekci vyhledávačů pro dotaz q a uživatelský profil p. (viz kapitola 3.3) Celkový rank dokumentu tedy bude určen vzorcem

$$rank(d) = \sum_{s \in S} R_{psq} \cdot rank_s(d).$$

Tato volba agregační funkce je zjevně monotónní a tudíž vyhovuje požadavkům algoritmu TA.

Algoritmus TA hledá vždy k dokumentů s největším rankem. (Přesněji řečeno hledáme jistou aproximaci této k-tice.) Hodnota této konstanty je zvolena $k = 10$, což odpovídá počtu hitů, které metavyhledávač vrací na jedné stránce. Vyžádá-li si uživatel další stránku výsledků, hledá se opět 10 hitů s nejvyšším rankem mezi těmi, které dosud nebyly uživateli prezentovány. Připomeňme, že aproximační verze algoritmu TA obvykle nezná přesnou hodnotu (celkového) ranku dokumentu, pouze aktualizuje informace

[6] V kontextu metavyhledávače budeme používat toto mnemotechnické značení. V obecném popisu algoritmu TA odpovídalo rankům dokumentu od jednotlivých vyhledávačů značení $x_1, \ldots, x_m$ a celkový rank byl značen $t(x_1, \ldots, x_m)$.

o jeho dolním a horním odhadu. Výstupem algoritmu TA je k nejrelevantnějších dokumentů, neřeší se už ovšem uspořádání v rámci této k-tice. Metavyhledávač třídí dokumenty v rámci jedné stránky výsledkové listiny podle dolního odhadu jejich ranků, který je v dané chvíli k dispozici.

Kapitola 4

Implementace

Podle návrhu popsaného v kapitole 3 byla vytvořena implementace metavyhledávače. Její význam přímo pro tuto práci spočívá především v získávání některých experimentálních výsledků, o nichž se zmíníme později. Metavyhledávač byl implementován v programovacím jazyce Java (konkrétně byla použita platforma J2SE 5.0). Volba byla motivována především velkým množství dostupných knihoven v tomto jazyce.

Implementace metavyhledávače je použitelná jednak jako knihovna a jednak jako samostatný program spustitelný z konzole. Pro klasické provozování metavyhledávače na Webu bylo vytvořeno webové rozhraní na základě technologie JSP, které tuto implementaci využívá právě jako knihovnu. Druhý režim je vhodný zejména pro ladění a některé experimenty.

4.1 Adresářová struktura a použité knihovny

Celá instalace metavyhledávače je umístěna v jediném adresáři (na jehož jméno ani umístění nejsou kladeny žádné zvláštní požadavky). Adresář obsahuje jeden Java archiv s vlastním kódem metavyhledávače a dále použité knihovny ve formě dalších Java archivů. Krom toho obsahuje ještě podadresáře `formats`, `logs` a `feedback`.

Kromě balíků, které jsou přímo součástí platformy J2SE 5.0, využívá tato implementace metavyhledávače knihovny HTTP Client 3.0 a HTML Entities 1.0. Knihovna HTTP Client poskytuje rozhraní pro komunikaci pomocí HTTP protokolu na straně klienta. HTTP Client 3.0 navíc závisí na knihovnách Logging 1.0.4 a Code 1.3. Všechny tři knihovny jsou dostupné pod licencí Apache 2.0 na stránkách

`http://jakarta.apache.org`

HTML Entities je jednoduchá knihovna obsahující statické metody pro konvertování speciálních znaků na HTML entity a naopak. Tato konverze se využívá při zpracování HTML stránky vrácené vyhledávačem. Knihovnu HTML Entities distribuovanou pod LGPL licencí lze nalézt na adrese

`https://htmlentities.dev.java.net`

Adresář `formats` obsahuje jeden soubor s příponou `.src` pro každý podporovaný vyhledávač. Tento soubor popisuje formát dotazu a výsledkové listiny, kterou příslušný vyhledávač používá. (viz kapitola 3.4)

K vypisování chybových a ladících zpráv je použito standardního mechanismu logování implementovaného balíkem `java.util.logging`. Všechny zprávy se vypisují na standardní chybový výstup, chybové zprávy se navíc zapisují do souborů v adresáři `logs`. Jestliže je příčinou chybového stavu vyhození výjimky, pak jméno této výjimky a stav zásobníku jsou součástí chybového výpisu.

Zvláštní pozornost je třeba věnovat chybovým zprávám reagujícím na zachycení výjimky `ResponseParseException`. Jak název napovídá, jde o chybu vzniklou při parsování HTML dokumentu s výsledkovou listinou, kterou vrátil některý vyhledávač. Pravděpodobnou příčinou je změna formátu této HTML stránky. V takovém případě je nutné provést aktualizaci příslušného souboru v adresáři `formats`.

Soubory v adresáři `feedback` reprezentují meta-index. Za běhu metavyhledávače jsou některé části meta-indexu kešovány v operační paměti a k jejich zapsání dochází s jistým zpožděním nebo při vypnutí metavyhledávače. Jinými slovy není zaručeno, že v době běhu programu tento adresář obsahuje meta-index v konzistentním stavu.

4.2 Zadávání dotazu

Kód metavyhledávače je obsažen v balíku `org.egothor.metasearch`. Metavyhledávač jako celek je reprezentován třídou `MetaSearch`. Ke spuštění metavyhledávače je třeba vytvořit instanci této třídy (typ MetaSearch je singleton) bezparametrickým konstruktorem a poté ji inicializovat voláním metody `initializeIfNeeded`. Inicializace musí proběhnout před položením prvního dotazu. Metoda `initializeIfNeeded` může být zavolána vícekrát, dokonce i z různých vláken. Je zaručeno, že inicializace proběhne pouze jednou, a klientský kód má po provedení této metody jistotu, že inicializace již úspěšně skončila. Pro vyhodnocení dotazu se na této instanci volá jedna z variant metody `dispatch`. Korektní ukončení činnosti metavyhledávače je třeba explicitně zajistit voláním metody `shutdown`.

Metoda `dispatch` má dvě verze. Jedna dostává jako parametr kolekci názvů vyhledávačů, kterým se pošle dotaz, druhá provádí selekci vyhledávačů automaticky (předá se jí pouze informace, kolik vyhledávačů má být vybráno), jak je popsáno v kapitole 3.3. První varianta je vhodná pro některé experimenty a téměř nepostradatelná pro ladění, druhá se využívá pro běžný provoz metavyhledávače. Obě dále dostávají jako parametry uživatelský profil, dotaz a počet hitů na jedné stránce výsledkové listiny. Uživatelský profil se zadává jako neprázdný řetěz. Nový profil není třeba explicitně

vytvářet. Pouze fakt, že u dvou volání metody `dispatch` se shodují tyto řetězce, informuje metavyhledávač, že se dotazy vztahují ke stejnému profilu. Dotaz se předává rovněž jako řetězec se standardní syntaxí, jakou používá rozhraní u většiny vyhledávačů na Webu (tj. termy oddělené mezerou či jinými nealfanumerickými znaky, fráze uzavřené v uvozovkách). Počet hitů na stránce se použije jako parametr slévacího algoritmu TA, jenž určuje, jaké množství hitů je výstupem tohoto algoritmu v jednom kroku (počet hitů na stránce odpovídá konstantě k - viz kapitola 3.5).

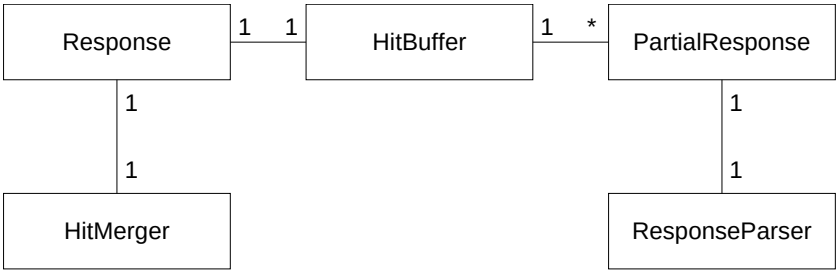
Metodu `dispatch` (tj. obě její varianty) lze volat současně z více vláken. Uvnitř metody dojde k selekci vyhledávačů (pokud nejsou jejich jména předána metodě přímo), těmto vyhledávačům se rozešle dotaz a bez čekání na jejich odpovědi metoda skončí. Návratovou hodnotou je instance třídy `Response` implementující rozhraní `Iterable<Hit>`. Jakmile, metavyhledávač zkonstruuje objekt typu `Response`, začne paralelně v několika dalších vláknech probíhat vyhodnocování dotazu. Správa těchto vláken je interní záležitostí metavyhledávače, tedy je zcela mimo zodpovědnost klientského kódu. `Hit` je jednoduchá třída, která pouze zapouzdřuje URI dokumentu, titulek, popisek a rank tvořící jednu položku výsledkové listiny. Jednotlivé hity odpovědi získáváme pomocí iterátoru. Volání metody `next` (resp. `hasNext`) na tomto iterátoru se může zablokovat do doby, než dodají vyhledávače dostatek informací, aby bylo možné vrátit následující hit celkové výsledkové listiny (resp. rozhodnout, zda výsledková listina ještě pokračuje).

Objekt typu `Response` lze uložit do keše metavyhledávače. To provedeme voláním metody `storeResponse` na instanci typu `MetaSearch`. Návratovou hodnotou této metody je číslo typu `long`, které slouží jako identifikátor uložené odpovědi. Z keše lze odpověď opět vyjmout voláním metody `restoreResponse`, která jako parametr dostane právě zmíněný identifikátor. Keš má pouze omezenou kapacitu a v případě nedostatku místa jsou některé odpovědi odstraňovány. K výběru objektu, který bude odstraněn, se používá algoritmus LRU. Uložíme-li tedy odpověď pomocí metody `storeResponse`, nemáme zaručeno, že se tuto odpověď podaří později pomocí `restoreResponse` obnovit. Tímto způsobem je vhodné odkládat odpovědi, u kterých si uživatel může za okamžik vyžádat další stránku výsledků, a předejít tak nutnosti vyhodnocovat dotaz opět od začátku.

Používá-li se metavyhledávač jako knihovna, jsou `MetaSearch`, `Response` a `Hit` jediné třídy (kromě výjimek), se kterými pracuje klientský kód.

4.3 Implementace agentů a slévání výsledků

Zmínili jsem se o tom, jaké rozhraní nabízí metavyhledávač klientskému kódu. Nyní se zaměříme na interní mechanizmy vyhodnocování dotazu. V této kapitole rozebereme třídy implementující agenta pro komunikaci s vyhledávačem a displej. Diagram těchto tříd zachycuje obrázek 4.1.



Obrázek 4.1: Agent a slévání výsledků - diagram tříd

Objekt `Response` vytvoří hned ve svém konstruktoru jednu instanci typu `HitMerger` a jednu instanci typu `HitBuffer`. Dále se pro každý vyhledávač vybraný pro vyhodnocení dotazu konstruuje jeden objekt typu `PartialResponse`.

Třída `PartialResponse` reprezentuje agenta. Hned v konstruktoru vytvoří nové vlákno, ve kterém probíhá komunikace s vyhledávačem pomocí HTTP protokolu. Pro parsování HTML kódu s výsledky, které vyhledávač vrací, používá pomocnou třídu `ResponseParser`.

Mezi agenty a displej je pro zvýšení efektivity umístěna vyrovnávací paměť implementovaná třídou `HitBuffer`. Jednotliví agenti prostor v této paměti nesdílejí, ale každý má k dispozici svou vlastní část, která může obsáhnout několik málo desítek hitů. Konkrétně v této implementaci je zmíněná kapacita nastavená na 21 hitů. Hodnota je volena tak, aby přesáhla počet hitů, které vyhledávače obvykle vracejí na jedné stránce odpovědi.[1] Agent (tj. instance `PartialResponse`) přidává do své části vyrovnávací paměti stále další hity, dokud ji nenaplní. (Ukládají se vždy všechny hity na stránce, takže výše popsaná kapacita může být i mírně překročena.) Dojde-li k tomuto stavu, vlákno komunikující s příslušným vyhledávačem se ukončí. Jakmile se paměť uvolní, vytvoří se nový objekt typu `PartialResponse`, který spustí nové vlákno, a získávají se další stránky odpovědi daného vyhledávače. Hity z této paměti jsou vybírány instancí třídy `Response`. Volání, které hity z vyrovnávací paměti vybírá, se zablokuje, pokud je tato paměť prázdná.

Třída `HitMerger` prakticky implementuje slévací algoritmus TA. Tato třída se sama chová pasivně, tj. nesnaží se získávat nové hity od jednotlivých vyhledávačů. Tento úkol plní třída `Response`. Jakmile jsou po instanci `Response` žádány další hity (prostřednictvím iterátoru), je tento požadavek delegován objektu `HitMerger`. Ten na žádost (volání metody `nextHits`) odpoví buď *„Ano, tady jsou požadované hity.“* nebo *„Nemám dost informací, potřebuji další hity od vyhledávačů.“* V druhém případě, objekt

[1] Nejčastěji má jedna stránka 10 hitů. U vyhledávače Morfeo ale například obsahuje hitů 20.

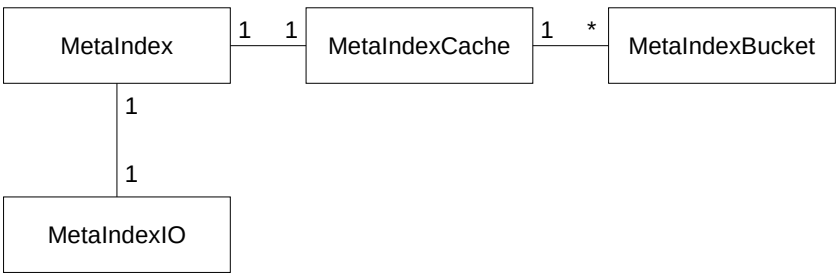
`Response` vyjme veškeré hity, které jsou v danou chvíli nashromážděné v objektu `HitBuffer`, a předá je objektu `HitMerger`. Poté znovu volá metodu `nextHits` a celý proces se opakuje.

Každá z instancí tříd `Response`, `HitMerger`, `HitBuffer`, `ResponseParser` nebo `PartialResponse` je asociována vždy s jedním konkrétním vyhodnocováním dotazu. Vzniká poté, co dojde k selekci vyhledávačů, a zaniká ve chvíli, kdy klientský kód už nemá zájem o další části výsledkové listiny. V případě třídy `PartialResponse` může navíc pro jeden vyhledávač v průběhu vyhodnocování jednoho dotazu vzniknout a zaniknout několik instancí.

Třída `Format` zapouzdřuje informace o formátu dotazu a výsledkové listiny při komunikaci s vyhledávačem. Jde o reprezentaci dat z adresáře `formats` v operační paměti. Pro každý vyhledávač vznikne jedna instance této třídy, která existuje po celou dobu běhu metavyhledávače. Objekt typu `Format` používá třída `PartialResponse` při konstrukci HTTP požadavku a třída `ResponseParser` při parsování odpovědi.

4.4 Zpětná vazba

Pro řešení selekčního problému sbírá metavyhledávač informace o kvalitě výsledkových listin a o rychlosti odpovědi jednotlivých vyhledávačů. Z těchto dvou skupin dat se pro vyhledávače počítají dvě složky skóre, které jsme označili P_s a Q_{psq}. (viz kapitola 3.3) Uchovávání rychlosti odpovědi na několik posledních dotazů pro všechny vyhledávače je triviální problém, který řeší třída `EnginePerformance`. Informace o kvalitě výsledků jsou uloženy v meta-indexu, na jehož implementaci se teď zaměříme.



Obrázek 4.2: Meta-index - diagram tříd

Klíčové třídy tvořící meta-index a vazby mezi nimi ukazuje obrázek 4.2. Třída `MetaIndex` je jediná třída viditelná ostatním komponentám metavyhledávače a pro správu meta-indexu používá pomocné třídy `MetaIndexIO` a `MetaIndexCache`. Meta-index je paměťově příliš náročná struktura na to, aby mohla být přítomna celá v operační paměti. K její reprezentaci

je použita hašovací tabulka. Vzhledem ke zvolenému typu hašování jsou páry klíč-hodnota rozděleny do kapes. Kapsa je implementována třídou `MetaIndexBucket`. Nedávno použité kapsy se kešují v operační paměti. Keš je implementována třídou `MetaIndexCache`. `MetaIndexIO` je třída zodpovídající za diskové operace, tj. za načítání kapes z disku a odkládání kapes z keše při nedostatku místa. Pro volbu kapsy, která je z keše odstraněna, se používá algoritmus LRU.

Klíč v hašovací tabulce (implementovaný třídou `MatrixKey`) tvoří trojice řetězců (uživatelský profil, vyhledávač, term). Meta-index je logicky tvořen souborem matic, kde každá matice odpovídá jednomu uživatelskému profilu, řádky matic korespondují s vyhledávači a sloupce s termy. Při výpočtu skóre při selekci vyhledávačů se pracuje s hodnotami T_{ps}, $|\{s \in S; M_{pst} > 0\}|$ a $|\{s \in S; M_{pst} \neq 0\}|$. (viz kapitola 3.3) Tyto hodnoty jsou odvozené z prvků meta-indexu, nicméně pro efektivitu selekce vyhledávačů je nelze počítat pokaždé znovu dle jejich definice. Proto jsou zmíněné hodnoty součástí hašovací tabulky, kde jsou uloženy pod speciálními klíči. (V případě T_{ps} získáme klíč tak, že jako term použijeme prázdný řetězec. Pro zbývající dva typy hodnot jsou použité klíče s prázdným řetězcem na místě vyhledávače. Pod tímto klíčem je pak uložena celá dvojice $|\{s \in S; M_{pst} > 0\}|$, $|\{s \in S; M_{pst} \neq 0\}|$.) Při aktualizaci hodnot T_{ps} může docházet k chybě způsobené reálnou aritmetikou. Dá se předpokládat, že tato chyba bude zanedbatelná. V případě potřeby je však možné tyto hodnoty přepočítat voláním metody `repairMetaIndex` na instanci typu `MetaSearch`.

4.5 Parametry metavyhledávače

Důležité konstanty ovlivňující chování metavyhledávače jsou definovány v třídách `AlgorithmSettings` a `PerformanceSettings`. Konstanty třídy `AlgorithmSettings` určují parametry použitých algoritmů, jako jsou faktor aproximace θ u slévacího algoritmu TA nebo hodnota α_{pt} určující maximální podíl profilové složky při výpočtu skóre vyhledávače. Třída `PerformanceSettings` definuje paměťové nároky metavyhledávače, jako je například velikost keše meta-indexu, a další parametry určující výkon, například maximální počet HTTP spojení.

Kapitola 5

Experimentální výsledky

V této kapitole jsou zdokumentována měření, na jejichž základě byly zvoleny některé parametry použitých algoritmů. Uvedeme také experiment, jehož cílem bylo změřit kvalitu výsledků, jež metavyhledávač produkuje.

Při těchto měřeních byla použita implementace metavyhledávače popsaná v kapitole 4. Veškeré experimenty, které zde popíšeme, byly provedeny na počítači s rychlostí síťového připojení 100 Mb/s. Metavyhledávač podporoval celkem 9 vyhledávačů. Některé vyhledávají dokumenty na celém Webu, některé se specializují na dokumenty v češtině. Jejich výčet obsahuje tabulka 5.1.

Název	Adresa	Rozsah vyhledávání
All the Web	`http://www.alltheweb.com`	celý Web
Alta Vista	`http://www.altavista.com`	celý Web
AOL Search	`http://search.aol.com`	celý Web
Ask Jeeves	`http://www.ask.com`	celý Web
Google	`http://www.google.com`	celý Web
Yahoo! Search	`http://search.yahoo.com`	celý Web
Jyxo	`http://jyxo.cz`	stránky v češtině
Morfeo	`http://morfeo.centrum.cz`	stránky v češtině
Seznam	`http://www.seznam.cz`	stránky v češtině

Tabulka 5.1: Vyhledávače použité při experimentech

5.1 Odhad ranku

Ve výsledkových listinách, které vracejí jednotlivé vyhledávače, bohužel obvykle chybí ranky pro jednotlivé dokumenty. Metavyhledávač je tedy nucen tyto ranky odhadnout. Za tímto účelem je použita funkce, která přiřazuje ranky v závislosti na pořadí dokumentu ve výsledkové listině. K vytvoření

této funkce použijeme vyhledávač Morfeo. Právě tento vyhledávač uvádí u každého hitu číslo určující jeho míru relevance vzhledem k dotazu. Toto číslo prakticky odpovídá ranku, pouze hodnoty, kterých může nabývat, nejsou z intervalu $\langle 0, 1\rangle$. Idea tohoto měření je položit vyhledávači Morfeo několik dotazů, extrahovat z výsledkových listin ranky a těmito hodnotami proložit funkci. Tuto funkci bude poté metavyhledávač používat k přiřazování ranků dokumentům pro všechny vyhledávače.

Množina testovacích dotazů byla získána pomocí vyhledávače na portálu Tiscali.[1] Zde je možné prohlédnout si několik posledních dotazů, které uživatelé vyhledávači položili. Z tohoto zdroje byla náhodně vybrána skupina dotazů a z ní poté odstraně takové dotazy, které na vyhledávači Morfeo vedou k prázdným výsledkovým listinám. Po této redukci zbylo 40 dotazů, jenž byly použity pro následující měření.

Morfeo počítá rank dokumentu na základě frekvence výskytu termů z dotazu, místě výskytů těchto termů (v běžném textu, v nadpisu atd.), vzdálenosti těchto termů v dokumentu a celkové významnosti dokumentu (celková významnost dokumentu je nezávislá na položeném dotazu). Tyto ranky jsou v případě Morfea celými nezápornými čísly. Maximální možná hodnota ranku je několik desítek tisíc, nicméně toto maximum je závislé na položeném dotazu. Rozsah ranků, které jsou dokumentům ve výsledkové listině přiřazeny, se tak pro různé dotazy výrazně liší.

Tuto situaci ilustruje obrázek 5.1. Graf ukazuje ranky v závislosti na pořadí hitu ve výsledkové listině pro tři dotazy - q_1 (*rybaření na Islandu*), q_2 (*usnesení vlády*) a q_3 (*seznam*). Je vidět, že pro další práci se získanými hodnotami je třeba pro všechny dotazy převést ranky na společný interval.

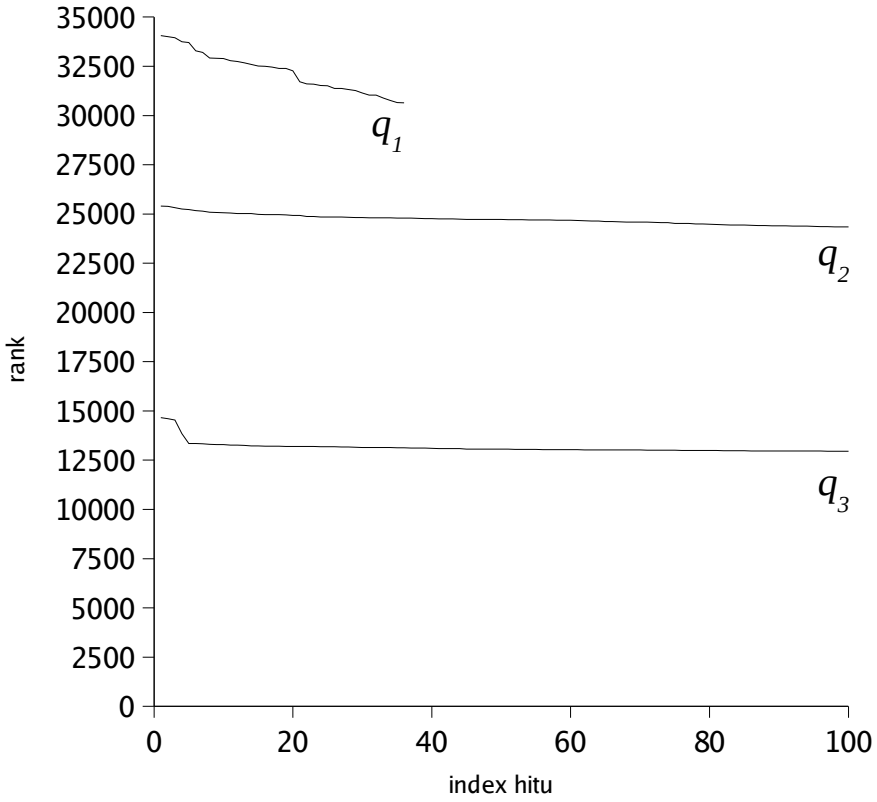
Mějme dotazy $q_1, \ldots, q_n$. (V našem případě $n = 40$.) Pro dotaz q_i označíme r'_{ij} rank j-tého dokumentu (indexováno od 1) na výsledkové listině vrácené vyhledávačem Morfeo. Číslo m_i bude značit počet hitů ve výsledkové listině pro dotaz q_i. Zřejmě platí, že pro každé i tvoří ranky $r'_{i1}, \ldots, r'_{im_i}$ nerostoucí posloupnost. Pro každé $i \in \{1, \ldots, n\}$ a každé $j \in \{1, \ldots, m_i\}$ definujeme hodnotu r_{ij} jako

$$r_{ij} = \frac{r'_{ij} - r'_{im_i}}{r'_{i1} - r'_{im_i}}.$$

V případě, že $r'_{i1} = r'_{im_i}$, položme $r_{ij} = 1$. Ranky jsou tímto znormovány do intervalu $\langle 0, 1\rangle$. Pro každé $i \in \{1, \ldots, n\}$ platí $r_{im_i} = 0$ (až na speciální případ, kdy $r_{i1} = r_{i2} = \cdots = r_{im_i} = 1$). Alternativně by bylo možné pro poslední dokument výsledkové listiny použít jisté malé kladné číslo. V provedeném měření však byla použita hodnota 0.

Při realizaci tohoto pokusu v praxi se dopouštíme jedné nepřesnosti. Vyhledávač Morfeo vrací pro daný dotaz maximálně 1000 hitů a to i v případě,

[1] http://hledani.tiscali.cz/web

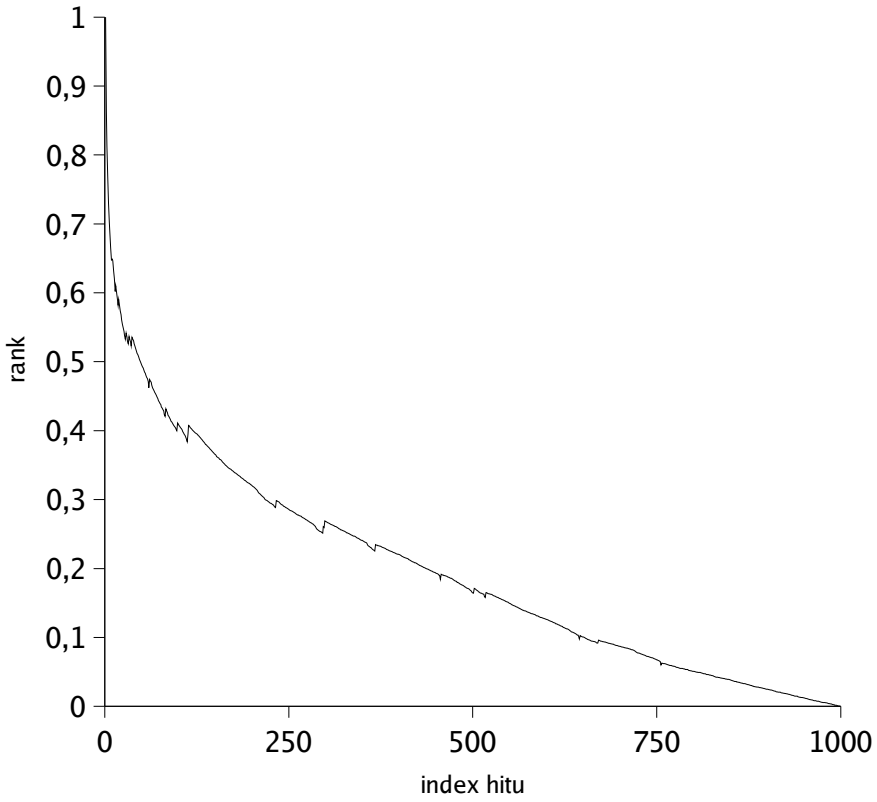


Obrázek 5.1: Rozsah ranků vyhledávače Morfeo

že nalezl větší počet relevantních dokumentů. Toto omezení je u vyhledávačů na Webu běžné a právě 1000 hitů je obvyklou hranicí. Pro testovaných 40 dotazů měla v 18 případech výsledková listina vrácená vyhledávačem Morfeo právě 1000 hitů. Pro těchto 18 dotazů bylo uvažováno $m_i = 1000$.

Dále pro každé $j \in \{1, \ldots, 1000\}$ spočteme hodnotu a_j jako aritmetický průměr ze všech hodnot r_{ij}, které jsou pro dané j definované. (r_{ij} není definováno, jestliže výsledková listina pro dotaz q_i má méně než j položek.) Takto naměřené hodnoty a_j zachycuje obrázek 5.2.

Jedním z dalších možných postupů je pokusit se proložit hodnotami a_j křivku pomocí metod statistické regrese. Charakter našich dat ovšem tomuto postupu příliš nenahrává. Hodnoty a_j nejprve pro malá j velmi strmě klesají. Přibližně od $j = 30$ je klesání mnohem mírnější. Zvolit jeden jednoduchý funkční předpis, který bude dobře aproximovat čísla a_j na celém intervalu $\langle 1, 1000 \rangle$ je proto problematické. Byl učiněn pokus proložit hodnotami a_j exponenciální funkci, tj. funkci tvaru $y = \alpha \cdot \beta^x$, přičemž parametry α, β hledáme tak, abychom minimalizovali součet čtverců chyb. Výsledek



Obrázek 5.2: Závislost průměrů a_j na pořadí hitu ve výsledkové listině

byl ovšem velmi neuspokojivý. Takto získaná křivka špatně vystihovala právě počáteční úsek posloupnosti průměrů a_j. Vzhledem k tomu, že nahrazením několika ranků jejich průměry ztrácíme informaci, můžeme prokládat křivku přímo hodnotami r_{ij}. Pro funkci $y = \alpha \cdot \beta^x$ dostaneme ale výsledek, který má stejný nedostatek jako v předchozím případě. Bylo by možné zkoušet naměřenými hodnotami prokládat další funkce. Pravděpodobně bychom k získání rozumné křivky museli posloupnost průměrů a_j (či posloupnosti ranků r_{ij}) rozdělit alespoň na dva úseky, získat pro každý úsek jinou funkci a ty pak spojitě navázat. Pro konstrukci funkce, kterou budou agenti používat k přiřazování ranků, byla nakonec zvolena zcela odlišná cesta.

Místo toho, abychom se snažili najít funkční předpis, můžeme hledanou funkci definovat přímo tabulkou hodnot pro její definiční obor $\{1, \ldots, 1000\}$. Jako základ této tabulky poslouží průměry a_j. Největším problémem je existence několika dvojic a_i, a_{i+1}, kde $a_i < a_{i+1}$. Tento jev je způsoben různou délkou výsledkových listin pro jednotlivé testovací dotazy. Lze se domnívat, že kvalitnější výsledky získáme zvýšením počtu dotazů. Podobné experi-

menty ale není snadné realizovat, protože obnášejí zaslání velkého počtu HTTP požadavků jednomu vyhledávači (v našem případě vyhledávači Morfeo).

Místo toho hodnoty a_j „vyhladíme“ pomocí metody klouzavých průměrů. Tato metoda má dva parametry - přirozená čísla s a k. Pro každé $j \in \{1, \ldots, 1000\}$ spočteme číslo r_j. Provedeme to tak, že hodnotami $a_{j-k}, \ldots, a_{j+k}$ proložíme polynom s-tého stupně (opět tak, abychom minimalizovali součet čtverců chyb). Číslo r_j pak definujeme jako hodnotu tohoto polynomu v bodě j. Definici v této podobě samozřejmě nemůžeme použít pro prvních k a posledních k členů posloupnosti r_j. Uvažujme polynom, který jsme použili k získání r_{k+1}. Definujme r_j jako hodnotu tohoto polynomu v bodě j pro $j \in \{1, \ldots, k\}$. Analogicky dopočteme i posledních k členů r_j. Více o této metodě je možné nalézt například v knize [4].

Popsat formálně kritérium pro volbu parametrů s a k by bylo pro tuto metodu dosti komplikované. Hodnoty r_j proto byly vypočteny pro několik různých dvojic parametrů. Testované dvojice parametrů jsou uvedeny v tabulce 5.2. Z jistého důvodu volíme stupeň polynomu s lichý (podrobnosti opět v [4]). Dle úsudku autora této práce byla vybrána dvojice $s = 3$, $k = 60$. Při porovnávání jednotlivých variant byly brány v úvahu odchýlení od hodnot a_j (s důrazem na počáteční úsek posloupnosti), hladkost křivky a monotonie.

s	k
3	$40, 45, 50, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 71, 81, 89$
5	$25, 31, 37, 40$

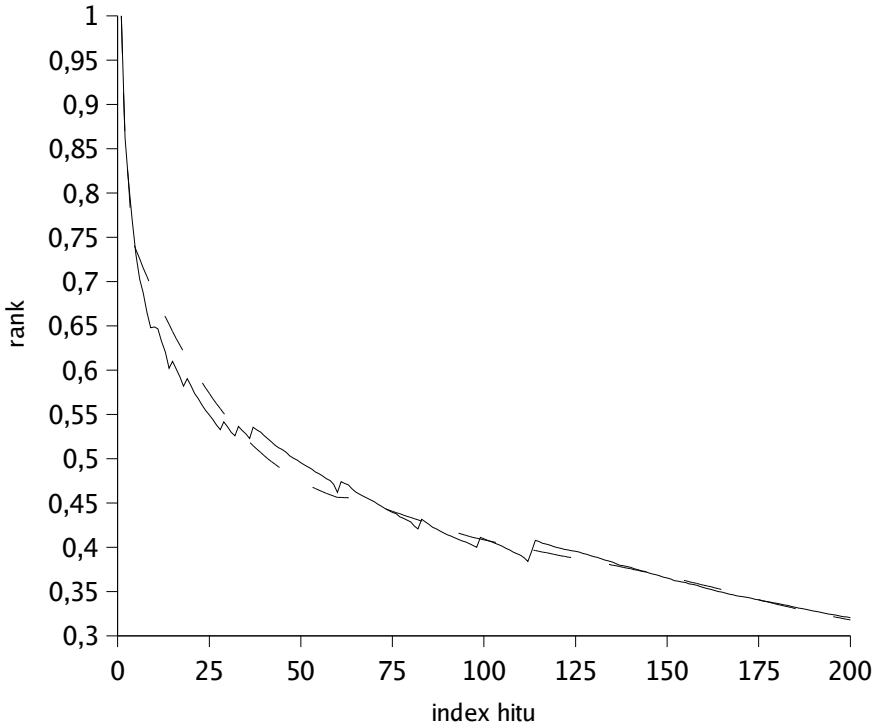
Tabulka 5.2: Testované parametry metody klouzavých průměrů

Přechodem od průměrů a_j k hodnotám r_j bylo dosaženo výrazného zlepšení. Získali jsme hladkou křivku a odstranili téměř všechny rostoucí úseky, které se vyskytovaly v posloupnosti průměrů a_j. Přesto se jevilo vhodné provést ještě jisté drobné korekce. První tři členy r_1, r_2, r_3 se příliš liší od průměrů a_1, a_2, a_3, jak ukazuje tabulka 5.3. Hodnoty r_1, r_2, r_3 tedy předefinujeme takto: $r_1 = a_1$, $r_2 = a_2$, $r_3 = a_3$. Dále byla provedena nepatrná úprava jisté hodnoty r_j (konkrétně číslo r_{61} se zvětšilo zhruba o $0,001$) tak, aby celá posloupnost $r_1, r_2, \ldots, r_{1000}$ byla nerostoucí. Situaci po těchto úpravách ilustruje obrázek 5.3. Průměry a_j jsou zobrazeny plnou čarou, hodnoty r_j čarou čárkovanou. Konkrétní číselné hodnoty prvních deseti členů posloupnosti $r_1, \ldots, r_{10}$ jsou uvedeny v tabulce 5.4.

Tímto jsme už vytvořili funkci, kterou agenti použijí pro odhad ranků. Ve výsledkové listině vyhledávače bude j-tému dokumentu (indexováno od 1) přiřazen rank r_j. Pro $j > 1000$ položme rank j-tého dokumentu roven nule. Toto dodefinování je spíše formalitou. Například vyhledávače uvedené v ta-

index hitu j	a_j	r_j
1	$1,0000000$	$0,7799057$
2	$0,8621195$	$0,7685094$
3	$0,8126759$	$0,7573823$
4	$0,7687979$	$0,7465208$
5	$0,7299028$	$0,7359216$

Tabulka 5.3: Porovnání průměrů a_j a hodnot r_j



Obrázek 5.3: Vyhlazení průměrů a_j

index hitu j	rank r_j
1	$1,0000000$
2	$0,8621195$
3	$0,8126759$
4	$0,7465208$
5	$0,7359216$
6	$0,7255811$
7	$0,7154960$
8	$0,7056629$
9	$0,6960782$
10	$0,6867386$

Tabulka 5.4: Ranky pro první stránku výsledkové listiny

bulce 5.1 pro jeden dotaz více než 1000 hitů nevracejí.[2]

5.2 Minimální váha vyhledávače

Při selekci vyhledávačů pro vyhodnocení dotazu je každému vyhledávači přiřazena váha R_{psq} (jak bylo popsáno v kapitole 3.3). Ta nabývá hodnot z intervalu $\langle R_{min}, 1\rangle$. Hodnota konstanty R_{min} určuje nejmenší možnou váhu vyhledávače a její volbou se v této kapitole budeme zabývat.

Zaměřme se na situaci, kdy pro jeden dotaz mají různé vyhledávače výrazně odlišné výsledkové listiny a kdy při selekci získá jeden vyhledávač váhu rovnou jedné a ostatním vyhledávačům je přiřazena váha blízká hodnotě R_{min}.

Pro metavyhledávač je zřejmě klíčové správně vybrat vyhledávače pro vyhodnocení dotazu a určit jejich váhy, právě když jsou jejich výsledky hodně rozdílné. Navíc k této situaci dochází v praxi velmi často. Není vůbec výjimkou položit dvěma vyhledávačům stejný dotaz a dostat odpovědi, kde mezi prvními deseti hity neobsahují jejich dvě výsledkové listiny ani jeden společný dokument.

Podobně případ, kdy všechny vyhledávače až na jeden mají váhu blízkou dolní hranici R_{min}, může nastat velmi snadno ve chvíli, kdy meta-index pro termy použité v dotazu obsahuje jen minimální informace. Uvažujme následující příklad. Uživatel zadá dotaz obsahující jediný term. Tento term nebyl v žádném předchozím dotazu, který metavyhledávač vyhodnocoval, použit. Dle meta-indexu tedy nelze jednotlivé vyhledávače rozlišit. Výběr vyhledávačů, kterým bude dotaz rozeslán, je pak víceméně náhodný (určený pouze na základě momentální rychlosti, jakou vyhledávače odpovídají). Předpo-

[2]Přesněji řečeno tuto hranici překračují vyhledávače All the Web, který vrací až 1010 hitů, a Seznam, jenž vrací maximálně 1009 hitů.

kládejme, že uživatel ve výsledkové listině vybral jako relevantní pouze jeden dokument a že tento dokument se vyskytoval ve výsledkové listině jen jediného vyhledávače. Takové chování uživatele se interpretuje jako pozitivní zpětná vazba pro zmíněný vyhledávač. Jestliže nyní dotaz zopakujeme (nebo ho budeme ladit tak, že přidáme další termy, pro které rovněž nejsou v meta-indexu žádné informace), získá právě tento vyhledávač výrazně vyšší váhu oproti všem ostatním. Při testování implementace metavyhledávače se ukázalo, že k tomu scénáři dochází poměrně často.

Za předpokladu, že jeden vyhledávač má váhu $R_{psq} = 1$ a ostatní vyhledávače mají váhu blízkou nule, a za předpokladu, že výsledkové listiny jednotlivých vyhledávačů jsou značně odlišné, ovšem dochází k tomu, že začátek (několik prvních stránek) výsledkové listiny metavyhledávače je stejný jako začátek výsledkové listiny vyhledávače, jenž má váhu rovnou jedné. Samotný fakt, že výsledek metavyhledávače je tvořen jediným vyhledávačem, může být nežádoucí. Navíc uvedený příklad ukazuje, že tato situace může snadno vzniknout pouze na základě vyhodnocení jednoho dotazu. Krom toho takový stav nelze jednoduše zvrátit, protože pro vyhledávače, jejichž výsledky si uživatel neprohlíží, není sbírána pozitivní zpětná vazba.

Na těchto úvahách je založena volba konstanty $R_{min} = 0,7$. Tato hodnota zaručuje, že jeden vyhledávač nemůže bez přispění ostatních vyhledávačů tvořit první stránku výsledků metavyhledávače (viz tabulka 5.4).

5.3 Míra aproximace pro slévání výsledků

Pro kombinování výsledkových listin vyhledávačů do jediné výsledkové listiny, kterou vrací metavyhledávač uživateli, byla použita aproximační verze algoritmu TA používající výhradně sekvenční přístup. Míra aproximace je určena konstantou θ. (viz kapitola 2.3) Nyní popíšeme experiment, který ukazuje, jak hodnota θ ovlivňuje rychlost vyhodnocování dotazů a přesnost odpovědí.

Idea je následující. Vezměme množinu testovacích dotazů. Necháme metavyhledávač dotazy vyhodnotit postupně pro různé hodnoty θ a tato vyhledávání porovnáme s případem $\theta = 1$, tj. s vyhledáváním, kde se pro slévání výsledků použije algoritmus TA bez jakékoli aproximace.

Vhodnější pro toto měření jsou spíše dotazy, pro které jednotlivé vyhledávače vracejí odpovědi obsahující velké množství hitů. Jestliže by například odpověď každého vyhledávače měla jen jednu stránku, potom metavyhledávač od každého vyhledávače tuto stránku získá a bude mít k dispozici kompletní výsledkové listiny. Pak ovšem slévací algoritmus není nucen provádět žádnou aproximaci a konkrétní hodnota θ nehraje žádnou roli. Zajímavější jsou tedy pro naše účely dotazy, kde metavyhledávač vrátí uživateli alespoň první stránku odpovědi dříve, než dosáhne konce výsledkových listin jednotlivých vyhledávačů.

Dotazy byly vždy rozeslány vyhledávačům Morfeo, Seznam a Jyxo. Volba vychází z toho, že všechny testovací dotazy jsou v češtině. Sada dotazů pro tento experiment byla opět získána ze seznamu naposledy položených dotazů vyhledávači na portálu Tiscali. Z tohoto zdroje bylo náhodně vybráno celkem 40 dotazů, na které byl kladen pouze jediný požadavek, jenž vychází z předchozí úvahy. Vybrán mohl být pouze takový dotaz, pro který každý z vyhledávačů Morfeo, Seznam a Jyxo vrací výsledek s alespoň 400 hity.

Při tomto měření nebyla prováděna standardní selekce vyhledávačů. Dotaz byl vždy rozeslán zmíněným třem vyhledávačům. Při slévání výsledků byly všem třem vyhledávačům přiřazeny stejné váhy.

Pro každou testovanou hodnotu θ bylo vyhodnoceno všech 40 dotazů. Metavyhledávač měl za úkol vytvořit vždy jen první stránku výsledkové listiny. Pro každý dotaz byl měřen čas, který trvalo vyhodnocení, počet HTTP požadavkům a počet společných hitů s vyhledáváním pro $\theta = 1$. Naměřený čas vyhodnocování dotazu by při standardním provozu byl delší ještě o dobu, kterou trvá selekce vyhledávačů. Budeme uvádět počet HTTP požadavků, které byly při vyhodnocování dotazu rozeslány všem třem vyhledávačům dohromady. Společné hity s přesným vyhledáváním počítáme následovně. Vezmeme první stránku výsledkové listiny a porovnáme ji s první stránkou odpovědi, kterou získáme vyhodnocením stejného dotazu tak, že při slévání použijeme neaproximující variantu algoritmu TA. Pro tyto dvě stránky nás zajímá velikost průniku, tedy počet hitů vyskytujících se na obou dvou. Shoda hitů je posuzována na základě rovnosti jejich URI. Počet hitů na stránce, tj. maximální možný počet shodných hitů, je v našem případě 10. Pro jednu hodnotu θ získáme popsaným postupem ke každému ze 40 dotazů hodnoty tří veličin (doba odpovědi, počet HTTP požadavků, shoda s přesným vyhledáváním). Pro každou veličinu nakonec ze 40 hodnot vypočteme aritmetický průměr. Výsledky měření ukazuje tabulka 5.5. Na jejich základě byla zvolena hodnota koeficientu aproximace $\theta = 1,7$.

θ	doba odpovědi	HTTP požadavky	shoda s přesným hledáním
1,0	35,450 s	185,175	10,000
1,2	11,191 s	86,175	9,425
1,4	5,940 s	52,250	8,425
1,6	3,099 s	29,200	7,875
1,7	2,208 s	23,175	7,625
1,8	1,939 s	20,075	7,325
2,0	1,213 s	16,125	7,150

Tabulka 5.5: Závislost kvality výsledků a rychlosti na míře aproximace

Při pohledu na naměřené hodnoty je dobré si uvědomit, jakou roli sehrála skutečnost, že mezi agenty a displejem je umístěna vyrovnávací paměť. Jak bylo popsáno v kapitole 4.3, agent získává od příslušného vyhledávače

další stránky výsledkové listiny, kdykoli má ve vyrovnávací paměti volné místo. Odešle se tedy několik HTTP požadavků, jejichž odpovědi se nepoužijí pro vytvoření první stránky výsledků metavyhledávače. Když jsme v tomto experimentu měřili počet HTTP požadavků potřebných pro vyhodnocení dotazu, byly do tohoto množství zahrnuty také požadavky, jejichž výsledky se při slévání nepoužili. Počet těchto nevyužitých HTTP požadavků je úměrný velikosti vyrovnávací paměti. V našem případě se jedná o 7 požadavků (2 stránky pro Morfeo, 2 stránky pro Jyxo a 3 stránky pro Seznam). Tyto „přebytečné požadavky“ ovšem nijak neovlivnily naměřenou dobu odpovědi. Ta je skutečně počítána jako čas, který uplyne od zadání dotazu do chvíle, kdy je první stránka výsledkové listiny vytvořena.

5.4 Rychlost odpovědi vyhledávače

Při selekci se pro každý vyhledávač počítá skóre. Jedna z jeho složek je odvozena z rychlosti, jakou daný vyhledávač odpovídá. Zajímá nás doba, která uplyne od chvíle, kdy byl vyhledávači odeslán HTTP požadavek, do okamžiku, kdy metavyhledávač dostane zpět HTTP odpověď. Pro výpočet této složky skóre potřebujme znát konstanty t_{th}, t_{to} a l. Hodnota t_{th} hraje roli nejkratší doby odpovědi, kterou obvykle potřebují i nejrychlejší vyhledávače. Konstanta t_{to} určuje maximální čas, po který metavyhledávač na odpověď vyhledávače čeká. Při výpočtu zmiňované složky skóre se použije rychlost odpovědi na l posledních HTTP požadavků, které byly vyhledávači zaslány. Více o výpočtu skóre vyhledávače bylo uvedeno v kapitole 3.3. Nyní nás zajímá odpověď na otázku, jak konstanty t_{th}, t_{to} a l nastavit.

S tímto cílem byl proveden následující experiment. Byla vygenerována sada 10 000 dotazů. Konkrétní podoba dotazů nemá příliš velký význam, nicméně pro úplnost dodejme, že každý dotaz obsahoval pouze jedno náhodně vybrané slovo anglického jazyka. Ke každému dotazu byl náhodně přiřazen jeden z vyhledávačů uvedených v tabulce 5.1. Poté byl každý dotaz odeslán příslušnému vyhledávači. Zajímala nás vždy pouze první stránka výsledkové listiny. Jinými slovy každý dotaz odpovídal právě jednomu HTTP požadavku. Pro každý požadavek byla zaznamenána doba odpovědi. Můžeme se domnívat, že jistou roli na rychlost, jakou vyhledávač odpovídá, má denní doba. Proto bylo zmiňovaných 10 000 požadavků rovnoměrně rozloženo do časového úseku 24 hodin v rámci pracovní části týdne.

Výsledky měření ukazuje tabulka 5.6. Časy odpovědí uváděné v milisekundách jsou rozděleny do několika intervalů. Jen zhruba 2 procenta odpovědí získal metavyhledávač rychleji než za 100 milisekund. (Většina z nich patří vyhledávači Morfeo.) A až na zanedbatelné množství požadavků byly odpovědi vráceny do 5 sekund. Hodnoty parametrů t_{th} a t_{to} tedy volíme $t_{th} = 0,1s$, $t_{to} = 5s$. Konstantu l z tohoto měření vyčíst nelze. Ta byla nastavena na $l = 15$, což je počet HTTP požadavků, který dle tabulky 5.5

a zvolenému koeficientu aproximace pro slévání $\theta = 1,7$ odpovídá zhruba dvěma dotazům (za předpokladu, že si uživatel vyžádá pouze první stránku výsledků).

doba odpovědi v ms	počet požadavků
$\langle 0, 100)$	215
$\langle 100, 200)$	1092
$\langle 200, 500)$	2173
$\langle 500, 1000)$	3735
$\langle 1000, 2000)$	2622
$\langle 2000, 5000)$	134
$\langle 5000, \infty)$	29

Tabulka 5.6: Rychlost odpovědi vyhledávačů

5.5 Testování uživateli

Na závěr byl proveden test, jehož cílem bylo získat nějakou informaci vypovídající o kvalitě výsledků produkovaných metavyhledávačem.

Metavyhledávač (tj. implementace popsaná v kapitole 4) byl zpřístupněn uživatelům. Po celou dobu běhu byly používány takové parametry algoritmů, které jsou uvedeny v předchozím textu. Dispečer pro vyhodnocení dotazu vybíral 3 vyhledávače z výčtu uvedeném v tabulce 5.1. Po zobrazení výsledků bylo možné na žádost uživatele vyhledávání zopakovat pro 5 vyhledávačů (což budeme dále považovat za nový dotaz). Výsledky, které zde popíšeme, pocházejí z 12 dnů provozu metavyhledávače. Za tuto dobu položilo 385 uživatelů[3] celkem 1043 dotazů. Pro vyhodnocování 116 dotazů bylo při selekci vybráno 5 vyhledávačů. Pro každého nového uživatele vygeneroval metavyhledávač identifikátor. Ten se pak uložil na straně klienta pomocí cookie a dle této informace se u dalších dotazů zjišťovalo, zda se jedná o téhož uživatele.

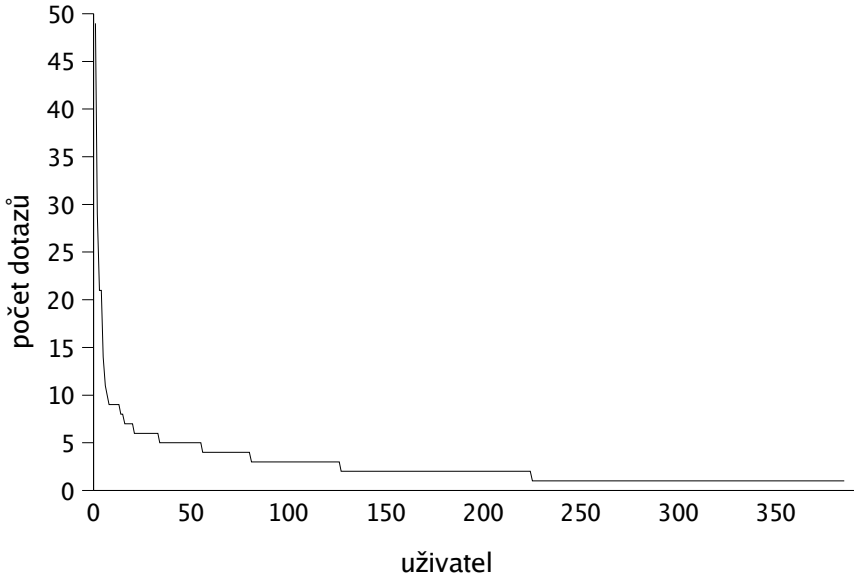
Na jednoho uživatele tedy v průměru připadá $2,71$ dotazu. Přesnější informací než průměr je distribuce dotazů mezi jednotlivé uživatele. Tu zachycuje tabulka 5.7. Uživatelé jsou zde rozdělení do skupin podle počtu dotazů, které metavyhledávači položili. Graf na obrázku 5.4 ukazuje jiným způsobem stejnou informaci. Všech 385 uživatelů je tu setříděno dle počtu dotazů. Graf pak zobrazuje, jak počet dotazů závisí na pořadí uživatele.

Předchozí data ukazují, že náš testovací vzorek je příliš malý na to, abychom mohli usoudit něco zajímavého o užitečnosti uživatelských profilů. Můžeme se však podívat na množinu dotazů jako celek a zaměřit se na

[3] jeden uživatel odpovídá jednomu uživatelskému profilu

počet dotazů	počet uživatelů
49	1
29	1
21	2
14	1
11	1
10	1
9	6
8	2
7	5
6	13
5	22
4	25
3	46
2	98
1	161

Tabulka 5.7: Distribuce dotazů mezi uživatele



Obrázek 5.4: Distribuce dotazů mezi uživatele

význam selekce vyhledávačů. Rozhoduje-li se, kterým vyhledávačům se dotaz rozešle a jaké se těmto vyhledávačům přiřadí váhy, měl by největší roli sehrát meta-index. V iniciálním stavu metavyhledávače je ale meta-index prázdný. Pro značné množství dotazů z našeho vzorku nebyly schopny hodnoty v meta-indexu jednotlivé vyhledávače rozlišit a jejich výběr byl víceméně náhodný. Dotazy můžeme tedy rozdělit do dvou skupin podle toho, zda meta-index ovlivnil volbu vyhledávačů. Množinu dotazů, pro které meta-index o výběru vyhledávačů nerozhodoval, označme Q_0. Zbývající dotazy pak budou tvořit množinu značenou Q_M. Každému dotazu dále přiřadíme kvalitu výsledků, které metavyhledávač vrátil. Tuto kvalitu pak můžeme pro množiny Q_0 a Q_M porovnat.

Je samozřejmě třeba rozhodnout, jakým způsobem budeme kvalitu výsledkové listiny pro daný dotaz měřit. Můžeme vyjít z toho, jaké dokumenty si uživatel prohlédl (tj. na které odkazy na HTML stránce s výsledky klikl myší). Lze předpokládat, že prohlédnuté dokumenty mají větší míru relevance než ostatní, protože uživatel získá hrubou představu o dokumentu už podle názvu a popisku. Za kvalitní výsledky považujme ty, kde si uživatel vyžádal dokumenty vyskytující se na začátku výsledkové listiny. Jestliže naopak uživatel prohlížel hity s vysokým pořadovým číslem, znamená to, že metavyhledávač nedal na přední místa relevantní dokumenty, nebo dokonce, že uživatel nemůže žádný relevantní dokument vůbec najít. Pro každý dotaz q spočítejme hodnotu r_q jako průměrné pořadí prohlédnutého dokumentu ve výsledkové listině (indexováno od 1). Čím bude toto číslo menší, tím jsou výsledky vrácené metavyhledávačem kvalitnější.

Definice čísla r_q neřeší situaci, kdy uživatel žádné dokumenty z výsledkové listiny neprohlížel. Takových případů je však v našem testovacím vzorku velice mnoho. Z celkového počtu 1043 dotazů do této kategorie spadá 663 z nich, což činí $63,6\%$. Přitom pouze pro 48 dotazů metavyhledávač vrátil prázdnou výsledkovou listinu. Přesto si však dovolíme tuto skupinu dotazů zanedbat a z dalšího měření ji vyloučit. Množství těchto dotazů je příliš velké na to, abychom se mohli domnívat, že uživatelé všechny hity ve výsledkové listině zamítli už podle názvu a popisku. Pravděpodobnějším vysvětlením je skutečnost, že se jednalo o nově spuštěný metavyhledávač a uživatelé zavítali na jeho stránky často ze zvědavosti, ne však proto, že něco skutečně hledali. V každém případě dotazy, kde uživatel žádné dokumenty neprohlížel, zastupují v množinách Q_0 a Q_M zhruba stejný podíl (pro Q_0 je to $64,7\%$ dotazů a pro Q_M jde o $59,6\%$), a proto je zajímavé zabývat se především ostatními dotazy.

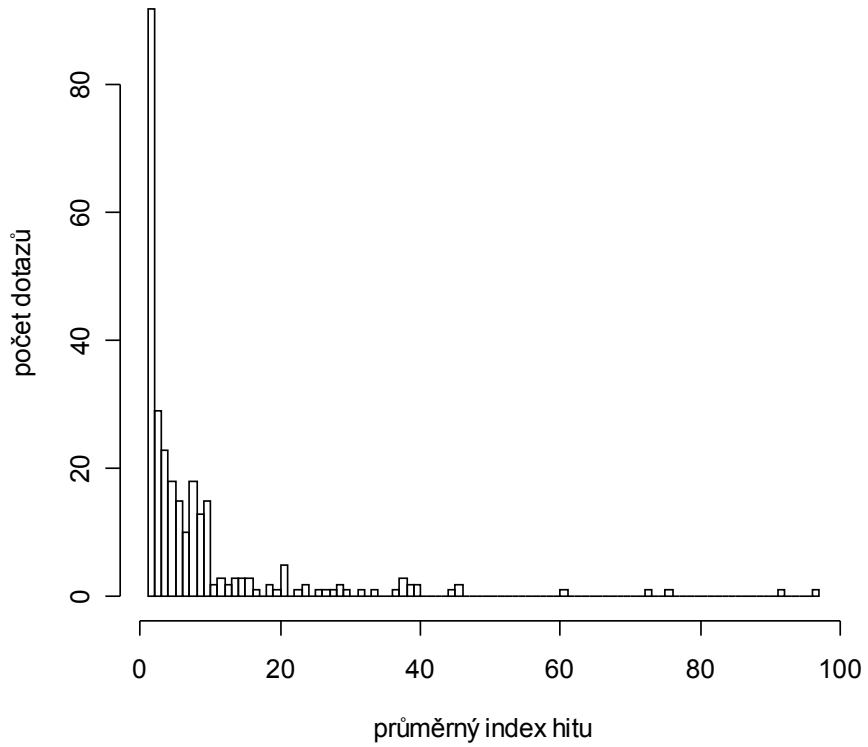
Dále byl z uvažovaného vzorku vyřazen ještě jeden dotaz. Jeho průměrný index prohlédnutého dokumentu byl $r_q = 287,3$. Přitom pro jakýkoli jiný dotaz q platí $r_q < 100$. U zmíněného vybočujícího dotazu si uživatel prohlížel i dokument s pořadovým číslem 801. I zde můžeme předpokládat, že jde o „rozmar testera“ a že tento uživatel neprohlédl všech předchozích 800 popisků dokumentů.

Symboly Q_0 či Q_M budeme dále značit množiny dotazů po odstranění výše zmíněných případů. Máme tedy dvě skupiny dotazů o velikostech $|Q_0| = 285$ a $|Q_M| = 94$.

Pro každou skupinu dotazů můžeme spočítat aritmetický průměr z hodnot r_q, tj.

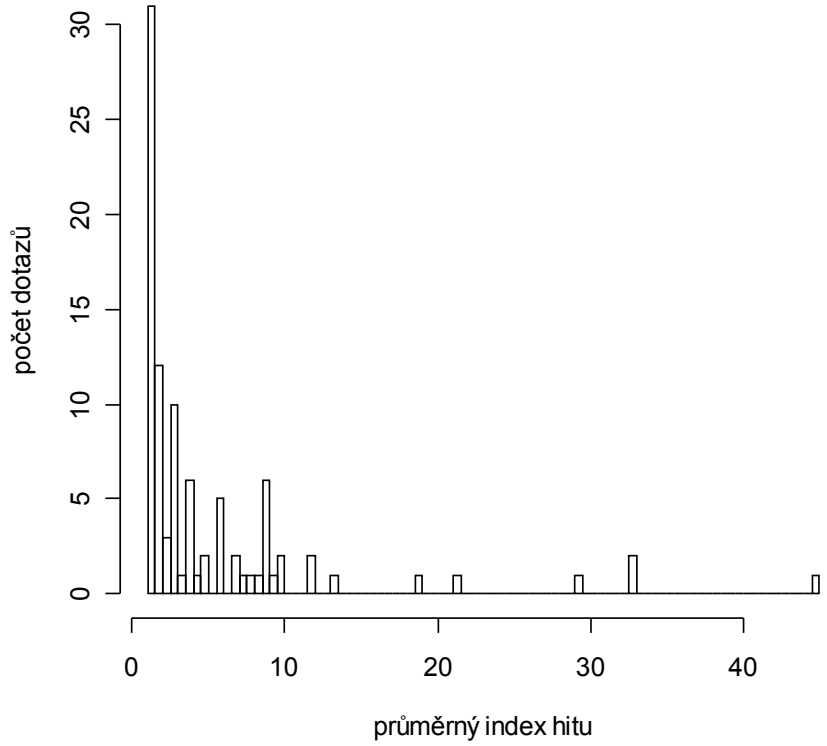
$$R_0 = \frac{1}{|Q_0|} \cdot \sum_{q \in Q_0} r_q,$$
$$R_M = \frac{1}{|Q_M|} \cdot \sum_{q \in Q_M} r_q.$$

Získáme tak hodnoty $R_0 = 8,69$ a $R_M = 5,35$. Distribuci hodnot r_q pro množiny Q_0 a Q_M ukazují histogramy na obrázcích 5.5 a 5.6.



Obrázek 5.5: Distribuce hodnot r_q pro množinu dotazů Q_0

Průměry R_0 a R_M poměrně přesvědčivě ukazují, že metavyhledávač dával na dotazy z množiny Q_M kvalitnější odpovědi než na dotazy z Q_0. Tuto



Obrázek 5.6: Distribuce hodnot r_q pro množinu dotazů Q_M

hypotézu můžeme podpořit i statistickým výpočtem. Lze předpokládat, že naměřené hodnoty $\{r_q \,|\, q \in Q_0\}$ tvoří náhodný výběr ze spojitého rozdělení s distribuční funkcí F_0 a dále že $\{r_q \,|\, q \in Q_M\}$ je na něm nezávislý náhodný výběr ze spojitého rozdělení s distribuční funkcí F_M. Ověříme, že platí nerovnost $F_0 < F_M$, pomocí dvouvýběrového Wilcoxonova testu. Tato metoda má pouze výše uvedené předpoklady. Více informací o ní je možné nalézt v knize [1]. Nulová hypotéza bude označovat tvrzení $F_0 = F_M$. Vztah $F_0 < F_M$ je alternativní hypotézou. Provedení testu na hladině $0,01$ vedlo k zamítnutí nulové hypotézy (tj. pravděpodobnost, že nulová hypotéza je zamítnuta, ačkoli platí, je nejvýše 1%). Přijmeme proto alternativní hypotézu, tedy tvrzení, že nerovnost $F_0 < F_M$ platí.

Prokázali jsme tak, že odpovědi na dotazy Q_M byly kvalitnější než odpovědi na dotazy Q_0. Tato skutečnost optimisticky vypovídá o významu meta-indexu. Je však dobré si uvědomit, že na příslušnost dotazu do mno-

žiny Q_M má vliv i ladění dotazu. Pokud bychom ale chtěli provést kvalitnější měření a tento vliv eliminovat, potřebovali bychom větší objem vstupních dat.

Kapitola 6

Možnosti rozšíření a příbuzné práce

V této kapitole se zmíníme o možnostech, jak na tuto práci dále navázat. Jako inspirace mohou sloužit existující metavyhledávače. Zmiňme se nyní o některých z nich a uveďme jejich zvláštnosti.

SavvySearch - `http://www.search.com`
: Metavyhledávač SavvySearch byl již v předchozím textu často zmiňován. Jde o projekt, který vznikl v akademickém prostředí. Jeho autoři se zaměřili především na řešení selekčního problému. Existují dokumenty (např. [5]) poměrně podrobně popisující některé algoritmy, které tento metavyhledávač používá. Později se přesunul do komerční sféry a jako důsledek už bohužel o jeho dalším vývoji neexistuje podobná volně dostupná dokumentace.

MetaCrawler - `http://www.metacrawler.com`
: I tento metavyhledávač vznikl na akademické půdě. Jeho zvláštností bylo vlastní uspořádávání hitů ve výsledkové listině. MetaCrawler sám přiřazoval ranky dokumentům, které od vyhledávačů získal, na základě analýzy textu. Stahoval všechny dokumenty ve výsledkové listině a prováděl jejich analýzu až ve chvíli, kdy vyhodnocoval dotaz uživatele. I tento metavyhledávač přešel do komerčního prostředí.

Dogpile - `http://www.dogpile.com`
: Dogpile patří mezi nejpopulárnější metavyhledávače na Webu. Jeho předností je přehledné zobrazování výsledkových listin od jednotlivých vyhledávačů i výsledkové listiny, která vznikla jejich slitím.

Vivísimo - `http://vivisimo.com`
: Vivísimo je metavyhledávač, který zobrazuje výsledkovou listinu klasickým způsobem uspořádanou dle ranků, ale zároveň je schopen i roztřídit nalezené dokumenty do kategorií (clusters). Tyto kategorie mo-

hou mít i několik úrovní. Jejich hierarchie je pak zobrazována jako strom.

KartOO - `http://www.kartoo.com`
U tohoto metavyhledávače rozhodně stojí za povšimnutí netradiční způsob zobrazování výsledků. Nalezené dokumenty jsou uživateli prezentovány v podobě grafu. Ten obsahuje dva typy vrcholů. Vrcholy prvního typu reprezentují nalezené dokumenty (u nichž je navíc graficky znázorněno, zda se jedná o významnější či méně významný dokument). Vrcholy druhého typu reprezentují termy, které jsou pro tyto dokumenty významné. Hrany pak vedou vždy mezi uzlem prvního a uzlem druhého typu. Každý dokument je tak spojen se svými charakteristickými termy.

Queryster - `http://www.queryster.com`
Queryster je jedním z metavyhledávačů, které umožňují uživateli vybrat si konkrétní vyhledávač, jenž bude jeho dotaz vyhodnocovat. Metavyhledávače tohoto typu se někdy označují termínem *All-In-One*. Nekombinují výsledkové listiny, pouze nabízejí uživateli jednotné rozhraní pro několik vyhledávačů.

Mluvíme-li o příbuzných pracích, stojí za to se alespoň velmi stručně zmínit o jednom alternativním přístupu, kterým se zabývají například články [11] nebo [12]. V našem případě obsahoval meta-index pro každou dvojici vyhledávač-term pouze jedno číslo (přesněji řečeno jedno číslo pro každý uživatelský profil) vyjadřující vhodnost vyhledávače pro vyhodnocení dotazů obsahujících daný term. Autoři zmíněných článků podrobně navrhli algoritmus, který se pro každou takovou dvojici snaží udržovat hned několik veličin odvozených od frekvence termů v dokumentech. Jejich hodnoty se snaží charakterizovat nejvýznamnější dokument v databázi vyhledávače vzhledem k příslušnému termu. Tento přístup ovšem vyžaduje buď velmi úzkou spolupráci metavyhledávače a jednotlivých vyhledávačů, které patrně nejsme schopni v prostředí Webu dosáhnout, nebo schopnost metavyhledávače analyzovat text dokumentů. Metoda se navíc zaměřuje na situaci, kdy počet podporovaných vyhledávačů je příliš vysoký (několik set) a při selekci není únosné počítat skóre na základě hodnot meta-indexu pro každý z nich.

Některá úskalí se ukázala také při návrhu metavyhledávače. Jejich řešení je další možnost, jak na tuto práci navázat. Uveďme nyní výčet těchto problémů.

- Možná největším problémem, který se vyskytl, je nestálost formátu dotazu a výsledkové listiny vyhledávačů. Obzvláště u výsledkové listiny se tento formát mění velmi často. Jde obvykle o změny, kterých si běžně uživatel komunikující s vyhledávačem přes jeho webové rozhraní vůbec nevšimne. Jsou to drobné modifikace týkající se některých atributů

HTML elementů, HTML komentářů apod. Tyto detaily jsou ale klíčové pro správné parsování HTML kódu s výsledky. Značným přínosem by byla schopnost metavyhledávače automaticky analyzovat (alespoň částečně) webové rozhraní vyhledávačů a dynamicky reagovat na jeho změny.

- Metavyhledávač je nucen odhadovat ranky dokumentů, které mu vyhledávače vrací. Jednou z možností, jak zlepšit tento odhad, je vzít v potaz délku výsledkové listiny. HTML stránka s výsledky vrácená vyhledávačem obvykle obsahuje přibližný počet hitů. (Tato informace je ovšem často velmi nepřesná.) K odhadu délky výsledkové listiny můžeme použít i počet a frekvenci výskytu termů použitých v dotazu.

- Slévací algoritmus TA neřeší uspořádání hitů v rámci jedné stránky výsledkové listiny. Metavyhledávač tedy tyto hity uspořádává podle dolního odhadu jejich ranku. Je samozřejmě možné, že zapojením i horního odhadu dosáhneme kvalitnějšího výsledku.

- Současný návrh metavyhledávače předpokládá, že počet vyhledávačů, kterým se rozesílá dotaz, zvolí uživatel. Alternativně může o tomto počtu samozřejmě rozhodovat přímo metavyhledávač. Určitě si lze také představit, že na tomto rozhodování budou uživatel i metavyhledávač spolupracovat.

- Při odvození skóre vyhledávače na základě hodnot meta-indexu se kombinují dvě složky. Jedna odpovídá konkrétnímu profilu uživatele, druhá vychází ze souhrnného hodnocení všech uživatelů. Při této kombinaci je maximální možný podíl profilové složky omezen konstantou α_{max} (viz kapitola 3.3). Pro získání konkrétní hodnoty této konstanty nebylo provedeno žádné měření. Návrh a realizace takového měření je jednou z dalších možností, jak navázat na tuto práci.

- Stárnutí meta-indexu bylo vyřešeno poměrně jednoduchým způsobem. Lze například přidat dynamickou změnu rychlosti stárnutí v závislosti na množství dat, která byla v poslední době do meta-indexu uložena.

- Meta-index po určitém čase sbírání zpětné vazby bude pro každý term upřednostňovat určité vyhledávače. To je účel, pro který byl meta-index vytvořen. Na druhou stranu se může stát, že meta-indexem znevýhodněné vyhledávače už nebudou používány pro vyhodnocování dotazů s příslušnými termy, tudíž pro ně nebude sbírána zpětná vazba a tedy nedostanou příležitost svou bilanci napravit. Zajímavá může být tato situace u nově vytvořeného uživatelského profilu. Jestliže pro tento profil bude v meta-indexu málo informací, bude se selekce vyhledávačů řídit společným hodnocením všech uživatelů. Některé vy-

hledávače, které budou takto znevýhodněny, by ale právě pro nového uživatele mohly být relevantní.

Jedním možným řešením je přidat do selekce vyhledávačů náhodný faktor. K vyhledávačům vybraných dle jejich skóre můžeme například s jistou pravděpodobností p_w přidat navíc jeden vyhledávač, kterému by jinak pro nízké skóre dotaz poslán nebyl. U tohoto přístupu, stojí za povšimnutí jeden jev. Takto náhodně zvolený vyhledávač bude mít evidentně nejnižší skóre mezi vybranými vyhledávači, tedy také nejmenší váhu při slévání výsledků. Vzhledem k jeho způsobu volby je vhodné, že má jen malou schopnost ovlivnit celkový výsledek. Na druhou stranu zmíněná váha nijak neovlivňuje sběr zpětné vazby, takže v tomto ohledu bude mít náhodně zvolený vyhledávač rovnocennou pozici s ostatními.

Kapitola 7

Závěr

Cíle této práce popsané v kapitole 1 byly splněny. Vznikl návrh metavyhledávače s podporou uživatelských profilů. Tento návrh komplexně pokrývá všechny klíčové problémy, které musí metavyhledávač na Webu řešit (tj. selekční problém, komunikaci s jednotlivými vyhledávači a slévání jejich výsledkových listin). Byla vytvořena implementace metavyhledávače včetně webového rozhraní. Dále se provedlo několik měření, ze kterých vycházejí volby parametrů pro použité algoritmy. Metavyhledávač byl také testován uživateli a analýza tohoto zkušebního provozu je rovněž součástí tohoto textu.

Problematika metavyhledávačů je však značně široká oblast. Je tu řada otevřených problémů, jejichž řešení už bylo mimo rámec této práce. Mezi největší z nich patří častá změna formátu jednotlivých vyhledávačů. Velkým přínosem by byla schopnost metavyhledávače automaticky se těmto změnám přizpůsobovat.

Podpora uživatelských profilů je součástí zde popsaného návrhu. Je velmi těsný vztah mezi uživatelským profilem a sběrem zpětné vazby. Preference jednotlivých uživatelů však ovlivňují pouze selekci vyhledávačů. Další výzvou je tedy navrhnout širší využití uživatelských profilů a přidat další typy zpětné vazby.

Příloha A

Obsah CD

K této práci je přiloženo CD, které obsahuje implementaci metavyhledávače (popsanou v kapitole 4) včetně webového rozhraní ve formě JSP stránky a dále také data vztahující se k provedeným experimentům.

Metavyhledávač je napsán v jazyce Java. Pro jeho spuštění je potřeba mít nainstalované prostředí J2SE Runtime Environment (JRE), verzi 5.0 (lze stáhnout na stránkách `http://java.sun.com`). Instalace programu spočívá pouze ve zkopírování adresáře `metasearch` z CD na pevný disk. Je třeba zajistit, aby měl program právo zápisu do podadresářů `feedback` a `logs`. Součástí adresáře `metasearch` jsou veškeré potřebné knihovny. Implementaci metavyhledávače obsahuje Java archiv `metasearch.jar`. V tomto archivu jsou také zdrojové kódy.

Metavyhledávač lze spustit jako samostatný program. Tento přístup je vhodný především k testování či ladění. Po spuštění dojde k vyhodnocení jednoho dotazu a poté se program ukončí. Dotaz, identifikátor uživatelského profilu a případné identifikátory vyhledávačů se předávají na příkazové řádce. Příkaz pro spuštění vypadá následovně.

```
java -jar metasearch.jar PROFILE QUERY [ENGINES]
```

- `PROFILE` může být libovolný neprázdný řetězec. Nové profily není třeba explicitně vytvářet. Metavyhledávač pouze rozlišuje jednotlivé uživatele na základě porovnávání těchto řetězců.
- `QUERY` specifikuje dotaz. Jednotlivé termy oddělíme mezerou, fráze se uzavírají do uvozovek. Celý dotaz je však třeba předat jako jediný parametr.
- `ENGINES` je výčet vyhledávačů, kterým bude dotaz rozeslán. Každý identifikátor vyhledávače se předává jako samostatný parametr. Je-li seznam vyhledávačů prázdny, provede metavyhledávač automatickou selekci, jak je popsána v kapitole 3.3. Adresář `formats` obsahuje jeden soubor pro každý podporovaný vyhledávač. Identifikátor vyhledávače je definován v tomto souboru pomocí atributu `name` elementu `search`.

Příkladem spuštění může být příkaz:

```
java -jar metasearch.jar Tom "weather Europe" Ask AOL
```

Uvozovky zde neoznačují frázi, pouze zabraňují interpretaci mezery mezi termy dotazu jako oddělovače parametrů příkazu.

Po spuštění zobrazí metavyhledávač na standardním výstupu další instrukce. Může být vhodné přesměrovat standardní chybový výstup, na který se vypisují logované zprávy (obvykle jde jen o ladící informace).

Je zamýšleno používat metavyhledávač spíše jako knihovnu. Příklad takového použití dává třída

```
org.egothor.metasearch.examples.UsageDemo
```

Tato třída je rovněž součástí archivu `metasearch.jar`.

Na tomto místě je vhodné ještě jednou připomenout, že při zpracování výsledků některého vyhledávače může dojít k problémům, které jsou způsobené neaktuální specifikací jeho formátu dotazu a odpovědi.

Webové rozhraní k metavyhledávači je na CD v adresáři `jsp`.

Dále CD obsahuje adresář `experiments`, kde jsou logy a některá další data pořízená při provádění experimentů popsaných v kapitole 5.

- Podadresář `rank` se vztahuje k hledání funkce pro odhad ranku (viz kapitola 5.1), kdy bylo vyhledávači Morfeo odesláno několik desítek dotazů. Jsou zde seznamy ranků pro každý dotaz tak, jak je vyhledávač Morfeo vrátil. Dále tento adresář v souboru `rank_list.txt` obsahuje hodnoty funkce, která z tohoto měření vzešla a kterou metavyhledávač pro odhad ranku používá.

- V podadresáři `theta` jsou logy vzniklé při hledání koeficientu aproximace θ pro slévací algoritmus a dále také použitá sada dotazů. Tento experiment je popsán v kapitole 5.3.

- Podadresář `response_time` obsahuje logy pořízené při měření, jehož cílem bylo zjistit, jakou rychlostí odpovídají vyhledávače na HTTP požadavky (viz kapitola 5.4).

- Data pocházející z testovacího provozu metavyhledávače (viz kapitola 5.5) jsou obsažena v podadresáři `user_test`. Kromě logů je zde také stav meta-indexu.

Příloha B

Formát vyhledávače Jyxo

```
<search
    name="Jyxo"
    description="Jyxo"
    method="GET"
    action="http://jyxo.cz/s"
    queryCharset="iso-8859-2"
>

<input name="s" user>
<input name="i" value="mozilla">
<input name="stem" value="1">
<inputnext name="skip" initial="0" factor="15">

<interpret
    browserResultType="result"
    resultListStart=""
    resultListEnd=""
    resultItemStart="<!--m-->"
    resultItemEnd="<!--n-->"
    itemURIStart="<A HREF=&#34;"
    itemURIEnd="&#34;"
    itemTitleStart=";&#34;>"
    itemTitleEnd="</A>"
    itemTitleSkip="(<b>)|(</b>)"
    itemSnippetStart="<div class=re>"
    itemSnippetEnd="</div>"
    itemSnippetSkip="(<b>)|(</b>)"
>

</search>
```

Literatura

[1] Anděl J.: *Základy matematické statistiky.* Preprint, MFF UK Praha, 2002

[2] Baeza-Yates R., Frakes W. B.: *Information Retrieval: Data Structures and Algorithms.* Prentice Hall, 1992, ISBN: 0-13-463837-9

[3] Baeza-Yates R., Ribeiro-Neto B.: *Modern Information Retrieval.* Addison Wesley, 1999, ISBN: 0-201-39829-X

[4] Cipra T.: *Analýza časových řad s aplikacemi v ekonomii.* SNTL, Praha, 1986, ISBN: 99-00-00157-X

[5] Dreilinger D.: *Description and Evaluation of a Meta-Search Agent.* Master Thesis, Department of Computer Science, Colorado State University, 1996

[6] Dreilinger D., Howe A. E.: *SavvySearch: A Meta-Search Engine that Learns which Search Engines to Query.* AI Magazine, 18(2), 1997

[7] Fagin R.: *Combining Fuzzy Information: an Overview.* ACM SIGMOD Record, June 2002, ISSN: 0163-5808

[8] Fagin R., Lotem A., Naor M.: *Optimal aggregation algorithms for middleware.* Journal of Computer and System Sciences, June 2003, ISSN: 0022-0000

[9] Jansen B. J.: *The effect of query complexity on Web searching results.* Information Research, Vol. 6 No. 1, 2000, ISSN: 1368-1613

[10] Korfhage R. R.: *Information Storage And Retrieval.* Wiley, 1997, ISBN: 0-471-14338-3

[11] Liu K.-L., Meng W., Yu C.: *Building Efficient and Effective Metasearch Engines.* ACM Computing Surveys, March 2002, ISSN: 0360-0300

[12] Liu K.-L., Meng W., Yu C.: *Efficient and Effective Metasearch for Text Databases Incorporating Linkages among Documents.* ACM SIGMOD Record, 2001, ISBN: 1-58113-332-4

[13] Mehlhorn K.: *Data Structures and Efficient Algorithms, Vol. 1: Sorting and Searching*. Springer Verlag, 1984, ISBN: 0-387-13302-X

[14] RFC 2616: Hypertext Transfer Protocol - HTTP/1.1