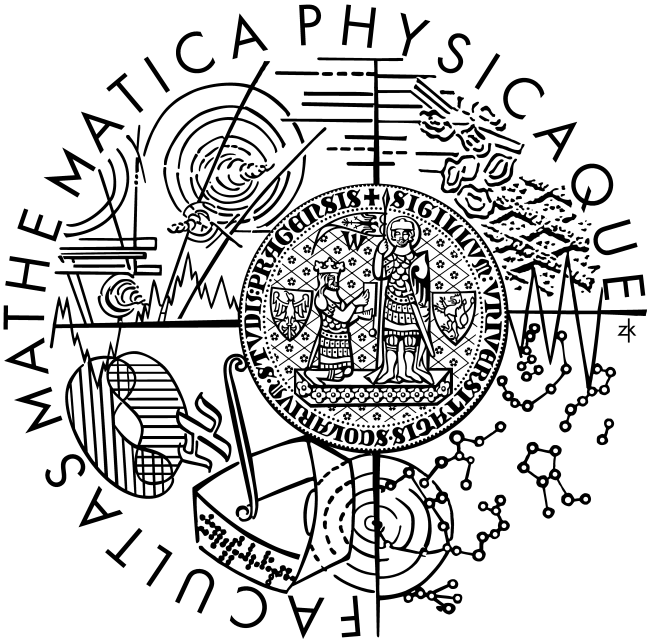
UNIVERZITA KARLOVA

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Daniel Benčík

Hledání téměř identických dokumentů ve velkých kolekcích

Vedoucí bakalářské práce: Mgr. Pavel Pecina

Studijní program: Informatika, Správa počítačových systémů

2007

Poděkování: Děkuji Mgr. Pavlu Pecinovi za jeho inspirativní nápady, trpělivost a odborné vedení, bez nichž by tato práce nikdy nevznikla.

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

V Praze dne …....

Obsah

Název práce: Hledání téměř identických dokumentů ve velkých kolekcích

Autor: Daniel Benčík

Katedra (ústav): ÚFAL MFF UK

Vedoucí bakalářské práce: Mgr. Pavel Pecina

e-mail vedoucího: pecina@ufal.mff.cuni.cz

Abstrakt: Tato práce se zabývá problematikou vyhledávání dokumentů, které jsou si natolik podobné, že je můžeme považovat za (téměř) stejné, a to v kolekcích čítajících až miliony dokumentů. Největší důraz práce je kladen na porovnání nových, rychlých algoritmů řešících danou úlohu s algoritmy stávajícími, které jsou díky své složitosti pro obrovské kolekce nepoužitelné. Práce obsahuje implementaci obou metod přístupu k dané problematice spolu s aplikacemi umožňujícími experimentální porovnání obou těchto metod.

Klíčová slova: blízké duplicity, podobnost dokumentů, signaturový model

Title: Near duplicate detection in large document collections

Author: Daniel Benčík

Department: ÚFAL MFF UK

Supervisor: Mgr. Pavel Pecina

Supervisor´s e-mail address: pecina@ufal.mff.cuni.cz

Abstract: This thesis deals with the problematics of detecting documents, which are so similair one to another, that we can consider them to be (nearly) identical and that in collections having up to millions of documents. The greatest aim of this thesis is a comparison of new, fast algorithms designed to solve this task with current algorithms, which due to their complexitiy cannot be used for large collections. The thesis contains an implementation of both new and current methods of solving the given task toghether with applications that are designed to experimentally compare these methods.

Keywords: near duplicate, document similarity, signature based model

Úvod

V posledních letech díky rozmachu Internetu a vývoji v oblasti uchovávání dat vznikly rozsáhlé kolekce dokumentů – vezměme pro příklad český internet, který je možné chápat jako obrovskou kolekci a jehož obsah je odborníky odhadován na sto milionů dokumentů. Jako následek řady faktorů (např. redundance/mirroring, spam, plagiátorství) mohou takovéto kolekce obsahovat některé dokumenty více než jednou – za kopii dokumentu v tomto případě považujeme buď naprostou shodu originálu a kopie či podobnost natolik velikou, že oba dokumenty mohou být považovány za stejné. Rozšíření téměř identických kopií dokumentů je často nepředvídatelné a potenciálně s sebou přináší problémy týkající se zvýšených nároků na uchovávání dat, snížení výkonu search enginů a kvality jejich odpovědí apod. Velké koncentrace (téměř) identických dokumentů by též mohly ovlivnit statistiky týkající se kolekcí, následkem čehož by mohly být nepříznivě ovlivněny výsledky některých aplikací založených na strojovém učení, které byly na takových kolekcích trénované.

Pojmem (téměř) identické dokumenty (nebo též blízké duplicity) budeme nazývat pár takových dokumentů, které svou vzájemnou podobností překonávají jistou hranici, která závisí na přání či úsudku uživatele (tedy není předem určena nějakou autoritou). Celkovým záměrem je nalezené páry téměř identických dokumentů z kolekce odstranit tak, aby nedocházelo k výše zmíněným negativní jevům (to už ale netvoří náplň této práce).

Cílem této bakalářské práce je navázat na ročníkový projekt, který se svou myšlenkou opíral o poznatky z [1], a vytvořit tak efektivní implementaci skupiny aplikací, pomocí nichž bude možné ve velmi rozsáhlých kolekcích dokumentů (řádově až miliony) nalézt velmi blízké duplicity a to, v porovnání s dnes běžně používanými metodami, ve velmi krátkém čase. Druhým cílem naší práce je experimentálně ověřit chování a použitelnost námi implementovaných nových přístupů v praxi, resp. tyto porovnat se stávajícími přístupy.

Text této práce je rozdělen na tři části. První část popisuje teoretický úvod, který tvoří myšlenkovou kostru našeho přístupu k problému. Druhá část práce je věnována provedeným experimentům a konkluzím na jejich základě vytvořeným, třetí část tvoří detaily týkající se samotných aplikací a to jak z uživatelského, tak z technického hlediska.

Kapitola 1

Teoretický úvod

1.1 Základní pojmy

V této kapitole definujeme pojmy a postupy používané v následujících částech práce.

1.1.1 Kosinus úhlu dvou n-dimenzionálních vektorů

Předpokládejme dva zcela libovolné *n*-dimenzionální vektory *u* a *v*. Pokud budeme chtít vypočítat velikost úhlu jimi svíraného, resp. kosinus tohoto úhlu, využijeme následujících vztahů:

$$\cos(u,v) = \frac{u.v}{|u|*|v|} \qquad u.v = \sum_{i=0}^{n} u_i * v_i \qquad |u| = \sqrt{\sum_{i=0}^{n} u_i^2}$$

kde $u.v$ značí skalární součin vektorů *u* a *v*, $|u|$ značí aritmetickou délku vektoru *u*.

1.1.2 Hammingova podobnost, resp. hammingova vzdálenost

Uvažujme dvě libovolné, stejně dlouhé, posloupnosti čísel 0 a 1, v našem případě je jejich délka nastavena na hodnotu 20. Hammingovou podobností dvou posloupností *A* a *B* nazveme počet pozic, na nichž mají obě posloupnosti shodnou hodnotu. hammingovou vzdáleností dvou posloupností *A* a *B* nazveme počet pozic, na nichž mají obě posloupnosti různou hodnotu, neboli

$$hamm\_podobnost(A,B) = \left|\{\ i : A[i] = B[i]\}\right|$$
$$hamm\_vzdalenost(A,B) = \left|\{\ i : A[i] \neq B[i]\}\right|$$

V příkladu uvedeném níže je hammingova podobnost posloupností *A* a *B* rovna dvanácti zatímco jejich vzdálenost je rovna osmi.

sig A	00101100100110010010
sig B	01100101111010110000
XOR	01001001011100100010

Obr. 1.1: Ilustrace hammingovy vzdálenosti/podobnosti

Jak ukazuje Obr. 1.1, je hammingova podobnost, resp. vzdálenost posloupností *A* a *B* rovna počtu nul, resp. jedniček ve výsledku operace XOR aplikované na obě posloupnosti.

1.1.3 Další definice

Definice: Poměrnou hammingovou podobností budeme nazývat výraz

$$pomerna\_hamm\_podobnost(A,B) = \frac{hamm\_podobnost(A,B)}{n}, n = |A| = |B|$$

Definice: Matice podobnosti je termín označující dvourozměrnou tabulku, v níž je v buňce s indexy *i, j* číslo udávající vzájemnou podobnost dokumentů doc_i a doc_j.

Definice: Vektorem dokumentu doc_i při použití slovníku V máme na mysli vektor reálných čísel dimenze $|V|$, jehož j-tá složka vyjadřuje zastoupení j-tého slova slovníku V v dokumentu doc_i.

Definice: Kosinovou podobností dvou dokumentů doc_i a doc_j máme na mysli stonásobek kosinu úhlu svíraného vektory obou dokumentů.

Definice: Pojmem term rozumíme slovo oddělené mezerou.

Definice: Pojmem bag of words rozumíme množinu všech slov nějakého dokumentu. V dalším textu budeme místo tohoto pojmu používat zkratku BOW.

Definice: Pojmem signatura dokumentu máme na mysli posloupnost alfanumerických znaků, které nějakým způsobem reflektují jeho obsah.

Definice: Pojmem transpozice máme na mysli libovolnou permutaci s právě jedním cyklem a to délky dva.

1.2 Klasické postupy hledání téměř identických dokumentů

Dvě skupiny do dnešní doby nejpoužívanějších metod na hledání téměř identických dokumentů jsou:

1) **Vektorový model dokumentů / kosinová podobnost** – pro každý dokument je vytvořen vektor vyjadřující jeho slovní obsah. Pro každou dvojici dokumentů je následně určena míra jejich podobnosti pomocí nějaké, nejčastěji kosinové, míry.

   VÝHODY
   - vysoká přesnost výsledků

   NEVÝHODY
   - vysoká časová složitost – díky nutnosti porovnat všechny dvojice dokumentů a naplnit matici podobnosti je časová složitost kvadratická (co do počtu dokumentů kolekce)
   - vysoká paměťová náročnost – po dobu trvání výpočtů je potřeba v paměti uchovávat vektory všech dokumentů kolekce

   V dalším textu se na tento výpočet model budeme odkazovat termínem vektorový model.

2) **Signaturový model dokumentů –** pro každý dokument je vytvořena signatura, tj. pomocí nějaké hashovací funkce (zmiňme např. MD5) je z obsahu dokumentu vytvořena posloupnost znaků a číslic dále reprezentujících daný dokument. Shodným dokumentům je přiřazena stejná signatura, což vede k odhalení identických dokumentů

   VÝHODY
   - příznivá lineární časová složitost (co do počtu dokumentů kolekce)
   - v některých případech lze nastavit požadovanou délku signatur dokumentů, čímž je možné ovlivnit paměťovou náročnost aplikace, stejně jako relevanci nalezených výsledků

   NEVÝHODY
   - generované signatury nejsou schopné zachytit malou změnu dokumentu, neboť není zajištěno, že velká míra podobnosti dokumentů se odrazí na velké míře podobnosti vytvořených signatur

   V dalším textu se na tento výpočetní model budeme odkazovat termínem signaturový model.

Pro obrovské kolekce dokumentů je čas na jejich zpracování pomocí vektorového modelu neúnosný. Při našich měřeních na procesoru AMD Opteron 275 dosáhla doba na vyhledání velmi podobných dokumentů pomocí této metody 10 hodin při velikosti kolekce 100 000 dokumentů, což, přihlédneme-li ke kvadratické složitosti algoritmu, dává při velikosti kolekce 10 000 000 dokumentů odhad doby běhu aplikace 12 let. Na druhou stranu z nevýhod uvedených u signaturového modelu vyplývá, že pro použití při hledání téměř identických dokumentů jsou tyto zcela nevhodné, takže vektorový model je jedinou cestou pro získání přesných výsledků.

Výhody a nevýhody řešení, jehož implementace byla cílem této bakalářské práce, tvoří kombinaci výhod a nevýhod obou výše zmíněných přístupů a to:

- příznivá časová složitost (rozhodně lepší než kvadratická)
- relevance získaných výsledků (srovnatelná s podobnostní metodou)

Naše řešení navíc nabízí možnost nastavení parametrů, pomocí nichž lze plně kontrolovat rychlost a přesnost výpočtu spolu s paměťovou náročností aplikací.

1.3 Locality Sensitive Hashing

Jak bylo řečeno dříve, nepoužitelnost signaturového modelu pro naše účely je zapříčiněna neschopností zaznamenat malou změnu dokumentu. Nejprve se pokusme vyjádřit, co znamená schopnost zaznamenat i malou změnu dokumentu. Jestliže hashovací funkce přiřadí dokumentu *A* signaturu S_A, pak pokud změníme dokument *A* jen málo (např. změnou nadpisu článku ap.), měla by signatura upraveného dokumentu být velmi podobná signatuře S_A. Můžeme tento požadavek vyjádřit též ekvivalencí

$$doc_i \approx doc_j \Leftrightarrow sig_i \cong sig_j \qquad (V1)$$

(≈ ... je podobno, podobnost dokumentů se určuje vektorovým modelem/kosinovou podobností)
(≅ ... je podobno, podobnost signatur se v našem případě určuje hammingovou podobností)

Takovou vlastnost splňují hashování označovaná jako Locality Sensitive Hashing, neboli LSH ([1]). Algoritmus, který bude naše řešení při převodu kolekce dokumentů na její signaturový model používat, je následující.

V..... slovník - množina termů získaných ze všech dokumentů
n..... mohutnost slovníku
k..... požadovaná délka tvořených signatur (v bitech)
h..... funkce, která pro dokument vytvoří na základě LSH algoritmu odpovídající signaturu čítající k bitů

Krok 1: Náhodně vygenerujme *k* vektorů dimenze *n*: $RND_1,...,RND_k$

Krok 2: Normalizujme vektory $RND_1,...,RND_k$ na aritmetickou délku jedna

Krok 3: Pro každý dokument *doc* vytvořme odpovídající vektor *v* dimenze *n*

```
for (int i=0; i < n; i++){
        if (V[i] is_in BOW_doc)
                v[i]=tf_idf_V[i];
        else
                v[i]=0;
};//for
```

Krok 4: Pro každý dokument doc_i spočteme $LSH(doc_i)$ následujícím způsobem

```
for (int j=0; j < k; j++){
        h(doc_i)[j] = 1 ⇔ v_i.RND_j >= 0
        h(doc_i)[j] = 0 ⇔ v_i.RND_j < 0
};//for
```

...kde *a.b* znamená skalární součin vektorů *a* a *b*.

V [1] je možné se dále dočíst, že při použití LSH platí pro libovolné dva vektory *u*, *v* následující rovnost

$$P[h(u) = h(v)] = 1 - \frac{\theta(u,v)}{\pi}$$

(kde $\theta(u,v)$ je velikost úhlu svíraného vektory *u,v* v radiánech)
která je jiným vyjádřením faktu, že pro dva podobné dokumenty (reprezentované podobnými

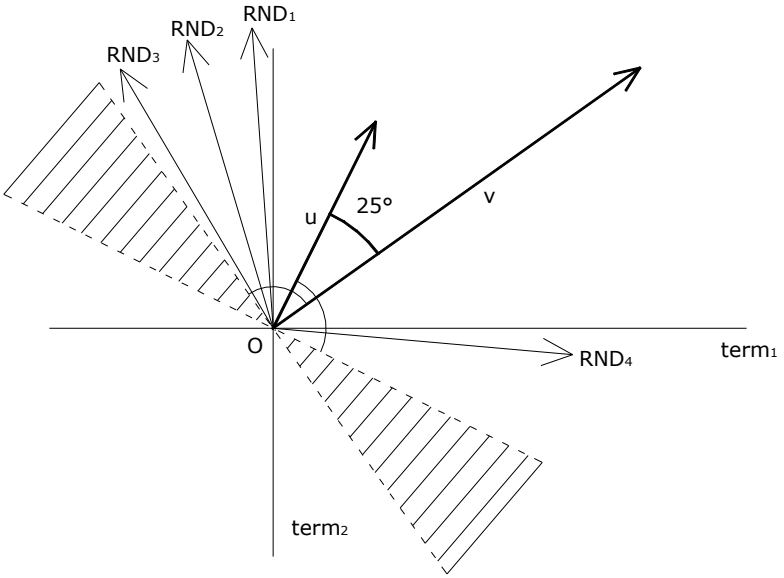
vektory *u*, *v* – podobnost dokumentů měřena pomocí kosinové míry uplatněné na jejich vektory) budou i odpovídající signatury podobné (podobnost signatur měřena pomocí hammingovy podobnosti).

Komentář k funkci LSH algoritmu

Rozeberme nyní funkci LSH algoritmu krok po kroku. V *Krok 1* jsou generovány náhodné vektory $RND_1,...,RND_k$ a to z rozdělení $N^n(0,1)$, čehož cílem je rozmístit náhodné vektory tak, aby se konvexní obal koncových bodů těchto náhodných vektorů (po normalizaci) co nejvíce podobal kouli resp. aby byl svým tvarem co nejblíže pravidelnému mnohostěnu. Následně porovnáváme polohu vektorů dokumentů s polohou vektorů $RND_1,...,RND_k$. Vytvářená signatura odpovídající dokumentu doc_i bude mít na pozici *j* jedničku právě tehdy, když vektor dokumentu doc_i s vektorem RND_j svírá úhel menší než 90°, v opačném případě bude na tomto indexu nula (viz *Krok 4)*. Výpočet kosinu úhlu svíraného vektorem dokumentu doc_i a vektorem RND_j je v *Krok 4* omezen pouze na skalární součin, neboť oba vektory jsou normalizované. V případě, že vektor dokumentu doc_i není před tvorbou signatur normalizován, vstupuje do hry ještě jeho aritmetická délka.

Pozn. Je zřejmé, že náhodné vektory, které mají všechny složky kladné stejně jako ty, které mají všechny složky záporné, nemají pro vektory dokumentů žádnou rozlišovací schopnost. První zmíněné musí s vektory všech dokumentů svírat úhel menší než 90°, zatímco druhé zmíněné musejí s vektory všech dokumentů svírat úhel větší než 90°, neboť vektory dokumentů mohou ve všech svých složkách nabývat pouze nezáporných hodnot. Nutno ovšem podotknout, že pravděpodobnost výskytu takového nevhodného vektoru je rovna 0.5^n**.**

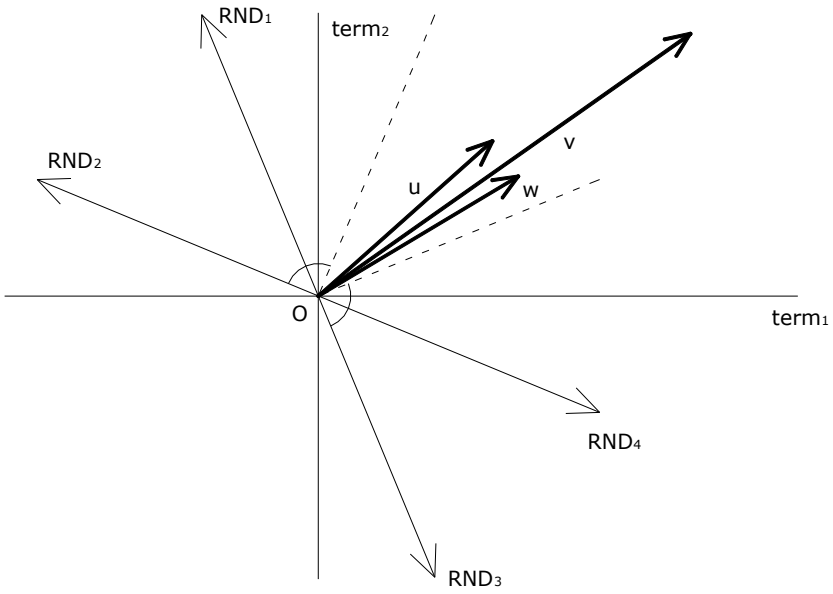
Na obrázcích níže je popsán jeden patologický jev, který se může v souvislosti s generováním vektorů $RND_1,...,RND_k$ a následným vytvářením signatur vyskytnout. Zde je tento jev popsán ilustrativně v dvourozměrném prostoru (každá dimenze reprezentuje počet výskytů jednoho termu v dokumentu, jehož vektor je na obrázku zakreslen - postup tvorby vektoru je uveden v *Krok 3* LSH algoritmu, viz výše). V reálném provozu je dimenze tohoto prostoru totožná s velikostí slovníku kolekce (tedy řádové desítky či stovky tisíc), nicméně tento patologický jev zůstává stejný nezávisle na dimenzi tohoto prostoru.



Obr. 1.2: Náhodné vektory jsou rozmístěny nepravidelně

Za situace, která je znázorněna na Obr. 1.2, je zřejmé, že vytvořené signatury všech dokumentů budou totožné, neboť náhodné vektory nejsou „dostatečně rozprostřeny“ po celém prostoru, v důsledku čehož každý z nich svírá se všemi vektory dokumentů příliš malý úhel.

Čárkovanými čarami jsou vyznačeny kolmice na oba vektory dokumentů *u* a *v*. K tomu, aby nějaký náhodný vektor měl schopnost rozlišit vektory *u* a *v*, by musel ležet ve vyšrafované oblasti.



Obr. 1.3: Vektory dokumentů kolekce nejsou dostatečně rozmanité

Na Obr. 1.3 jsou znázorněny tři vektory *u*, *v*, *w* reprezentující tři různé dokumenty, které jsou si navzájem velice podobné. I přesto, že jsou náhodné vektory RND_1 až RND_4 rozmístěny, dle konkluze vytvořené na základě prvního patologického jevu, rovnoměrně, nemají nagenerované náhodné vektory žádnou schopnost rozlišit dané vektory dokumentů, neboli bity, kterými tyto náhodné vektory přispějí do signatur, budou shodné. V případě, že jsou si vektory *u,v,w* podobné natolik, že je možné je považovat za blízké duplicity, je toto kýžený stav, neboť zaručuje, že velmi podobným dokumentům přiřadíme velmi podobné (v tomto případě dokonce shodné) signatury.

1.4 PLEB Algoritmus

Jak bylo řečeno výše, jednou z vlastností, které od našeho řešení očekáváme, je snížení časové složitosti oproti vektorovému modelu, v opačném případě jeho konstrukce pozbývá významu. Pokud porovnáváme, stejně jako to dělá vektorový model, všechny dvojice dokumentů, řešíme úlohu mnohem složitější, než je ta, kterou řešit máme – namísto hledání téměř identických dokumentů hledáme podobnosti každých dvou dokumentů kolekce. Ale pokud je naším záměrem odhalit pouze téměř identické dokumenty, je velmi pravděpodobné, že drtivá většina výpočtů podobností dvojic dokumentů nám do výsledku nepřispěje, neboť téměř identické dokumenty tvoří zpravidla velmi malou část všech párů. Řešení problému časové náročnosti výpočtu tím, že odstraníme výpočet podobností, které nám pro řešení naší úlohy nepřinesou relevantní informace, je uvedeno v [1] pod názvem PLEB – Point Location in Equal Balls. Obecnou podmínkou funkce algoritmu je existence hashovací funkce splňující *(V1) z* odstavce *1.3*.

Algoritmus pro hledání signatur, které jsou si velmi podobné (pomocí hammingovy podobnosti), je následující. Vstupem algoritmu je seznam signatur všech dokumentů kolekce, výstupem jsou pak ty dvojice, které svou vzájemnou podobností překračují nějakou, uživatelem definovanou, prahovou hodnotu.

for (int i=0, i < pocet_iteraci; i++){
Lexikograficky setřiď seznam signatur *(Krok 1)*
Projdi seznam a najdi téměř identické dokumenty *(Krok 2)*
Zvol libovolnou transpozici a aplikuj ji na všechny signatury *(Krok 3)*
}; //for

Komentář k *Krok 1*: Pro každou signaturu ze setříděného seznamu lze očekávat, že v jejím okolí jsou signatury jí podobné a díky *(V1)* platí, že i odpovídající dokumenty jsou si podobné – pro každou signaturu seznamu je tedy potřeba prohlížet signatury položené v seznamu výše a níže a to v nějakém zvoleném rozsahu, například o 30 signatur výše a níže (tento rozsah budeme dále nazývat *beam*). Prohledáním tohoto rozsahu všech signatur v seznamu algoritmus objeví část požadovaného výsledku.

Komentář k *Krok 2*: Mějme nyní dvě libovolné signatury:

$$sig_A : 01010101011100$$
$$sig_B : 01110101011100$$

Obě signatury se liší v jediném bitu, vycházeje tedy z *(V1)* se budou i dokumenty *A* a *B* lišit málo. V seznamu signatur ale pravděpodobně obě signatury leží díky lexikografickému uspořádání daleko od sebe a tato dvojice tedy v *Krok 2* nebude odhalena a zařazena do výstupu aplikace. K nápravě slouží poslední krok for-cyklu. Náhodně zvolíme transpozici a tuto aplikujeme na všechny signatury. Při vhodné volbě signatury (v našem případě cyklus {3,14}) bude

$$sig'_A = perm[sig_A] : 01010101011100$$
$$sig'_B = perm[sig_B] : 01010101011101$$

Při lexikografickém setřídění seznamu signatur v další iteraci se signatury sig'_A a sig'_B dostanou vedle sebe a tudíž budou objeveny.

1.5 Motivace zavedení parametru *shift_count*

Algoritmus použitý v [1] jsme modifikovali přidáním dalšího parametru a to *shift_count*. Tento označuje počet transpozic aplikovaných na seznam signatur v rámci každé iterace algoritmu PLEB. Termínem *treshold* máme v dalším odstavci na mysli hodnotu procentuální podobnosti takovou, že dvojice dokumentů (signatur) překonávající tuto (prahovou) procentuální podobnost prohlásíme za blízké duplicity.

Díky výpisům počtu již odhalených párů velmi podobných dokumentů po každé iteraci jsme mohli (ve fázi experimentování) velmi jednoduše sledovat „úspěšnost“ jednotlivých iterací, tedy počet párů, které byly objeveny právě v poslední iteraci. Pozorováním běhu se *shit_count=1* (nastavení dle článku [1]) bylo zjištěno, že velice často dochází k jevu, kdy v několika (jednotkách až desítkách) po sobě jdoucích iteracích nejsou objeveny žádné páry, které nebyly objeveny dříve, ačkoliv párů, které na své objevení ještě čekaly, bylo mnoho. Zjistili jsme, že důvodem tohoto chování je skutečnost, že jediná aplikace transpozice na seznam signatur pořadí signatur nijak

výrazně nezmění a okolí každé signatury se změní jen o málo. Důvodem této malé změny je fakt, že leží-li dvě velmi podobné signatury ve vzájemné vzdálenosti menší nebo rovné *beam*, potom jsou dle definice *treshold* procent bitů jejich signatur totožné a vybráním jakýchkoliv dvou indexů z těchto a následným prohozením se na postavení obou signatur nic nezmění (jedině v případě, že by se mezi obě signatury dostaly nějaké, které mezi nimi před aplikací transpozice nebyly). Je zřejmé, že pravděpodobnost takového „špatného“ náhodného výběru indexů k prohození je *(treshold/100)^2*. Rozšíření naší modifikace algoritmu PLEB o parametr *shift_count* bylo tedy nasnadě a později se ukázalo, že parametr *shift_count* je pro zvyšování efektivity algoritmu PLEB mnohem významnější než počet iterací a *beam*. Z výše uvedené pravděpodobnosti plyne, že pro zachování efektivity algoritmu PLEB je vhodné zvyšovat *shift_count* s rostoucí hodnotou parametru *treshold*.

Kapitola 2

Experimentální část

2.1 Úvod

Implementace aplikací, pracujících na základě výše uvedených algoritmů, nebyla jedinou náplní této práce. Její stěžejní částí byla experimentální analýza chování signaturového modelu, který využívá algoritmu LSH (tento model budeme dále v textu nazývat LSH-signaturový). Cílem této experimentální analýzy bylo poskytnout srovnání mezi LSH-signaturovým a vektorovým modelem tak, aby bylo možné určit, zda LSH-signaturový model může být při hledání blízkých duplicit plnohodnotnou alternativou vektorového modelu, co se přesnosti získaných výsledků týče.

Dalším cílem experimentální analýzy bylo formulovat taková pravidla popisující konfiguraci algoritmu PLEB (nastavení počtu iterací, *shift_count* a *beam*), která zajistí, nebo alespoň přispějí k rychlému nalezení velké části výsledku.

2.2 Experimentální data

Experimenty byly provedeny nejdříve na 100 000 dokumentech posbíraných z českých periodik. Na dokumenty této kolekce jsme nekladli žádné podmínky, byly náhodně vybrány z větší české kolekce. Druhá testovaná kolekce obsahovala množinu 10 000 dokumentů první kolekce s nejmenšími počty různých slov (mohutnost BOWů těchto dokumentů nepřesáhla 150 termů). Třetí testovaná kolekce obsahovala 10 000 dokumentů první kolekce, které měly nevyšší počty různých slov (mohutnost BOWů těchto dokumentů byla vyšší než 500, obsahovala i dokumenty mající BOWy obsahující desetitisíce termů). Na jednotlivé kolekce se budeme dále odkazovat jako na *Coll_Mixed*, *Coll_Small* a *Coll_Large*.

2.3 Analýza efektivity konfigurace algoritmu PLEB

Chování algoritmu PLEB ovlivňují tři faktory – *iter_count*, *beam* a *shift_count*. Nastavením různých hodnot těchto parametrů lze ovlivnit efektivitu procesu hledání velmi podobných signatur a tím zkrátit čas potřebný k nalezení určité procentuální části všech velmi podobných signatur a/nebo zvětšit množinu nalezených párů při zachování časové náročnosti hledání.

Obecně je určit optimální nastavení parametrů *iter_count*, *beam* a *shift_count* zcela nemožné, neboť by to znamenalo určit extrémy funkce tří proměnných (*iter_count, beam, shift_count*), což by bylo nemožné i při řádově vyšším počtu měření, než který jsme provedli my. Tato podkapitola experimentální části této práce si tedy neklade za cíl formulovat přesná pravidla platná za každých okolností a na jakékoliv kolekci. Spíše se snažíme popsat vzájemné působení jednotlivých parametrů a na jejich základě vytvořit doporučení, jejichž dodržováním je možné dosáhnout dobrých výsledků.

2.3.1 Výsledná doporučení

Analýzu nastavení algoritmu PLEB jsme prováděli na kolekci *Coll_Mixed*. Na základě přibližně stovky experimentů jsme sestavili následující doporučení týkající se nastavení jednotlivých parametrů algoritmu PLEB.
Pozn. Údaj v sloupci „nalezených_párů (%)“ určuje počet párů nalezených daným během aplikace *hamming* v poměru ke všem podobným dokumentům nalezitelným pomocí signaturového modelu (tedy aplikací *hamming_full*) na daných datech.

1/ parametr *beam* nastavit na vysokou hodnotu (40 – 50). Horní mez těchto hodnot je těžké odhadnout. Zvyšováním hodnoty parametru *beam* a poměrným snižování hodnoty *iter_count* bychom sice dosáhli totožného počtu aplikací funkce na zjištění hammingovy vzdálenosti, s velmi nízkými hodnotami *iter_count* se ale vyhledávání stane neefektivní (jak uvidíme níže). Pro posouzení vlivu velkých hodnot *beamu* na efektivitu hledání velmi podobných signatur jsme provedli následující měření.

beam	nalezených_párů (%)
60	69,8
70	73,3
80	76,2
90	78,6

Tab. 2.1: Vliv veličiny beam (*iter_count =100*, *shift_count=50)*

Je vidět stálý nárůst množství nalezených párů, nicméně nelze celý proces hledání podobných signatur opřít pouze o parametr *beam*, jak bude vysvětleno v bodu týkajícím se parametru *iter_count*. Také je viditelné, že vztah efektivity a velikosti *beamu* není lineární a snaha zvyšovat efektivitu pouze *beamem* by vedla k vysokým časovým nárokům.

2/ parametr *shift_count* nastavit na vysokou hodnotu (v našich měřeních 40 – 50). Jak již bylo řečeno výše, je nutné hodnotu tohoto parametru přizpůsobovat hodnotě *treshold*, navíc je ale potřeba ji přizpůsobovat i hodnotě parametru *beam*. To z toho důvodu, že proto, abychom využili potenciál rozšířeného obzoru, který vůči každé signatuře prohledáváme, je nutné zajistit, že se signatury v každé iteraci v rámci tohoto obzoru dostatečně obmění a toho lze docílit pouze výší hodnoty parametru *shift_count*. Odhadnout horní mez tohoto parametru je ještě těžší než v případě parametru *beam*. Pro posouzení vlivu velmi vysokých hodnot parametru *shift_count* jsme provedli následující experimenty.

shift_count	nalezených_párů (%)
10	34,59
20	50,34
30	61,65
40	69,12
50	74,32
60	77,68
70	80,08
80	82,62
90	84,34

shift_count	nalezených_párů (%)
10	50
20	67,71
30	79,05
40	85,39
50	89,42
60	91,6
70	93,03
80	94,53
90	95,45

Tab. 2.2, 2,3: Vliv veličiny shift_count (beam=10, iter_count=500 a *beam=20, iter_count*=500)

Z dat v těchto tabulkách uvedených je možné vypozorovat, že ani vztah velikost hodnoty *shift_count* a počet nalezených párů není ve vztahu přímé úměrnosti, zejména u nejvyšších hodnot již zvýšení hodnoty o 10 přinese pouze malý nárůst počtu nalezených párů. Proto bychom doporučili hodnoty v rozmezí 50 – 60.

3/ parametr *iter_count* volit téměř výlučně podle nastavení parametru *beam*. Na rozdíl od předchozích dvou parametrů, u tohoto je třeba hledat spodní mez. Opodstatnění tohoto tvrzení je následující: mějme signaturu *A*, a počet párů *A:B*, které vyhovují hodnotě *treshold*, označme *k*. Potom potřebujeme za ideálních podmínek alespoň *k/2*beam (*)* iterací k tomu, abychom všechny podobné signatury *B* našli. V praxi se ale s ideálními podmínkami nesetkáme (ne v každé iteraci se signatury promíchají, jak bychom si přáli – navíc do hry vstupuje pravděpodobnost, takže kvalitu promíchání nedokážeme nikdy zaručit), proto je tento odhad příliš nízký. Pro ověření toho, jak se algoritmus PLEB chová při volání s nízkým počtem iterací byly provedeny následující experimenty.

iter_count	**nalezených_párů (%)**
50	37,5
100	60,3
150	72,9
200	78,7
250	84,4

Tab. 2.4: Vliv veličiny iter_count (*beam=50*, *shift_count=40)*

I při výše doporučovaných hodnotách parametrů *beam* a *shift_count* je počet iterací velmi významný faktor ovlivňující schopnost algoritmu nalézt co nejvíce podobných párů. Z (*) také vyplývá, že minimální hodnota parametru *iter_count* závisí na volbě parametru *treshold*. Pro *treshold=80* bychom za spodní mez, pod níž se nevyplatí hodnotu parametru *iter_count* snižovat, považovali hodnotu 250, přičemž hodnotu 300 - 400 bychom mohli označit jako zlatý střed. Pro hodnotu *treshold=95* bychom za minimální považovali hodnotu 50, hodnoty 100 – 150 by pak tvořily zlatý střed.

2.4 Porovnání kosinového a LSH-signaturového modelu

2.4.1 Metodika měření kvality aproximace

K porovnání obou metod přístupu k řešené úloze posloužily aplikace *similarity* a *hamming_full*. Záměrně nebyla vybrána aplikace *hamming*, neboť jsme chtěli porovnat pouze oba přístupy a nezkreslovat tak výsledky použitím algoritmu PLEB, díky kterému bychom neměli jistotu, že jsme v prostoru signatur našli všechny nalezitelné páry vyhovující našim požadavkům na podobnost. Parametry, jejichž vliv jsme v experimentech na jednotlivých kolekcích posuzovali, byly délka signatur, redukce slovníku, průměrná mohutnost BOWů dokumentů, použití *tf/tf_idf*.

Úspěšnost aproximace kosinového modelu LSH-signaturovým se určovala pomocí dvou evaluačních měr označených L_1 a L_2, jejichž vzorce jsou uvedeny níže. Zároveň, při posuzování úspěšnosti v oblasti hledání blízkých duplicit, byly zavedeny ještě klasické míry *Precision*, *Recall* a *F-Measure*, jejichž vzorce rovněž následují.

$$L_1 = \frac{\sum_{\substack{doc_i \in sample\_list \\ doc_j \in kolekce}} \left| kosinová\_podobnost(doc_i, doc_j) - hamm\_podobnost(doc_i, doc_j) \right|}{|sample\_list| * |kolekce|}$$

$$L_2 = \frac{\sum_{\substack{doc_i \in sample\_list \\ doc_j \in kolekce}} \left(kosinová\_podobnost(doc_i, doc_j) - hamm\_podobnost(doc_i, doc_j) \right)^2}{|sample\_list| * |kolekce|}$$

$$R = \sum_{doc_i \in sample\_list} \frac{\frac{|hamm(doc_i) \cap cos(doc_i)|}{|cos(doc_i)|}}{|sample\_list|}$$

$$P = \sum_{doc_i \in sample\_list} \frac{\frac{|hamm(doc_i) \cap cos(doc_i)|}{|hamm(doc_i)|}}{|sample\_list|}$$

$$hamm(doc_i) = \{doc_j : hamm\_podobnost(doc_i, doc_j) \geq treshold\}$$

$$cos(doc_i) = \{doc_j : kosinová\_podobnost(doc_i, doc_j) \geq treshold\}$$

$$F_{(1)} = \frac{2 * P * R}{P + R}$$

Evaluační míry L_1 a L_2 dvěma různými způsoby měří odchylku hammingových podobností od kosinových. Jak bude zřejmé dále v textu, nejsou tyto míry vždy dostačující, proto jsme zavedli další míru S, jejíž výpočet probíhá následujícím způsobem :

1/ pro testované páry dokumentů spočtěme hammingovu i kosinovou podobnost

2/ $\forall i \in [0,1..99] : X_i = \{(doc_1, doc_2) : kosinová\_podobnost(doc_1, doc_2) \in (i, i+1]\}$

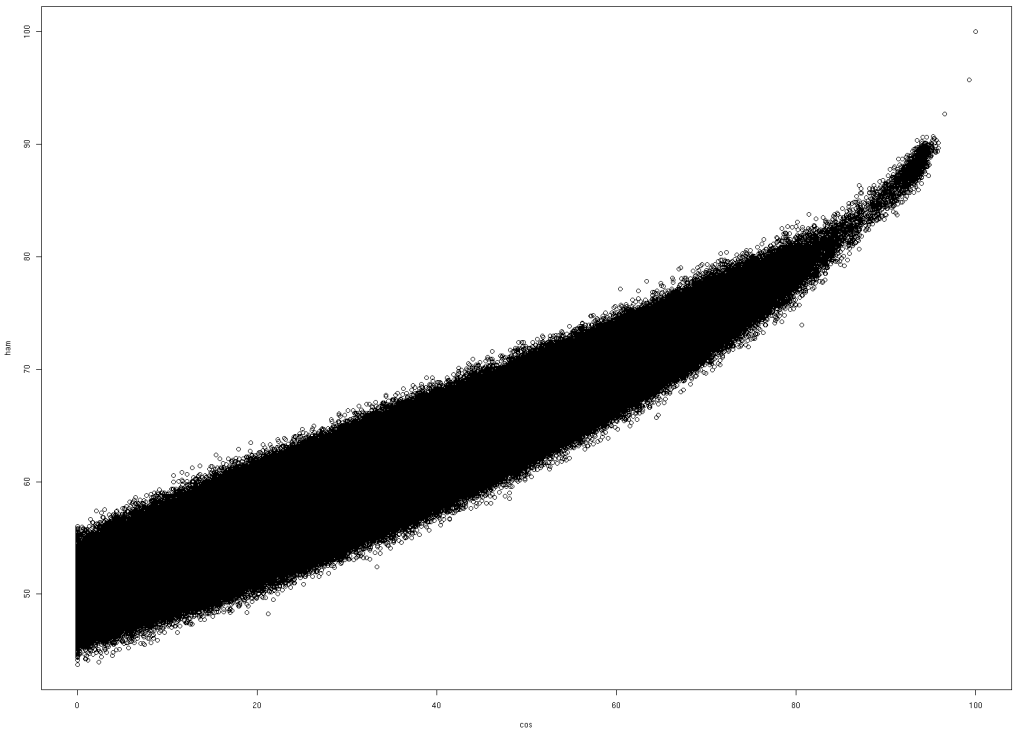
3/ $\forall i, \forall (doc_j, doc_k) \in X_i : V_i = Var(hamm\_podobnost(doc_j, doc_k))$

4/ $S = Mean(V_i)$

Míra *S* tedy lidově řečeno vyjadřuje „průměrnou tloušťku“ závislosti hammingových podobností na kosinových vykreslené na Obr. 2.5 (níže).

2.4.2 Problém s rozsahy

Na Obr. 2.5 je vykreslena kosinová a hammingova podobnost všech párů dokumentů zjištěná v rámci jednoho z provedených experimentů (*Coll_Mixed, treshold=70*). Negativním jevem je rozmezí hodnot na jednotlivých osách. Zatímco hodnoty na vodorovné ose nabývají hodnot [0, 100] (přibližně), na ose svislé je nabýváno hodnot pouze v intervalu [45, 100] (přibližně). Kosinové podobnosti 0 tedy odpovídá hammingova podobnost 45, což jinými slovy znamená, že i z vektorů dokumentů, které jsou si v kosinové míře zcela nepodobné, jsou vygenerovány signatury, které se shodují v 45% bitů.

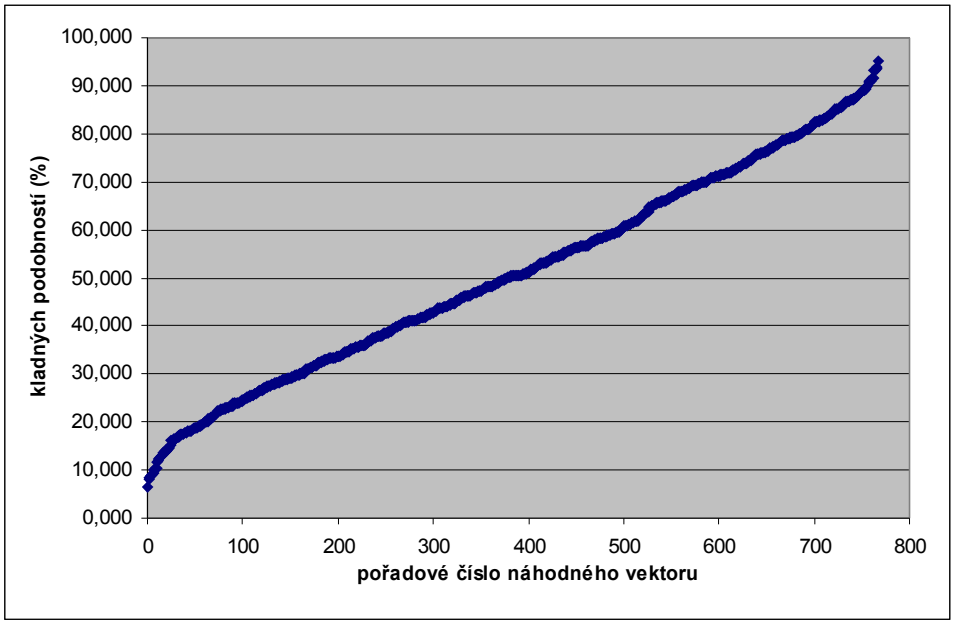


Obr. 2.5: Rozsahy hammingových a kosinových podobností nejsou totožné

Popsaný jev se nám po sérii neúspěšných pokusů o vysvětlení podařilo vysvětlit pomocí důkazu „nevhodně“ generovaných náhodných vektorů v aplikaci *create_sigs*. Ze všeho nejdříve jsme ověřili kvalitu generátoru náhodných čísel, který daná aplikace používá a díky aplikaci *analyze_rnd_vecs_quality* bylo zjištěno, že náhodné vektory jsou kvalitně generovány z prostoru $N^{velikost\_slovniku}(0,1)$, takže zavinění rozdílných rozsahů kosinových a hammingových podobností nekvalitním generátorem náhodných čísel jsme mohli vyloučit.

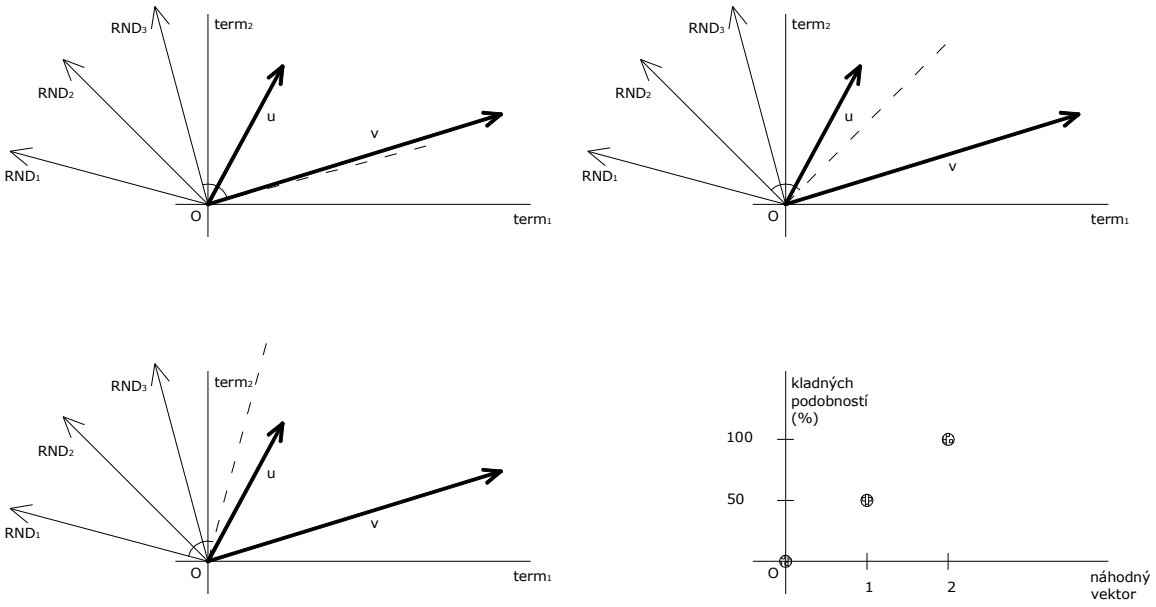
Při následujícím experimentu jsme se snažili prozkoumat, zda za různé rozsahy kosinových a hammingových podobností nemůže nerovnoměrná distribuce vektorů dokumentů ve vektorovém prostoru. Pomocí aplikace *implant_rnd_vecs* jsme vygenerovali stovky náhodných vektorů (totožným postupem jako v *create_sigs)*. Pro každý z těchto jsme pomocí aplikací *similarity* a *extract_similarities* počítali, jaké procento vektorů dokumentů dané kolekce svírá s daným náhodným vektorem úhel menší než 90° (neboli kosinus tohoto úhlu je kladný). Tímto způsobem jsme mohli zjistit, na jak velké části rozdělí každý náhodný vektor dokumenty kolekce tím, že do signatury na příslušnou pozici přispěje nulou či jedničkou.

Získané informace pro všechny náhodně generované vektory jsme shrnuli do grafu níže (počty vektorů dokumentů, se kterými náhodně generované vektory svíraly úhel menší než 90°, v něm jsou pro přehlednost seřazeny vzestupně). Počet nagenerovaných vektorů byl 768, což je číslo dostatečně vysoké pro to, abychom výsledky našeho měření mohli považovat za relevantní.



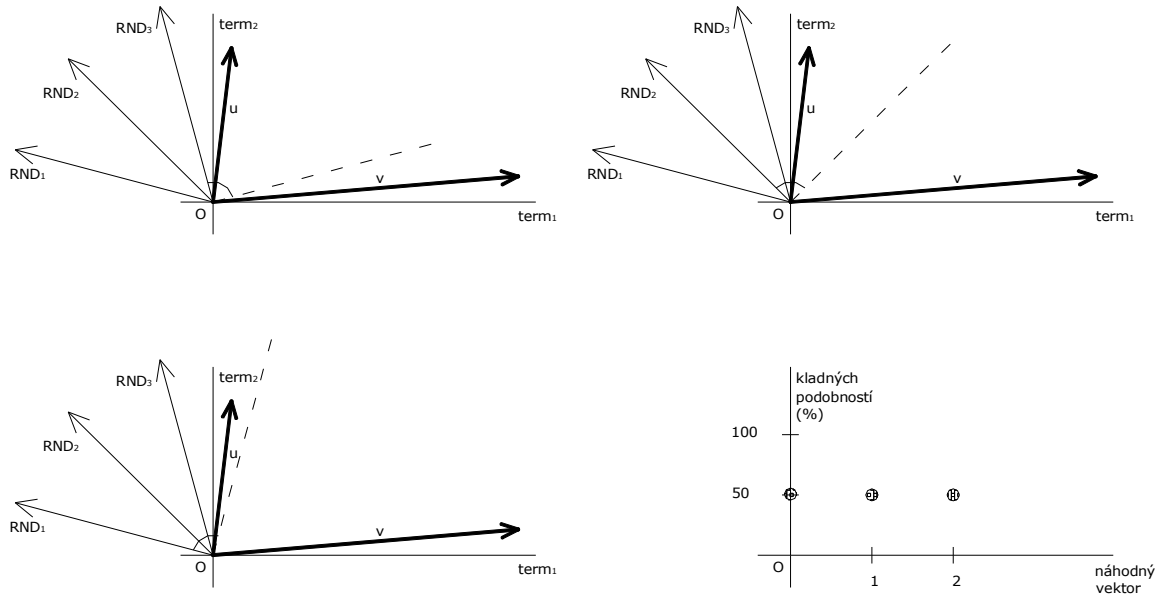
Obr. 2.6: Vhodnost náhodně generovaných signatur

Výsledný graf podává informaci o tom, jak pravidelně jsou vektory dokumentů rozmístěny ve vektorovém prostoru. Díky tomuto grafu se nám podařilo pochopit příčinu problému různých rozsahů hammingových a kosinových podobností. Vysvětlení podávají obrázky níže (Obr. 2.7, 2.7)



Obr. 2.7: Rovnoměrně generované náhodně vektory a vhodná kolekce

Na Obr. 2.7 je vidět, že jestliže jsou vektory dokumentů rozmístěny ve vektorovém prostoru rovnoměrně a jestliže máme různorodě vygenerované náhodné vektory, je graf (stejný jako na Obr. 2.6) podobný ose I. kvadrantu, neboli neplatí, jak bychom si přáli, že každý náhodný vektor množinu vektorů dokumentů rozdělí na dvě stejně mohutné podmnožiny.



Obr. 2.8: Rovnoměrně generované náhodně vektory a nevhodná kolekce

Naproti tomu, jestliže pracujeme s kolekcí, jejíž vektory dokumentů jsou nerovnoměrně rozmístěny po svém vektorovém prostoru, ani přes rozmanité nagenerování náhodných vektorů se vektory kolekce nepodaří rozlišit a důsledkem je kromě předpokládané špatné aproximace vektorového modelu LSH-signaturovým i graf, který se od osy I. kvadrantu liší a má spíše tvar úsečky rovnoběžné s osou x. Ačkoliv se může zdát, že toto je kýžený stav (každý náhodný vektor množinu vektorů dokumentů půlí), z výše řečeného vyplývá, že takového grafu je možné dosáhnout pouze na kolekci, v níž jsou vektory dokumentů nerovnoměrně rozmístěny.

Z toho můžeme vyvodit, že čím více se výše zmíněný graf přibližuje horizontále, tím horší je distribuce vektorů dokumentů v I. kvadrantu vektorového prostoru a naopak čím více se tento graf podobá ose I. kvadrantu, tím rovnoměrněji jsou vektory dokumentů rozmístěny.

Konkluze vyplývající z výše popsaného je taková, že neschopnost zachytit hammingovy podobnosti menší než nějaká prahová mez (v našem případě cca 45%) patří k vlastnostem LSH-signaturového přístupu a nelze jej nijak odstranit. Jak později uvidíme, při hledání blízkých duplicit nás tato nedokonalost LSH-signaturového přístupu nebude nijak omezovat.

Přímým důsledkem výše zmíněného je fakt, že se nám při měření v rámci celého spektra kosinových podobností nikdy nepodaří dosáhnout výsledků $L_1=0$, $L_{2=}0$. V oblasti hledání blízkých duplicit už toto nemusí platit a hodnoty L_1, L_2 nižší než jedna jsou určitě pro vhodné kolekce dosažitelné.

2.4.3 Redukce slovníku

Bylo by skvělé, kdybychom při hledání velmi blízkých duplicit mohli nějakým způsobem redukovat slovník, čímž bychom snížili délku vektorů dokumentů a následně dobu výpočtu aplikací *create_sigs* a *similarity*. V rámci relevance námi naměřených dat jsme si museli položit otázku, zda se dobré výsledky naší aproximace dosažené při použití redukovaného slovníku dají vztáhnout i na měření, při nichž redukce slovníku nebyla použita. Modelovou situaci uvádí tabulka níže.

	vektorový model	***LSH-signaturový model***
bez redukce slovníku	sim (a,b)=85 %	ham(a,b)=82%
s redukcí slovníku	sim (a,b)=85.5 %	ham(a,b)=82.5%

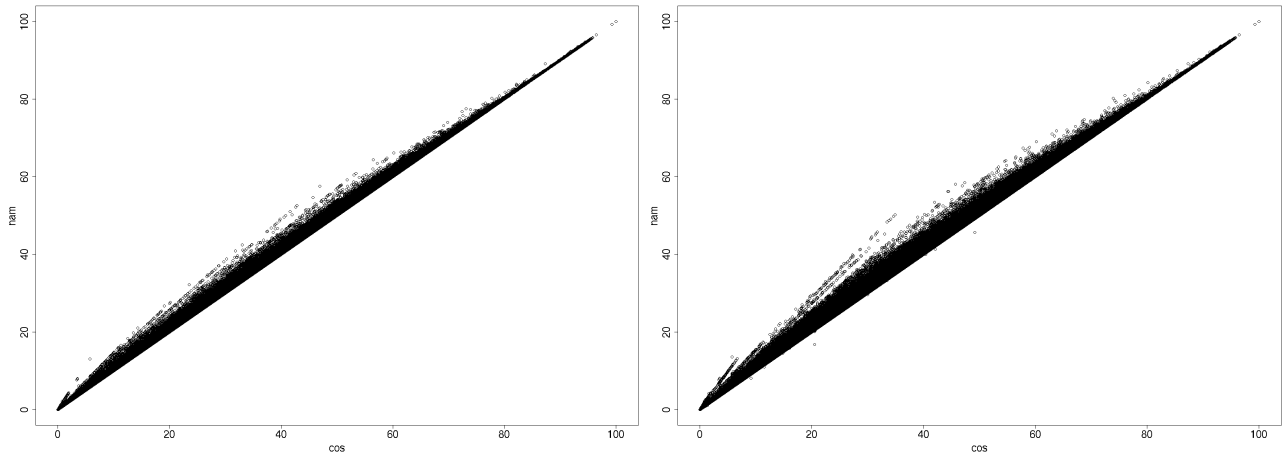
Tab. 2.9: Modelové podobnosti párů dokumentů

Mějme dva různé experimenty na téže kolekci lišící se pouze použitím plného, resp. redukovaného slovníku. Zvolme nějaký pár a naměřme v obou experimentech podobnost tohoto párů jak ve vektorovém, tak v LSH-signaturovém modelu. Pokud výsledky dopadnou tak, jak shrnuje tabulka výše, mohlo by se zdát, že máme vyhráno (a že výsledky naměřené s pomocí redukovaného slovníku můžeme vztáhnout na měření, při nichž redukce slovníku použita nebyla), ovšem k takovému posouzení nám chybí ještě jeden údaj a tím je míra zkreslení vektorového modelu získaná redukcí slovníku – podobnost výsledků získaných na základě neredukovaného a redukovaného slovníku je pouze náhodná a nereflektuje míru zkreslení vektorového prostoru.

Cílem naší práce je porovnat kosinové podobnosti získané na plném slovníku s výsledky hammingovy podobnosti získané na redukovaném slovníku, což z dat uvedených v tabulce nelze. V rámci zjištění míry zkreslení vektorového prostoru získané redukcí slovníku jsme vytvořili vektory sta tisíc dokumentů (*Coll_Mixed*), v jednom případě byly vektory vytvořeny pomocí plného slovníku, podruhé pomocí plného slovníku ochuzeného o slova s *DC=1* (slovník střední) a potřetí pomocí plného slovníku ochuzeného o slova s *DC < 5* (slovník malý). Následně jsme proti *sample_listu* čítajícímu 100 dokumentů spočetli kosinové podobnosti 10 000 0000 párů na všech třech sadách vektorů a výsledky jsme zpracovali pomocí aplikace *compare_results*.

	L1	***L2***	***maximální rozdíl podobností***	***S***
Plný ku Střednímu	0,173	0,082	10,588	0,121
Plný ku Malému	0,458	0,398	15,308	0,213

Tab. 2.10: Zkreslení vektorového prostoru vlivem redukce slovníku



Obr. 2.11,2.12: Zkreslení vektorového prostoru vlivem redukce slovníku (vlevo Plný ku Střednímu, vpravo Plný ku Malému)

Zejména z obrázků je vidět, že vliv je zřetelný, nicméně v oblasti velmi podobných dokumentů téměř k žádnému zkreslení nedochází,z čehož můžeme usoudit, že hodnoty funkcí L_1, L_2 naměřené na signaturách vytvořených s použitím redukovaného slovníku můžeme vztáhnout na původní dokumenty kolekce (obsahující slova, která byla při redukci slovníku odstraněna).

2.5 Experimenty na jednotlivých kolekcích

Při hledání velmi blízkých duplicit jsme volili *treshold_sim* takovým způsobem, aby množina vyhovujících párů byla dostatečně mohutná a naměřené výsledky tak byly relevantní. K danému *treshold_sim* jsme získali hodnotu *treshold_ham* jako minimum hammingových podobností odpovídajících kosinovým podobnostem vyšším než *treshold_sim*. U tabulek prezentujících kvalitu aproximace v oblasti velmi podobných dokumentů je *Precision*, díky volbě této *treshold_ham*, vždy rovna jedné, proto ji do tabulek nezahrnujeme.

2.5.1 Coll_Mixed

Experimenty probíhaly s použitím tří slovníků – plného, méně redukovaného (bez slov s *DC=1*, tzv. střední slovník) a více redukovaného (bez slov s *DC* < *5*, tzv. malý slovník).

délka signatur	***velikost slovníku***	***počet různých signatur***	***L1***	***L2***	***S***
768	1189721	97361	30,134	990,443	2,390
768	586284	97357	29,794	971,132	2,381
768	306909	97357	29,619	961,869	2,501

Tab. 2.13: Porovnání vlivu různých slovníků za použití hodnot tf

délka signatur	***velikost slovníku***	***počet různých signatur***	***L1***	***L2***	***S***
768	1189721	97364	49,357	2442,19	2,263
768	586284	97359	49,352	2439,94	2,391
768	306909	97357	49,312	2436,03	2,469

Tab. 2.14: Porovnání vlivu různých slovníků za použití hodnot tf_idf

délka signatur	***velikost slovníku***	***R***	***L1***	***L2***	***S***
768	1189721	0,167	2,750	9,556	0,888
768	586284	0,143	3,504	17,043	0,935
768	306909	0,152	3,156	12,563	2,501

Tab. 2.15: Porovnání různých slovníků za použití hodnot tf při hledání velmi blízkých duplicit treshold_sim=80, treshold_ham=76

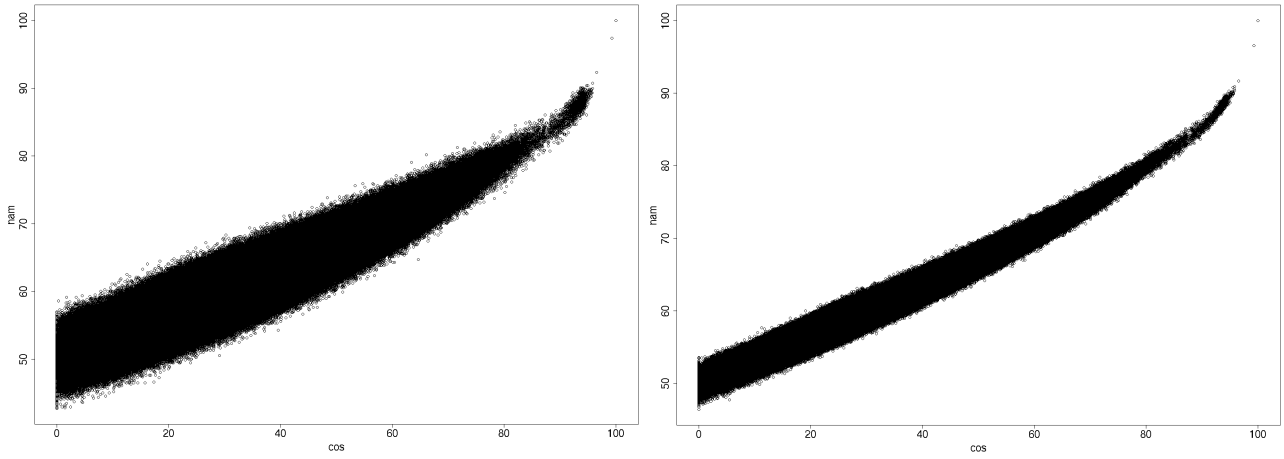
délka signatur	***velikost slovníku***	***R***	***L1***	***L2***	***S***
768	1189721	0,766	2,606	8,455	0,765
768	586284	0,803	3,348	12,943	1,396
768	306909	0,811	3,282	12,621	0,710

Tab. 2.16: Porovnání různých slovníků za použití hodnot tf_idf při hledání velmi blízkých duplicit treshold_sim=80, treshold_ham=75

Velikost slovníku nemá na kvalitu aproximace vliv, žádná z veličin L_1, L_2, S, R nevykazují žádnou závislost na zmenšujícím se slovníku. Na druhou stranu silnější redukce, než odstranění slov s $DC=1$, resp. $DC < 5$ bychom nedoporučovali, neboť by tím zřejmě vzrostlo zkreslení vektorů dokumentů a výsledky takto naměřené bychom nemohli vztáhnout na původní dokumenty kolekce.

délka signatur	***počet různých signatur***	***L1***	***L2***	***S***
768	97357	29,794	971,132	2,381
1024	97359	29,430	950,830	1,858
1536	97359	29,660	962,762	1,175
2048	97359	29,732	966,247	0,895
2560	97359	29,681	962,859	0,721
3072	97359	29,735	966,412	0,649
3584	97359	29,533	953,991	0,511
4096	97359	29,897	974,079	0,477

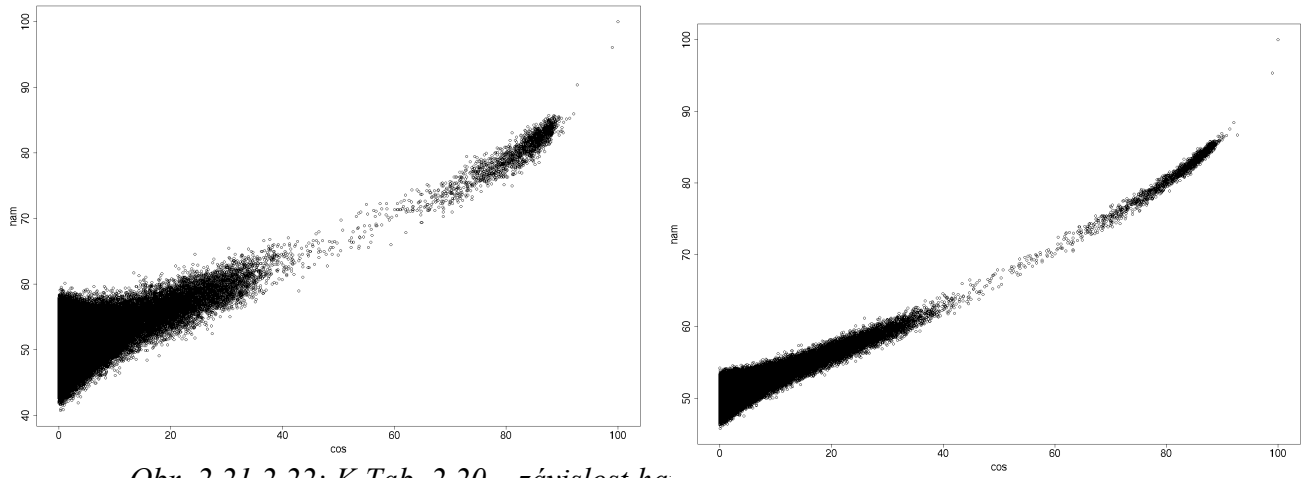
Tab. 2.17: Porovnání různých délek signatur za použití hodnot tf a středního slovníku



Obr. 2.18,2.19: K Tab. 2.17 – závislost hammingových podobností na kosinových délka signatur 768 (vlevo), 4096 (vpravo)

délka signatur	***počet různých signatur***	***L1***	***L2***	***S***
768	97357	49,352	2439,942	2,391
1024	97359	49,356	2439,508	1,914
1536	97359	49,364	2439,466	1,251
2048	97359	49,343	2437,054	0,848
2560	97359	49,363	2438,761	0,747
3072	97359	49,366	2438,843	0,585
3584	97359	49,358	2437,980	0,505
4096	97359	49,350	2437,122	0,481

Tab. 2.20: Porovnání různých délek signatur za použití hodnot tf_idf a středního slovníku



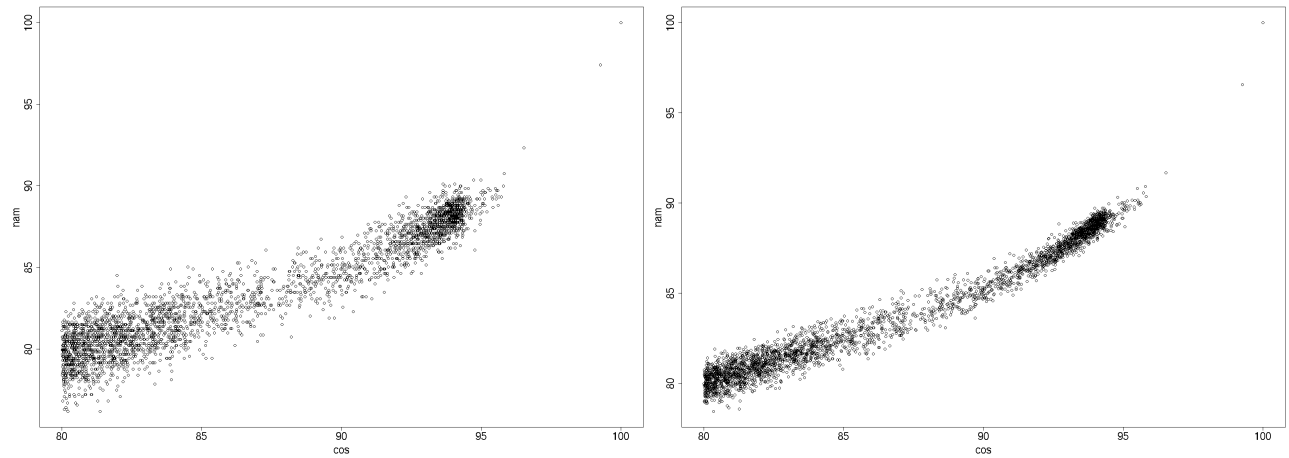
Obr. 2.21,2.22: K Tab. 2.20 – závislost hammingových podobností na kosinových délka signatur 768 (vlevo), délka signatur 4096 (vpravo)

S rostoucí délkou signatur se hodnoty L_1, L_2 nijakým způsobem nezlepšují. Důvod je viditelný na obrázcích, sklon objektu (resp. jeho vzdálenost od osy kvadrantu) je stále totožná. Naopak se s rostoucí délkou signatur výrazně mění veličina *S*. Dá se předpokládat, že zvyšováním délky signatur bychom vykreslenou závislost stále zeštíhlovali, až by jednou dosáhla vzhledu grafu funkce, který pro danou kolekci představuje funkci závislosti hammingovy podobnosti na kosinové.

Vysoké hodnoty funkcí L_1, L_2 naměřené při aproximaci *tf_idf* vektorového modelu jsou způsobeny body „nahrnutými“ do levé části grafu, kde je rozdíl kosinové a hammingovy podobnosti největší.

délka signatur	***R***	***L1***	***L2***	***S***
768	0,143	3,504	17,043	0,935
1024	0,189	3,431	15,070	0,766
1536	0,200	3,151	12,826	0,529
2048	0,169	2,716	10,581	0,374
2560	0,202	3,140	12,887	0,331
3072	0,182	3,140	10,671	0,328
3584	0,196	3,673	15,202	0,226
4096	0,158	3,004	13,227	0,272

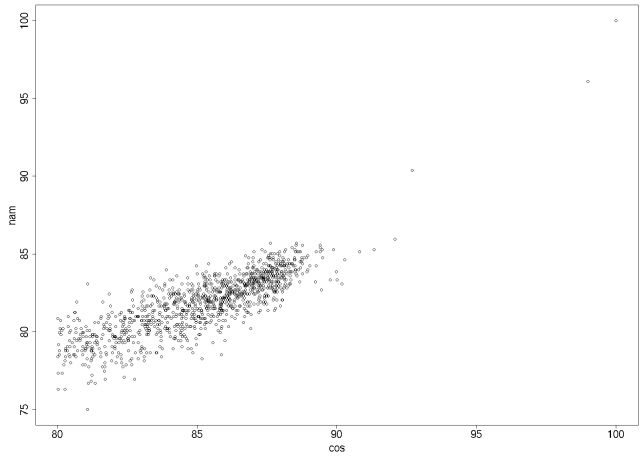
Tab. 2.23: Porovnání různých délek signatur, tf vektory, střední slovník, treshold_sim=80, treshold_ham=76

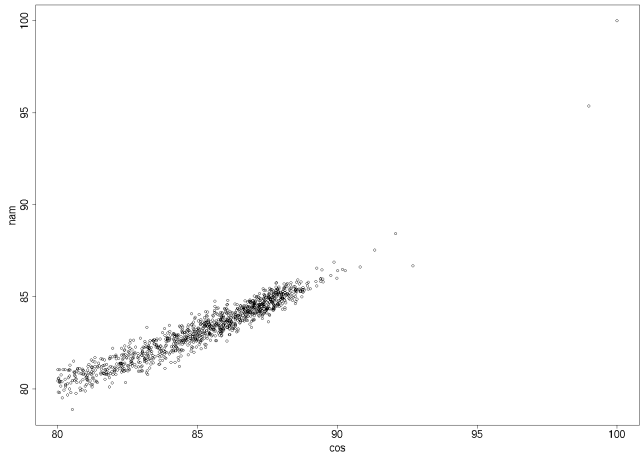


Obr. 2.24,2.25: K Tab. 2.23 – závislost hammingových podobností na kosinových délka signatur 768 (vlevo), délka signatur 4096 (vpravo)

délka signatur	*R*	*L1*	*L2*	*S*
768	0,803	3,348	12,943	1,413
1024	0,803	2,826	9,364	0,836
1536	0,788	2,846	9,560	0,400
2048	0,807	3,471	13,400	0,308
2560	0,792	2,781	9,017	0,288
3072	0,776	2,416	6,905	0,219
3584	0,780	2,816	9,186	0,191
4096	0,768	2,816	5,031	0,234

Tab. 2.26: Porovnání různých délek signatur tf_idf vektory, střední slovník, treshold_sim=80, treshold_ham=76



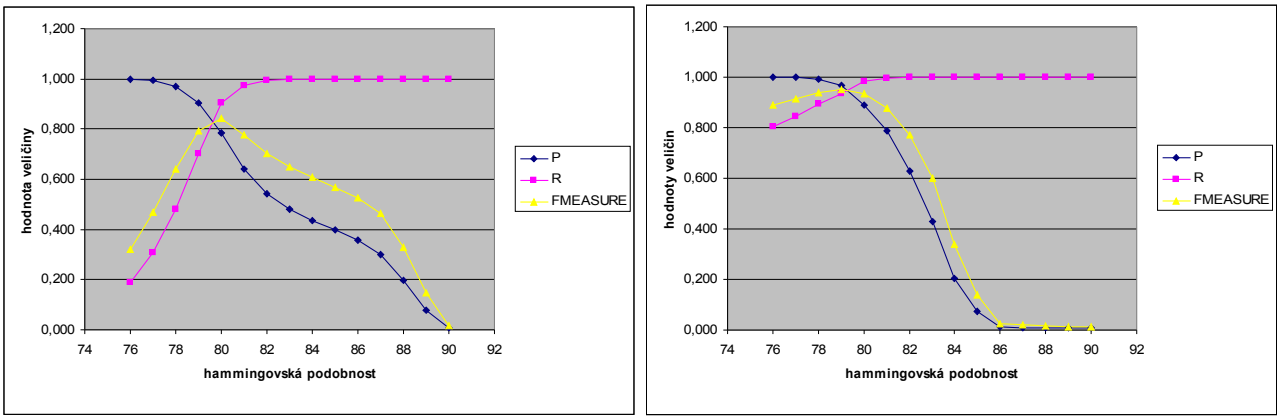


Obr. 2.27,2.28: K Tab. 2.26 – závislost hammingových podobností na kosinových délka signatur 768 (vlevo), délka signatur 4096 (vpravo)

Při hledání blízkých duplicit nemá délka signatur vliv ani na funkce L_1, L_2, ani na výši míry *Recall*, naopak značný (takřka lineární) vliv má na veličinu *S*. Nízké hodnoty míry *Recall* jsou zapříčiněny postupem, kterým pro *treshold_sim* hledáme *treshold_ham* (vysoké *Precision* je vykoupeno nižším *Recall)*. Diametrálně odlišné hodnoty míry *Recall* při hledání blízkých duplicit v *tf*, resp. *tf_idf* vektorovém prostoru jsou způsobeny tím, že zatímco podobnosti v *tf* vektorovém modelu jsou rozprostřeny po celém intervalu [0,100], v *tf_idf* vektorovém modelu jsou podobnosti rozděleny do dvou skupin (málo podobné, hodně podobné), jak je možno vidět na Obr. 2.18 – 2.22. Toto rozdělení není způsobeno žádnou chybou v našem postupu a výpočtu, jedná se o vlastnost *tf_idf* vektorového modelu.

Příjemným faktem je nízká hodnota funkcí L_1, L_2 v oblasti hledání blízkých duplicit. V této úloze je dokonce aproximace *tf_idf* vektorového prostoru přesnější než aproximace *tf* vektorového prostoru.

Pro určení nejvhodnějšího *treshold_ham* k danému *treshold_sim* jsme provedli experimenty zkoumající vztah měr *Precision* a *Recall* při *treshold_sim=80* a *treshold_ham* měnícím se po jednotkách v intervalu [76, 90].



Grafy 2.29,2.30: Průběhy veličin Precision a Recall v závislosti na parametrech aproximace slovník=střední, délka signatur=1024, tf vektory (vlevo), tf_idf vektory (vpravo)

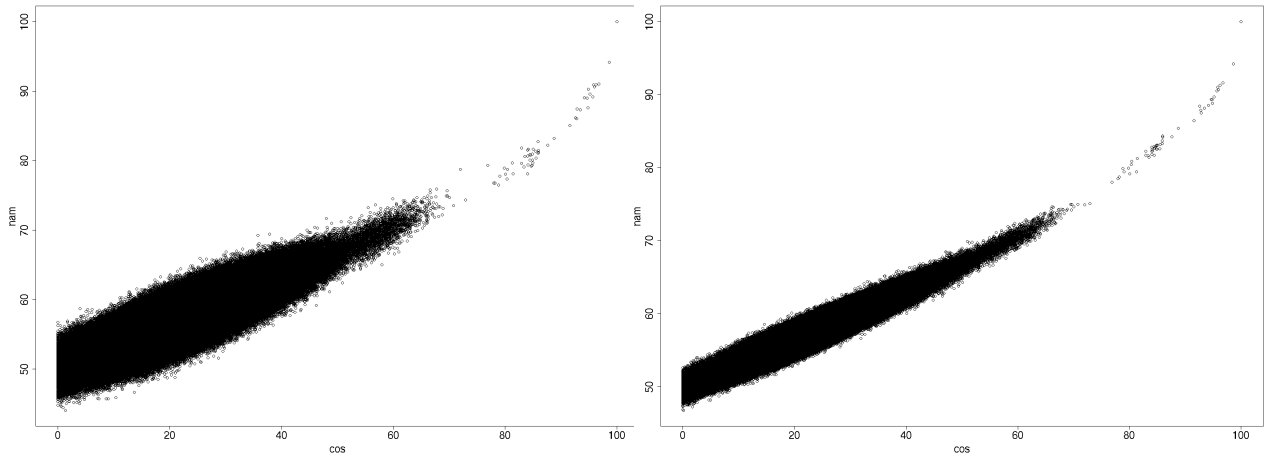
Velmi pozitivní jsou vysoké hodnoty měr *Precision* a *Recall* při aproximaci obou vektorových modelů. Ideální hodnota parametru *treshold_ham* je pro *treshold_sim=80* rovna přibližně hodnotě 79 - 80 u obou vektorových modelů.

2.5.2 Coll_Small

Jelikož tato kolekce obsahovala pouze malé dokumenty, čítal slovník vytvořený z tého kolekce pouze 166861 termů a proto jsme se rozhodli nevytvářet více sad vektorů pomocí redukovaných slovníků. V měřeních na *Coll_Mixed* jsme viděli, že délka signatur nemá vliv na funkce L_1, L_2 a vliv na veličinu S byl prokázán zcela nezpochybnitelně, proto jsme nepovažovali za nutné provádět experimenty na dalších kolekcích s tolika různými nastaveními.

délka signatur	***počet různých signatur***	***L1***	***L2***	***S***
1024	9850	35,426	1283,226	1,826
2048	9850	35,321	1274,364	0,990
3072	9850	35,510	1287,128	0,649
4096	9850	35,370	1276,989	0,499

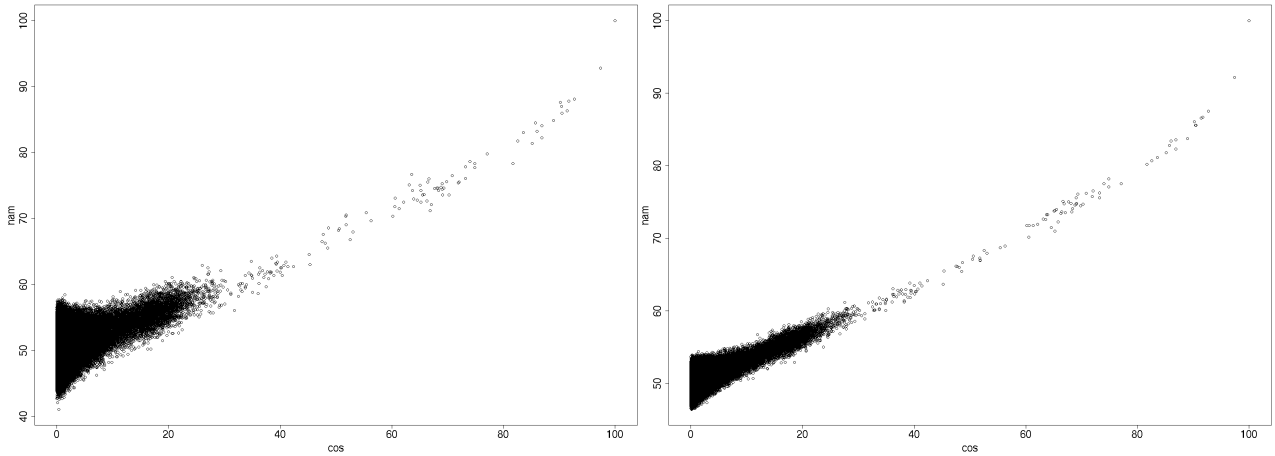
Tab. 2.31: Porovnání různých délek signatur za použití hodnot tf



Obr. 2.32,2.33 K Tab. 2.31 – závislost hammingových podobností na kosinových délka signatur 1024 (vlevo), délka signatur 4096 (vpravo)

délka signatur	*počet různých signatur*	*L1*	*L2*	*S*
1024	9850	49,541	2457,130	1,703
2048	9850	49,543	2456,143	0,922
3072	9850	49,548	2456,292	0,606
4096	9850	49,545	2455,693	0,473

Tab. 2.34: Porovnání různých délek signatur za použití hodnot tf idf

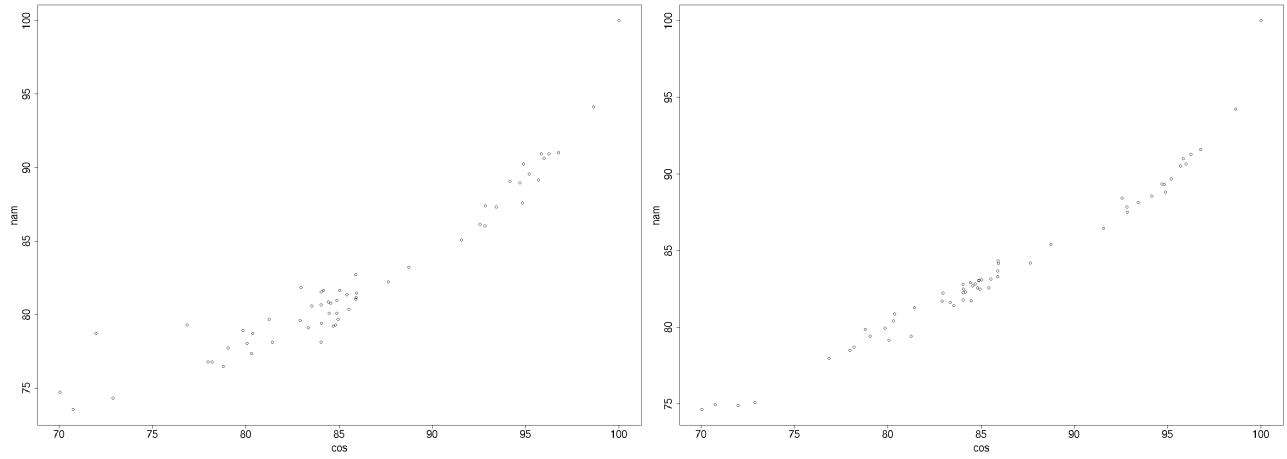


Obr. 2.35,2.36: K Tab. 2.34 – závislost hammingových podobností na kosinových délka signatur 1024 (vlevo), 4096 (vpravo)

Na této kolekci se znovu potvrdila neměnnost funkcí L_1, L_2 s rostoucí délkou signatur, taktéž se potvrdil vztah mezi klesající veličinou S a rostoucí délkou signatur. Hodnoty funkcí L_1, L_2 naměřené při tomto měření jsou oproti stejným měřením na předchozí kolekci vyšší proto, že v této kolekci je větší podíl méně podobných dokumentů (většina bodů na grafu je vlevo), kde je rozdíl hammingovy a kosinové podobnosti větší (shodně jak u *tf*, tak *tf_idf* vektorového modelu).

délka signatur	*R*	*L1*	*L2*	*S*
1024	0,540	3,162	16,100	0,656
2048	0,685	2,770	13,544	0,473
3072	0,593	2,224	8,706	0,329
4096	0,654	2,244	9,468	0,299

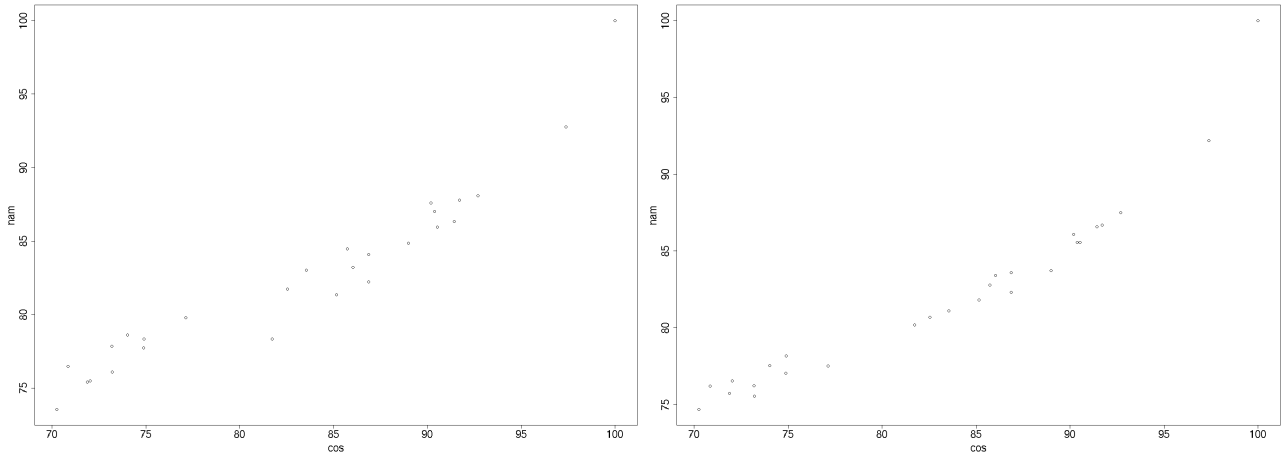
Tab. 2.37: Porovnání různých délek signatur, tf vektory, treshold_sim=70, treshold_ham=73



Obr. 2.38,2.39: K Tab. 2.37 – závislost hammingových podobností na kosinových délka signatur 1024 (vlevo), délka signatur 4096 (vpravo)

délka signatur	*R*	*L1*	*L2*	*S*
1024	0,769	2,063	8,304	1,030
2048	0,896	2,041	8,581	0,208
3072	0,886	2,153	9,314	0,141
4096	0,769	2,312	10,285	0,245

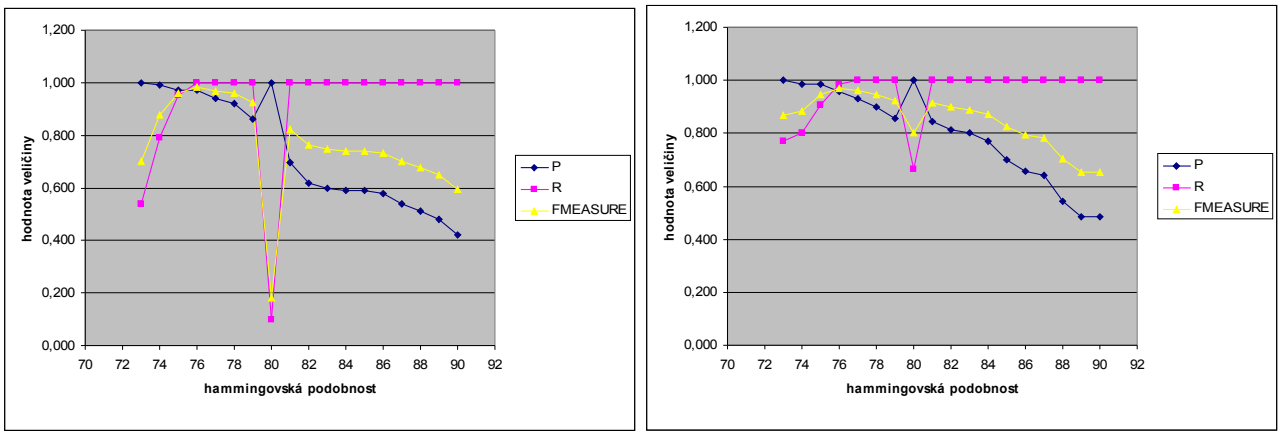
Tab. 2.40: Porovnání různých délek signatur, tf_idf vektory, treshold_sim=70, treshold_ham=73



Obr. 2.41,2.42: K Tab. 2.40 – závislost hammingových podobností na kosinových délka signatur 1024 (vlevo), délka signatur 4096 (vpravo)

Při vyhledávání blízkých duplicit je aproximace opět kvalitní, hodnoty funkcí L_1, L_2 jsou nízké (při aproximaci *tf_idf* vektorového prostoru jsou obecně nižší než při aproximaci *tf* vektorového modelu). Vysoké hodnoty *Recall* při aproximaci *tf* vektorového modelu jsou způsobeny tím, že hranici *treshold_sim=70* a *treshold_ham=73* vyhovělo velmi málo párů, neboť většina párů dokumentů je v levé části grafů (Obr. 2.32,2.35).

Překvapivým jevem je zvýšení hodnoty veličiny *S* při přechodu z délky signatur 3072 na délku 4096 při aproximaci *tf_idf* vektorového prostoru. Měření na této kolekci ale nemůžeme v tomto ohledu brát jako směrodatná, neboť je v oblasti blízkých duplicit příliš málo párů.



Grafy 2.43,2.44: Průběhy veličin Precision a Recall v závislosti na parametrech aproximace, délka signatur=1024, tf vektory (vlevo), tf_idf vektory (vpravo)

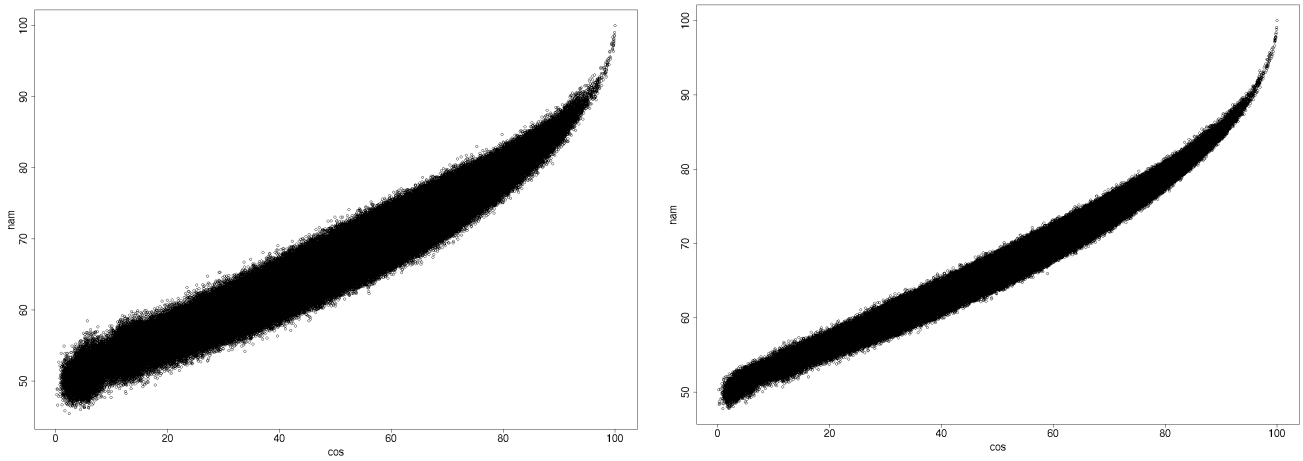
Na rozdíl od předchozí kolekce je v tomto případě ideální hodnota *treshold_ham* větší než hodnota *treshold_sim*, konkrétně pro *treshold_sim=70* je to hodnota cca 76 (u obou vektorových modelů). Skok v oblasti okolo hammingovy podobnosti 80 zřejmě poukazuje na nerovnoměrné rozdělení párů v oblasti blízkých duplicit a je také zapříčiněn nízkým počtem těchto párů.

2.5.3 Coll_Large

Všechny dokumenty obsažené v této kolekci svou velikostí přesahovaly hranici 500 různých termů. Slovník této kolekce obsahoval 889469 termů, proto jsme se rozhodli vytvořit redukovaný slovník, který z plného vznikl odstraněním slov splňujících *DC=1*. Poučeni z předchozích experimentů, kdy velikost slovníku kromě délky výpočtů nic neovlivňovala, jsme experimenty na této kolekci v rámci úspory času prováděli pouze na redukovaném slovníku.

délka signatur	***počet různých signatur***	***L1***	***L2***	***S***
1024	9554	12,579	219,692	1,673
2048	9557	13,121	233,425	0,872
3072	9558	12,886	227,466	0,590
4096	9558	12,372	215,566	0,482

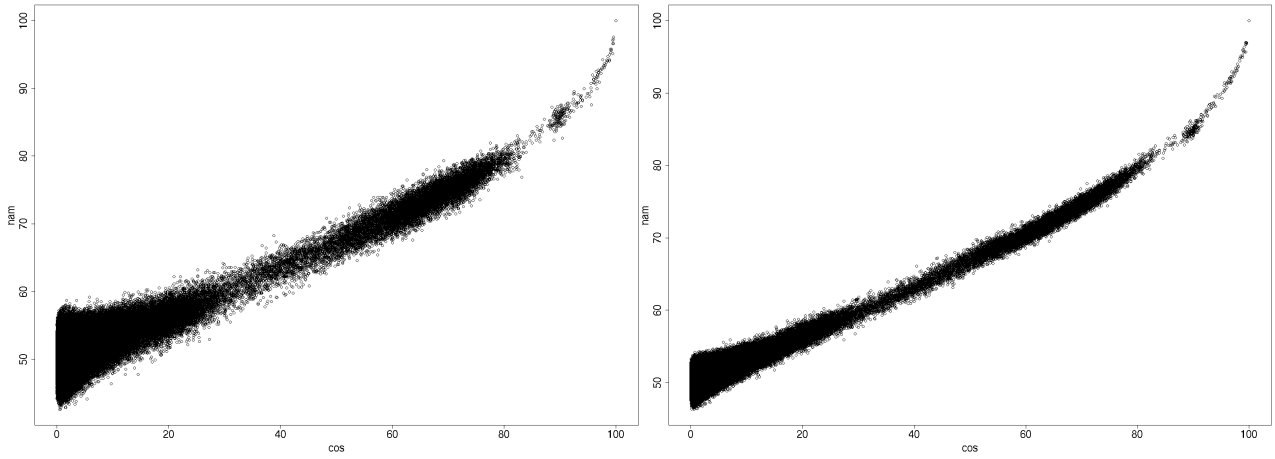
Tab. 2.45: Porovnání různých délek signatur za použití hodnot tf a redukovaného slovníku



Obr. 2.46,2.47: K Tab. 2.45 – závislost hammingových podobností na kosinových délka signatur 1024 (vlevo), délka signatur 4096 (vpravo)

délka signatur	***počet různých signatur***	***L1***	***L2***	***S***
1024	9558	48,937	2402,615	1,785
2048	9558	48,915	2399,381	0,880
3072	9558	48,906	2398,101	0,663
4096	9558	48,927	2399,976	0,472

Tab. 2.48: Porovnání různých délek signatur za použití hodnot tf_idf a redukovaného slovníku

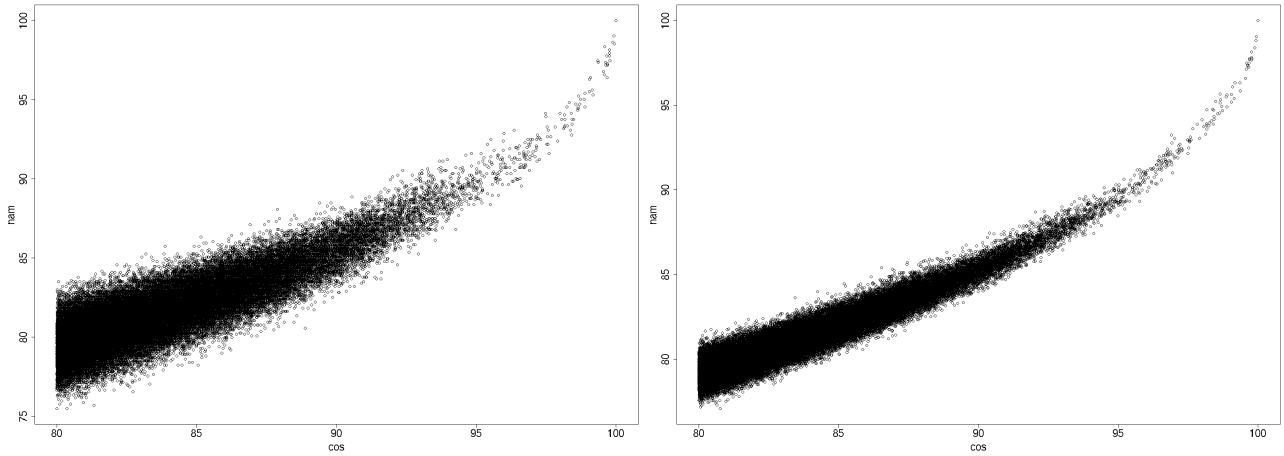


Obr. 2.49,2.50: K Tab. 2.48 – závislost hammingových podobností na kosinových délka signatur 1024 (vlevo), délka signatur 4096 (vpravo)

Díky rovnoměrnému zastoupení všech kosinových podobností v *tf* vektorovém modelu (Obr. 2.46,2.47) není hodnota funkcí L_1, L_2 ovlivněna body nakupenými u nízkých kosinových podobností a proto jsou hodnoty tyto naměřené na *tf* vektorovém modelu nejlepší ze všech naměřených (na všech kolekcích). Hodnoty L_1, L_2 měřené na *tf_idf* vektorovém prostoru jsou opět zhoršeny charakteristikou tohoto prostoru. Na délce signatur v obou prostorech opět zcela nezávisí funkce L_1, L_2, závislost veličiny *S* na délce signatur je zcela zřejmá.

délka signatur	***R***	***L1***	***L2***	***S***
1024	0,125	2,152	6,643	0,957
2048	0,108	1,639	4,148	0,518
3072	0,119	1,761	4,639	0,380
4096	0,152	2,328	6,979	0,356

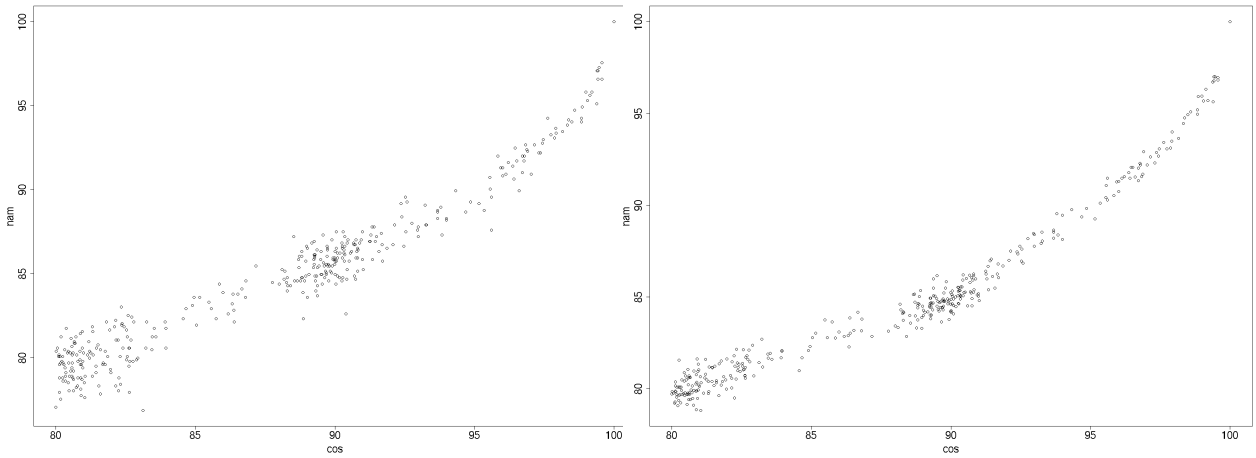
Tab. 2.51: Porovnání různých délek signatur, tf vektory, treshold_sim=80, treshold_ham=75



Obr. 2.52,2.53: K Tab. 2.51 – závislost hammingových podobností na kosinových délka signatur 1024 (vlevo), délka signatur 4096 (vpravo)

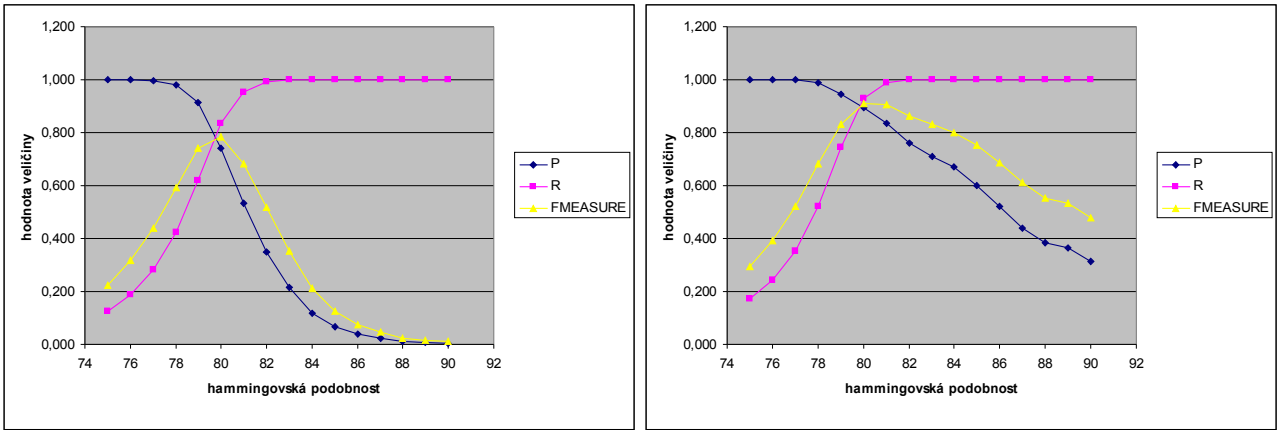
délka signatur	*R*	*L1*	*L2*	*S*
1024	0,173	2,749	11,486	0,800
2048	0,186	2,824	11,726	0,405
3072	0,197	2,832	11,873	0,346
4096	0,201	2,852	12,603	0,288

Tab. 2.54: Porovnání různých délek signatur, tf_idf vektory, treshold_sim=80, treshold_ham=75



Obr. 2.55,2.56: K Tab. 2.54 – závislost hammingových podobností na kosinových délka signatur 1024 (vlevo), délka signatur d4096 (vpravo)

V oblasti velmi podobných dokumentů je aproximace do třetice velmi kvalitní, hodnoty funkcí L_1, L_2 jsou u obou vektorových prostorů nízké. I zde se ukazuje závislost mezi hodnotou veličiny *S* a délkou signatur. Nízké hodnoty míry *Recall* jsou dány rovnoměrnou distribucí kosinových podobností v celém intervalu [0,100] (distribuce je nejlepší ze všech tří kolekcí) - viz Obr. 2.46,2.49.



Grafy 2.57,2.58:Průběhy Precision a Recall

V obou grafech křivky veličiny *Recall* strmě stoupají, což je dle našeho názoru způsobeno faktem, že páry dokumentů jsou rovnoměrně rozprostřeny i v okolí blízkých duplicit. Jako ideální hodnota parametru *treshold_ham* pro *treshold_sim=80* se jeví hodnota 80 (u obou vektorových modelů).

2.6 Časový přínos LSH-signaturového modelu

Ke zjištění zrychlení, kterého použitím LSH-signaturového modelu dosáhneme oproti klasickému vektorovému modelu, jsme provedli experimenty na kolekci obsahující 644 566 českých dokumentů posbíraných z českých periodik (všechny výše zmíněné kolekce byly podmnožinami této). Neredukovaný slovník obsahoval 2 918 334 termů, proto jsme se rozhodli redukovat slovník o termy s *DC=1* (redukovaný slovník měl velikost 1 543 251 termů). Na základě tohoto slovníku jsme vytvořili *tf_idf* vektory a následně signatury délky 1024. Poté jsme změřili délky běhů aplikací *similarity*, *hamming_full* a *hamming* (s různými hodnotami parametru *iter_count)* - všechna volání byla bez použití parametru -*sample_list*. Parametr *treshold* byl nastaven na hodnotu 95. Výsledky měření jsou uvedeny v tabulce.

aplikace	***délka běhu (min)***	***objevených párů (%)***
hamming (iter_count=50)	18,22	99,40
hamming (iter_count=100)	41,75	99,61
hamming (iter_count=250)	107,47	100,00
hamming (iter_count=500)	222,55	100,00
hamming (iter_count=1000)	396,77	100,00
hamming_full	912,47	
similarity	36259,11	

Tab. 2.59: Porovnání časové náročnosti LSH-signaturového a vektorového modelu (beam=50, shift_count=70)

Pozn. Pro výpočet dat uvedených ve třetím sloupečku byl použit počet všech objevitelných párů pomocí LSH-signaturového přístupu (celkem 24 306 párů).

V souladu s 2.3.1 odstavec *3/* se i malý počet iterací projevil jako dostačující pro nalezení všech blízkých duplicit při vysoké hodnotě parametru *treshold*. Naměřené časy ukazují, že časové zrychlení získané použitím LSH-signaturového modelu je obrovské (necelých 20 minut oproti 25 dnům). Ve prospěch LSH-signaturového modelu hovoří i fakt, že algoritmus PLEB je lineární ve velikosti kolekce, zatímco aplikace *similarity* je v téže veličině kvadratická. Můžeme tedy odhadnout, že na kolekci velikosti 10M dokumentů (předpokládejme přibližně stejně dlouhé dokumenty jako v měřené kolekci a totožné nastavení *beam* a *shift_count*), by výpočet pomocí aplikace *hamming* (*iter_count*=100) trval přibližně 650 minut, zatímco pomocí aplikace *similarity* přibližně 17 let.

Zřejmě je při použití LSH-signaturového modelu potřeba započítat i dobu během níž se vytvářejí signatury. Tato je ale vzhledem k délce běhu aplikace *similarity* zanedbatelná, neboť zatímco v aplikaci *similarity* je počítáno *velikost_kolekce * velikost_kolekce* skalárních součinů, v aplikaci *create_sigs* je těchto třeba pouze *velikost_kolekce * délka_signatur* (rozdíl je v několika řádech).

Kapitola 3

Závěr

Na základě experimentů na všech kolekcích se nám podařilo prokázat, že LSH-signaturový přístup je při hledání velmi podobných dokumentů vhodnou alternativou, která na jedné straně přináší nižší přesnost aproximace, na druhou stranu se může pochlubit neporovnatelně kratším výpočetním časem, takže je díky němu možné upočítat dříve neupočitatelné.

Otázku volby optimální délky signatury nedokážeme přesně odpovědět, závisí totiž na požadavcích každého experimentu. Je překvapivé, že růst počtu různých signatur se na nějaké délce signatur zastaví a dále neroste. Potěšivým faktem je klesající hodnota veličiny *S*, která potvrzuje intuitivní představu, že více náhodných vektorů musí každý dokument lépe popisovat. Volba optimální délky signatury tedy závisí na tom, jakou hodnotu veličiny *S* uživatel vyžaduje. Nutno dodat, že dvojnásobná délka signatur s sebou kromě cca dvojnásobného zlepšení míry S (ve všech tabulkách je vztah délky signatur a veličiny *S* téměř nepřímá úměrnost) přináší i dvakrát delší čas běhů aplikací *create_sigs* a *hamming*(*_full*).

Za velmi dobré považujeme hodnoty *Precision* a *Recall* naměřené při experimentech na všech kolekcích (nezávisle na použitém vektorovém modelu), které se při nejlepších nastaveních *treshold_ham* pohybovaly v intervalu [0,8 ; 1]. *Tf_idf* vektorový model je díky své charakteristice kvalitněji aproximovatelný, než *tf* vektorový (menší hodnoty funkcí L_1, L_2, vyšší hodnoty měr *Precision* a *Recall)*, nicméně rozdíl při správném nastavení *treshold_ham* není nijak zásadní.

Prokázali jsme také, že pomocí algoritmu PLEB lze najít velmi velké části nalezitelných výsledků, aniž bychom museli přikročit k prohledávání „každý s každým“. Zároveň se nám podařilo formulovat obecné principy ovlivňující chování algoritmu PLEB (úspěšnost, časová složitost).

Naopak se neprokázala závislost kvality aproximace na počtu nenulových složek vektorů vztažených k velikosti slovníku (prozkoumání této vlastnosti bylo důvodem pro volbu charakteristik všech tří experimentovaných kolekcí). Proto se LSH-signaturový model jeví jako vhodný i pro kolekce malých dokumentů.

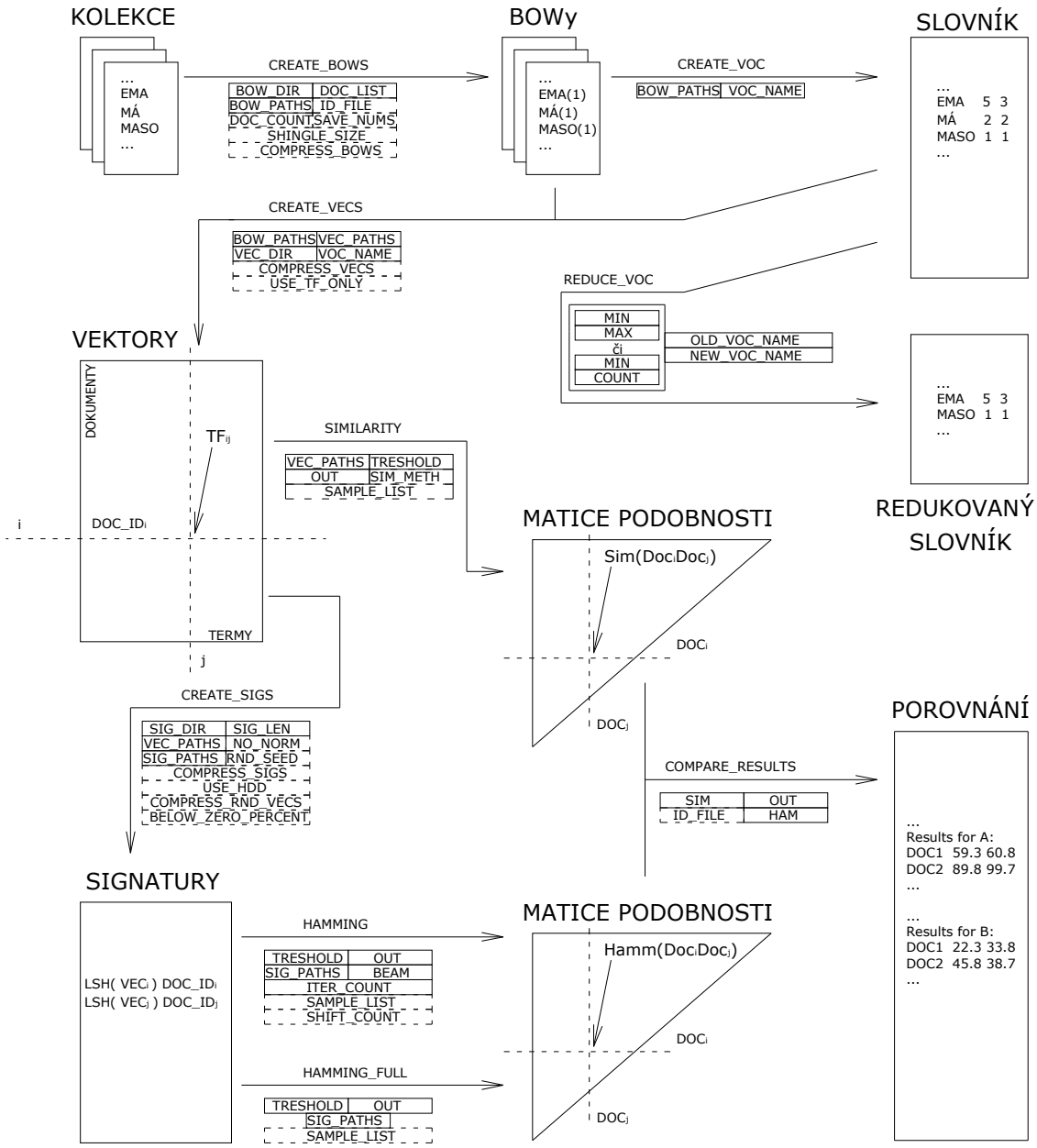
Hodnoty *treshold_sim* a *treshold_ham* bychom doporučovali volit stejně, případně snížit o 1-2 hodnotu *treshold_ham* pro zvýšení *Precision*, či zvýšit o 1-2 touž hodnotu pro zvýšení *Recall*.

Příloha A

Implementační část

A.1 Dekompozice úlohy

Na implementační úrovni je úkol řešen balíkem aplikací, z nichž každá se stará o jinou fázi hledání téměř identických dokumentů. Vstupně výstupní závislosti jednotlivých aplikací ukazuje diagram níže. Každá šipka představuje právě jednu aplikaci, jejíž povinné parametry jsou orámovány plnou čarou, zatímco čárkovanou čarou jsou označeny parametry volitelné. Všechny aplikace v případě volání s parametrem "-h", či bez parametrů podají informaci o způsobu volání a významu svých jednotlivých parametrů.



A.2 Funkce jednotlivých aplikací (uživatelská dokumentace)

CREATE_BOWS

Pro všechny dokumenty kolekce sestaví bag of words (dále jen BOW), neboli množinu slov dokumentu spolu s počtem výskytů slov v rámci daného dokumentu.

CREATE_VOC

Vytvoří slovník celé kolekce, navíc uchová celkový počet výskytů každého slova v celé kolekci a počet dokumentů, v nichž je dané slovo obsaženo.

REDUCE_VOC

Ze slovníku vytvořeného aplikací *create_voc* odstraní ta slova, která nevyhovují uživatelem daným podmínkám na *tf, df* či *tf_idf*.

CREATE_VECS

Ze slovníku a BOWu každého dokumentu kolekce vytvoří vektor odpovídající danému dokumentu. V závislosti na použitých přepínačích jsou vektory tvořeny hodnotami *tf* či *tf_idf*.

CREATE_SIGS

Z vektorů vytvořených v aplikaci *create_vecs* vytvoří signatury pomocí LSH algoritmu.

SIMILARITY

Na vektory vytvořené aplikací *create_vecs* aplikuje uživatelem danou metodu na porovnávání vektorů dokumentů (implicitně kosinová míra) a průchodem „každý s každým“ odfiltruje ty dvojice dokumentů, které svou podobností (vůči použité porovnávací metodě) přesahují uživatelem požadovanou míru podobnosti.

HAMMING

Pomocí algoritmu PLEB vyhledá v signaturách vytvořených aplikací *create_sigs* ty dvojice signatur (resp. dokumentů), které svou podobností (poměrná hammingova podobnost) překročí uživatelem zadanou hodnotu.

HAMMING_FULL

Pomocí průchodu „každý s každým“ vyhledá v signaturách vytvořených aplikací *create_sigs* ty dvojice signatur (resp. dokumentů), které svou podobností (poměrná hammingova podobnost) překročí uživatelem zadanou hodnotu.

COMPARE_RESULTS

Ze souborů obsahujících výsledek aplikací *hamming*, resp. *hamming_full* a *similarity* vytvoří přehledné porovnání podobností vytvořených oběma aplikacemi pro jednotlivé dvojice dokumentů.

A.3 Význam jednotlivých přepínačů aplikací

Aplikace, které mají pouze jeden výstup, konkrétně se jedná o aplikace *create_voc, reduce_voc, hamming, hamming_full, similarity a compare_results*, mohou svůj výstupní soubor rovnou komprimovat, čímž dojde k významné úspoře místa na disku. Tohoto efektu lze dosáhnout použitím koncovky „.gz“ u jména výstupního souboru. U ostatních aplikací je komprimaci výstupu nutno vynutit voláním s přepínačem *-compress_****.

CREATE_BOWS

- **doc_list** jméno_souboru : soubor obsahující jména souborů kolekce
- **bow_paths** jméno_souboru : soubor k uložení cest k BOW souborům
- **bow_dir** jméno_adresáře : jméno adresáře k uložení BOW souborů
- **[compress_bows]** : BOW soubory budou komprimovány.

- **[doc_count** prirozene_cislo**]** : počet prvních řádků souboru doc_list, které budou zpracovány.

- **[shingle_size** prirozene_cislo**]** : parametr udávající číslo *k* takové, že teprve k-tice po sobě jdoucích slov dokumentu jsou považována za právoplatná slova dokumentu. Implicitní hodnota je nastavena na jedna.

- **[save_nums]** : při použití tohoto přepínače budou brána v potaz i slova skládající se pouze z číslic.

- **[id_file** jméno_souboru**]** : při použití toho přepínače bude zavedeno mapování jmen dokumentů na ID, se kterým se bude pracovat v průběhu celého výpočtu a dojde tím k výrazné úspoře velikosti výstupů aplikací *hamming*, *hamming_full* a *similarity*. Do zadaného souboru bude uloženo vzniklé mapování.

Soubor doc_list je jediným uživatelským vstupem aplikace. Jeho obsahem jsou cesty (absolutní či relativní) k dokumentům kolekce, každá oddělená bílým znakem.

CREATE_VOCABULARY

- **bow_paths** jméno_souboru : soubor obsahující cesty k BOW souborům
- **voc_name** jméno_souboru : jméno souboru k uložení slovníku

REDUCE_VOCABULARY

- **old_voc_name** jméno_souboru : jméno souboru obsahující slovník k redukci
- **new_voc_name** jméno_souboru : jméno souboru pro uložení redukovaného slovníku
- vlastnosti slov, která budeme chtít zachovat do redukovaného slovníku:
  - **min_tf** float, **max_tf** float : rozsah hodnot TF
  - **min_tf** float, **count** prirozene_cislo : minimální hodnota TF a požadovaná mohutnost redukovaného slovníku
  - **min_df** float, **max_df** float : rozsah hodnot DF
  - **min_df** float, **count** prirozene_cislo : minimální hodnota DF a požadovaná mohutnost redukovaného slovníku
  - **min_tf_idf** float, **max_tf_idf** float : rozsah hodnot TF_IDF
  - **min_tf_idf** float, **count** prirozene_cislo : minimální hodnota TF_IDF a požadovaná mohutnost redukovaného slovníku

POZN. Aplikace automaticky rozpoznává, zda *old_voc_name* je soubor komprimovaný či nikoliv, bez explicitního uvedení, takže je kompatibilní se všemi možnými výstupy *create_voc*. Kdybychom, například, chtěli do redukovaného slovníku zahrnout pouze slova s TF v rozmezí 1 až 10, obsahovalo by volání mimo jiné také *-min_tf 1 -max_tf 10*. Uplatňovat lze pouze ty páry podmínek, které jsou výše uvedené (například pár *-min_tf* a *-max_df* je nepřípustný).

CREATE_VECTORS

- **bow_paths** jméno_souboru : jméno souboru se jmény BOW souborů
- **voc_name** jméno_souboru : jméno souboru se slovníkem
- **vec_dir** jméno_adresáře : jméno adresáře k uložení VEC souborů
- **vec_paths** jméno_souboru : jméno souboru k uložení cest k VEC souborům
- **[use_tf_only]** : vektory dokumentů budou sestaveny z hodnot *tf* a nikoliv *tf_idf*
- **[compress_vecs]** : při použití tohoto přepínače budou VEC soubory komprimovány.

SIMILARITY

- **vec_paths** jméno_souboru : jméno souboru obsahujícího cesty k VEC souborům
- **out** jméno_souboru : jméno souboru k uložení výsledku
- **treshold** float : prahová hodnota podobnosti, pouze dvojice s podobností vyšší nebo rovnou treshold budou zařazeny do výsledku
- **[sim_meth** retezec**]** : určení metody porovnávající vektory dvou dokumentů. Implicitní nastavení je kosinová míra
- **[sample_list** jméno_souboru**]** : soubor obsahující vzorek, tedy podmnožinu dokumentů kolekce, vůči kterým se bude podobnost počítat

CREATE_SIGNATURES

- **vec_paths** jméno_souboru : jméno souboru obsahujícího cesty k VEC souborům
- **sig_len** prirozene_cislo : požadovaná délka signatur
- **sig_dir** jméno_adresáře : jméno adresáře pro uložení SIG souborů
- **sig_paths** jméno_souboru : jméno souboru pro uložení cest k SIG souborům
- **[no_norm]** : pokud je tento parametr uveden, vektory dokumentů nebudou před převodem na signatury normalizovány
- **[rnd_seed** prirozene_cislo**] :** seed generátoru náhodných čísel
- **[compress_sigs]** : při použití přepínače budou SIG soubory komprimovány
- **[use_hdd** prirozene_cislo**]** : při použití přepínače budou náhodné vektory generovány na disk. Hodnota uvedená za přepínačem určuje, kolik MB RAM může být využito k uložení náhodných vektorů
- **[compress_rnd_vecs]** : při použití tohoto přepínače budou soubory s nageneravanými náhodnými vektory ukládány komprimovaně
- **[below_zero_percent** prirozene_cislo**]** : hodnota tohoto parametru určuje, kolik procent složek náhodných vektorů musí být kladných, resp. záporných. Např. při hodnotě 30 musí podíl kladných i záporných složek přesáhnout 30%

POZN. SIG soubory jsou soubory obsahující dvojice signatura - jméno souboru. V zájmu úspory místa na disku a následné úspory času při čtení souboru signatur je vhodné ukládat signatury nikoliv jako řetězce číslic 0 a 1, ale jako řetězce šestnáctkových cifer (každá šestnáctková cifra odpovídá čtveřici nul a jedniček). Z toho vyplývá podmínka, aby požadované délky signatur byly dělitelné čtyřmi. Navíc je vhodné zvolit délku signatury (v naší implementaci) jako číslo dělitelné 32, resp. 64 (na 32-bitových, resp. 64-bitových strojích), jinak nebude plně využit paměťový potenciál aplikace.

HAMMIMG

- **iter_count** přirozené_číslo : počet iterací PLEB algoritmu
- **sig_paths** jméno_souboru : jméno souboru obsahujícího cesty k SIG souborům
- **treshold** float : prahová hodnota podobnosti, pouze dvojice s podobností vyšší nebo rovnou treshold budou zařazeny do výsledku
- **out** jméno_souboru : jméno souboru k uložení výsledku
- **beam** přirozené_číslo : rozpětí, v rámci kterého se v algoritmu PLEB hledají podobné signatury vůči aktuálně zpracovávané signatuře
- **[shift_count** přirozené_číslo**]** : počet transpozic aplikovaných na vektor signatur v rámci jedné iterace PLEB algoritmu. Implicitní hodnota rovna jedné.
- **[sample_list** jméno_souboru**]** : soubor obsahující vzorek, tedy podmnožinu dokumentů kolekce, vůči kterým se bude podobnost počítat.

HAMMING_FULL

- **sig_paths** jméno_souboru : jméno souboru obsahujícího cesty k SIG souborům
- **treshold** float : prahová hodnota podobnosti, pouze dvojice s podobností vyšší nebo rovnou treshold budou zařazeny do výsledku
- **out** jméno_souboru : jméno souboru k uložení výsledku
- **[sample_list** jméno_souboru**]** : soubor obsahující vzorek, tedy podmnožinu dokumentů kolekce, vůči kterým se bude podobnost počítat.

COMPARE_RESULTS

- **sim** jméno_souboru : jméno souboru obsahujícího výsledek aplikace *similarity*
- **ham** jméno_souboru : jméno souboru obsahujícího výsledek aplikace *hamming*, resp. *hamming_full*
- **out** jméno_souboru : jméno souboru k vytištění výsledného porovnání
- **[id_file** jméno_souboru**]** : jméno souboru obsahujícího mapování ID dokumentů na jejich jména. Volitelný parametr k použití pouze v případě, že aplikace *create_bows* byla volána s parametrem *id_file*

POZN. Aplikace automaticky rozpoznává, zda jsou soubory *ham* a *sim* komprimované či nikoliv a je tedy plně kompatibilní se všemi možnými výstupy aplikací *hamming(_full)* a *similarity*.

A.4 Technická dokumentace

A.4.1 Úvod k technické dokumentaci

Rychlost a správná funkce celého balíku aplikací v naprosté míře závisí na několika hlavních datových strukturách, které, po jedné, tvoří ústřední část některých aplikací. Pro aplikaci *create_voc* je to *hashmapa* mapující typ *struktura* na typ *char* * a pro *create_sigs*, *hamming* a *hamming_full* je to struktura *TSignatura* uchovávající binární signatury přímo v bitech. Míra efektivity a úspornosti kódu těchto struktur hraje kritickou úlohu pro rychlost celého výpočtu (implementační detaily viz níže). Nyní rozebereme implementaci každé ze jmenovaných struktur.
(Pozn. implementace celého balíku aplikací samozřejmě zahrnuje mnohem více (datových) struktur. Pro příklad uveďme řídké vektory využívané aplikací *create_vecs* a *create_sigs*. Jejich implementace je však zcela klasická a proto nemá smysl se o nich více rozepisovat).

A.4.2 Hlavní datové struktury

A.4.2.1 Struktura TSignatura

```
struct TSignatura
{
     unsigned int Zaloz(char * Signatura_V_Retezci);
     unsigned int Zaloz(unsigned int Delka_V_Bitech);
     void Znic();
     void Nastav_Bit(unsigned int Index, unsigned int
                                              Hodnota);
     unsigned int Zjisti_Bit(unsigned int Index) const;
     void Prohod_Bity(unsigned int Prvni_Index, unsigned int
                                              Druhy_Index);
     unsigned long* Adresa_Signatury(void){ return _data;};
     unsigned long * _data;
     void Vypis_Signaturu(void);
};
```

Z výše uvedených metod struktury je zajímavá pouze implementace metody *Zjisti_Bit* a *Nastav_Bit*, implementace ostatních je většinou závislá na těchto dvou. Struktura *TSignatura* uchovává svá data v poli typu *unsigned long*. Jednotlivé bity jsou ve struktuře uloženy přesně v tom pořadí, v jakém za sebou jdou v signatuře (tedy stejně jako pomocí Big-Endianu). Čtení a nastavení jednotlivých bitů probíhá pomocí aplikace bitových masek na příslušné položky *_data* (lehce k dohledání ve zdrojovém kódu - nejdříve je pomocí operace *modulo* určena složka *_data*, ve kterém je hledaný *bit* uložen a následně je na vybranou složku *_data* aplikována bitová maska v závislosti na tom, zda se jedná o operaci *Zjisti_Bit* či *Nastav_Bit* a na tom,zda je nastavovaná či čtená hodnota rovna jedné či nule).

Datový typ *unsigned long* z datových typů obsažených v jazyce C nejvíce odráží architekturu daného stroje (tedy na 32b strojích má tento 4B a na 64b strojích má tento nejčastěji ze všech datových typů 8B) a proto je nejvhodnější použít právě tento, pokud chceme zaručit, že procesor bude moci pracovat vždy s daty velikosti svého slova a nebude tak muset extrahovat menší datové typy z větších, pro daný stroj nativních. Toto je důvodem, proč jsou data struktury *TSignatura* ukládána do pole typu *unsigned long*.

V závislosti na použitém nastavení překladače, resp. v závislosti na konkrétním počtu *bytů* datového typu *unsigned long* jsou nadefinovány příslušné konstanty, které struktura *TSignatura* využívá, aby byla zajištěna správná funkčnost této a též je vybrána správná funkce zjišťující hammingovu vzdálenost signatur, jejíž implementace se jak pro *32b*, tak pro *64b unsigned long* liší (více o této v dokumentaci aplikace *hamming*).

*A.4.2.2 Struktura typu hash_map < char * , struct X >*

Tato struktura, která se od normální implementace *hash_mapy* poněkud liší, je použita pouze v aplikaci *create_voc* a *compare_results*. V ostatních aplikacích jsou použity klasické implementace *hash_mapy*, neboť pouze v aplikaci *create_voc* má použití této nestandardní implementace odůvodnění z důvodu úspory procesorového času, v aplikaci *compare_results* je použití této datové struktury spíše záležitostí přehlednosti kódu, rychlost aplikace *compare_results* totiž drtivou měrou ovlivňuje rychlost načítání dat z disku.

Všechny implementace *hash_mapy*, včetně této, používají následující hashovací_funkci:

```
unsigned int Prvociselne_Hashovani(char * Slovo)
{
     unsigned int i;
     unsigned long Vysledek = 0, Nasobitel;
     for (i = 0; i < strlen(Slovo); i++){
          Nasobitel= pow(PRVOCISLO, i);
          Nasobitel *= (unsigned long)Slovo[i];
          Vysledek += Nasobitel;
          Vysledek = Vysledek % VELIKOST_MAPY;
          };
     return (unsigned int)Vysledek;
};
```

Pozn. `PRVOCISLO` a `VELIKOST_MAPY` jsou konstanty definované (za překladu) v rámci projektu.

Při ukládání slova je pomocí funkce (viz výše) spočten index do pole kontejnerů typu vektor, do nějž jsou ukládána data o všech slovech, která na daném indexu kolidují (princip totožný s klasickou implementací *hash_mapy*). Do tohoto vektoru není uloženo samotné slovo

ani data ke slovu se vztahující, nýbrž index do vnějších polí, na kterém jsou slovo a k němu se vztahující data uložena.

V aplikaci *create_voc* jsou tato pole konkrétně *Pole_Slov, Pole_TF, Pole_DF* a *Pole_Pridano_V_Tomto_Kole* (poslední je typu *bool* * a údaje v něm uložené slouží jako pomůcka při výpočtu DC daného slova, je nastavena na `false` při otevírání dokumentu a při prvním výskytu daného slova je nastavena na `true`). Všechna pole jsou stejně velká (co do hranice indexů) a na jednotlivých indexech jsou ve všech polích údaje týkající se jednoho slova. Důvodem, pro který byla zvolena tato implementace namísto klasické, kdy jsou data ukládána rovnou do vektorů obsahujících kolidující záznamy v podobě struktur, je ten, že před zpracováním BOWu každého dokumentu kolekce je nutno položku *Pridano_V_Tomto_Kole* všech slov nastavit na *false*. Při klasickém přístupu je tato operace realizovaná jako iterace přes celou `hashmapu` a její rychlost silně závisí na velikosti pole, na kterém jsou pověšené vektory obsahující kolidující záznamy a do něhož se pomocí hashovací funkce získává index (viz pozn. na další straně). Při použití našeho přístupu je tato operace řešena jediným *memsetem* pole *Pole_Pridano_V_Tomto_Kole*. Experimenty prokázaly, že použití našeho přístupu dokáže oproti použití klasického přístupu zkrátit čas výpočtu z jednotek hodin na desítky vteřin. Nevýhodou našeho řešení je redundance dat, neboť s každým slovem je nezbytně nutné si navíc uchovávat i index do zmíněných vnějších polí (poznamenejme ale, že tento nedostatek je pouze "teoretického rázu" - pokud je aplikace *create_voc* používána v rámci celého balíku aplikací, je při velikosti slovníku 1 milion slov paměťový overhead způsobený redundancí dat velký 4MB (resp. 8MB na 64b strojích), což je dnes zanedbatelné množství paměti).

A.4.3 Technická dokumentace jednotlivých aplikací

CREATE_BOWS

VSTUPNÍ DATA

Jediným vstupem aplikace je soubor obsahující cesty (ať už absolutní či relativní vůči místu spouštění aplikace) k dokumentům kolekce, které mají být zpracovány. Jednotlivé cesty musí být odděleny bílým znakem.

VÝSTUPNÍ DATA

Výstupem aplikace jsou BOW soubory (tedy soubory obsahující bag of words všech dokumentů kolekce nebo prvních několika dokumentů kolekce, jejichž počet je určen přepínačem *-doc_count*) spolu se souborem obsahujícím informace o kolekci a cesty ke všem vytvořeným BOW souborům.

V případě, že je aplikace spuštěna s přepínačem *-id_file*, je výstupem aplikace i soubor mapující jména dokumentů kolekce na jejich ID.

FORMÁT VÝSTUPNÍCH DAT

Formát bag of words jednotlivých dokumentů je následující:

jméno_souboru:mohutnost_BOWu term1:počet_výskytů_termu1 term2:...\n

Formát souboru obsahujícího jména BOW souborů je následující:

VELIKOST_KOLEKCE prirozene_cislo \n
cesta_k_BOW_souboru1 \n
cesta_k_BOW_souboru2 \n ...

Cesty k BOW souborům jsou relativní vůči místu spuštění aplikace.

Formát ID souboru:

id1 jméno_dokumentu1\n
id2 jméno_dokumentu2\n...

IMPLEMENTACE

Aplikace obsah každého dokumentu kolekce uloží do kontejneru typu *hash_map < char *, unsigned int>* (vlastní implementace, klasický přístup), její obsah po načtení celého dokumentu vytiskne do BOW souboru dbaje na to, aby v žádném BOW souboru nebyl překročen maximální počet zapsaných BOWů (maximální počet BOWů zapsaných do jednoho BOW souboru je definován konstantou za překladu). Pokud je zadán i přepínač *id_file*, je před zapsáním BOWu do BOW souboru spočteno ID dokumentu, ID se do BOW souboru vytiskne namísto jména dokumentu a dvojice ID - jméno dokumentu je zapsána do *id_file*.

CREATE_VOC

VSTUPNÍ DATA

Jediným vstupem aplikace je soubor s cestami k BOW souborům vytvořený aplikací *create_bows*.

VÝSTUPNÍ DATA

Výstupem aplikace je soubor obsahující slovník celé zpracovávané kolekce.

FORMÁT VÝSTUPNÍCH DAT

Formát souboru, obsahujícího slovník, je následující:

VELIKOST_SLOVNIKU přirozené_číslo \n
VELIKOST_KOLEKCE přirozené_číslo \n
term1 TC_termu1 DC_termu1 \n
term2 TC_termu2 DC_termu2 \n
...

IMPLEMENTACE

Aplikace postupně prochází všechny BOW soubory a jejich obsah ukládá do kontejneru typu *hash_map*, který je podrobněji popsán v 3.4.2.2. Po přečtení všech BOW souborů je kompletní slovník uložený v *hash_mapě* vytištěn.

REDUCE_VOC

VSTUPNÍ DATA

Jediným vstupem aplikace je soubor obsahující slovník určený ke zredukování.

VÝSTUPNÍ DATA

Výstupem aplikace je soubor obsahující redukovaný slovník.

FORMÁT VÝSTUPNÍCH DAT

Formáty vstupního i výstupního souboru aplikace jsou totožné (viz výše).

IMPLEMENTACE

Implementace této komponenty jako jediná využívá šablonovaných kontejnerů knihovny STL. Důvodem pro tuto volbu byly nevelké časové úspory získané užitím vlastní implementace těchto kontejnerů. Aplikace, v závislosti na vstupních parametrech, vybere funkci, která redukci slovníku obstará. Její pomocí jsou z neredukovaného slovníku odstraněna ta slova, která nevyhovují uživatelem zadaným požadavkům. V případě, že uživatelem zadaná podmínka na redukovaný slovník obsahuje i velikost redukovaného slovníku, je vektor obsahující slova vyhovující podmínce setříděn dle veličiny, jejíž minimální hodnota byla uživatelem zadána (přepínače *-min_***), a nepotřebná část vektoru je odstraněna. Následně je veškerý obsah vektoru vypsán do výstupního souboru.

CREATE_VECS

VSTUPNÍ DATA

Vstupy aplikace tvoří soubor s cestami k BOW souborům zpracovávané kolekce a soubor se slovníkem, vůči kterému mají být vektory vytvořeny.

VÝSTUPNÍ DATA

Výstupem aplikace jsou VEC soubory obsahující vektory dokumentů kolekce a soubor obsahující cesty k VEC souborům spolu s informacemi o zpracovávané kolekci.

FORMÁT VÝSTUPNÍCH DAT

Formát VEC souboru je následující:

jméno_dokumentu1:počet_nenulových_složek_vektoru index1:hodnota1 index2:.... \n
jméno_dokumentu2:počet_nenulových_složek_vektoru index1:hodnota1 index2:.... \n

Formát souboru obsahující cesty k VEC souborům je následující:

USING_TF (nebo USING_TF_IDF) \n
VELIKOST_SLOVNIKU přirozené_číslo \n
VELIKOST_KOLEKCE přirozené_číslo \n
POCET_ZAZNAMU_VE_VEC_SOUBORECH přirozené_číslo \n
cesta_k_VEC_souboru1 \n
cesta_k_VEC_souboru2 \n...

IMPLEMENTACE

Prvním krokem běhu aplikace je načtení celého slovníku do kontejneru typu *hash_map* (vlastní implementace, klasický přístup). Do této *hash_mapy* je ke každému slovu také uloženo odpovídající TC, DC a pořadové číslo slova ve slovníku. Následně jsou tvořeny vektory jednotlivých dokumentů kolekce – pro každé přečtené slovo z BOWu dokumentu je zkontrolována jeho přítomnost ve slovníku (slovo mohlo být odstraněno při redukci slovníku), následně je do vektoru daného dokumentu přidána dvojice sestávající z pořadového čísla tohoto slova ve slovníku následovaného buď TF daného slova vzhledem k právě zpracovávanému dokumentu (při použití přepínače *-use_tf_only)* nebo hodnotou TF_IDF daného slova vzhledem k právě zpracovanému dokumentu a kolekci. Množina dvojic *index:hodnota*, tvořící vektor zpracovávaného dokumentu je uchovávána v kontejneru typu vektor dvojic – na konci tvorby vektoru každého dokumentu je vektor setříděn pomocí algoritmu QuickSort vzestupně dle pořadových čísel slov ve slovníku a následně zapsán do VEC souboru. Setříděním vektoru zajistíme jednodušší načítání vektorů v komponentách *create_sigs* a *similarity* a hlavně mnohem jednodušší operace s vytvořenými vektory v těchto komponentách (např. skalární součin v aplikaci *similarity* ap.)

CREATE_SIGS

VSTUPNÍ DATA
Vstupy aplikace tvoří soubor s cestami k VEC souborům zpracovávané kolekce.

VÝSTUPNÍ DATA
Výstupem aplikace jsou SIG soubory obsahující signatury dokumentů kolekce a soubor obsahující cesty k SIG souborům spolu s informacemi o zpracovávané kolekci.

FORMÁT VÝSTUPNÍCH DAT
Formát SIG souboru je následující:

signatura1:INFO_signatury_1 seznam_odpovídajících_dokumentů \n
signatura2:INFO_signatury_2 seznam_odpovídajících_dokumentů \n ...
(kde INFO_signatury_X je počet dokumentů kolekce sdílejících danou signaturu)

Formát souboru obsahující cesty k SIG souborům je následující:

VELIKOST_KOLEKCE přirozené_číslo \n
DELKA_SIGNATUR_V_BITECH přirozené_číslo \n
POCET_RUZNYCH_SIGNATUR přirozené_číslo \n
cesta_k_SIG_souboru1 \n
cesta_k_SIG_souboru2 \n ...

IMPLEMENTACE

Po naalokování a naplnění potřebného paměťového prostoru náhodnými čísly aplikace vytváří signatury. V rámci minimalizace paměťových nároků aplikace je vždy načten pouze jeden vektor, pomocí LSH algoritmu je vytvořena příslušná signatura, která je okamžitě zapsána do tzv. *temp_sig* souboru, tj. souboru, který obsahuje dvojice *signatura:odpovídající_dokument*, kde ale soubor vzniklý konkatenací všech *temp_sig_souborů* (po skončení konverze všech vektorů) může obsahovat řádky s toutéž signaturou. Pro každý VEC soubor je vytvářen zvláštní *temp_sig soubor*. Po konverzi posledního vektoru jsou uvolněna data naalokovaná pro náhodné vektory a začíná načítání *temp_sig* souborů. Signatury jsou ukládány do *hash_mapy* (klasický přístup, vlastní implementace) mapující odpovídající dokumenty na signatury (chceme zjistit, jestli nějaké signatuře neodpovídá více než jeden dokument a pokud ano, pak si tuto skutečnost zapamatovat), *temp_sig* soubory jsou po svém přečtení ihned smazány. Při úspěšném načtení všech signatur do *hash_mapy* začíná výpis mapy do SIG souborů.

Pokud je aplikace volána s přepínačem *-use_hdd*, jsou nejdříve do tzv. *RND_VEC* souborů na disku nagenerovány jednotlivé náhodné vektory, počet vektorů uložených do jednoho souboru, potažmo počet těchto souborů je určen konstantou, která následuje za přepínačem *-use_hdd*. Po nagenerování náhodných vektorů je do paměti načten první *RND_VEC* soubor a pomocí vektorů obsažených v něm jsou vytvořeny první části signatur všech dokumentů, které jsou následně uloženy do dočasných souborů (tzv. *Partial_Temp_Sig* soubory). Následně je načten druhý *RND_VEC* soubor, pomocí vektorů v něm uložených jsou spočteny druhé části signatur a tyto jsou připojeny k dříve nagenerovaným začátkům signatur. Tento proces je iterován přes všechny soubory obsahující náhodné vektory. Po skončení čtení *RND_VEC* souborů jsou tyto smazány a v dočasných (*Partial_Temp_Sig*) souborech dříve obsahujících začátky signatury jsou nyní uloženy

signatury celé. Formát těchto souborů je totožný s formátem *Temp_Sig* souborů vytvořených po nagenerování všech signatur při volání aplikace bez přepínače *-use_hdd*. Zpracování *Partial_Temp_Sig* souborů a další výpočet je tedy od této chvíle totožný s tím uvedeným v odstavci výše.

SIMILARITY

VSTUPNÍ DATA

Vstupy aplikace tvoří soubor s cestami k VEC souborům zpracovávané kolekce.

VÝSTUPNÍ DATA

Výstupem aplikace je soubor obsahující jména dvojic dokumentů, které svou podobností převyšují uživatelem zadanou prahovou hodnotu.

FORMÁT VÝSTUPNÍCH DAT

Formát výsledku výpočtu je následující:

dokument1:podobny_dokument1:podobnost11:podobny_dokument2:podobnost12.. \n
dokument2:podobny_dokument1:podobnost21:podobny_dokument2:podobnost22.. \n
...

neboli každý řádek začíná jménem dokumentu doc_i následovaným seznamem dokumentů dokumentu doc_i podobných.

IMPLEMENTACE

Aplikace načte obsah všech VEC souborů do paměti do kontejneru typu vektor řídkých vektorů. Poté se pomocí dvou indexů, nazvěme je A a B, prohledává prostor dvojic řídkých vektorů. Index A je na počátku nastaven na nulu a ukazuje na první řídký vektor, index B je nastaven vždy na hodnotu indexu A a následně stoupá, vždy po spočtení kosinu řídkých vektorů ležících ve vektoru řídkých vektorů na pozicích A a B. Jestliže hodnota kosinu této dvojice přesáhne prahovou hodnotu, je tato informace připsána do souboru *out* a zároveň jsou indexy A a B přidány do datové struktury sdružující indexy (strukturu nazvěme Výsledek) do vektoru řídkých vektorů na dvojice, jejichž podobnost převyšuje prahovou hodnotu. Tuto informaci je nutno uložit z toho důvodu, že pokud počítáme podobnost doc_i s doc_j , je jisté, že podobnost doc_j vůči doc_i nikdy počítat nebudeme, neboť bychom za cenu zdvojnásobení doby běhu aplikace přicházeli dvakrát na tytéž informace. Při vypisování dokumentů podobných doc_j ale doc_i vypsat musíme – proto je nutné si indexy vektorů podobných dokumentů spolu s vypočtenou pravděpodobností ukládat a datovou strukturu, kam jsou tato data uložena, při vypisování výsledného souboru procházet.

HAMMING

VSTUPNÍ DATA

Vstupy aplikace tvoří soubor s cestami k SIG souborům zpracovávané kolekce.

VÝSTUPNÍ DATA

Výstupem aplikace je soubor obsahující jména dvojic dokumentů, které jsou podobností převyšují uživatelem zadanou prahovou hodnotu.

FORMÁT VÝSTUPNÍCH DAT

Formát výsledku výpočtu je následující:

dokument1:podobny_dokument1:podobnost11:podobny_dokument2:podobnost12... \n
dokument2:podobny_dokument1:podobnost21:podobny_dokument2:podobnost22... \n
...

neboli každý řádek začíná jménem dokumentu doc_i následovaným seznamem dokumentů dokumentu doc_i podobných. V případě, že má být do výsledku vypsáno jméno dokumentu, který sdílí signaturu s jinými dokumenty, jsou do výsledku zapsána jména všech těchto dokumentů jako celek ve tvaru $doc_1(doc_2\ doc_3\ ...doc_N)$.

IMPLEMENTACE

Aplikace načte obsah všech SIG souborů do paměti do vektoru signatur. Poté pomocí algoritmu PLEB hledá dvojice signatur, jejichž hammingova podobnost vyhovuje uživatelem zadané prahové hodnotě podobnosti. Pokud je taková dvojice nalezena, jsou extrahovány adresy obou signatur a tyto jsou spolu s podobností uloženy do *hash_mapy* obsahující výsledek podobně jako v aplikaci *similarity*. S pomocí této mapy je na konci vyhledávání vypsán výsledek (na rozdíl od aplikace *similarity* je v *hamming* celý výsledek vypisován až na konci běhu).

Zajímavými rysy implementace této aplikace jsou

1/ Autoři článku [1] při vyhledávání podobných signatur v rámci algoritmu PLEB postupují tak, že pro každou signaturu (označme ji sig_1) zkoumají jistý počet (*beam)* signatur ležících ve vektoru signatur „nad“ a „pod“ signaturou sig_1. Předpokládejme, že při posuzování signatur, které leží „pod“ signaturou sig_1 nalezneme signatur sig_2, přičemž vzájemná podobnost obou signatur bude postačovat pro to, abychom tyto prohlásili za blízké duplicity. Následně, až bude prozkoumáváno okolí signatury sig_2 a to konkrétně signatury ležící „nad“ signaturou sig_2, bude dříve nalezený pár objeven znovu. Autoři článku tedy zbytečně prohledávají okolí každé signatury na obou stranách, přičemž ke stejnému výsledku vede prohledávání pouze horní či dolní poloviny okolí každé signatury. Naše implementace využívá prohledávání pouze jedné poloviny okolí, což s sebou přináší téměř dvojnásobnou rychlost aplikace `hamming`.

2/ funkce zjišťující hammingovu vzdálenost signatur je implementována následovně. Na začátku běhu aplikace je inicializováno pole *Pocet_Jednicek_Bin_Zapisu,* v němž je na *i*-té pozici uložen počet jedniček binárního zápisu čísla *i*. Počet složek tohoto pole je 2^{16}, neboli dokáže uchovávat počty jedniček binárního zápisu pro všechna šestnáctibitová čísla. Při výpočtu hammingovy vzdálenosti dvou signatur uložených ve struktuře T*Signatura* (viz 3.4.2.1) jsou brány jednotlivé šestnáctibitové složky obou *unsigned long* polí, na ně je aplikována operace XOR a výsledné číslo je použito jako index do pole *Pocet_Jednicek_Bin_Zapisu.* Hodnota na tomto indexu je přičtena k výsledné hammingově vzdálenosti, která je na počátku výpočtu inicializována hodnotou nula, a tímto postupem se iteruje přes všechny šestnáctibitové složky datových polí obou signatur.

HAMMING_FULL

Všechny vlastnosti aplikace *hamming_full* jsou totožné s vlastnostmi aplikace *hamming*, v této aplikaci se ale nevyužívá algoritmus PLEB, vypisování a princip tvorby výsledku je totožný s tím v aplikaci *similarity*.

COMPARE_RESULTS

VSTUPNÍ DATA

Vstupy aplikace tvoří soubor s výsledkem výpočtu aplikace *similarity*, soubor s výsledkem výpočtu aplikace *hamming* (či *hamming_full)* (soubory vytvořené aplikacemi *similarity* a *hamming*, resp. *hamming_full* by měly být vytvořeny s použitím stejného *sample_listu*), při použití přepínače *-id_file* v aplikaci *create_bows* tvoří poslední část vstupu aplikace *compare_results* i *id_file*, tedy soubor mapující ID dokumentů na jejich jména.

VÝSTUPNÍ DATA

Výstupem aplikace je soubor obsahující přehledné porovnání podobností vytvořených aplikacemi *similarity* a *hamming*, resp. *hamming_full* pro všechny dvojice dokumentů nalezených ve vstupních souborech.

FORMÁT VÝSTUPNÍCH DAT

Formát výsledku výpočtu je následující:

Results for X:
Y *Sim_Podobnost_Y_Vuci_XHam_Podobnost_Y_Vuci_X \n*
Z Sim_Podobnost_Z_Vuci_X Ham_Podobnost_Z_Vuci_X \n ...
(kde *X* je jméno dokumentu ze *sample_listu*, vůči němuž aplikace *similarity* a *hamming,* resp. *hamming_full* tvořily svůj výsledek)

IMPLEMENTACE

Aplikace pracuje blokově. V každém bloku načte jednu řádku souboru obsahujícího výsledek aplikace *similarity*, tedy seznam jmen dokumentů podobných jednomu dokumentu obsaženém v *sample_listu*, se kterým bylo *similarity* voláno. Tuto řádku uloží aplikace do *hash_mapy* (vlastní implementace, principielně shodná s tou v 3.4.2.2) mapující hammingovu podobnost a *similarity* podobnost na *char* ***.** Po uložení této řádky aplikace načte řádku souboru obsahujícího výsledek aplikace *hamming*, resp. *hamming_full* a tuto ukládá do též *hash_mapy*, do které se ukládala data z výsledku aplikace *similarity*. Pokud je po načtení řádky souboru vytvořeného aplikací *hamming*, resp. *hamming_full* zjištěno, že se oba řádky vztahují k jinému vztaženému souboru (ze *sample_list* obou aplikací), je výpočet ukončen, neboť vstupní soubory aplikací se netýkají stejného *sample_listu*. V opačném případě je pro každé jméno dokumentu kolekce po zpracování řádků obou vstupních souborů v *hash_mapě* obsažena hammingova a kosinová podobnost (či speciální hodnoty, kterými mapu inicializujeme pro indikaci toho, že některá z podobností nebyla pro daný dokument nalezena – typicky v případě, kdy v aplikaci *similarity* pár dokumentů odpovídá požadavku treshold zatímco v *hamming*, resp. *hamming_full* nikoliv, či naopak). Následuje vytištění výsledku ve formátu uvedeném výše, vyprázdnění *hash_mapy* a čtení následujícího řádku souboru vytvořeného aplikací *similarity*.

Příloha B

Obsah přiloženého CD

K práci přiložené CD obsahuje:

- adresář ./src, v němž jsou uloženy soubory se zdrojovými kódy aplikací
- unixový skript ./install.sh, který vytvoří adresář ./bin, zkompiluje všechny aplikace a přemístí vytvořené binárky do vytvořeného adresáře. Dále vytvoří adresář ./exp pro uložení dat vytvořených v průběhu výpočtu (vektory dokumentů, BOWy, signatury) a adresář ./res určený pro uložení výsledku aplikací *hamming*, *hamming_full*, *similarity* a *comapare_results*.
- unixové skripty ./bows_through_vecs_no_red.sh a ./bows_through_vecs_red.sh. Oba mají právě jeden argument, kterým je jméno souboru obsahujícího cesty ke všem dokumentům, které mají být zpracovány. Cesty k těmto souborům musejí být odděleny bílým znakem. Při použití prvního skriptu k redukci slovníku nedochází, při použití druhého dochází k redukci slovníku o termy, které jsou přitomné v právě jednom dokumentu v rámci celé zpracovávané kolekce. Při použití obou skriptů jsou vektory dokumentů vytvářeny pomocí veličiny *TF*.
- unixový skript ./sigs.sh, který pro zpracovávané dokumenty vytvoří signatury délky 1024
- unixové skripty ./hamm_through_cmp_full.sh a ./hamm_through_cmp_pleb.sh. Oba mají povinný právě jeden argument, kterým je jméno souboru obsahujícího cesty k dokumentům zpracovávané kolekce, vůči kterým bude počítána podobnost s ostatními dokumenty (viz. přepínač *sample_list* u aplikací *hamming*, *hamming_full* a *similarity*). Oba skripty provádí výpočet podobností dokumentů vůči danému *sample_listu*, první k tomu používá aplikace *hamming_full*, druhý aplikaci *hamming*, oba dále počítají podobnosti dokumentů pomocí aplikace *similarity* a vytvořené výsledky jsou zpracovány aplikací *compare_results*. Hodnota *treshold_sim* je nastavena na 80, hodnota *treshold_ham* na 79, *iter_count* je 500, *shift_count* má hodnotu 80 a *beam* je nastaven na hodnotu 50.

Bibliografie

[1] Ravichandran, D., Pantel,P., Hovy, E. (2005): Randomized Algorithms and NLP: Using LocalitySensitive Hash Functions for High Speed Noun Clustering, *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics, strany 622 – 629*

[2] Ricardo A. Baeza-Yates,Berthier Ribeiro-Neto (1999): Modern Information Retrieval, Addison-Wesley Longman Publishing Co., Inc.