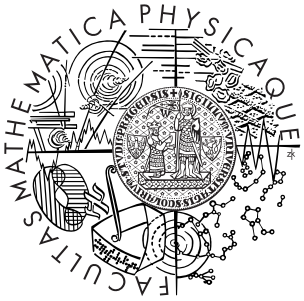
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

# BAKALÁŘSKÁ PRÁCE




Lenka Smejkalová

## Vizualizace podobnosti dokumentů

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Pavel Pecina
Studijní program: Informatika - programování

2007

Chtěla bych poděkovat svému vedoucímu Pavlu Pecinovi za cenné rady, poskytnutou literaturu a testovací data.

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

V Praze dne 8. srpna 2007 Lenka Smejkalová

# Obsah

Název práce: Vizualizace podobnosti dokumentů
Autor: Lenka Smejkalová
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: Mgr. Pavel Pecina
e-mail vedoucího: pecina@ufal.mff.cuni.cz

Abstrakt: V této práci se zabýváme vizuálním porovnáváním textů, především hledáním podobností v textových dokumentech pomocí techniky dotplotu. Pro tento způsob porovnávání byl implementován algoritmus a vytvořena uživatelská aplikace, se kterou je možno pracovat interaktivně. Pro zpracování více textů najednou byla vytvořena i command-line verze. Ovládání obou verzí je popsáno v uživatelské příručce v Příloze A. Dále se v práci zabýváme tím, jaké vzorky vznikají v dotplotu a co vypovídají o původním textu. Je možné hledat podobnosti v rámci jednoho souboru stejně dobře jako porovnávat dva různé soubory.

Klíčová slova: dotplot, podobnost, matice, vizuální, porovnání

Title: Document similarity visualization
Author: Lenka Smejkalová
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Pavel Pecina
Supervisor's e-mail address: pecina@ufal.mff.cuni.cz

Abstract: In the present work we study visual comparision of texts, especially by finding similarity in text documents by dotplot technique. An algorithm for this type of comparison was implemented and graphic user interface was made to allow user to work interactively. Also, command-line version of the application was created to allow batch processing of multiple documents. Instructions to use these programs are written in user documentation which is added to this work in Appendix A. Further we study which patterns could grow up in dotplot and what these patterns predicate about documents. It is possible to find self-similarity in one documents or similarity in two different documents.

Keywords: dotplot, similarity, matrix, visual, comparision

# Kapitola 1

# Úvod

Úloha zkoumání podobnosti dokumentů se velmi často řeší automaticky v rámci velkých kolekcí s přísnými požadavky na co nejmenší časovou a paměťovou náročnost použitých metod. Jiným případem je ovšem situace, kdy uživatel potřebuje porovnat jedinou dvojici dokumentů a získat relativně detailní přehled o jejich podobnosti (případně podobnosti jejich částí), aniž by je musel číst a detailně studovat. V takovém případě by bylo ideální dokumenty a jejich podobnost zobrazit vizuálně.

Cílem bakalářské práce je implementovat aplikaci s grafickým rozhraním umožňující komfortní vizuální porovnávání dvojic dokumentů prostřednictvím různých náhledů, jako je např. dotplot nebo také diff.

Je mnoho nástrojů, které se zabývají porovnáváním textových dokumentů. Většinou tyto nástroje hledají drobné rozdíly mezi dvěma dokumenty nebo se zabývají vyhledáváním nejvíce podobných dokumentů z velkého množství textů. Aplikace Dotplot je přizpůsobena především na zobrazení podobnosti dvou dokumentů tak, aby uživatel tuto podobnost mohl zkoumat i do velkých detailů. Kromě maticového zobrazení (dotplot) nabízí také klasický UNIX diff.

Nástroj diff je starší než dotplot a byl implementován již mnohokrát. Cílem této práce není pokusit se o nové vylepšení, ale je zde uveden jen pro doplnění, a proto se budeme zabývat především technikou dotplotu.

## 1.1 Pro koho je aplikace určena

Aplikace je určena především lidem, kteří se nějakým způsobem zajímají o lingvistiku. Dále je určena také programátorům, kteří se chtějí na svůj

zdrojový kód podívat trošku jinak. Tímto pohledem mohou objevit i nějaké nedostatky, např. redundanci.

Základní funkcí projektu Dotplot je grafické zobrazení podobnosti dvou souborů nebo self-similarity jednoho souboru.

K aktuálnímu vygenerovanému obrázku jsou k dispozici také odpovídající části souborů. Dále aplikace nabízí grafický diff. Dotplot a diff jsou na sobě vzájemně nezávislé, lze je však vyvolat najednou. Kromě grafického rozhraní existuje také command-line verze Dotplotu. Pro některé uživatele může být nepohodlné, že musejí zadávat spoustu parametrů do příkazové řádky, ale jiní uživatele naopak ocení, že si mohou napsat vhodný script a zpracovat tak spoustu textů bez dalšího zásahu. To je obzvlášť výhodné u velmi dlouhých vstupních souborů (např. několik MB textu se zpracovává několik sekund).

Pohodlnější uživatelé, kteří se nechtějí zabývat zadáváním parametrů do příkazové řádky, mohou využít některých výchozích nastavení.

## 1.2 Reference na podobné projekty

Při hledání podobných softwarových projektů, které by se zabývaly stejnou problematikou, jsem nebyla příliš úspěšná. Je možné, že podobných aplikací neexistuje mnoho a nebo nejsou zveřejněny. Druhou možností je, že mé hledání nebylo důkladné.

Nalezla jsem projekt Dotplot (http://sourceforge.net/projects/dotplot), který je velmi podobný mé aplikaci. Má několik nedostatků, ale i kladů, které budou popsány v závěru v rámci srovnání obou aplikací.

Dále jsem nalezla program na porovnávání DNA sekvencí, který bohužel nebyl ke stažení. Tento projekt zřejmě najde své uplatnění v biologii při výzkumu genetiky, ale k porovnání textů asi nebude použitelný.

## 1.3 Možnosti aplikace

Aplikace Dotplot může být používána ve dvou provedeních - jako interaktivní grafická aplikace nebo command-line verze. Obě verze se lehce liší, protože na grafickou aplikaci jsou kladeny vyšší nároky na komunikaci s uživatelem. Jinak nabízejí stejné možnosti nastavení parametrů a vytvoření obrázků.

V aplikaci je možné s obrázkem ještě dále pracovat, podívat se na soubory, z kterých se dotplot počítá, ukládat obrázky v některém z podporo-

vaných formátů nebo výpočet zopakovat pro jiné soubory.

V command-line verzi je možné jen spustit program s parametry a pak si prohlédnout výsledek v souboru. Obrázky se ukládají pouze ve formátu PNG.

Popis ovládání můžete nalézt v Uživatelské příručce v Příloze A.

# Kapitola 2

# Teorie

## 2.1 Úvodní definice

**dotplot** – v použité literatuře je tímto termínem označována technika vytváření obrázku nebo interaktivní program, v tomto textu budeme výraz *dotplot* používat především pro hotový obrázek

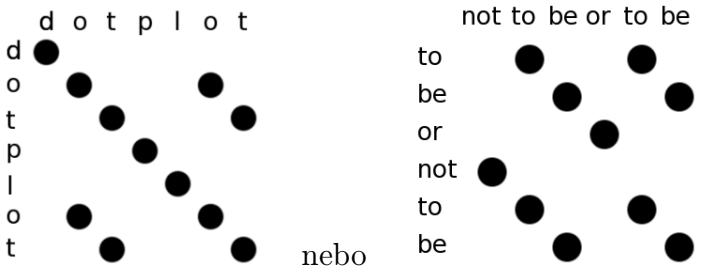
**token** – element ve vstupním textu (neprázdná posloupnost znaků), který již dále nechceme dělit, např. jedno slovo nebo jeden řádek

**typ (type)** – unikátní element ve vstupním textu, např. fráze *„to be or not to be“* obsahuje 6 tokenů, ale jen 4 typy, protože slova *to* a *be* se opakují

## 2.2 Princip vytvoření dotplotu

Ačkoliv hotové obrázky mohou vypadat i velmi komplexně, teorie, která za vším stojí, je velmi jednoduchá. Představme si, že máme čtverečkovaný papír a dvě posloupnosti písmenek nebo slov. První posloupnost zapíšeme na papír svisle shora dolů (osa y) a druhou posloupnost vodorovně zleva doprava (osa x) tak, aby obě posloupnosti tvořily levou a horní hranu obdélníku, popř. čtverce.

Poté projdeme tento obdélník resp. čtverec po řádcích a když narazíme na políčko, pro které se shoduje řádkové písmenko se sloupcovým, tak ho barevně označíme. Pro jednoduchost můžeme používat jen černou barvu. Výsledný obrázek může vypadat například takto:



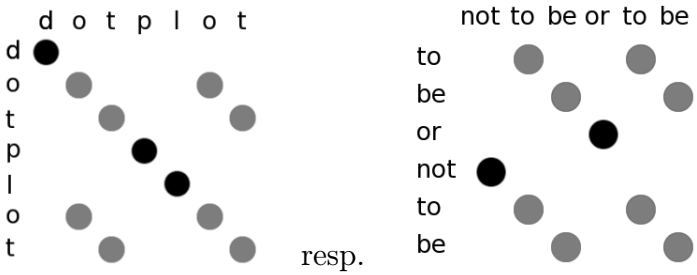

Obrázek 2.1: základní technika

Dále tuto metodu můžeme obohatit o rozlišování váhy shod. V jazyce máme mnoho slov, která se často opakují. V češtině jsou to například předložky, spojky, zájmena (ze, se, a, ale, ...), v jiných jazycích se zase hojně vyskytují členy (a, an, the, la, les, ...). Takové shody nás příliš nepřekvapují, protože jich bude mnoho v každém delším textu. Pak by taková matice podobnosti byla značně přesycená a zanikly by ostatní informace. Toto by mohlo nastat i v případě, že by oba články pojednávaly o úplně rozdílných tématech (nesouvisejících) a jediné shody by nastávaly právě v těchto slovech, bez kterých se v jazyce prakticky neobejdeme. Tomuto se chceme při porovnávání vyhnout. Řešením může být, že si spočítáme váhu $w$ řetězce $r$ jako funkci $w(r) = 1/freq(r)$, kde $freq(r)$ je počet výskytů daného řetězce $r$. Tuto hodnotu uložíme do matice (zapíšeme na čtverečkovaný papír).

V předchozím příkladu jsme rozlišovali pouze černou a bílou (nevybarvenou). Nyní bychom těmto dvěma barvám přiřadili hodnoty 1 (černá) a 0 (bílá). Získáme tak nekonečně mnoho hodnot mezi nulou a jedničkou. Těmto hodnotám bychom mohli přiřadit barvy v odstínech šedi.

Pokud si znovu projdeme předchozí příklad, tak v posloupnosti *„d, o, t, p, l, o, t“* máme dvakrát písmeno *o* a *t*, tedy jim přiřadíme váhu 0,5 a ostatní písmena budou mít váhu 1. Stejně tak ve frázi *„to be or not to be“* resp. *„not to be or to be“* se dvakrát vyskytují slova *to* a *be*. Graficky to bude vypadat tak, jak je uvedeno na obrázku 2.2.

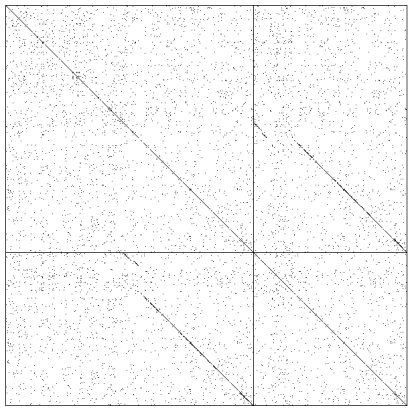
Nyní jsme popsali základní myšlenku vytváření dotplotu. Pokud si představíme, že bychom použili stejný princip např. na celou knihu, tak by to asi bylo velmi náročné na výpočet a výsledek by nebyl příliš přehledný. Pro větší vstupní data se používají další techniky, ke kterým se dostaneme později v kapitole Implementace. Výše popsaný princip nám prozatím postačuje pro pochopení interpretace vzorků, které mohou vznikat v dotplotu.




Obrázek 2.2: použití vyvažování

## 2.3 Historie - DNA řetězce

Původně se dotplot používal v biologii ke studování podobností v genetických sekvencích. Toto porovnávání by mohlo být realizováno i za pomoci jiných nástrojů (např. UNIX diff), ale pro člověka je příjemnější grafický výstup, kde na první pohled rozezná diagonálu, než složité pročítání textových výstupů.



Obrázek 2.3: Dvě DNA sekvence

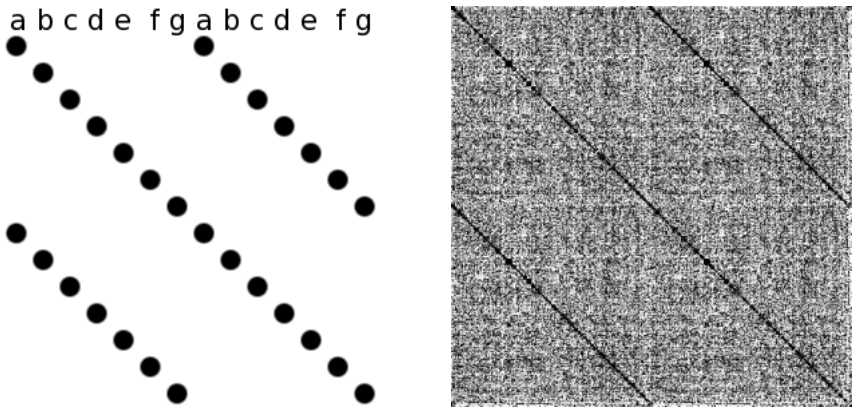
Na obrázku 2.3 vidíme dvě DNA sekvence (jsou odděleny svislou resp. vodorovnou čarou). Z toho můžeme poznat, že druhá je kratší a obsahuje velkou část první sekvence, do níž byla vložena subsekvence, která se v prvním řetězci nevyskytuje, což je znázorněno přerušením diagonály. Interpretaci vzorků v dotplotu bude věnována následující podkapitola.

## 2.4 Vzorky vznikající v dotplotu a jejich interpretace

Abychom si mohli udělat představu o původním textu jen ze znalosti obrázku, musíme rozumět vizuálnímu jazyku dotplotu. S jeho pomocí pak budeme schopni lépe analyzovat dokumenty.

Dalo by se říci, že čtverce v dotplotu znamenají velkou hustotu neseřazených shod tokenů, zatímco diagonály označují seřazené shody. Tmavší oblast vždy znamená větší počet shod tokenů. Existuje spousta způsobů, jak tyto úkazy mohou být v dotplotu kombinovány, a proto se jimi budeme v následujícím textu zabývat podrobněji. Vždy uvedeme schematický obrázek a vedle něj dotplot vytvořený z reálných dat.

**Diagonály**



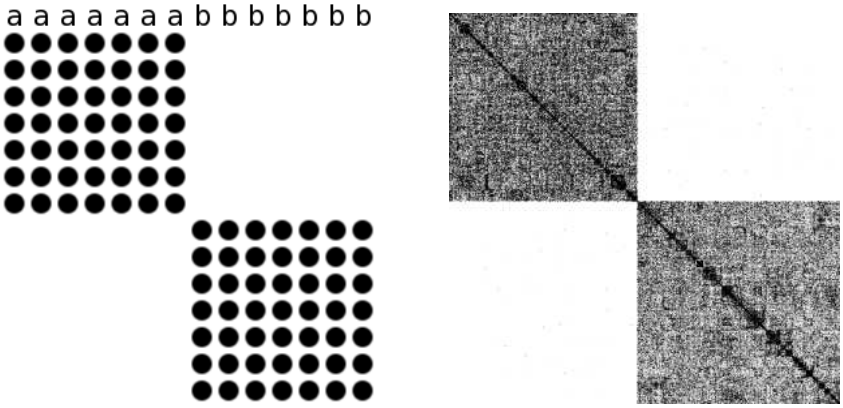

Obrázek 2.4: Diagonály a) schéma b) příklad

Pokud v obrázku zpozorujeme jen jednu diagonálu, znamená to, že se jedná o posloupnost neopakujících se tokenů, stejnou v obou dokumentech. V případě dvou (či více) diagonál, vypovídá obrázek o tom, že se posloupnosti tokenů opakují.

Obecně můžeme říci, že diagonály poukazují na uspořádanou sekvenci shod tokenů ve vstupním textu.

**Čtverce (Squares)**

Čtverce znamenají vyšší hustotu shod tokenů. Například pokud máme dva texty v různých jazycích (dokumenty jsou napojeny za sebe), tak tokeny




Obrázek 2.5: Čtverce a) schéma b) Dva nesouvisející články

se mezi sebou budou shodovat jen v rámci jednoho jazyka. Samozřejmě existují i slova, která jsou shodná ve více jazycích, ale pokud předpokládáme, že se nejedná o příbuzné jazyky (např. čeština a slovenština) a máme dostatečně dlouhý text, tak ostatní shody jsou zanedbatelné.

V reálném příkladu je zobrazen dotplot dvou textů napojených za sebe, které spolu vůbec nesouvisí. První z nich je článek pojednávající o Africe, druhý je známá povídka Proces od německého autora Franze Kafky v němčině, čímž je zajištěn opravdu velký kontrast mezi čtverci.

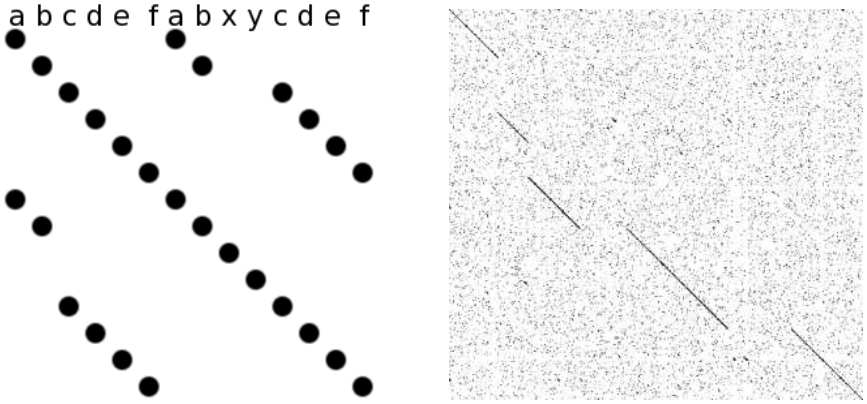
Obecně, jeden čtverec znamená velkou hustotu neuspořádaných shod tokenů, dva a více čtverců obvykle indikují, že celý text není jen v jednom jazyce nebo pojednává o velmi rozdílných tématech.

**Přerušené diagonály (broken diagonals)**

Toto znamená, že do posloupnosti byla vložena nová podposloupnost, která s původní nemá nic společného. Na schematickém obrázku takto byla vložena posloupnost *xy* do posloupnosti *abcdef*. Tento úkaz se často objevuje ve zdrojových kódech, pokud např. porovnáváme starší verzi s novější, protože v nové verzi je většinou něco přidáno. Naopak pokud něco odmažeme (vypustíme), tak je diagonála také přerušena, ale v opačném směru - v dotplotu je vložena horizontální mezera, zatímco při vložení nové podposloupnosti je vložena mezera vertikální.

V příkladu je uveden dotplot vytvořený z vygenerovaného textu Lorem Ipsum[1] , kde na vertikální ose je původní text, zatímco na horizontální je

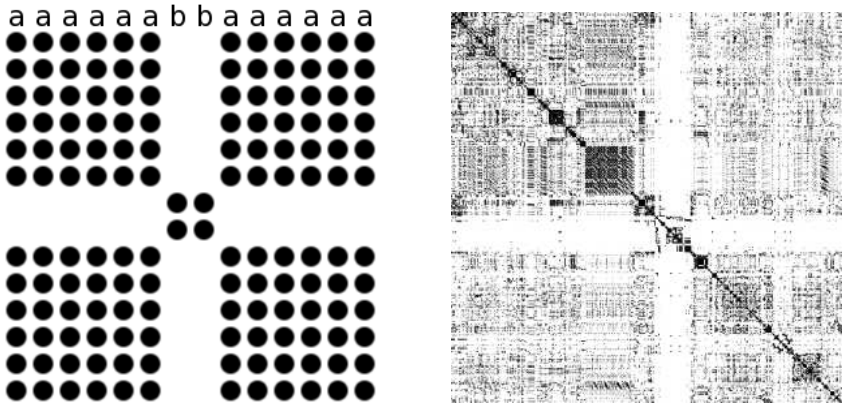
[1]Lorem Ipsum je vygenerovaný pseudolatinský text, který se většinou používá jako




Obrázek 2.6: Přerušené diagonály a) schéma b) vypuštění a vložení

upravený text tak, že ze začátku bylo několik odstavců vypuštěno a ke konci dokumentu byly naopak přidány nové odstavce. Těmito dvěma akcemi se vytvořily jak horizontální, tak vertikální mezery.

**Světlý kříž (Light cross)**




Obrázek 2.7: Světlý kříž a) schéma b) zdrojové kódy

Světlý kříž v dotplotu vznikne v případě, že do homogenní posloupnosti[2] , vložíme jinou (homogenní) posloupnost. V obecných dokumentech můžeme

výplňový text při návrhů grafického rozvržení HTML stránek, časopisů atp.

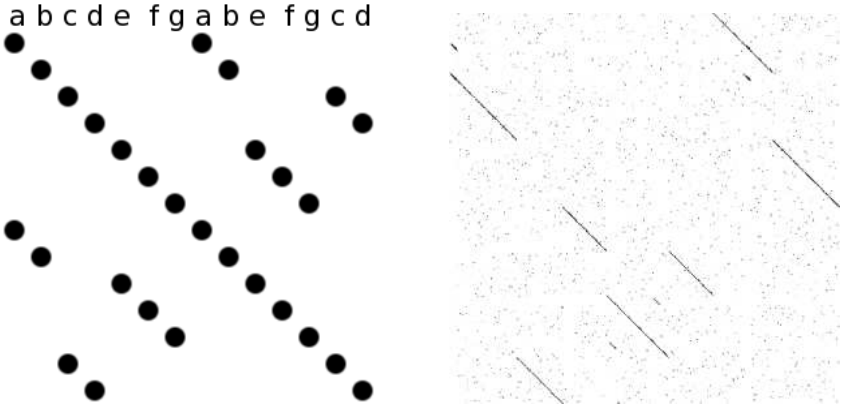
[2]homogenní posloupností rozumíme posloupnost, která má všechny prvky stejné

jako příklad uvést opět text Lorem Ipsum, do kterého bude vložen např. český text nebo zdrojový kód, obecně něco, co nijak více nesouvisí s majoritní částí dokumentu (v našem případě s Lorem Ipsum). Obecně to může být text v jednom jazyce, do kterého vložíme výrazně kratší text v jiném jazyce (mělo by se jednat o jazyky, které si nejsou příliš příbuzné).

V našem příkladu je zobrazen dotplot zdrojových kódů, kde světlý kříž je tvořen souborem, který nebyl napsán přímo programátorem, ale byl vytvořen pomocí *designeru* při návrhu grafického prostředí nebo nástrojem *moc*[3].

Obecně můžeme říci, že světlé kříže indikují vkládání do neuspořádaných sekvencí a přerušené diagonály znamenají vložení do uspořádaných sekvencí.

**Přeuspořádané diagonály (Reordered diagonals)**




Obrázek 2.8: Přeuspořádané diagonály a) schéma b) Lorem Ipsum - změněné pořadí odstavců

Tento vzorek vznikne, pokud máme na jedné ose původní posloupnost a na druhé tuto posloupnost přeuspořádanou. Např. pokud budeme mít dvě studentské práce, přičemž jeden student práci opsal od druhého tak, že pouze změnil pořadí vět či odstavců.

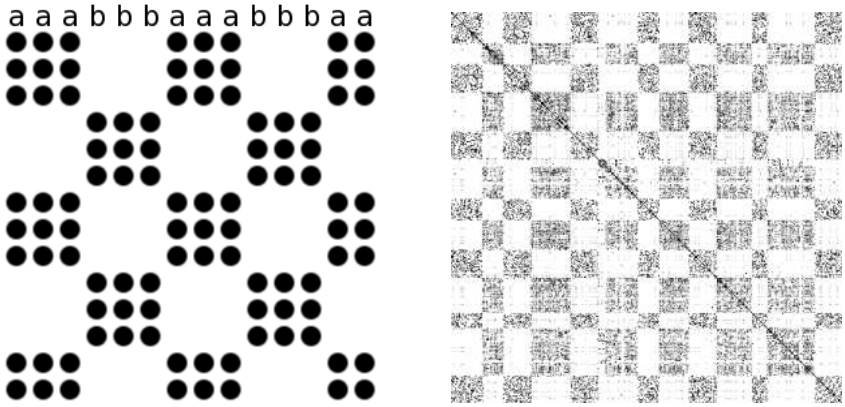
Tento vzorek se může objevit, když spojíme několik dokumentů za sebe do jednoho velkého souboru. Ve druhém souboru budou ty samé dokumenty v jiném pořadí. Neměly by mít příliš velkou hustotu shod tokenů, jinak by

[3]moc = Meta Object Compiler, nástroj generující meta informace o třídách použitých v programu. Díky těmto informacím Qt umožňuje použití programátorských mechanismů jako např. systém slotů a signálů, které nejsou podporovány v C++

to byl následující případ přeuspořádaných čtverců. Velmi názorně je to vidět v textu, kde se slova neopakují příliš často (způsobeno to je také tím, že není dlouhý).

Ideálním příkladem je vygenerovaný text pomocí Lorem ipsum. Pro první dokument bylo vygenerováno 7 odstavců, ve druhém dokumentu byl stejný text, ale odstavce byly náhodně přeuspořádáné.

**Přeuspořádané čtverce (Reordered squares)**



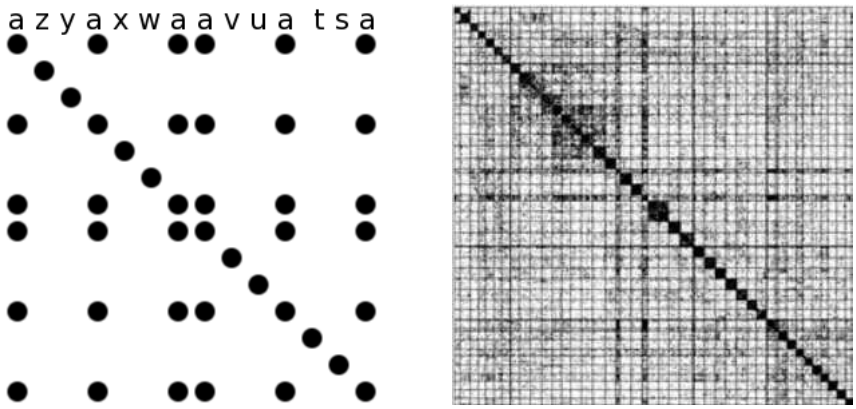

Obrázek 2.9: Přeuspořádané čtverce a) schéma b) Lorem Ipsum & Qt manuál

Dotplot, ve kterém se střídají tmavé plochy se světlými, vypovídá o tom, že máme dva odlišné texty, které jsou do sebe navzájem propleteny. Tyto texty by samy o sobě tvořili oblast s velkou hustotou. Schematicky to můžeme znázornit pomocí posloupnosti znaků $a$ a znaků $b$, přičemž se tyto posloupnosti pravidelně střídají.

V reálném příkladu (obrázek 2.9 b) byl použit opět vygenerovaný text Lorem Ipsum, který byl prokládán odstavci z dokumentace Qt. Tyto dva dokumenty nemají rozhodně nic společného, a proto dochází k tak názornému střídání světlých a tmavých oblastí.

**Tmavý kříž (Dark cross)**

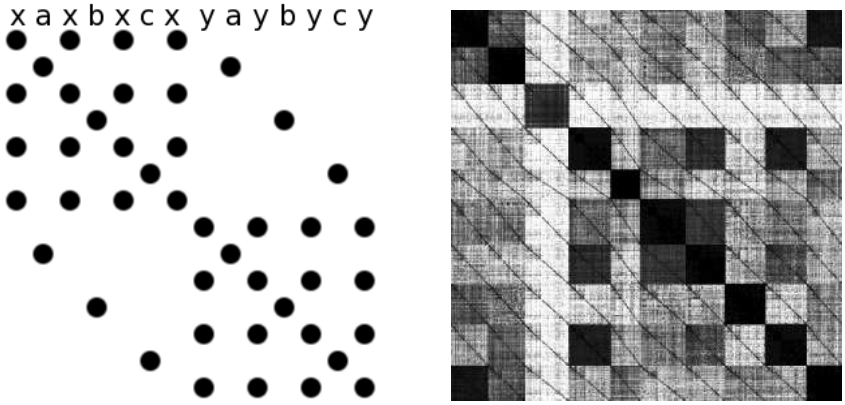
Tmavý kříž vznikne v dotplotu pokud posloupnost obsahuje podposloupnost, jejíž prvky jsou obsaženy i ve zbytku posloupnosti. V obecném textu vznikne v případě, kdy část textu je natolik obecná, že se tokeny často opakují i ve zbytku dokumentu. Takovým pěkným příkladem je soubor všech Shakespearových dramat poskládaných za sebou (obrázek 2.10 b).




Obrázek 2.10: Tmavý kříž a) schéma b) Shakespearova dramata

Poblíž středu doplotu je naznačen tmavý kříž. Konkrétně se jedná o dramata *Znásilnění Lukrecie (The Rape of Lucrece)* a *Venuše a Adonis (Venus and Adonis)*, která obsahují spoustu obecných slov vyskytujících se i v ostatních dramatech, jako např. *doth, eyes, love, life, death*, atd.

**Shuffle**



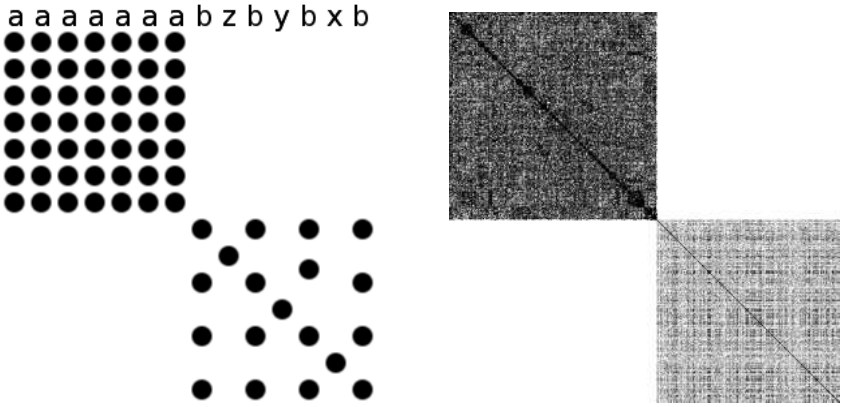

Obrázek 2.11: Shuffle a) schéma b) zápisky z parlamentu

Shuffling (zde zůstáváme u původního anglického označení, jak je uvedeno např. v [2]) je v podstatě zobecnění akcí vkládání a přeuspořádání. Ve schematickém obrázku vkládáme posloupnost *abc* do posloupnosti složené pouze ze znaků $x$ a následně do posloupnosti znaků $y$.

Velmi zajímavým příkladem (obrázek 2.11 b) jsou multijazyčné zápisky z parlamentu. V naší ukázce se jedná o zápisky z jednoho dne v jedenácti evropských jazycích (po řadě: angličtina, dánština, němčina, řečtina, španělština, finština, francouzština, italština, holandština, portugalština a švédština). Na hlavní diagonále vznikají tmavé čtverce, protože slova se v rámci jednoho jazyka opakují. Dále si můžeme všimnout výrazné diagonální struktury, která je způsobena tím, že některé tokeny se shodují ve všech jazycích, například vlastní jména, zeměpisné názvy, číselné údaje atd. Také tomu přispívá fakt, že dokumenty mají určitou strukturu - vyskytují se v nich různé značky podobné HTML[4] tagům, např. pro označení nového odstavce, pro zaznamenání jména mluvčího atd.

Dalším zajímavým úkazem je proměnlivá světlost čtverců mimo hlavní diagonálu. Řečtina (čtvrtý řádek/sloupec) vytváří v dotplotu tzv. světlý kříž (viz dříve na str. 14), což znamená, že se velmi odlišuje od ostatních evropských jazyků, zatímco dvojice jazyků španělština - portugalština nebo dánština - švédština vytvářejí velmi tmavé čtverce mimo diagonálu. To dokazuje, že jsou si tyto jazyky velmi blízké a některá slova jsou v nich shodná.

**Proměnlivá hustota (Density variation)**




Obrázek 2.12: Proměnlivá hustota

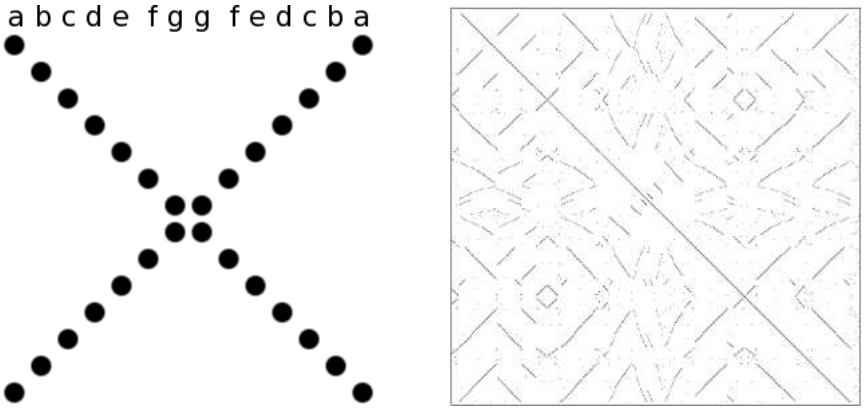
K poklesu hustoty v dotplotu dochází, když posloupnost proložíme jinou posloupností. Dobře je to vidět také u příkladu v předchozím odstavci (obrázek 2.11 b), kde čtverce znázorňující shody napříč jazyky nejsou tak

[4]HyperText Markup Language je značkovací jazyk pro hypertext, používá se např. pro vytváření webových stránek

tmavé, protože je v nich menší počet shod tokenů. Na obrázku (obrázek 2.12) jsou vykresleny světlejší šedou barvou.

Jako další příklad můžeme uvést opět uměle vytvořený text. Tentokrát jsme použili článek o Africe a vygenerovaný Lorem Ipsum text (oba dokumenty jsme spojili za sebe). Český text si je soběpodobný více než pseudolatinský text, což se v dotplotu projeví tak, že horní levý čtverec je tmavší než pravý dolní. Jelikož tyto dva jazyky si nejsou nijak podobné, tak nás jistě nepřekvapuje, že zbytek dotplotu zůstává prázdný (bílý).

**Palindromy**




Obrázek 2.13: Palindrom a) schéma b) 900 jmen souborů

Palindrom je zvláštní posloupnost, která se čte stejně odpředu i pozpátku. Vznikne například tak, že vezmeme obyčejnou posloupnost, obrátíme ji a tu pak připojíme za původní. Takováto posloupnost v dotplotu vytvoří hlavní a vedlejší diagonálu.

Přestože palindromů existuje velké množství, většinou se však jedná jen o krátké věty (např. *„Jelenovi pivo nelej“*) a ve větších dokumentech bychom takový jev zřejmě těžko hledali. V našem příkladu (obrázek 2.13 b) použijeme dotplot který vznikl z 900 jmen souborů. Jména se objevila nejprve v jednom pořadí a později se objevila také v opačném pořadí. Tyto soubory byly generovány nástrojem *make*, který nejprve dealokoval všechny i-nodes (make clean) a pak je znovu alokoval. Tento nástroj zřejmě používá k ukládání i-nodes zásobník[5], což vysvětluje zpětné diagonály, protože jsou alokovány v opačném pořadí než byly dealokovány.

[5]datová struktura Last In First Out - LIFO

# Kapitola 3

# Implementace

Grafická aplikace se od command-line verze liší především v ovládání. Následující část je shodná pro obě verze, takže o rozdílech mezi verzemi se zmíníme až v závěru kapitoly.

Vlastní výpočet dotplotu se skládá ze tří operací. Vstupem je prostý text, ze kterého musíme vyrobit tokeny (tokenizace), pak vypočteme f-image, což je pouze interní reprezentace dotplotu a nakonec je potřeba tento f-image zobrazit uživateli, tedy vyrobit tzv. q-image, který už může být zobrazen na výstupním zařízení.

Schematicky znázorněno takto: text -> tokens -> f-image -> q-image

## 3.1 Datové struktury

Hlavní datové struktury byly vytvořeny celkem tři. Nyní si je postupně popíšeme.

1. Struktura **TTypes** je realizována pomocí standardní šablony std::map, klíčem je načtený token typu string (v aplikaci QString) a hodnotou je číslo, které je přiřazováno inkrementálně.

2. Struktura **TTokens** je také realizována pomocí std::map. Zde je klíčem hodnota z TTypes a hodnotou je vektor výskytů.

3. **TMatrix** je třída, která obsahuje ukazatel na dvourozměrné pole typu double, které reprezentuje vlastní f-image. Dále obsahuje výšku, šířku a položky, které nyní nejsou důležité pro vytvoření f-image (např. cpp či pix).

## 3.2 Tokenizace (Tokenization)

Tokenizace (angl. tokenization) znamená rozdělení vstupního textu na slova, věty, řádky nebo jinak definované úseky znaků (obecně tokeny). Program toto dělení provádí tak, že mu zadáme, jaké znaky má chápat jako oddělovač. Ostatní znaky načte jako součást tokenu. Pokud jsou dva oddělovací znaky vedle sebe a tokenem by bylo prázdné slovo, což není přípustné, tak prázdný token jednoduše vynecháme.

V aplikaci jsou přednastaveny čtyři skupiny oddělovačů a je zde možnost definovat si oddělovače vlastní. První skupinou jsou všechny bílé znaky, tedy mezera, tabulátor a nový řádek. V běžném textu by tedy jedno slovo odpovídalo jednomu tokenu. Je tu problém, že v textech se vyskytují také interpunkční znaménka, jako tečky, čárky, otazníky atd. a před těmito znaky se většinou nedělá mezera. V důsledku to znamená, že se při načítání přidají k nějakému slovu a s ním pak tvoří token. Na jiném místě v textu se toto slovo vyskytne samotné a pro aplikaci to pak znamená zcela jiný token.

Dalším typem oddělovače je nový řádek. Toho může být využito například při zkoumání zdrojových kódů.

Třetí skupinou oddělovačů jsou speciální znaky, které jsou dány výčtem (patří mezi ně i bílé znaky):

```
.,:?!;"\/-+='{}()[]<>|*&^%@~‘$#
```

Měly by tak pokrýt všechny možné znaky, které by se mohly vyskytnout v běžném textu a způsobit tak nechtěný efekt popsaný výše, kdy jsou oddělovačem pouze bílé znaky. Mohli bychom pak dosáhnout toho, že tokeny budou tvořit právě slova.

Čtvrtá skupina oddělovačů je zadána opačným způsobem - je zadán výčet znaků, které oddělovačem nejsou. Jsou to velká a malá písmena anglické abecedy, číslice 0 – 9 a znak _ (podtržítko). Toto jsou znaky, které se mohou vyskytovat v názvech proměnných (tam většinou platí omezení, že musí začínat písmenem), takže tokeny jsou pak proměnné a číselné konstanty (pozor - desetinná tečka je chápána jako oddělovač!). Tento způsob by tedy mohl být použit například pro zkoumání výskytů názvů proměnných ve zdrojových kódech, případně jako obměna předchozího způsobu (oddělovači jsou speciální znaky) ale pouze pro anglické texty.

Po rozpoznání tokenu ho uložíme do struktury TTypes, kde uchováváme jen unikátní řetězce. V grafické verzi používáme QString, v command-line verzi bez Qt std::string. Pokud už je ve struktuře TTypes obsažen, pouze

uložíme jeho pozici ve vstupním textu do struktury TTokens. V případě, že porovnáváme dva různé soubory, máme také dvě instance TTokens. Instance TTypes existuje vždy jen jedna.

Jinak je způsob načítání nezávislý na tom, zda máme jeden nebo dva vstupní dokumenty.

## 3.3 Výpočet f-image

V této chvíli máme všechny tokeny načteny a uloženy ve struktuře TTokens. V této fázi vytvoříme f-image (z angl. floating point image), což je matice hodnot. V jednoduchém případě je hodnota 1 tam, kde dochází ke shodě tokenů, a 0 tam, kde ke shodě nedochází.

Tato část se již bude mírně lišit v závislosti na tom, zda porovnáváme dva různé soubory nebo hledáme podobnost pouze v rámci jednoho souboru. Pro každý případ máme dvě funkce, celkem tedy čtyři. Další dělení je podle toho, zda počítáme dotplot z celých souborů nebo jen z určitého úseku. Zde je jejich výčet a stručný popis:

**computing_fimage** – vypočtení pouze části dotplotu ze dvou souborů

**computing_fimage_all** – vypočtení celého dotplotu ze dvou souborů

**computing_fimage_ss** – vypočtení pouze části dotplotu z jednoho souboru

**computing_fimage_ss_all** – vypočtení celého dotplotu z jednoho souboru

### 3.3.1 Algoritmus

Algoritmus je převzatý z textu [1], který se zabývá pouze zkoumáním self-similarity. Tato práce je rozšířena o zkoumání podobnosti i mezi dvěma různými soubory, a proto bylo nutné tento algoritmus mírně modifikovat.

K tomuto algoritmu by se dalo dospět následující úvahou. Jedno z možných řešení by bylo mít každý soubor načtený ve vektoru, postupně je procházet a tvořit tak matici po řádcích. Pokud označíme počet tokenů v prvním souboru písmenem $M$ a počet tokenů ve druhém souboru jako $N$, pak by bylo zapotřebí $M * N$ porovnání.

Jak už bylo zmíněno dříve, tak typů je méně než tokenů, v nejhorším případě jich je stejný počet (k tomu dochází v případě, že vstupní text rozdělíme na unikátní tokeny - každý se vyskytuje pouze jednou). Zde se

nabízí možnost, že bychom neprocházeli token po tokenu, ale typ po typu. Hodnoty typů jsou uloženy ve struktuře TTokens, kde tvoří klíč v mapě (`std::map`) a hodnotou je vektor výskytů daného typu. Pokud je počet typů v prvním souboru $V$ ($V \leq M$) a ve druhém $U$ ($U \leq N$), tak potřebujeme $U * V$ porovnání, což v nejhorším případě znamená opět $M * N$ porovnání.

Stačí nám projít strukturu TTokens a pokud zjistíme shodu, tak projdeme vektor(y) všech výskytů a do matice TMatrix zaneseme informaci o shodě do příslušných políček.

Tato složitost jde však ještě snížit. Ukázalo se, že výhodnější je procházet strukturu TTypes a každý typ zkusit vyhledat v tokens1 a tokens2 (při self-similarity máme jen jednu instanci TTokens a není třeba nic vyhledávat). Pokud typ nalezneme v obou strukturách, pokračujeme dále, v opačném případě přejdeme k dalšímu typu. Díky použití `std::map` je vyhledávání velmi rychlé, protože vyhledáme klíče, které jsou setříděné. Z původních $U * V$ porovnání se tak dostaneme až na $(U + V) * (log_2(U) + log_2(V))$, což se vyplatí až pro větší hodnoty $U$ a $V$. Výraz $U + V$ odpovídá nejhoršímu případu, kdy se žádný typ nenachází v obou souborech. Výrazy $log_2(U)$ a $log_2(V)$ vyjadřují složitost vyhledávání mezi setříděnými hodnotami.

Upravený algoritmus použitý ve funkci `computing_fimage_all`:

```
TTokens::iterator it1, it2;
TTypes::iterator it_types = types->begin();
while (it_types != types->end()) {
   it1 = tokens1->find(it_types->second);
   if (it1 != tokens1->end()) { // prvni nalezen
      it2 = tokens2->find(it_types->second);
      if (it2 != tokens2->end()) { // druhy nalezen
         // pocty vyskytu v souborech
         TInt f1 = it1->second.size();
         TInt f2 = it2->second.size();
         double f; // prumerny pocet vyskytu
         f = (double)(f1+f2)/2;
         if (f < T || T == NO_APROXIMATION) {
            double w; // vaha typu (ohodnoceni)
            w = weight(f);
            for (TInt i = 0; i < f1; i++) {
               int k = it1->second[i]*m/M;
               for (TInt j = 0; j < f2; j++) {
                  int l = it2->second[j]*n/N;
```

```
                fimage[k][l] += w;
             }
          }
       }
    }
    it_types++;
}
```

Pro výpočet self-similarity stačí projít pouze TTokens (obsahuje stejný počet položek jako TTypes, jelikož máme jen jeden soubor). Zde můžeme použít jiné vylepšení. Matice f-image je vždy symetrická, proto stačí spočítat jen jednu polovinu, kterou zkopírujeme do druhé po skončení výpočtu. Tento postup se vyplatí opět až při větších souborech, kde je potřeba použít kompresi. Algoritmus (použitý ve funkci `computing_fimage_ss_all`) vypadá takto:

```
TTokens::iterator it;
while (it != tokens->end()) {
   // pocet vyskytu v souboru
   TInt f = it->second.size();
   if (f < T || T == NO_APROXIMATION) {
      double w; // vaha typu (ohodnoceni)
      w = weight(f);
      for (TInt i = 0; i < f; i++) {
         int k = it->second[i]*m/M;
         for (TInt j = i; j < f; j++) {
            int l = it->second[j]*m/M;
            fimage[k][l] += w;
         }
      }
   }
   it_types++;
}
// zkopirovani poloviny matice
for (int i = 0; i < m; i++) {
   for (int j = i; j < m; j++) {
      fimage[i][j] = fimage[j][i];
}
```

Při výpočtu f-image z částí souborů se používají funkce `computing_fimage` a `computing_fimage_ss`. Jediná změna je v tom, že četnost výskytů se počítá pouze ze zadaného intervalu a musíme kontrolovat, zda-li výskyt tokenu patří do tohoto intervalu.

### 3.3.2 Vyvažování (Weighting)

Vyvažování je nutno použít v případě, kdy máme velká vstupní data. Jinak by mohlo dojít k tomu, že se několik shod tokenů nakumuluje do jednoho zobrazovaného bodu (viz následující oddíl) a celková matice by byla přesycená (saturation). Ve výsledku bychom viděli začerněný dotplot, ve kterém by hledaná informace pravděpodobně zanikla. Proto při výpočtu f-image nerozlišujeme pouze hodnoty 1 a 0, ale ukládáme vypočítanou hodnotu podle toho, jak je shoda významná.

Zřejmě není překvapivé, že v obsáhlých textech se vyskytuje velký počet předložek, spojek, zájmen atd. Díky těmto slovům by mohly zaniknout shody, které jsou pro nás daleko zajímavější, a proto je nutné tento případ nějak ošetřit. Nabízí se řešení, že do matice místo 1 uložíme hodnotu $w = 1/f$, kde $f$ je počet (frekvence) výskytů daného tokenu. Tedy čím více výskytů, tím menší bude hodnota $w$. V případě, že porovnáváme dva soubory, spočteme hodnotu $w$ jako $w = 1/((f_1 + f_2)/2) = 2/(f_1 + f_2)$, kde $f_1$ je počet výskytů daného tokenu v prvním souboru a $f_2$ ve druhém souboru.

Díky vyvažování dosáhneme toho, že shody předložek atd. budou v dotplotu zobrazeny světlejší barvou, zatímco shody vzácnějších slov budou nejtmavší.

### 3.3.3 Komprese (Compression)

Pokud vytváříme dotplot z krátkého článku nebo jen z několika vět, tak si bez problémů vystačíme s tím, co už známe. Krátký článek může mít například 200 – 300 slov (tokenů), pokud pro každou shodu použijeme jeden pixel, výsledný dotplot se nám vejde na obrazovku i na papír. Ale co se stane v případě, kdy máme knihu s několika tisíci či desetitisíci slovy a chceme z ní vytvořit dotplot. Pro každý pár tokenů musíme alokovat v paměti 8B (double). Výsledný dotplot bychom museli nějakým způsobem zmenšit, abychom ho mohli zobrazit na monitoru. Proto by bylo lepší použít kompresi už při vytváření f-image. Ušetří se tak místo v paměti a převod na q-image bude rychlejší.

Metoda komprese spočívá v tom, že shody několika tokenů budou zobrazeny v jednom bodu na obrazovce a také pro ně bude vyhrazeno pouze jedno políčko v matici. Pokud je počet tokenů v prvním souboru $M$ a ve druhém $N$, původní matice by měla velikost $M \times N$. My máme například prostor jen pro $m \times n$, kde $m \ll M$ a $n \ll N$. Shoda $i$-tého tokenu s $j$-tým se zapíše do matice f-image na souřadnice $fimage[i * m/M][j * n/N]$ ($i$ je pořadí tokenu v prvním souboru, $j$ ve druhém). Analogicky bychom udělali přepočet pro jeden porovnávaný soubor.

Shody jsou ohodnocené (weighting) a v jednom poli matice se budou postupně přičítat. Důsledkem toho je, že už nebudeme mít hodnoty jen v intervalu $< 0, 1 >$, ale mohou narůstat až do několika desítek či stovek. To znamená, že nebude možné při převodu na q-image jednoduše vypočítat odstín šedi. Převod bude o něco náročnější a více se o něm dozvíme později.

### 3.3.4 Aproximace (Aproximation)

Nyní už víme, jak si poradit s velkými daty a jak dosáhnout toho, aby dotplot nebyl příliš přesycený. V této podkapitole se dozvíme, jak urychlit výpočet. Pokud máme velmi velká vstupní data, kde se vyskytuje příliš předložek, členů atp., které do matice přispějí velmi malou hodnotou a přitom tuto hodnotu musíme do matice zapsat na několik míst, bylo by lepší ji ignorovat a zapisování vynechat.

Musíme však stanovit minimální hranici, kolikrát se musí token v textu vyskytovat, abychom jej mohli ignorovat. Tuto hranici nazveme *treshold* (raději se opět budeme držet anglického termínu, český překlad by byl *práh*) a její hodnotu označíme písmenem $T$. V případě, že porovnáváme dva soubory, tak nás bude zajímat, zda průměrný počet výskytů tokenu je menší než $T$. Pokud počítáme f-image jen pro část souborů, uvažujeme počty výskytů jen v daném intervalu.

Diskuzi o optimální hodnotě treshold a ukázku několika dotplotů vytvořených s různými hodnotami $T$ naleznete v kapitole Experimenty na stránce 34.

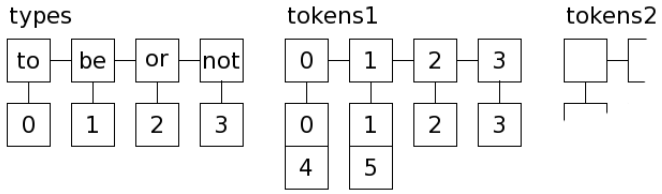
### 3.3.5 Praktická ukázka

Popíšeme si podrobný postup vytváření dotplotu na malém příkladu. V prvním souboru budeme mít větu „To be or not to be“ a ve druhém „Not to be or to be“. Některá písmena jsou sice velká, ale pro náš příklad budeme

velikost ignorovat. V aplikaci je možné nastavit, zda chceme brát na velikost písmen ohled (volba case-sensitive).

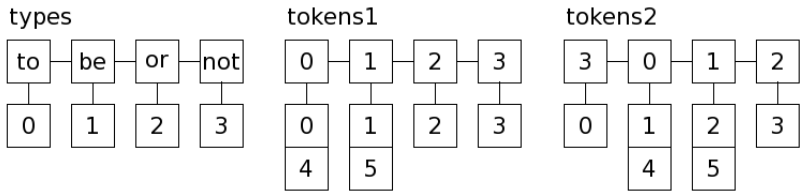
1. Načteme slova z prvního souboru. Oddělovačem jsou všechny bílé znaky (mezera, tabulátor, konec řádku), takže jedno slovo odpovídá tokenu. Pro každý token se nejdříve podíváme do *types*, zda už je uloženo, pokud ne, přidáme ho s hodnotou o 1 vyšší. Pak tuto hodnotu zkusíme vyhledat v *tokens*1, pokud ji nenalezneme, tak ji přidáme a zároveň uložíme pozici výskytu v souboru (číslujeme od nuly).

   Po prvním kroku budou vypadat struktury následovně:




2. Načteme druhý soubor. Obsahuje stejná slova, akorát v jiném pořadí, takže do *types* nic nepřidáváme, ale do *tokens*2 postupně přidáme všechny hodnoty a pozice výskytů tokenů.

   Po druhém kroku vypadají struktury takto:




3. Vytvoříme f-image. V každém souboru je 6 tokenů, takže matice bude mít velikost $6 \times 6$. Použijeme algoritmus, který je uveden výše. Pro náš malý příklad je použití komprese a aproximace (i vyvažování) celkem zbytečné.

   Matice f-image vypadá takto:

| | not | to | be | or | to | be |
|---|---|---|---|---|---|---|
| to | 0 | 0,5 | 0 | 0 | 0,5 | 0 |
| be | 0 | 0 | 0,5 | 0 | 0 | 0,5 |
| or | 0 | 0 | 0 | 1 | 0 | 0 |
| not | 1 | 0 | 0 | 0 | 0 | 0 |
| to | 0 | 0,5 | 0 | 0 | 0,5 | 0 |
| be | 0 | 0 | 0,5 | 0 | 0 | 0,5 |

4. A po převodu na q-image (pouze odstíny šedi) získáme výsledek, jaký je na obrázku 2.2 b).

## 3.4 Výpočet q-image

V této části se budeme zabývat tím, jak z matice čísel (f-image) vyrobíme obrázek (q-image), který již můžeme zobrazovat a objevovat v něm různé vzorky, které jsme si popsali v kapitole 2.4. Na počátku jsme měli v matici (f-image) hodnoty pouze 1 a 0, kterým by v q-image odpovídaly barvy černá a bílá. Pak jsme provedli rozšíření a zavedli odstíny šedi. Nakonec jsme připustili, že v matici mohou být hodnoty vyšší než 1 a navíc tuto maximální hodnotu předem neznáme.

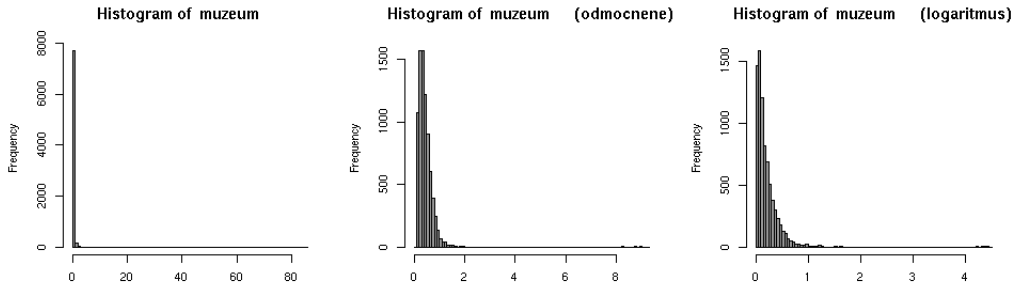
Ve vytváření q-image máme na výběr z několika způsobů. Zjistíme, že ne všechny vedou k pěkně vypadajícím výsledkům. Prvním takovým nápadem může být vydělení všech hodnot v matici jejich maximem. Získáme tak hodnoty v intervalu $< 0, 1 >$, které už můžeme jednoduše převést do odstínů šedi. Tento způsob má však jednu nevýhodu. Pokud se více zabýváme tím, jaké hodnoty se objevují v matici, tak zjistíme, že většina jich je velmi malých ($< 1$) a několik je naopak velmi velkých ($> 100$), což v důsledku znamená, že dotplot je příliš světlý a tmavou barvou jsou zaplněna pouze místa, kde dochází k velké shodě. Pak se může stát, že nám nějaká informace unikne.

Další pokus spočíval v tom, že jsme hodnotám $\leq 1$ přiřadili odstíny šedi a hodnotám $> 1$ černou barvu. Malou obměnou bylo zavedení další barvy (např. červené) pro hodnoty $> 1$. Byl učiněn i pokus přidat ještě jednu barvu pro extra velké hodnoty.

Tato metoda má však také jednu nevýhodu. Díky tomu, že předem

neznáme, jaké hodnoty se v matici objeví, nemůžeme pevně určit hranici resp. hranice, při jejichž překročení dojde ke změně barvy. Pokud nastavíme hranici pevně, tak se může stát, že každý dotplot bude jinak tmavý - stejná informace bude znázorněna jinak. V extrémním případě v něm přerostou přidané barvy a informace opět zanikne. Tyto pokusy jsou podrobněji popsány v kapitole Experimenty, kde jsou uvedeny i ukázky (str. 35).

Nabízí se však myšlenka, že tyto hranice spočítáme z hodnot, které bude obsahovat již vypočtená matice. Mohli bychom použít např. průměr, medián nebo nějaký kvantil.

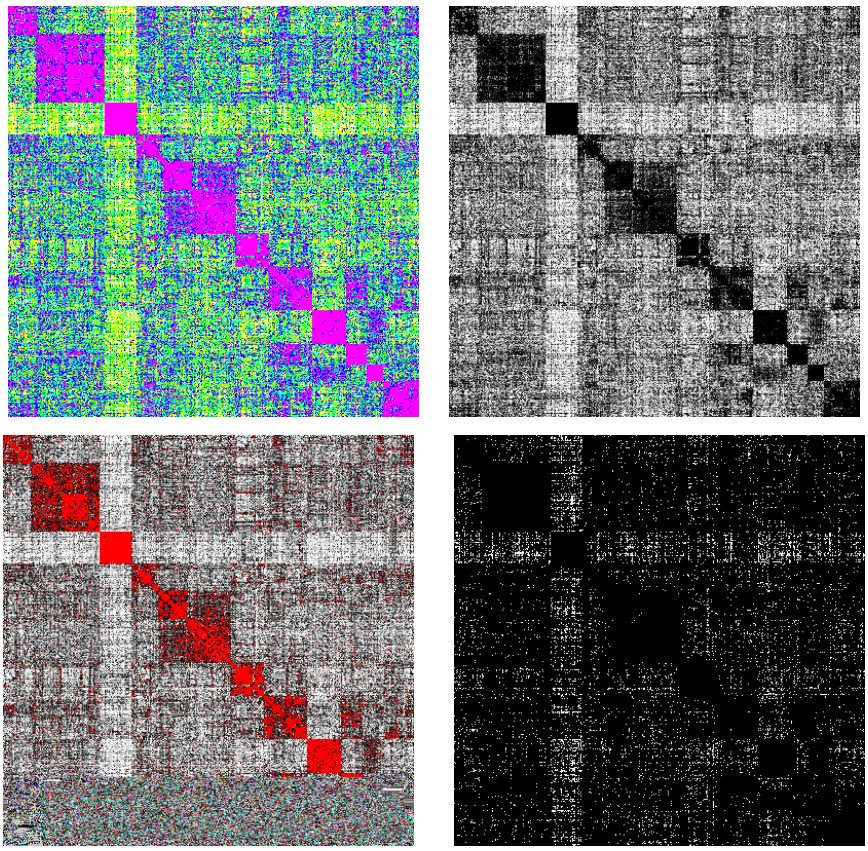



Obrázek 3.1: Histogramy a) původní hodnoty b) odmocněné c) zlogaritmované

Když se podíváme na obrázek 3.1 a), na kterém je zobrazen histogram[1] hodnot v matici, vidíme, že všechny hodnoty jsou blízké nule. Můžeme použít nějakou funkci, např. druhou odmocninu nebo přirozený logaritmus, čímž dosáhneme toho, že se nám hodnoty „roztáhnou“. První histogram zobrazuje původní hodnoty, druhý je vytvořen z odmocněných hodnot a třetí ze zlogaritmovaných. (Histogramy na obrázku vychází z článku v Českém národním korpusu nazvaném „muzeum“.)

V programech byly nakonec použity čtyři způsoby vykreslení (type of q-image), které si nyní popíšeme.

[1] histogram je sloupcový graf, v němž každému intervalu přiřadíme počet hodnot, které do něj spadají



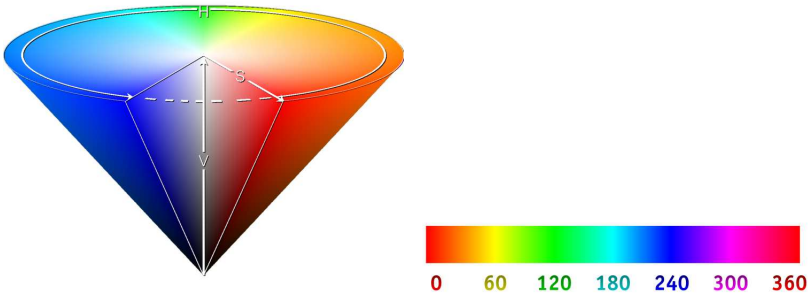
Obrázek 3.2: Různé druhy q-image

1. **HSV barvy**

   Pro vykreslení používáme barvy, které mají v HSV modelu[2] složky $S = 1$ a $V = 1$ a složka $H$ se pohybuje v rozmezí 60° – 360°. Jsou to takové barvy, jaké můžeme najít na obvodu HSV kuželu, který je zobrazen na obrázku 3.3. Na tomto obrázku vidíme také HSV scale.

   Hodnoty nejprve zlogaritmujeme a zjistíme hranice intervalů. Každému intervalu odpovídá jeden barevný odstín.

[2]HSV = Hue, Saturation, Value.
Hue - barevný tón, převládající. Měří se jako poloha v barevném kuželi (0° až 360°).
Saturation - sytost barvy, příměs jiné barvy. V kuželi vzrůstá od středu k okrajům.
Value - hodnota jasu, množství bílého světla. V kuželi vzrůstá směrem k podstavě.




Obrázek 3.3: a) HSV kužel b) HSV scale

2. **odstíny šedi**

   Hodnoty opět zlogaritmujeme, nalezneme 90%-kvantil, 10% největších hodnot obarvíme černě, pro prvních 90% použijeme odstíny šedi, které přiřazujeme opět podle intervalů. Pro nulové hodnoty použijeme bílou barvu.

3. **odstíny šedi + červená**

   Totožný postup jako v předchozím případě, jen pro 10% největších hodnot nepoužijeme černou, ale červenou barvu.

4. **černobílý**

   Pokud je v matici nenulová hodnota, vykreslíme ji černě, ostatní místa zůstanou prázdná (bílá). Tento způsob se hodí jen pro velmi malé soubory. Na obrázku 3.2 je vidět, jak nevhodné je použití pro větší soubor (cca 90 000 slov).

## 3.5 Uživatelské rozhraní

Vše, co bylo popsáno v této kapitole, je implementováno v command-line verzi. V grafické aplikaci je možné s takto vytvořeným dotplotem dále pracovat, vytvářet z něj subdotploty (porovnání pouze určitých úseků souborů), sledovat původní text, uložit si některý dotplot nebo všechny najednou.

Navíc je zde diff. Je implementován v základní podobě - program převezme výstup z UNIX diffu a zobrazí soubory s obarvenými řádky podle toho, zda byl řádek přidán, smazán či změněn.

Oba programy jsou napsány v jazyce C++. Zatímco command-line verze funguje bez dalších speciálních knihoven (používá jen standardní a knihovnu libpng, která se nachází většinou v každé Linuxové distribuci), grafická aplikace používá Qt toolkit verze 4.1.4. Podrobný popis realizace by byl nad rámec tohoto textu, ale můžete jej nalézt na přiloženém CD v programátorské dokumentaci.

# Kapitola 4

# Experimenty

V této kapitole seznámím čtenáře s pokusy, které jsem prováděla při řešení některých problémů implementace, jako např. hledání nejlepšího způsobu převodu f-image na q-image nebo jaká je optimální hodnota parametru treshold.

Během výkladu o vzorcích v dotplotu bylo uvedeno již několik obrázků vytvořených z textů. Některé z nich vznikly uměle, aby byly co nejvíce názorné. Podíváme se na reálná data, která pocházejí z Českého národního korpusu[1] a z vícejazyčného korpusu (European Parliament Proceedings Parallel Corpus).

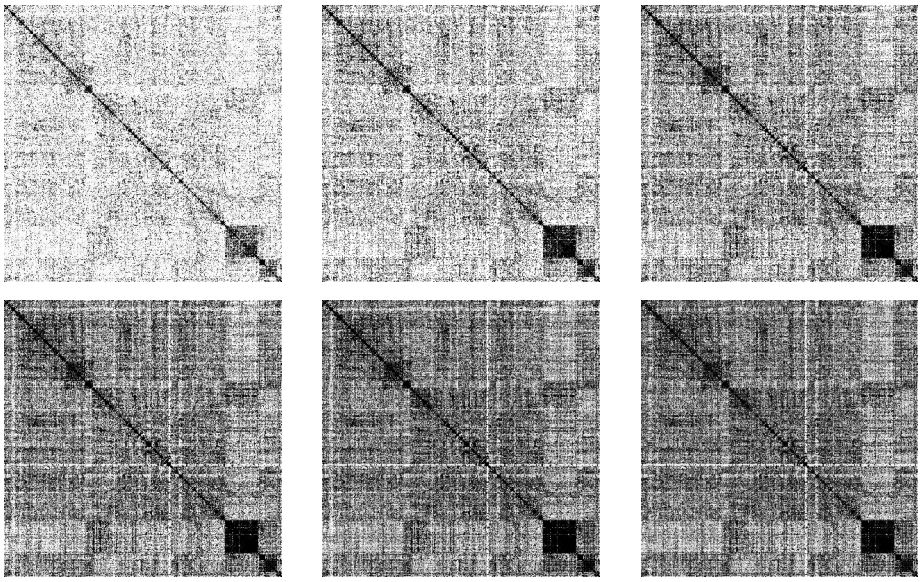
Nakonec se podíváme na některé pokusy z hlediska času.

## 4.1 Treshold

Už dříve jsme si vysvětlili, že čím nižší je kritická hodnota výskytů (typ s vyšším počtem výskytů ignorujeme), tím rychlejší je výpočet f-image. Pokud vytvoříme několik obrázků s různým tresholdem, zjistíme, že se od sebe mohou celkem lišit. Pokud je treshold příliš malý, může se stát, že některá informace nebude příliš patrná. Pokud je naopak příliš velký, dotplot bude přesycen a informace může zaniknout.

V příkladu byl použit text pojednávající o akvaristice, který obsahuje přibližně 40 000 slov. Dotploty jsou vytvořeny postupně s hodnotami treshold 20, 50, 100, 200, 500 a 1000.

[1] jazykový korpus je soubor textů (většinou velmi rozsáhlý), které bývají označkovány (anotovány)



Obrázek 4.1: Ukázka použití různých hodnot parametru treshold

V dotplotu poukážeme na několik znaků a porovnáme je na obrázcích. Prvním takovým znakem je diagonála. Na prvním obrázku je velmi dobře rozpoznatelná, se zvyšujícím se tresholdem se její viditelnost mírně zhoršuje.

V levém dolním rohu je tmavší čtverec, který je vidět přibližně stejně dobře na všech obrázcích. Tento čtverec leží v oblasti křížení světlejších pruhů. Ty už nejsou vidět stejně dobře na všech obrázcích, možná nejlépe jsou zobrazeny na posledním obrázku.

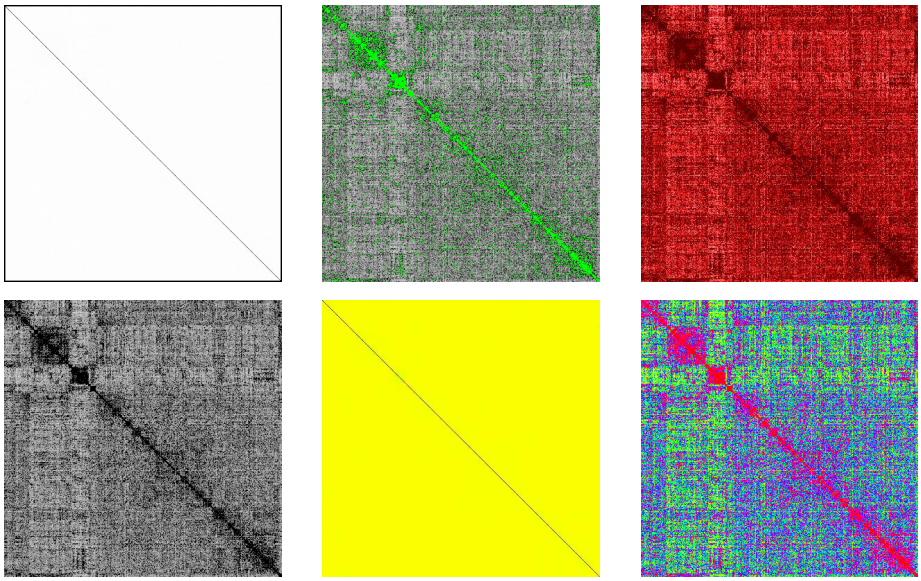
Přibližně v první čtvrtině na diagonále je malý čtvereček. Ten je opět vidět lépe až na obrázcích s vyšším tresholdem. Pokud budete obrázky zkoumat podrobněji, jistě naleznete takovýchto drobných rozdílů více.

Ačkoliv je toto posuzování velmi subjektivní, některé prvky jsou patrné při menším treshold, jiné při vyšším, což vede k závěru, že optimální hodnota bude někde uprostřed. Třetí obrázek ($T = 100$) by mohl být takovým příjemným kompromisem. To je také důvod, proč ve výchozím nastavení programu je právě $T = 100$.

## 4.2 Q-image

Původně bylo navrženo více způsobů převodu na q-image. Bohužel ne všechny se ukázaly jako použitelné. Ve finální verzi aplikace zůstaly pouze čtyři způsoby, které byly popsány už v kapitole Výpočet q-image na str. 28

Na obrázku 4.2 byl použit text knihy Amatérský přírodovědec (132 000 slov), který se také nachází v jazykovém korpusu. Zde nemá cenu se zabývat tím, na kterém obrázku je co lépe vidět, to si může čtenář rozhodnout sám. Popíšeme, který obrázek byl vytvořen jakým způsobem a popř. se zmíníme o tom, proč byl tento způsob zavržen.



Obrázek 4.2: Pokusy s převodem na q-image

První obrázek vznikl tak, že jsme hodnoty vydělili maximem a převedli do odstínů šedi. Většina jich byla příliš malá, maximum naopak příliš velké, a proto se zachovaly jen body na diagonále. Ostatní hodnoty se po vydělení zaokrouhlily na nulu. Proč nebyl tento způsob použit je celkem jasné - veškerá informace byla ztracena.

Pro druhý obrázek se hodnoty v intervalu $< 0, 1 >$ převedly do odstínů šedi, hodnoty $(1, d >$, kde $d$ je průměr všech hodnot, byly červeně a hodnoty větší než $d$ byly barvou zelenou. Není to až tak špatný postup, ale jak jste si mohli povšimnout, v tomto příkladu se červená barva vůbec nevyskytuje. Je

to tím, že rozdělení hodnot je odlišné pro každý f-image a hranice intervalu by neměla být zadána pevně (tady je pevně nastavena 1).

Třetí obrázek je asi největším krokem stranou. Metoda nebyla řádně promyšlena, ale obrázky vypadaly celkem pěkně, a proto také stojí za zmínku. Jedná se o to, že hodnoty rozdělíme do intervalů, každému intervalu přiřadíme barvu a podle velikosti hodnoty určíme odstín barvy. V tomto příkladu byly intervaly tři a barvy byly různé druhy červené.

Způsob převodu použitý ve čtvrtém obrázku se používal téměř od začátku vývoje programu a přesto nebyl zahrnut v aplikaci. Princip je stejný jako ve druhém obrázku, ale hodnoty větší než 1 jsou černé. To má za následek, že pokud máme velmi velký vstup, tak dotplot je celý černý (tak se tomu stalo např. při vytváření dotplotu knihy o 1 600 000 slovech). Problémem tedy je, že hodnota hranice je opět zadána pevně jako v předchozím případě.

Předposlední ukázka je obdoba prvního obrázku s tím rozdílem, že nepoužíváme odstíny šedi, ale HSV barvy, kde složky saturation a value jsou nastaveny na maximum, složka hue, která určuje barevný odstín, nabývá hodnot 60° – 360°. Hodnoty z f-image opět dělíme jejich maximem a pak jim přiřadíme barvu.

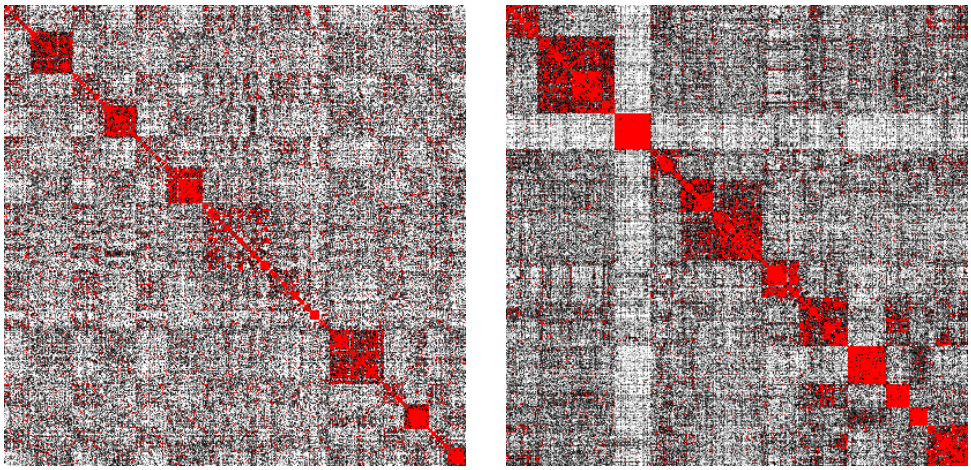
Poslední obrázek vznikl sloučením metody 3 a 5. Používáme zde opět HSV barvy, ale hodnoty nejdříve rozdělíme do intervalů. Podle toho, ve kterém intervalu se hodnota nachází, dostane přiřazenu barvu. V této fázi vývoje to byl asi nejlepší způsob, který mohl být použit pro různě velké vstupy bez ztráty informace, protože se zde objevuje myšlenka dělení do intervalů. Přestože obrázky vypadají poněkud křiklavě, byla obměna tohoto způsobu zařazena do aplikace. Ovšem z hlediska tisku jsou lepší obrázky v odstínech šedi.

## 4.3 Příklady z praxe

První dva příklady byly vytvořeny každý z jednoho souboru. Jedná se tedy o self-similarity nebo-li soběpodobnost. Data pochází z Českého národního korpusu.

V první ukázce (obrázek 4.3 a) jsou za sebou seřazena díla Járy Cimrmana[2]. Každé dílo se skládá ze dvou částí - vědecký seminář a divadelní hra, která má podobu scénáře. To znamená, že v textu je napsáno jméno

[2]Jára Cimrman je fiktivní postava univerzálního českého génia, vytvořená Jiřím Šebánkem a Zdeňkem Svěrákem.



Obrázek 4.3: Příklady z praxe: a) Cimrman b) muzeum - vědecké práce

postavy, která daný text říká. Důsledkem je, že tato jména se velmi často opakují (Cimrmanovy hry jsou většinou pro malý počet postav - cca 4 – 5).

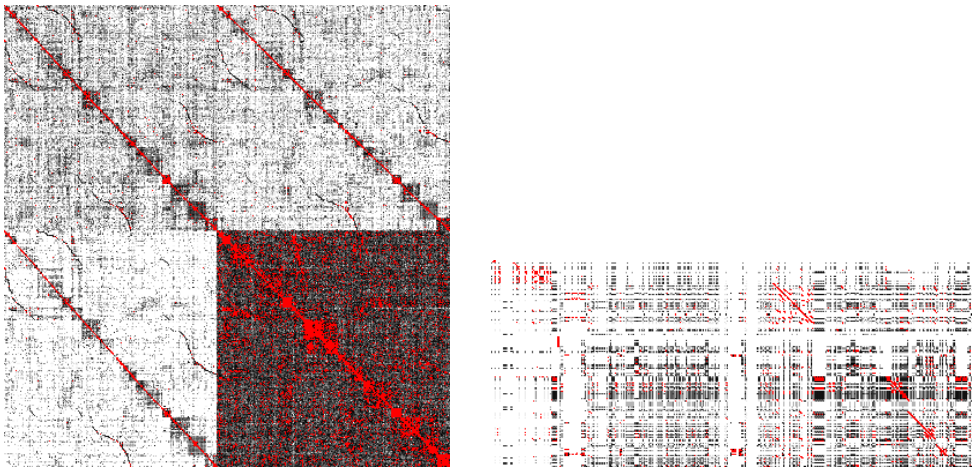
Na diagonále vidíme, že se střídají světlejší a tmavší čtverce (ty světlejší jsou hůře rozpoznatelné). To odpovídá tomu, že světlejší čtverec obsahuje vědecký seminář a tmavší čtverec je divadelní hra. Tento rozdíl by se dal vysvětlit tím, že právě ve hře se velmi často vyskytují jména postav, která se nevyskytují už v žádné jiné hře.

Zhruba uprostřed nejsou čtverce na diagonále příliš zřetelné. Tato část odpovídá dílu *Němý Bobeš* a *Cimrman v říši hudby*. Němý Bobeš má odlišnou strukturu od ostatních, protože se nedělí na seminář a hru, ale je to jeden velký seminář, ve kterém je popisováno, jak byly rekonstruovány divadelní hry. Seminář je prokládán ukázkami ze hry, což se projevuje jako tmavší malé čtverečky, které se vyskytují i mimo hlavní diagonálu.

Dílo Cimrman v říši hudby také nemá standardní strukturu. Jde o jeden větší seminář, na jehož konci je uvedena opereta, která v dotplotu tvoří také tmavší čtvereček (lépe je vidět až při zvětšení).

Ještě bychom si mohli povšimnout větších, ale ne příliš tmavých čtverců mimo diagonálu. Díla jsou si velmi podobná, jak strukturou, tak i obsahem, zvláště v seminářích se často vyskytuje umělcovo jméno a navíc díla pocházejí od stejných autorů (Zdeněk Svěrák a Ladislav Smoljak). Dá se předpokládat, že používají podobné jazykové prostředky, čímž by se dalo vysvětlit proč dochází k četným shodám napříč celým souborem děl.

Druhý příklad je typickou ukázkou souboru více prací. V tomto případě se jedná o vědecké práce. Každá z nich je reprezentována čtvercem na diagonále. Můžeme zde vidět světlý kříž, který způsobuje práce, která je tematicky zaměřena jinak než ostatní.



Obrázek 4.4: Příklady z praxe 2: a) Vícejazyčný korpus b) zdrojové kódy

Následující ukázky (obr. 4.4) jsou vytvořeny porovnáním dvou souborů, které si jsou nějakým způsobem blízké. V prvním případě počítáme dotplot z vícejazyčného korpusu. Soubory jsou zápisky z parlamentu stejně jako v příkladu použitém při vysvětlování vzorku Shuffle. Na svislé ose jsou data v angličtině a španělštině, na vodorovné ose v němčině a portugalštině. Protože se jedná o překlady, tak vzniknou diagonály ve všech čtvercích. Menší čtverce, které se objevují na diagonálách, jsou způsobeny opakujícími se výskyty vlastních jmen nebo slov společných ve všech jazycích.

Z tohoto příkladu můžeme usoudit, že nejméně podobné jsou si jazyky španělština a němčina (levý dolní čtverec), za nimi následuje dvojice angličtina a portugalština, o něco podobnější jsou si angličtina s němčinou. Nejvíce podobné jsou si však jazyky španělština s portugalštinou.

Všimněte si, že zde není zaznamenáno např. porovnání angličtiny a španělštiny. Pokud bychom chtěli analyzovat shody všech dvojic, museli bychom soubory spojit za sebe a provést self-similarity porovnání (podobně jako na obrázku 2.11 b).

Na druhém obrázku je porovnání zdrojových kódů obou verzí přiložených na CD. Command-line verze je výrazně kratší (svislá osa) oproti grafické

aplikaci. Při spojování souborů byly nejprve použity hlavičkové soubory a za nimi následují zdrojové kódy. Zde byla použita tokenizace po řádcích, tzn. že oddělovačem byl konec řádku. Některé části jsou shodné v obou verzích, proto zde můžeme nalézt diagonály. Výrazně světlejší svislý pruh přibližně uprostřed je způsoben spojováním slotů se signály, což je specifické právě pro Qt.

Při vytváření dotplotů ze všech souborů, které jsem měla k dispozici z jazykového korpusu, jsem objevila spoustu zajímavých obrázků, které by určitě stály za pozornost, ale bohužel pro ně v této práci již není prostor. Ty nejzajímavější z nich jsou přiloženy na CD i s texty, z kterých vznikly.

## 4.4 Časová náročnost

Poslední část této kapitoly bude věnována především tomu, jak dlouho trvá vytvořit dotplot. Uvedeme si konkrétní čísla, aby uživatel získal lepší představu. Všechny pokusy byly realizovány na počítači s procesorem Intel Celeron M420 s frekvencí 1,6 GHz.

Výpočet dotplotu je z hlediska času (ale i paměti) celkem náročná akce. Na začátku vývoje, když nebyl brán ohled na složitost, trval výpočet dotplotu ze středně velkého souboru řádově i desítky minut. Nejnáročnější bylo načítání vstupu, protože se vždy musí kontrolovat, zda už jsme takový token načetli. Původně byly typy uchovávány v nesetříděném spojovém seznamu, takže vyhledávání bylo velmi náročné. Po použití struktury std::map se celkový výpočet o řád zlepšil.

Po tom, co byl výpočet celkově zlepšen (ale ještě ne ve finální podobě), bylo vygenerováno ze souboru textů soubor obrázků. Jeden dokument obsahoval řádově desítky tisíc slov, počet dokumentů byl přes 2000 o celkovém počtu slov 102 milionů a celkové velikosti 650 MB textu. Z toho vzniklo 520 MB obrázků ve formátu PNG. Všechny výpočty byly provedeny s $T = 100$. Celá tato operace proběhla za 20 minut, což byl velmi dobrý výsledek.

Dnes tento výpočet proběhl na stejném souboru dokumentů za ještě lepší čas - 18,25 minut. Obrázky ale byly vytvořeny v jiném způsobu vykreslení, který je o něco náročnější, protože musíme nejdříve ze setříděných hodnot vytvořit intervaly. Celková velikost výstupu je 590 MB, průměrná doba vytvoření jednoho dotplotu vychází na 0,5 s.

V podkapitole Treshold byly použity obrázky, které byly vytvořeny za

cca tři sekundy. Díky různému nastavení treshold se jednotlivé časy mohou lišit až o 250 ms. Zde byla použita command-line verze ve finální podobě.

Zpracování 108 souborů o celkovém počtu slov 9 500 000 (62 MB), kdy jsme chtěli vyrobit pro každý soubor 6 obrázků s různým nastavením treshold, trvalo celkově necelých 10 minut. Pro každý soubor musel být program spuštěn šestkrát. V průměru je tedy jeden dotplot vytvořen a uložen za 1 s. Ještě bychom mohli poznamenat, že mezi těmito soubory se nacházel jeden extra velký, jehož zpracování trvalo 12 s

# Kapitola 5

# Závěr

## 5.1 Další možnosti vývoje

V každém programu se dá najít něco, co by se dalo vylepšit, předělat, udělat jinak nebo doplnit, a tak ani tento nebude výjimkou. Co se týče grafické aplikace, tak by se daly veškeré texty, jako názvy tlačítek, popisky atd., jednoduše přeložit do jiného jazyka pomocí nástroje Qt Linguist. Uživatel by si pak mohl zvolit jazyk, jakým s ním má aplikace komunikovat.

Dále by se dalo vylepšit zobrazování textů k dotplotu, např. když se celý dotplot nevejde najednou do okna, tak se text posune na začátek zobrazované části. Tento text by také mohl být podbarven světlými odstíny barev použitých v dotplotu.

Pokud uživatel dvakrát rychle za sebou klikne do dotplotu, tak by se text mohl posunout na aktuální pozici. Když bude text dostatečně přiblížen tak, že jeden bod odpovídá jedné dvojici tokenů, tak by se tato dvojice mohla zobrazit po najetí myši na bod v dotplotu.

Nyní aplikace podporuje práci jen s jedním dotplotem. Není možné mít dotploty z více souborů, protože by se muselo v paměti uchovávat velké množství dat, aby uživatel s těmito dotploty mohl dále pracovat. Toto omezení by šlo zrušit nebo alespoň upravit tak, že by dotploty v aplikaci zůstaly, ale už by se s nimi nedalo dále pracovat.

Další rozšíření spočívá v tom, že by uživatel mohl vybrat více souborů najednou, které by se spojily za sebe do jednoho velkého souboru a z toho by se spočítal dotplot (self-similarity).

Aplikace by mohla podporovat funkce, které zůstaly v command-line verzi z doby vývoje, jako je např. ukládání souboru s četnostmi (zde by

mohl být i zobrazen seřazený podle abecedy či podle četností).

Možnosti vylepšení nabízejí i části, které jsou společné pro aplikaci i její command-line verzi. Můžeme nalézt další vylepšení použitého algoritmu či nový způsob převodu f-image na q-image. Dále by se dal vymyslet nový parser, třeba takový, jaký používají kompilátory programovacích jazyků. Uživatel by tak mohl k definici tokenů použít regulární výrazy.

Soubor pro ukládání nastavení by mohl mít XML strukturu, pak by bylo jasné, co který parametr znamená a pokročilejší uživatel by mohl modifikovat soubor i mimo aplikaci.

V neposlední řadě by se dalo vylepšit ukládání obrázků tak, že by se do jména souboru vložila informace, s jakými parametry byl dotplot vytvořen, popř. by vznikl ještě průvodní soubor, ve kterém by tyto informace byly zaznamenány, aby uživatel mohl zpětně zjistit, jaká nastavení byla použita, až se k obrázku vrátí po delší době.

V této práci byla implementována jen základní verze diffu, proto i v této části nalezneme mnohá vylepšení.

Program byl vyvíjen pod systémem Linux a přestože Qt je multiplatformní toolkit, tak aplikace nebyla testována na jiných systémech. V dalším vývoji by se dala upravit (včetně command-line verze) tak, aby byla také multiplatformní.

## 5.2 Srovnání s podobnými projekty

V úvodu jsme zmínili existenci podobného projektu se stejným názvem - Dotplot (někde je uveden název Dotploter, pro odlišení budeme v následujícím článku používat tento název). V této podkapitole si popíšeme hlavní rozdíly a zdůrazníme, jaké klady a zápory má tato aplikace oproti našemu Dotplotu.

Dotploter je napsán v jazyce Java, a proto je velmi pomalý. Problémy s rychlostí nenastávají při samotném výpočtu matice (f-image), ale už při manipulaci s aplikací, například při snaze vybrat zdrojový text.

Dále jsem zjistila, že Dotploter neumí pracovat s libovolným souborem, ale pouze s formátem TXT (možná i s dalšími, ale soubory bez koncovky se nezobrazí a tedy nemohou být vybrány).

Pozitivní na tomto projektu je, že pokud chceme porovnávat více souborů najednou, tak je to velmi snadné, protože program je připojí za sebe (cat) a provede self-similarity výpočet celkového souboru. Na druhou stranu není možné porovnat dva různé soubory. Výběr souborů probíhá zaškrtáváním

v panelu, může se tedy lehce stát, že omylem zapomeneme nějaký soubor odškrtnout, když už ho nechceme zahrnout do výpočtu.

Dalším drobným nedostatkem je porovnávání velmi malých souborů. Dotploter vykreslí pro každou shodu jen malou tečku a pokud máme v souboru pouze pár slov, tak jsou tyto tečky velmi nevýrazné.

Posledním problémem, kterého jsem si na první pohled všimla, bylo uživatelsky nepříliš přívětivé rozhraní. Uživatel nemá k dispozici příliš mnoho nastavení a některé věci jsou řešeny podivně - neintuitivně. Například ukládání obrázku se musí nastavit ještě před zahájením výpočtu. (Při pokusu se dotplot vytvořil v pořádku, ale soubor na disku měl nulovou délku.)

Vygenerované obrázky vypadají na pohled celkem příjemně, možná lépe než výstupy z mé aplikace. Je zde spousta nastavení vykreslení, např. ve stylu Matrix. Ale přesto mi zde chybí například nastavení parsování vstupního textu a celkově je to příliš pomalé.

## 5.3 Použité zdroje

Literaturu, která je uvedena na konci práce, jsem využila hlavně k pochopení problematiky a implementaci algoritmu ( [1], [2]). Dále k napsání tohoto textu v typografickém systému LaTeXvděčím především publikaci [3].

Kromě těchto zdrojů byla použita k vývoji aplikace dokumentace Qt toolkitu (http://doc.trolltech.com/4.1/index.html), k vytvoření command-line verze byl použit příklad s vysvětlením vytváření PNG obrázků pomocí libpng na serveru Root.cz (http://www.root.cz/clanky/jak-vytvorit-rastrovy-obrazek-v-jazyce-c/).

Velkým zdrojem byl také Internet, kde např. poznámky pod čarou byly získávány ze stránek wikipedia.org a wikipedia.cz a mnohé další stránky, které není možné uvést jmenovitě.

# Dodatek A

# Uživatelská příručka

## A.1 Grafická aplikace

Nejprve si popíšeme, jak je členěno okno aplikace. Můžeme ho rozdělit na tři hlavní části. V horní části je panel, kde si uživatel vybírá soubor nebo soubory, ze kterých chce vytvořit dotplot. Pro každý soubor je zde jedno tlačítko - *File y* a *File x*. Po stisknutí tlačítka se zobrazí klasický dialog pro výběr souboru. Po potvrzení výběru se název souboru objeví v příslušném políčku (Line Edit). Jméno je možné i manuálně editovat. Pokud chceme tento název jednoduše odstranit, slouží nám k tomu tlačítko *cl*.

Soubor *y* bude v dotplotu použit vertikálně (osa y) a soubor *x* horizontálně (osa x). Pokud budeme mít vybraný pouze jeden soubor, aplikace bude počítat dotplot jako self-similarity.

Dále zde nalezneme tlačítko *Encoding*, které slouží pro výběr kódování v případě, že je jiné než defaultní (UTF-8).

Poslední tlačítko *dotplot & diff* na horní liště je určeno pro spuštění výpočtu dotplotu i diffu zároveň.

Dolní (větší) část aplikace obsahuje tři záložky. První z nich, *dotplot*, je nejdůležitější částí celé aplikace. Obsahuje nastavení pro výpočet dotplotu, nástroje pro další práci s ním a samozřejmě také samotnou plochu pro vykreslení. Druhá záložka *dotplot - text* obsahuje pouze okna pro zobrazení textu, ze kterého je právě spočten dotplot. To znamená, že nezobrazuje celý soubor, ale jen aktuální část. Není možné mít zobrazený text aniž bychom neměli spočtený dotplot.

Poslední záložka *diff* slouží pro zobrazení grafického diffu souboru. Zatímco pro výpočet dotplotu stačí zadat pouze jeden soubor, pro diff musíme

mít vyplněny oba soubory.

Nyní máme popsány hlavní části aplikace a můžeme se podívat na podrobnější popis té nejdůležitější části - záložka *dotplot*.

V levé části nalezneme ovládací panel, který se skládá z několika částí.

*Type of q-image*

První z nich je okno pro výběr způsobu vykreslení (type of q-image). Zde má uživatel na výběr ze čtyř možností. Ty byly podrobněji popsány již dříve v kapitole o vytváření q-image. Každý způsob je reprezentován pouze barevným obrázkem.

První z nich (horní levý) reprezentuje barevné vykreslování, druhý (horní pravý) je pro odstíny šedi, třetí je pro odstíny šedi s červenou barvou pro 10% nejvyšších hodnot a poslední pro černobílé vykreslení. Pokud na tomto panelu přepneme na jiné vykreslení, tak se změna projeví okamžitě - obrázek se překreslí. Při změnách jiných nastavení musíme nejdříve stisknout tlačítko *Compute dotplot* nebo *Recompute* (bude vysvětleno později).

*Fit image to screen*

Zaškrtávací tlačítko *Fit image to screen* je určeno pro zobrazení dotplotu tak, aby co nejlépe vyplňoval prostor pro něj určený. Pokud je dotplot větší než tato plocha, tak se zmenší, v opačném případě se zvětší. Transformace probíhá vždy tak, že je zachován poměr stran. Ve výchozím nastavení toto pole není zaškrtnuté.

Ve střední části levého panelu je objekt s dvěma záložkami. Nejprve si popíšeme záložku *Settings* a pak bude následovat popis záložky *Browsing*.

### Záložka Settings

*Delimeters*

Delimiter neboli oddělovač umožňuje uživateli nastavit, jak se má provést načítání vstupního textu. Zde jsou přednastaveny čtyři základní skupiny oddělovačů mezi tokeny. Pro všechny typy oddělovačů platí, že pokud se vyskytnou dva oddělovače vedle sebe, tak prázdný token, který by mezi nimi vznikl, se bude ignorovat.

První skupinou jsou bílé znaky (tabulátor, mezera, nový řádek), dalším typem oddělovače je pouze nový řádek, třetí skupinu tvoří bílé znaky a speciální znaky:

```
.,:?!;"\/-+='{}()[]<>|*&^%@~‘$#
```

Předposlední skupina je zadána výčtem znaků, které nejsou oddělovači. Patří sem velká a malá písmena anglické abecedy, číslice a podtržítko. Pokud uživateli nevyhovuje ani jedna z přednastavených skupin, má možnost zadat vlastní výčet oddělovačů - zaškrtnout volbu *others* a do políčka zadat řetězec znaků (bez mezer).

Možnosti oddělovačů jsou podrobněji popsány v kapitole Implementace na straně 21.

*N-grams*

*N-grams* neboli n-tice znamená, kolik tokenů (vytvořených dle zadaného oddělovače) odpovídá jednomu suptokenu, který bude dále používán. Uživatel například může chtít zkoumat v textu ustálená slovní spojení, tak si zde nastaví hodnotu 2 a tokeny se budou pojit do dvojic (v dalším zpracování bude token označovat celou dvojici). Toto nastavení může mít hodnoty 1 až 10, výchozí nastavení je 1.

*Case-sensitive*

Zaškrtávací políčko *Case-sensitive* slouží k tomu, zda nám záleží na velikosti písmen či ne. Pokud toto políčko bude zaškrtnuté, tak slova „kočka“ a „KočKa“ budou chápána jako dva různé typy (types), v opačném případě pro vnitřní potřeby aplikace všechna slova převádí na malá písmena a slova „kočka“ a „KočKa“ (i „KOČKA“ atp.) budou reprezentovány jedním typem. Defaultně je políčko nezaškrtnuté (Case-insensitive).

*Treshold*

Treshold (neboli práh) je nastavení kritické hodnoty počtu výskytů typu, podle které rozhodujeme, zda daný typ budeme zpracovávat nebo zda-li ho budeme ignorovat, pokud se vyskytuje příliš často (podrobněji viz část o aproximaci na str. 26).

Uvedeme si názorný příklad. *Treshold* bude nastaveno na hodnotu 50, token „kočka“ bude mít v textu 60 výskytů, a protože $60 \geq 50$, tak tento typ budeme ignorovat. Pokud nechceme ignorovat žádný typ, tak nastavíme treshold na 0. Výchozí hodnota je 100.

*Max width, max height*

Původní myšlenka byla, že *max width* a *max height* (tedy maximální šířka a výška) jsou maximální povolené rozměry dotplotu (v px). S vývojem aplikace je výstižnějším vysvětlením, že tyto maximální rozměry jsou pro omezení velikosti matice f-image a tedy pro míru komprese. Podle těchto rozměrů se spočítá velikost jednoho bodu v dotplotu, takže svým způsobem zachovávají i původní myšlenku. Výchozí nastavení je $400 \times 400$, maximální

možné hodnoty jsou $800 \times 800$, ale tím bude výpočet f-image i převod na q-image (a samotné zobrazování dotplotu) celkově pomalejší.

*Point size*

*Point size* označuje velikost jednoho bodu v dotplotu. Jedním bodem je myšleno jedno políčko z matice f-image. Pokud je nastavena nula, tak je po spočtení f-image vypočtena velikost jednoho bodu podle maximální šířky a výšky. Při zadání nenulové hodnoty je velikost bodu právě tato hodnota. Maximální povolená hodnota je 50, ale uživatel by měl mít na paměti, že pokud samotná matice f-image je dost velká, tak při vyšších hodnotách point size ($\geq 4$) bude obrázek q-image velmi velký a jeho zobrazení bude trvat příliš dlouho. Vyšší hodnoty point size je doporučeno používat jen při velmi malých vstupech (počet tokenů $\leq 50$). Výchozí hodnota je 0, což znamená, že se velikost bodu dopočítá automaticky.

*Max size of dotplot*

Zaškrtávací políčko *Max size of dotplot* označuje, zda si uživatel přeje, aby aplikace sama dopočítala maximální velikost dotplotu dle aktuální velikosti okna. Tím se také určí míra komprese a velikost bodu. Při zaškrtnutí tohoto pole bude ignorováno nastavení *max width*, *max height* a *point size*. Defaultně není zaškrtnuté.

Pokud změníme nastavení, můžeme stisknutím tlačítka *Recompute* vyvolat spočtení nové matice. Při této akci se soubor znovu nenačítá, takže tlačítko není citlivé například na změnu oddělovače. Pokud uživatel změní pouze *point size*, tak se nebude znovu počítat ani f-image, ale pouze q-image. Tlačítko je citlivé pouze na změny od jeho posledního stisknutí nebo stisknutí tlačítka *Compute dotplot*. Není tedy možné například provést změnu nastavení a pak přepočítat několik již vytvořených dotplotů. Tlačítko *Recompute* je citlivé jen pro změny těchto položek: *treshold*, *max width*, *max height*, *point size* a *max size of dotplot*.

Pokud je použito před prvním spočtení dotplotu, provede se stejná akce jako při stisknutí *Compute dotplot*.

### Záložka Browsing

Záložka *Browsing* je určena pro další práci s dotplotem. Pokud uživatel ještě žádný nevyrobil, pak žádná akce na této záložce nebude mít význam. Pro další popis budeme předpokládat, že dotplot již existuje.

V horní části se nachází okno s historií, které obsahuje jednotlivé položky. Každá z těchto položek reprezentuje jeden dotplot. Jméno této položky je

dáno rozmezím, pro které je dotplot počítán. Čísla z tohoto rozmezí odpovídají pozici tokenu ve vstupním souboru, které uvažujeme pro výpočet. Položce, která je právě vyznačena, odpovídá zobrazený dotplot. Mezi těmito položkami je možno překlikávat, aplikace vždy zobrazí příslušný dotplot v aktuálním módu vykreslení. Pokud takový dotplot ještě nebyl zobrazen, není uložen v paměti, a proto se může stát, že nebude zobrazen ihned, protože se nejdříve musí vyrobit nová pixmapa. Jak dlouho bude tato operace trvat záleží na tom, jaká má uživatel nastavení - jak je velká matice f-image a jaká je velikost jednoho bodu.

S již vytvořeným dotplotem můžeme dále pracovat.

Pomocí myši je možné v dotplotu označit část, která se má zvětšit. Označení je jednoduché, je to jako nakreslit obdélník v obyčejném kreslicím programu. Označení se projeví žlutooranžovým orámováním. Poté se stiskne tlačítko *Zoom* (nebo klik pravým tlačítkem myši do doplotu) a tato oblast se přepočítá a zobrazí. V okně s historií přibude další položka se souřadnicemi právě vypočteného dotplotu. Tato akce lze i několikrát opakovat. Pokud ale označíme celý dotplot, tak už se zoomování neprovede, protože výsledek by byl totožný s aktuálním dotplotem. Díky historii můžeme procházet předchozí dotploty.

Dále je možno měnit nastavení vykreslení. Tuto změnu není nutno potvrzovat, aplikace ihned změní dotplot. Pokud už byl vykreslen, tak se pouze zobrazí, v opačném případě je nutno vytvořit q-image.

Tlačítka *Save* a *Save all* slouží k ukládání dotplotu. *Save* uloží pouze aktuální dotplot, který je právě zobrazen. Při ukládání je nutné zadat také příponu, která zároveň znamená formát, v jakém má být obrázek uložen. Pokud přípona není zadána nebo je neznámá, automaticky se doplní `.png` a obrázek bude uložen ve formátu `PNG`. Podporované formáty (a tedy i přípony) jsou `BMP`, `JPEG` (přípony .jpg i .jpeg), `PNG`, `PPM`, `XBM` a `XPM`.

Druhé tlačítko (*Save all*) slouží pro uložení více dotplotů najednou. Uživatel si může vybrat, zda chce uložit od jednoho f-image všechny q-image nebo všechny f-image v jednom módu vykreslení. Pro první případ musí být zaškrtnuto políčko *all toqs* a pro druhý políčko *all dotplots*. Také je možné obě možnosti zkombinovat a uložit vše najednou. V tomto případě budou zaškrtnuta obě políčka. Pokud nebude zaškrtnuto ani jedno, provede se stejná akce jako při jednoduchém uložení.

Uživatel by si měl uvědomit, že pokud některé z požadovaných dotplotů dosud nebyly zobrazeny, aplikace je musí nejprve vytvořit (převést f-image do q-image) a teprve pak je může uložit. Tato operace může trvat

nějakou dobu (opět v závislosti na nastavení a na počtu ukládaných dotplotů). Uložené soubory budou označeny příznaky, aby bylo možné je od sebe rozeznat. Pokud ukládáme více dotplotů, za jméno souboru se připojí pořadové číslo (pořadí je dáno dle pozice v historii). V případě uložení více q-image přidáváme za jméno `_q0` - `_q3`, kde číslo označuje typ vykreslení.

Poslední dvě tlačítka slouží ke smazání již nepotřebných dotplotů, aby zbytečně nezabíraly místo v paměti. *Delete* smaže pouze aktuální dotplot (matici f-image a všechny příslušné pixmapy s různými módy vykreslení - pokud byly vytvořeny).

*Delete all* smaže všechny dotploty a také uvolní alokovanou paměť, takže už není možné prohlížet text, ze kterého byl vytvořen dotplot. Toto tlačítko ale není nutné použít v případě, že chceme spočítat dotplot z jiných souborů, jelikož se všechny dotploty smažou po stisknutí tlačítka *Compute dotplot*.

*Compute dotplot*

Toto nejdůležitější tlačítko celé aplikace jsme si nechali až na závěr. Jeho popis je velmi jednoduchý. Pokud je použito poprvé, tak spočítá dotplot a zobrazí ho. Pokud už byly nějaké dotploty zobrazeny (a uživatel je nesmazal sám), tak budou smazány a bude uvolněna paměť, která byla potřeba k předchozím výpočtům.

### Diff

Před spuštěním diffu je nutné zadat oba soubory a vybrat správné kódování pro každý soubor. Tím se zajistí, že se porovnání provede správně (v případě různého kódování se nejprve oba soubory převedou na stejné) a že se správně zobrazí diakritika.

Některé řádky ve výstupu se zobrazí podbarvené. Modrá barva označuje řádky, které jsou navíc pravém souboru (File x), červená je pro řádky, které jsou v levém souboru (File y). Zelenou barvou jsou takové řádky, které byly nějakým způsobem pozměněny.

### Menu

*File*

V nabídce *File* se nachází položky, které odpovídají tlačítkům popsaným výše. Jsou u nich uvedeny i klávesové zkratky. Navíc jsou zde položky *FullScreen* a *Exit*. Jak už názvy napovídají, tak *FullScreen* zobrazí aplikaci přes celou obrazovku a *Exit* aplikaci ukončí úplně.

*Options*

Nabídka *Options* obsahuje celkem tři položky. *Choose encoding* odpovídá tlačítku *Encoding*. Otevře se dialog, ve kterém uživatel může pro každý soubor zvlášť vybrat kódování. V tomto dialogu je také možno nastavit defaultní kódování (UTF-8). Tlačítko se šipkou nastaví kódování pro druhý soubor stejné jaké je nastaveno pro první.

Položky *Save current settings* a *Load settings* slouží k ukládání a nahrávání nastavení. Nastavení se ukládá do souboru, jednotlivé hodnoty jsou na samostatných řádcích a jsou v pevném pořadí. Struktura souboru je blíže popsána v programátorské dokumentaci na přiloženém na CD. Běžnému uživateli stačí vědět, že by neměl tento soubor editovat mimo aplikaci.

*Help*

Poslední nabídka *Help* obsahuje položky *Help!*, *About Dotplot* a *About Qt*. *Help!* zobrazí stručnou nápovědu s odkazem na uživatelskou dokumentaci. Položka *About Dotplot* zobrazí malé info o této aplikaci jako je např. jméno autorky a datum vzniku. *About Qt* zobrazuje základní informace o Qt.

Nakonec uvedeme stručný přehled všech položek i s jejich klávesovými zkratkami (pokud existují).

```
File
  -- FullScreen              Ctrl+F
  -- Compute dotplot         Ctrl+D
  -- Compute diff            Ctrl+G
  -- Compute all             Ctrl+A
  -- Recompute               Ctrl+R
  -- Zoom                    Ctrl+Z
  -- Save current dotplot    Ctrl+S
  -- Save all dotplots       Ctrl+Shift+S
  -- Exit                    Ctrl+Q
Options
  -- Choose encoding         Ctrl+E
  -- Save current settings   Ctrl+K
  -- Load settings           Ctrl+L
Help
  -- Help                    Ctrl+H
  -- About Dotplot           ------
  -- About Qt                ------
```

## A.2 Command-line verze

Command-line verze programu Dotplot sice zdaleka neumí to, co grafická aplikace, ale má i své klady. Program nevykresluje žádné okno na obrazovku ani nepotřebuje žádnou zvláštní knihovnu (v počátcích byla potřeba knihovna SDL, program vykreslil pouze výsledný dotplot, se kterým už nešlo nic dalšího dělat). Nyní je potřeba pouze knihovna libpng, která je většinou přítomna v každé Linuxové distribuci.

Funkčnost je značně omezena, chybí interaktivita či výběr formátu obrázku. Naopak byly ponechány funkce z fáze vývoje, jako např. vytvoření souboru četností, který není zahrnut v grafické aplikaci.

Jelikož má mnoho parametrů, tak ovládání není zrovna nejjednodušší, ale ne všechny parametry je nutné zadat - uživatel může využít výchozího nastavení. A pokud je v rozpacích, může si nechat zobrazit nápovědu (parametr `-h`).

Kladem command-line verze naopak je, že ji uživatel může spustit vzdáleně nebo si vytvořit jednoduchý script, který mu vyrobí (a uloží) dotploty souborů třeba z celého adresáře.

Nyní si popíšeme, jak program spustit a co jednotlivé parametry znamenají. Nakonec si uvedeme krátký příklad.

Pokud chcete spustit program a jste v adresáři, ve kterém se nachází spouštěcí soubor, stačí napsat do příkazové řádky:

```
dotplot_cl [options] -o <image_output> | -a <image_output>
  -f <filename1> [-g <filename2>] | -r <matrix_input>
```

To, co je v hranatých závorkách, je nepovinné. Výrazy uvedené ve špičatých závorkách mají být nahrazeny konkrétními jmény (závorky nepíšeme). Znak `|` znamená nebo. Interpretace tohoto zápisu je, že musíme zadat parametr `-o` nebo `-a` společně se jménem souboru, dále musí být zadán jeden nebo oba soubory se vstupním textem nebo soubor s maticí. Pro podrobnější popis parametrů viz dále.

Pokud nezadáte jméno souboru pro obrázek s příponou .png, tak bude připojena automaticky. Jiné přípony budou ignorovány a .png bude rovněž doplněno.

Následuje popis jednotlivých parametrů:

- `-f <filename>` filename je jméno prvního souboru, který se promítne na vertikální osu y
- `-g <filename>` filename je jméno druhého souboru, který nemusí být zadán (pak se počítá self-similarity prvního souboru), jinak se promítne na horizontální ose x

  Pokud je zadán parametr `-g` a není `-f`, tak se parametr `-g` ignoruje.
- `-r <filename>` pokud soubor filename obsahuje matici f-image, spočítá se výsledný q-image z této matice. Soubor musí být ve správném formátu, jinak by mohlo dojít k chybě. Tento parametr byl použit pro ladící účely, proto není načítání nijak ošetřeno. Pokud nemá uživatel jen dobré úmysly nebo do souboru manuálně zasahoval, nedoporučuje se používat tento parametr.
- `-a <filename>` vytvoří dotploty ve všech módech vykreslení, které uloží do souboru filename_x.png, kde $x$ je číslo 0 .. 3 charakterizující konkrétní typ vykreslení
- `-q n` $n$ je číslo typu vykreslení (0 .. 3)

  `0` vytvoří se intervaly a přiřadí se jim barvy s hue = 60 .. 360

  `1` vytvoří se intervaly, posledním 10% hodnot se přiřadí černá, zbytek bude v odstínech šedi

  `2` jako 1, místo černé je červená (default)

  `3` černobílé vykreslení 0 - bíla, $> 0$ - černá, vhodné jen pro malé soubory!
- `-c` (bez argumentu) case-sensitive porovnávání
- `-i` case-insensitive (default)

  V případě, že jsou uvedeny oba přepínače `-c` i `-i`, považuje za platný ten, který je uveden později.
- `-h` zobrazí nápovědu
- `-n n` n-tice (n-grams) (1 .. 10) default 1

- `-o <obr_output>` výsledek se uloží do souboru obr_output.png
- `-p n` typ oddělovače (0 .. 4)

  `0` oddělovači jsou bíle znaky - mezera tabulátor, nový řádek (default)

  `1` oddělovačem je pouze nový řádek

  `2` oddělovači jsou bílé znaky a tyto speciální znaky:

  `.,:?!;"\/-+='{}()[]<>|*&^%@~‘$#`

  `3` oddělovači jsou všechny znaky kromě: a-z,A-Z,0-9 a _

  `4` vlastní oddělovače - zadáte pomocí parametru `-d <str>`
- `-d <str>` str je řetězec jednopísmenných oddělovačů
- `-t n` treshold (míra aproximace - čím vyšší číslo, tím menší aproximace)

  default 100, `NO_APROXIMATION` = 0
- `-v <matrix_output>` uloží matici hodnot f-image do souboru matrix_output (na prvním řádku budou uloženy rozměry matice)
- `-x n` maximální šířka matice f-image
- `-y n` maximální výška matice f-image Z těchto rozměrů se pak určí také velikost bodu v q-image. Výchozí hodnoty jsou $400 \times 400$
- `-s n%` počáteční souřadnice pro výpočet pouze části dotplotu (default 0)
- `-e n%` koncové souřadnice default (100) Procenta mohou být i desetinná čísla, použijte desetinnou tečku (ne čárku). Nejprve zadejte souřadnice prvního souboru (osa y) poté znak ':' jako oddělovač a souřadnice v druhém souboru (osa x). Pokud mají být souřadnice (procentuálně) stejné pro oba soubory, stačí číslo zadat pouze jednou. Pokud bude zadán pouze jeden z těchto parametrů, pro druhý se použijí defaultní hodnoty.
- `-k <filename>` uloží četnosti slov do souboru filename

- `-b n` velikost jednoho bodu v dotplotu (point size), default 0 - point size se dopočítá

Ke správnému fungování programu musí být zadány minimálně přepínače `-f` (s volitelným `-g`) nebo `-r` pro vstupní data a `-a` nebo `-o` pro výstupní data. Ostatní parametry jsou volitelné a pokud nebudou zadány, tak se použijí výchozí hodnoty. Pokud je u přepínačů uveden argument, musí být tento argument uveden v příkazové řádce bezprostředně za tímto přepínačem. Dále není možné zadat přepínače `-f` a `-r`, protože `-r` bude ignorován. Pokud budou zadány oba parametry `-a` a `-o`, tak se bude `-o` ignorovat.

Nyní následuje malý příklad spuštění dotplotu:

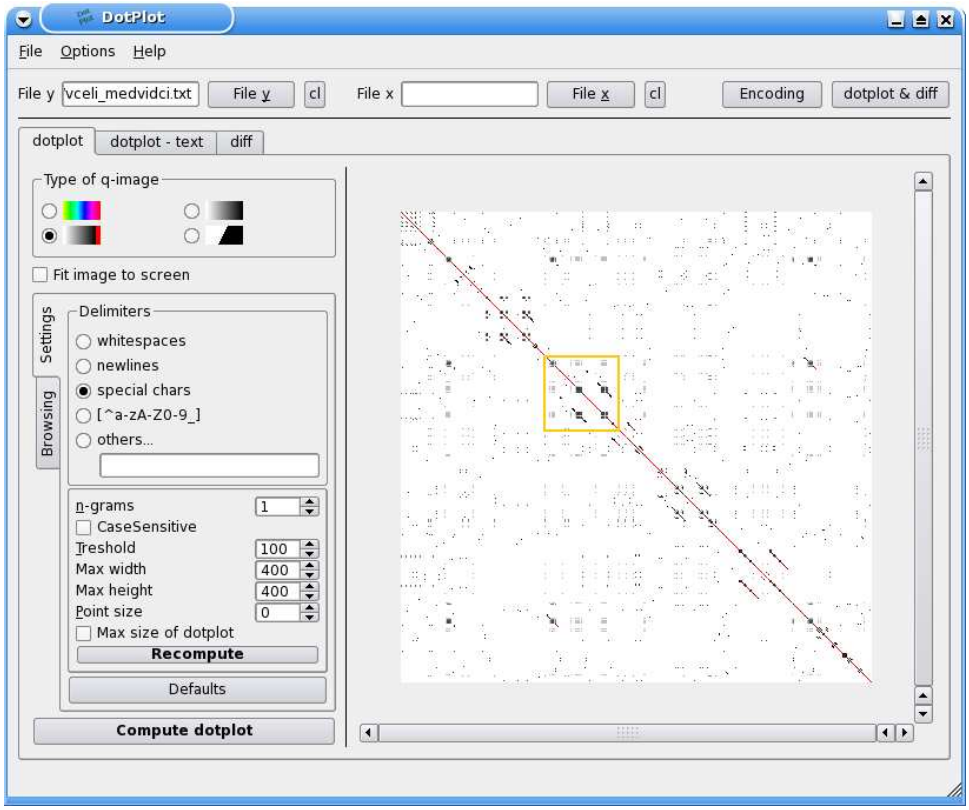
```
dotplot_cl -f input_file1.txt -g input_file2.txt -p 0
  -x 400 -y 400 -q 2 -o output_image.png -k cetnosti.txt
  -t 100 -n 1 -s 20:40.5 -e 100:100
```

Pozorný čtenář si jistě všiml, že ne všechny parametry je třeba zadat. Jelikož v našem příkladu jako hodnoty argumentů používáme defaultní hodnoty, šlo by program spustit i takto (se stejným výsledkem):

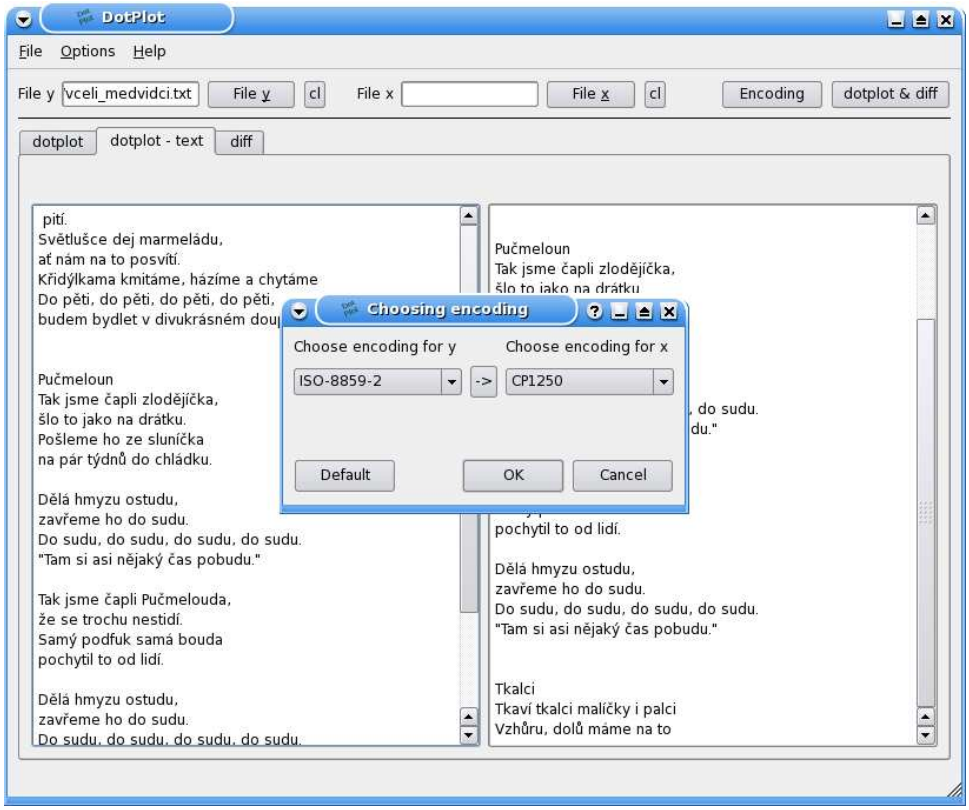
```
dotplot_cl -f input_file1.txt -g input_file2.txt
  -o output_image.png -k cetnosti.txt -s 20:40.5
```

# Dodatek B

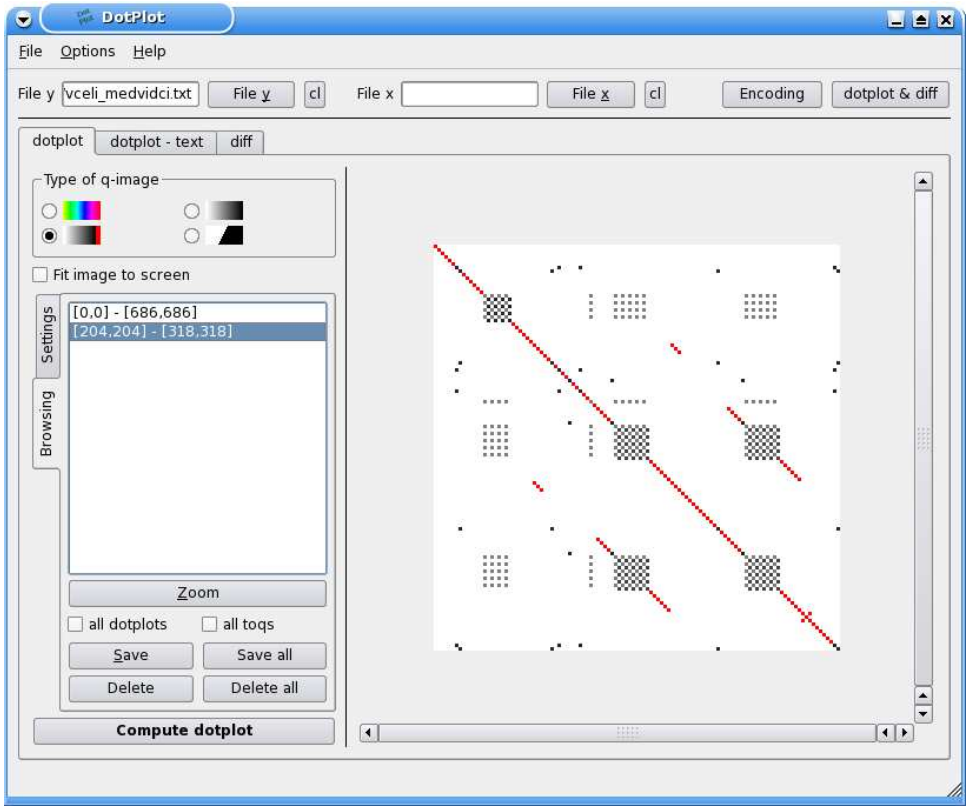
# Náhledy uživatelského rozhraní



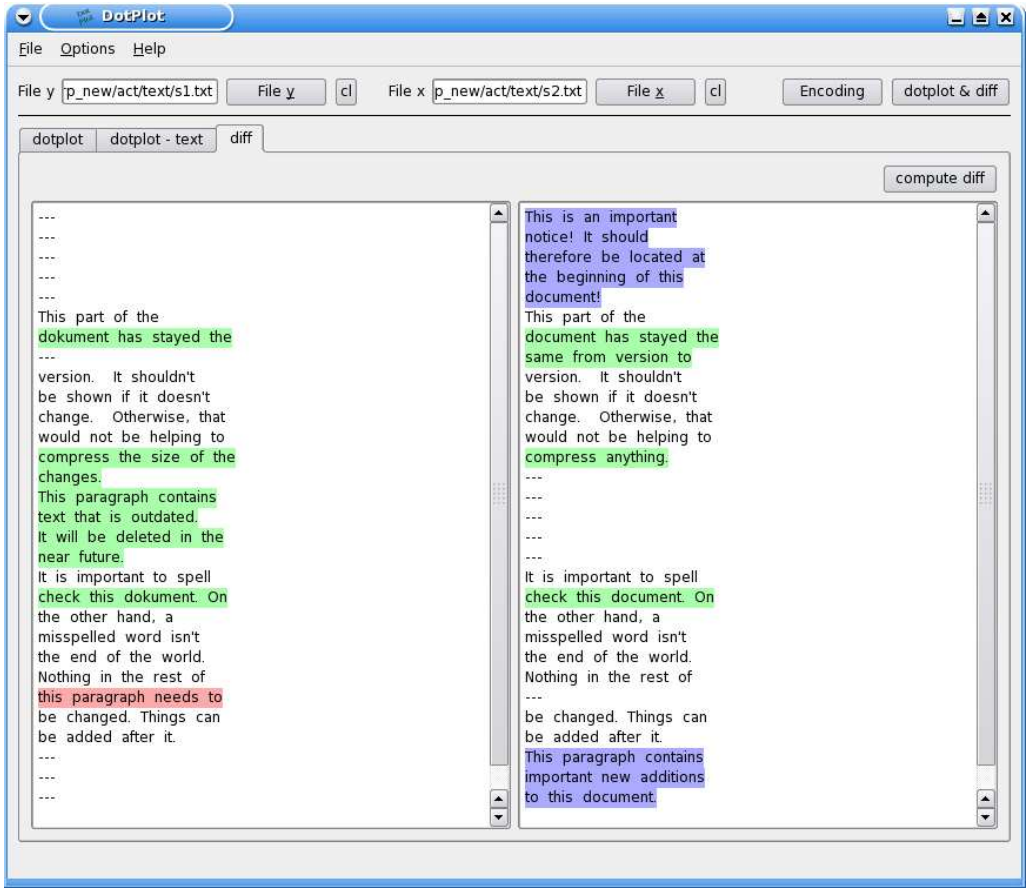

Obrázek B.1: Aplikace - nastavení




Obrázek B.2: Aplikace - dialog pro výběr kódování




Obrázek B.3: Aplikace - Práce s dotplotem




Obrázek B.4: Aplikace - Diff

# Dodatek C

# Obsah přiloženého CD

**Dotplot** – grafická aplikace

**Dotplot_cl** – command-line verze

**Uživatelská příručka**

**Programátorská dokumentace**

**Bakalářská práce** – tento text

**Příklady**

# Literatura

[1] Church K. W., Helfman J. I.: *A Program for Exploring Self-Similarity in Millions of Lines for Text and Code*, Journal of Computational and Graphical Statistics, June 1993.

[2] Helfman J. I.: *Similarity Patterns in Language*, IEEE Symposium of Visual Languages, 1994.

[3] Rybička J.: *Latex pro začátečníky* 2.vydání, KONVOJ, Brno, 1999.