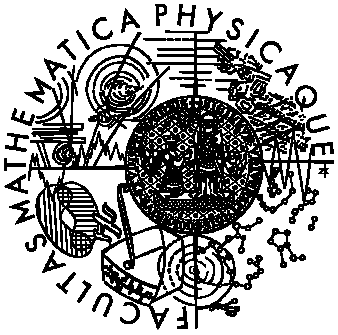
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Antonín Wimberský

Grafové algoritmy a závislostní parsing

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Pavel Pecina

Studijní program: Informatika, Obecná informatika

2007

Děkuji vedoucímu práce Pavlu Pecinovi za cenné podněty, připomínky a všestrannou pomoc.

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

V Praze dne 7.8.2007

Obsah

Název práce: Grafové algoritmy a závislostní parsing
Autor: Antonín Wimberský
Ústav: Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: Mgr. Pavel Pecina
e-mail vedoucího: pecina@ufal.ms.mff.cuni.cz

Abstrakt: V předložené práci studujeme praktické řešení problému závislostního parsingu pomocí grafového algoritmu hledání maximální kostry v orientovaném grafu (multigrafu). Výhodou tohoto přístupu je velmi snadné parsování jak projektivních, tak i neprojektivních větných konstrukcí. Parsovanou větu reprezentujeme orientovaným multigrafem, jehož vrcholy představují slova dané věty a hrany označují (potenciální) vazby mezi jednotlivými dvojicemi slov. Ohodnocení hran se získá z trénovacích dat, vypočítá se například jako pravděpodobnost vazby mezi danou dvojicí slov, případně v kombinaci s dalšími pokročilejšími metodami. Výslednou maximální kostru potom považujeme za závislostní strom dané věty.

Klíčová slova: parsing, neprojektivní, maximální kostra, orientovaný graf

Title: Graph algorithms and dependency parsing
Author: Antonín Wimberský
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Pavel Pecina
Supervisor's e-mail address: pecina@ufal.ms.mff.cuni.cz

Abstract: In the present work we study practical solution of dependency parsing's problem with help of graph algorithm for finding maximal spanning tree in oriented graph (multigraph). Advantage of this approach is very easily parsing of non-projective constructions. We represent the parsing sentence as an oriented multigraph, which vertices constitutes words of our sentence and edges symbolize (potential) relation between single pairs of words. Evaluation of edges we get from training data, it can be count for example as probability of relation between given two words, possibly in combination with other more advanced methods. Resulting maximal spanning tree gives then the dependency tree of our sentence.

Keywords: parsing, non-projective, maximal spanning tree, directed graph

Úvod

Cílem této práce je implementace závislostního parsingu pomocí grafového algoritmu. Obecné informace o závislostním parsingu včetně motivace můžeme nalézt v kapitole 1. V kapitole 2 je pak rozebrán postup, který použijeme. Kapitola 3 se zabývá použitými daty a v kapitole 4 se nachází informace o programu SVM (Support Vector Machine), který budeme v naší práci využívat. Kapitola 5 konečně přináší komplexní popis celého parseru včetně jeho použití. Výsledky experimentů provedených naším parserem pak uvádíme v kapitole 6.
Je třeba zdůraznit, že postup, který využíváme pro naší implementaci není originální, ale vychází z článku Ryana McDonalda *Spanning Tree Methods for Discriminative Training of Dependency Parsers* [7]. Narozdíl od uvedeného článku však k ohodnocování hran grafu představujícího větu (viz později – kapitola 2.1) nepoužíváme algoritmus MIRA (The Margin Infused Relaxed Algorithm), ale přikláníme se k jiným metodám, zejména výše zmíněné SVM.

Kapitola 1
Úvod do závislostního parsingu

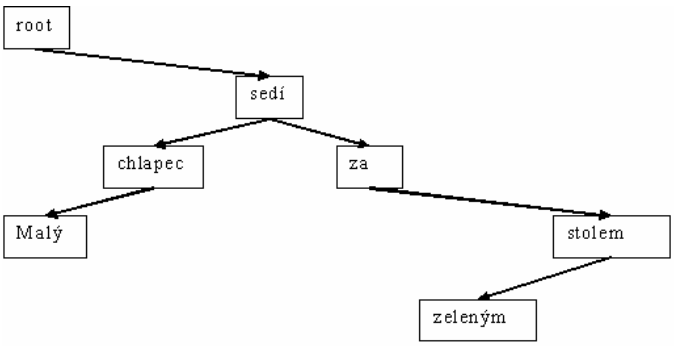
1.1 Závislostního parsingu

Závislostní parsing jako součást větného rozboru znamená popis závislostí mezi jednotlivými slovy ve větě. (V našem pojetí se mezi slova počítá i interpunkce a další „neslovní“ členy, jako třeba závorky a podobné symboly.) To, že určitá dvě slova jsou spolu ve vazbě znamená, že jedno z nich (*závislý člen*) je syntakticky závislé na tom druhém (*řídící člen*). Příkladem může být například slovní spojení „malé auto“, kde „auto“ tvoří řídící člen a adjektivum „malé“ je členem závislým. Pokud tento vztah zakreslujeme, směřujeme šipku ve směru od řídícího členu k závislému (lze se setkat i s opačným směrem šipky, náš zápis však lépe odpovídá výsledné datové struktuře).

Pro řešení našeho problému na počítači je důležité uvědomit si, že každé slovo závisí vždy právě na jednom jiném slově, a dále, že žádná skupina slov na sobě není závislá cyklicky (věta, jejíž závislostní graf by cyklus obsahoval, by neměla smysl a ani by nebyla syntakticky správně). Výsledná struktura, která popisuje závislosti v dané větě je tedy orientovaný strom, běžně nazývaný jako *závislostní strom* věty. Aby mohla být zachována podmínka, že každý větný člen závisí právě na jednom jiném členu, přidává se do závislostního stromu uměle kořen, označovaný jako NODE, a výsledný strom má tedy n+1 vrcholů (n je počet větných členů).

Na obrázku 1 je vidět příklad závislostního stromu pro větu *„Malý chlapec sedí za zeleným stolem.“*

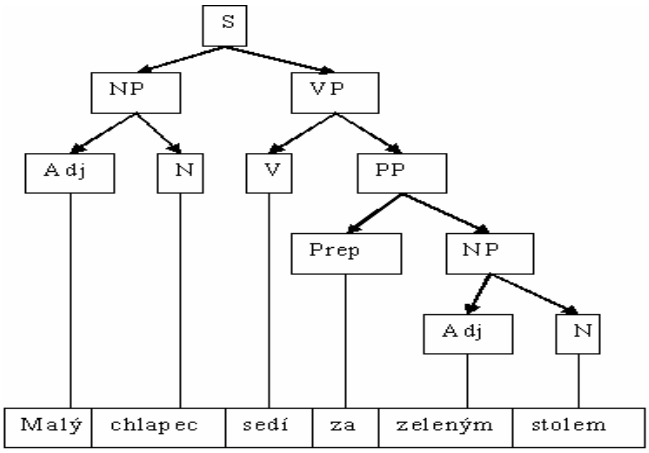
Obrázek 1 – Závislostní strom



V praxi se kromě závislostního stromu používá ještě jedna stromová reprezentace věty, a to *strom složkový*. Narozdíl od závislostního stromu se zde nezachycují závislosti mezi jednotlivými slovy, ale spíše bezprostředně sousedící složky. Tento přístup se používá spíše v anglickém jazyce, pro český jazyk se ukázal jako ne zcela vhodný.

Na *obrázku 2* je pro srovnání zobrazen složkový strom věty z předcházejícího příkladu. Složkový strom lze jednoduše sestavit na základě uzávorkování souvisejících složek:
((Malý)(chlapec))((sedí)((za)((zeleným)(stolem))))

Obrázek 2 – Složkový strom



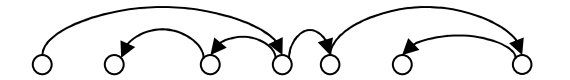
1.2 Neprojektivní konstrukce

Důvodem, proč se v českém jazyce příliš neujala reprezentace věty složkovým stromem je, že některé, v češtině obvyklé, větné konstrukce nedokáže správně zachytit. Jedná se o tzv. *neprojektivní větné konstrukce*, které obsahují jakési vybočení z větného rámce. Strom na obrázku 1 je projektivní. Lze to rozpoznat jednoduše tak, že pokud umístíme slova do jedné řádky v jejich původním pořadí a před ně umístíme kořen, můžeme mezi nimi zakreslit hrany závislostního stromu, aniž by se křížily (předpokládáme, že hrany vedeme nad řádkou). Z obrázku 3 je zřejmé, že náš první příklad je opravdu projektivní.

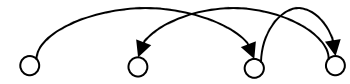
Příkladem neprojektivní konstrukce může být například věta „*Vánoční nadešel čas.*“ V tomto případě neexistuje způsob, jak nakreslit závislosti bez křížení hran (viz obrázek 4).

Neprojektivní věty se velmi často vyskytují vedle českého jazyka i v jazyku německém nebo holandském.

Obrázek 3 – Projektivní konstrukce



Obrázek 4 – Neprojektivní konstrukce



1.3 Využití závislostního parsingu

Závislostní parsing sám o sobě by byl sice zajímavou, ale nepříliš užitečnou aplikací. Je třeba si však uvědomit, že znalost závislostního stromu věty, je základním předpokladem ke správnému pochopení jejího významu. Zejména pro strojové aplikace, které se nějakým způsobem zabývají jazykem, je závislostní parsing velmi užitečný.

Asi největší oblastí, kde se závislostní parsing využívá, je oblast strojových překladů. Způsob překládání vět stylem „slovo od slova“ se neukazuje jako příliš vhodný, už z toho důvodu, že různé jazyky mají různý slovosled. Ke správnému překladu je třeba rozlišit jednotlivé větné členy, jako je podmět a přísudek (tedy základní stavebnou dvojici), různé přívlastky, objekty a doplňky. To by však bez znalosti závislostního stromu byl značný problém – stejné slovo může být v různém kontextu různým větným členem. Pokud ovšem známe vazby mezi slovy, není už těžké identifikovat základní stavební dvojici a další větné členy. To může pomoci nejen při sestavování správného slovosledu, ale také k odhadnutí správného významu jednotlivých slov, což je důležité zejména u vícevýznamových výrazů (jako třeba slovo „*stát*“).

Další aplikací, ve které může závislostní parsing nalézt uplatnění, je generování synonym. Pokud generujeme synonyma k určitému slovu v textu, je opět vhodné chápat význam onoho slova. Například ke slovu *kolo* může být synonymem slovo *bicykl*, ale také slovo *kružnice*, záleží jen na kontextu.

Určitě by se našla i mnohá další uplatnění, jako třeba vyhledávání v textu podle významu, významové porovnávání textů, nástroje pro kontrolu gramatiky a v neposlední řadě by se závislostní parsing mohl uplatnit i v umělé inteligenci – pro správné pochopení významů vět.

1.4 Klasické parsery

Většina stávajících systémů určených pro závislostní parsing funguje na principu bezkontextových gramatik a přepisovacích pravidel. Použitá gramatika jakožto generativní systém umožňuje generovat řetězce příslušné danému jazyku. Kromě generátoru se používá také akceptor, který je opakem generátoru a bývá popsán nějakým zásobníkovým automatem. Generátor a akceptor jsou pak základem parseru. Během své činnosti parser simuluje stavbu derivačního stromu, který vyjadřuje reprezentaci postupně aplikovaných pravidel pomocí stromu. [5]

Používají se dva různé přístupy – odshora dolu nebo zdola nahoru. V prvním případě se začíná s přepisem počátečního neterminálu (symbolu) gramatiky na levé straně pravidla za pravou stranu. Pravidla se volí v takovém pořadí, aby se jejich postupnou aplikací získala celá věta. Naopak při použití přístupu zdola nahoru se začíná s celou větou a postupným přepisem jejich částí pomocí pravidel se získá počáteční neterminální symbol.

Dále je uveden příklad generativní gramatiky a zpracování věty.

Příklad 1 – Generativní gramatika

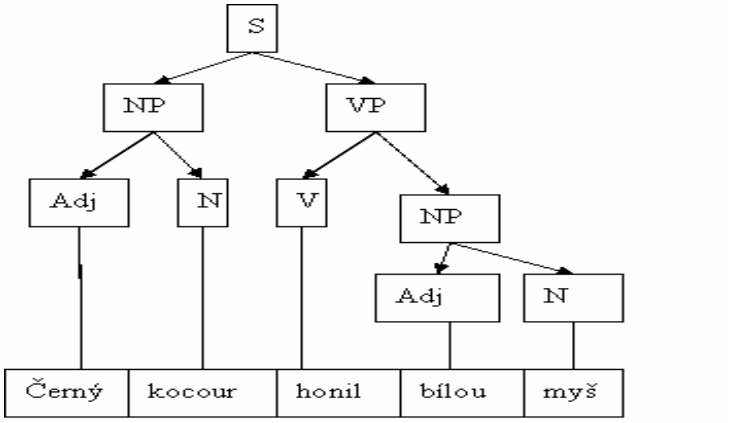
Mějme bezkontextovou gramatiku:

S → NP VP
VP → V NP
NP → N
NP → Adj N
V → honil | …
N → myš | *kocour* | …
Adj → bílou | *černý* |...

Použitím této gramatiky lze generovat například syntakticky správnou větu: „*Černý kocour honil bílou myš*.“

Posloupnost derivačních kroků (při nejlevější derivací) je pak následující:
S ⇒ NP VP ⇒ D N VP ⇒ Černý N VP ⇒ Černý kocour VP ⇒ Černý kocour V NP ⇒ Černý kocour honil NP ⇒ Černý kocour honil D N ⇒ Černý kocour honil bílou N ⇒ Černý kocour honil bílou myš

Zde je tento postup znázorněn pomocí stromu:



Jak je vidět, výsledkem tohoto postupu je složkový, nikoliv závislostní strom. I přesto, že existuje převod na závislostní strom (není to triviální úloha, ale je možné ji provést), vyplývá z toho omezení právě popsaného postupu. A sice neschopnost řešit neprojektivní větné konstrukce.

Mimo generativních gramatik se v praxi používají postupy založené na statistických údajích. Základem těchto postupů bývá jazykový korpus, který obsahuje rozsáhlé reprezentativní vzorky jazyka. Obvykle (alespoň v korpusech používaných pro parsing) kromě samotných textů obsahuje i morfologické údaje o jednotlivých slovech a samozřejmě informace o syntaktických vazbách. Funkce takovýchto parserů se obvykle skládá ze dvou částí – trénování a samotného parsování. Trénování spočívá v tom, že se procházejí data ze závislostního korpusu a přitom se sledují některé statistické hodnoty, například výskyt určitých slovních dvojic (bigramů) a zda je mezi nimi syntaktická závislost. Tyto údaje se pak použijí při parsování.

V této kategorii parserů se používají nejrůznější algoritmy, většinou se však jedná o algoritmy grafové (např. [13]). Mezi tyto parsery bude zcela nepochybně patřit i ten náš.

Kapitola 2

Využití grafového algoritmu

2.1 Řešení závislostního parsingu pomocí grafového algoritmu

Budeme vycházet z postupu uvedeném v článku [7]. V tomto pojetí budeme větu reprezentovat jako orientovaný graf (přesněji multigraf), kde vrcholy grafu odpovídají jednotlivým slovům a hrany odpovídají možným závislostním vazbám. Tedy možnost vazby mezi slovy *A* a *B*, kde *A* je člen řídící a *B* člen závislý, je zaznamenána jako orientovaná hrana od *A* do *B*. V našem grafu jsou propojena všechna slova stylem „každé s každým“, přičemž mezi každými dvěma slovy vedou dvě hrany s opačnou orientací (záleží nám na tom, který člen z dvojice je závislý a který řídící). Pouze k uzlu představujícímu kořen žádné hrany nevedou, vedou jen od něj. Každá hrana v grafu má nezáporné ohodnocení, které vyjadřuje míru „pravděpodobnosti,“ že vazba, kterou daná hrana vyjadřuje, je skutečná a že se objeví ve výsledném závislostním stromu. Narozdíl od [7] však pro ohodnocování hran nebudeme používat algoritmus MIRA, ale přikloníme se k jiným metodám (SVM). Přesným způsobem výpočtu hodnot hran se budeme zabývat podrobněji ve 4. kapitole, nyní předpokládáme, že máme k dispozici takovýto ohodnocený graf.

V této situaci aplikujeme algoritmus na hledání maximální kostry v orientovaném grafu. Tento algoritmus je jednoduchou modifikací původního algoritmu pro hledání minimální kostry v orientovaném grafu Chu-Liu-Edmons [1],[2]. Výsledná maximální kostra uvedeného grafu bude závislostním stromem věty, kterou graf reprezentuje.

Největší výhodou tohoto přístupu je, mimo jeho relativně snadné implementace, možnost velmi efektivního parsingu i pro neprojektivní větné konstrukce, přesněji se zpracování projektivních a neprojektivních konstrukcí při tomto postupu vůbec neliší.

2.2 Algoritmus na hledání maximální kostry v orientovaném grafu

Algoritmus pro hledání maximální kostry vychází původně z algoritmu pro hledání kostry minimální. Modifikace je však pouze drobná, spočívá jen v záměně nerovností, vlastní algoritmus zůstává nezměněn a je možné ho snadno použít i pro hledání maximální kostry. Dále v textu budeme mluvit již jen o algoritmech *pro hledání maximální kostry* a nebudeme výslovně popisovat, že se jedná o pozměněné algoritmy.

Pro hledání maximální kostry v grafu existuje značné množství algoritmů. Mezi všeobecně známé patří Kruskalkův a Primův (v českém prostředí též známý jako Jarníkův), méně známý je třeba Borůvkův algoritmus. Všechny tyto algoritmy

jsou velmi jednoduché i snadno implementovatelné pro jakékoliv neorientované grafy.

Náš problém je však podstatně složitější. Při hledání maximální kostry v orientovaném grafu požadujeme, mimo běžnou podmínku na souvislost a acyklitu, aby výsledkem byl orientovaný strom a to takový, kde všechny orientované hrany vedou směrem od kořene k listům.

Pro řešení naší úlohy tedy použijeme algoritmus Chu-Liu-Edmons, který nejenže dokáže efektivně nalézt maximální kostru orientovaného grafu, ale nedělá mu problém ani multigraf. Jedná se o rekurzivní algoritmus, který postupuje následujícím způsobem: Pro každý vrchol (vyjma kořene) vybírá příchozí hranu s největší kapacitou. Pokud výsledek neobsahuje cyklus, byla nalezena maximální kostra. Pokud vybrané hrany cyklus tvoří, je tento cyklus zkontrahován do jediného vrcholu a pokračuje se znova s upraveným grafem. Při návratu z rekurze se vrchol vzniklý kontrakcí samozřejmě opět rozebere. Přesný popis algoritmu včetně průběhu kontrakce následuje:

Chu-Liu-Edmons– algoritmus pro hledání maximální kostry v orientovaném grafu

VSTUP: seznam slov, ohodnocení hran
VÝSTUP: závislostní strom

Popis algoritmu – převzato z [3], upraveno a přeloženo

Chu-Liu-Edmons(G)

1. Vyhoď všechny hrany grafu G vedoucí do kořene. Gh buď výsledný graf.
2. Pro každý vrchol $n \in Gh$ vyber hranu s maximální vahou vedoucí do n. Th buď výsledný graf sestávající se z vybraných hran a příslušných koncových vrcholů.
3. Pokud Th neobsahuje cyklus, vrať Th (Th je maximální kostra).
4. Jinak najdi cyklus C v Th a vytvoř kontrahovaný graf
   Gc = Contract(Gh, C)
5. y = Chu-Liu-Edmons(G)
6. Najdi vrchol $x \in C$ takový, že hrana $(x', x) \in y$ a hrana $(x'', x) \in C$ pro nějaké vrcholy $x' \in y$ a $x'' \in C$.
7. Vrať $y \cup C - \{(x'', x)\}$

Contract(Gh = (V, E), C)

1. Gc buď podgraf G bez vrcholů z C
2. Do grafu Gc přidej vrchol c reprezentující cyklus C
3. Pro každý vrchol $x \in V - C$ najdi hranu $(x',x) \in E$, kde $x' \in C$
   Přidej hranu (c, x) do Gc, kde cena(c, x) = $\max_{x' \in C}$ cena(x', x)
4. Pro každý vrchol $x \in V - C$ najdi hranu $(x, x') \in E$, kde $x' \in C$
   Přidej hranu (x, c) do Gc, kde
   cena(x, c) = $\max_{x' \in C}$ [cena(x, x') – cena(pred(x'), x') + cena(C)],
   kde pred(v) je předchůdce vrcholu *v* v cyklu C
5. vrať Gc

Časová složitost algoritmu při naivní implementaci je $O(n^3)$, z toho $O(n^2)$ zabere práce v jednom rekurzivním volání a máme až n volání. Při vhodné implementaci je však možné dosáhnout časové složitosti $O(n^2)$ [8].

2.3 Implementace algoritmu na hledání maximální kostry Chu-Liu-Edmons

V našem pojetí je graf G reprezentován maticí sousednosti. Ta má tu výhodu, že je z ní možné snadno, bez dohledávání, přečíst jak hrany, které z jednotlivých vrcholů vycházejí (řádky matice), ale také hrany, které do nich vstupují (sloupce). Pro matici velikosti NxN (pro N rovno počtu vrcholů) je také matice dobře využita - vyjma diagonály a prvního sloupce pro kořen je zcela zaplněna.

Ve skutečnosti však pro uložení grafu použijeme matici o velikost $2N^2$, N je počet vrcholů. Takto zvětšená matice umožní snadné přidávání vrcholů (těch, co vznikají v důsledku kontrakce) v průběhu algoritmu, aniž by docházelo ke zbytečnému přesouvání a kopírování údajů.

Dále využijeme jako pomocnou strukturu vektor obsahující maximální (ve smyslu ohodnocení) hrany vstupující do jednotlivých vrcholů. Tento vektor načteme pouze jednou – na začátku – a dále ho už jen vhodným způsobem obnovujeme. Díky tomuto vektoru snížíme práci v průběhu rekurzivního volání na O(N), protože nemusíme příslušné hrany hledat v lineárním čase.

Během celé práce algoritmu přistoupíme ke každé hraně grafu nejvýše konstantě krát. Výsledná časová složitost je tedy $O(N^2)$, protože počet všech hran včetně hran přidaných během kontrakcí je vždy menší než $4N^2$ (vyplývá z velikosti matice $(2N)^2$.

Podrobnější popis implementace algoritmu včetně všech technických detailů zde uvádět nebudeme, kompletní zdrojový kód v jazyce C++ včetně komentářů se nachází na přiloženém CD.

Kapitola 3

Pražský závislostní korpus (Prague Dependency Treebank)

3.1 Základní údaje

PDT (Prague Dependency Treebank) vznikl jako reakce na podobný projekt v anglickém jazyce - Penn Treebank. Jedná se o rozsáhlý soubor manuálně anotovaných textů v češtině. Jako zdroje těchto textů byly použity následující tiskoviny: Lidové noviny, Mladá fronta Dnes, Českomoravský profit a časopis Vesmír.[10]

Data v PDT jsou anotována na 3 rovinách - na morfologické, analytické a tektogramatické rovině. V morfologické rovinně jsou k jednotlivým slovům (přesněji slovním jednotkám - počítáme mezi ně i interpunkci apod.) přiřazeny rozšiřující informace "lemma" a "tag". Lemma značí základní tvar daného slova (např. infinitiv u slovesa nebo 1.pád jednotného čísla u podstatného jména) a tag obsahuje morfologické informace jako slovní druh, rod, pád, číslo apod. (viz PŘÍLOHA A).

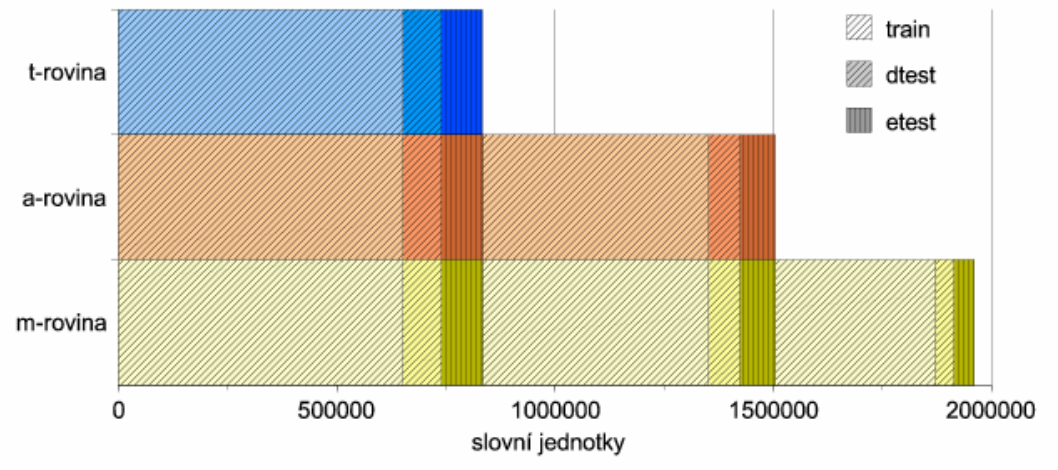
„Na analytické rovině jsou jednotlivým větám přiřazeny odpovídající závislostní stromy. Každý prvek morfologické roviny zde odpovídá právě jednomu uzlu. Většina hran reprezentuje závislostní vztah, ostatní odrážejí různé další lingvistické či technické jevy, např. koordinaci, apozici, interpunkci apod. Zaznamenáno je i lineární uspořádání uzlů, odpovídající pořadí slovních jednotek ve větě, což umožňuje „správné“ grafické zobrazení stromu.“ [10]

Poslední rovina v PDT je rovina tektogramatická. Zde jednotlivé uzly zachycují pouze „plnovýznamová“ slova, a tedy počet uzlů věty neodpovídá počtu slov téže věty v morfologické rovině. Touto vrstvou se více zabývat nebudeme, v naší práci si vystačíme s vrstvou analytickou.

Součástí každé roviny jsou samozřejmě i příslušné údaje z nižších rovin. Důvodem, proč nepoužít rovnou nejvyšší vrstvu (tektogramatickou) je, že čím vyšší vrstva je, tím méně dat obsahuje (viz obr. 5 - rozdělení dat PDT).

Data analytické roviny jsou uložena ve formátu PDT (podrobněji v další podkapitole) a jsou rozdělena do tří typů souborů: soubory *.w obsahují pouze slovní jednotky, *.m obsahují data z morfologické vrstvy a v *.a jsou popsány příslušné závislostní stromy. Tato data jsou rozdělena do celkem deseti adresářů, osm z nich slouží jako trénovací data (train1–train8), adresář dtest obsahuje vývojová testovací data a konečně poslední dtest je určen jako evaluační. Množství dat v jednotlivých adresářích je přibližně stejné.

Obrázek 5 – Rozdělení dat PDT [převzato z [10]]



3.2 PML

Pro reprezentaci dat v PDT se používá zvláštní formát – PML (Prague Markup Language). Jedná se o formát dat založený na XML, který je určen pro reprezentaci lingvistické anotace textů ve všech čtyřech (počítáme i slovní rovinu) rovinách anotace. Každý PML soubor obsahuje mimo vlastní anotace i hlavičku, z níž je odkazováno na PML schéma daného souboru. Kromě toho hlavička obsahuje i seznam všech souborů, na které je odkazováno a případné další potřebné informace.

Jednotlivé roviny anotace jsou popsány v souborech PML schéma. Jedná se o formalizaci abstraktního anotačního schématu pro konkrétní roviny anotace. PML schéma obsahuje informace o elementech, které se vyskytují na dané rovině a o jejich struktuře.[10]
Vlastní anotace je vyjádřena pomocí XML elementů a atributů, které jsou pojmenovány podle příslušného PML schématu. PML formát obsahuje dostatek prostředků pro jednotnou reprezentaci většiny běžných anotačních konstrukcí. Obsáhlejší informace o PML a jeho struktuře lze nalézt například v [9] nebo v [10].

3.3 Využití PDT pro parsing

Pro řešení našeho problému - závislostního parsingu, je PDT nezbytnou součástí. Trénovací data analytické roviny využijeme k nalezení správného ohodnocení hran a testovací data budou sloužit k ověření funkčnosti celého systému.

Samotné načítání trénovacích dat bude probíhat tak, že budou zvlášť zpracovávány údaje o jednotlivých bigramech (dvojících slov vyskytujících se spolu ve větě; záleží zde na pořadí – bigramy *a-b* a *b-a* jsou rozdílné). Každý bigram je v grafu představujícím větu (viz oddíl 2.1) reprezentován právě jednou hranou, a tedy informace, které získáme o jednotlivých bigramech z PDT můžeme použít k ohodnocení příslušných hran.

Relevantní údaje, které můžeme o jednotlivých bigramech získat, jsou jednak morfologické vlastnosti, dále informace o poloze jednotlivých slov ve větě a samozřejmě z trénovacích dat i údaj, které bigramy představují skutečné vazby. Způsoby, jak z těchto údajů získat ohodnocení hran jsou popsány v následující kapitole.

Kapitola 4

Ohodnocování hran

4.1 Klasická pravděpodobnost

Asi nejjednodušší metodou ohodnocování hran (v grafu představujícím větu) je odhad klasické pravděpodobnosti vazby. Pro každou hranu se spočítá pravděpodobnost, že daná hrana představuje skutečnou vazbu, z trénovacích dat jako
P(A_B) = (Počet nalezených vazeb dvojice A_B / počet výskytů dané dvojice),
kde počtem výskytů dané dvojice se myslí, kolikrát se daná dvě slova vyskytla spolu v jedné větě v trénovacích datech. Pokud se daná dvojice ve větě spolu nikdy nevyskytla, není tato hodnota definována a můžeme ji nahradit buď nulou nebo některou jinou hodnotou.

Tento způsob ohodnocení, ač je implementačně velmi jednoduchý, je také velice přesný a pro dostatečně velkou množinu trénovacích dat by byl sám o sobě dostačující. Trénovací data, která máme k dispozici z PDT však bohužel tak rozsáhlá nejsou. Při reálných experimentech prováděných na testovacích datech (dtest) se ukazuje, že pro většinu hran (cca 2/3) je klasická pravděpodobnost nedefinována (viz kapitola 6). Z toho důvodu přistupujeme k dalším hodnotícím metodám popsaným dále.

Mimo klasické pravděpodobnosti je možné využít i některých dalších pravděpodobnostních hodnocení – asociačních měr. Pracuje se přitom z tzv. kontingenční tabulkou, která pro každý bigram A_B určuje čtyři hodnoty *a*, *b*, *c* a *d*, kde *a* je počet vazeb A_B v trénovacích datech, *b* znamená počet vazeb A_X, kde X ≠ B, *c* je naopak počet vazeb X_A, X ≠ A a d je doplněk do celkového počtu bigramů (což je totéž, jako počet vazeb X_Y, kde X ≠ A a Y ≠ B). Na obrázku 6 je znázornění takové tabulky.

Obrázek 6 – Kontingenční tabulka

A_B	B	not(B)
A	a	b
not(A)	c	d

Přestože z těchto hodnot lze dopočítat mnoho relevantních údajů (viz příloha D, nebo podrobněji [11]), narážíme zde na stejný problém jako u klasické pravděpodobnosti, totiž na velikost množiny trénovacích dat. U dvojice slov, která se v trénovacích datech nevyskytovala, žádná z těchto metod příliš nepomůže.

4.2 Support Vector Machine

Support Vector Machine je jednou z metod strojového učení. Její funkce je následující. Na základě vstupních trénovacích dat vytvoří ohodnocovací model a ten použije k ohodnocení dat testovacích.

Vstupní data budou mít podobu $\{(c_1, x_1), (c_2, x_2), ... ,(c_n, x_n)\}$. Zde každé c_i nabývá hodnot 1 nebo -1 a určuje třídu i-tého příkladu (zda se jedná o kladný či záporný příklad); x_i je p-dimenzionální reálný vektor, který reprezentuje polohu bodu v p-dimenzionálním prostoru. Práce SVM spočívá v tom, že hledá optimální p-1 dimenzionální nadrovinu, která oddělí tyto body podle jejich třídy. Optimální nadrovinou se zde myslí taková nadrovina, která obě třídy bodů rozděluje co nejvíce a co nejlépe, tedy se nachází mezi oběma třídami bodů a zároveň je od nejbližšího bodu co nejvíce vzdálená. Více o této problematice lze nalézt například v [4] nebo v [12].

V naší práci použijeme implementaci SVM_light od Thorstena Joachimse (konkrétně ve verzi 6.01; podrobné informace a download lze nalézt na webové stránce *http://www.joachims.org*). Skládá se ze dvou částí. První část provádí samotné trénování a jejím výstupem je ohodnocovací modul, na základě kterého druhá část implementace provede ohodnocení testovacích dat. Ve vstupním souboru jsou data uložena textově, přičemž na každé řádce se nachází právě jeden příklad (tedy dvojice (c_i, x_i)) v následující podobě:

[c] [1:vlastnost_1] [2:vlastnost_2] ... [n:vlastnost_n],

kde c je třída příkladu (tedy buď 1 nebo -1) a jednotlivé vlastnosti jsou reálná čísla. Pro reálné použití je ještě důležité vědět, že vlastnosti s nulovou hodnotou není nutné zapisovat – tedy údaje typu [X:0] lze vynechat, což v praktickém použití může znamenat značnou úsporu místa.

4.3 Využití SVM

Při řešení závislostního parsingu využijeme SVM následovně. Z trénovacích dat načteme množství vektorů obsahující předem určené vlastnosti. Přesněji pro každý bigram slov (myslíme dvojici slov A_B vyskytující se spolu ve větě) a pro dvojice „*Kořen_slovo*“ načteme jeden vektor, který použijeme jako příklad pro trénování SVM. V případě, že mezi danou dvojicí slov A_B je vazba (zjistíme z analytické roviny), jedná se o kladný příklad, v opačném případě se jedná o příklad záporný. To, že záporné příklady budou převažovat, je asi zřejmé: V každé větě je n kladných příkladů (n-1 vazeb mezi slovy a 1 vazba vedoucí od kořene) a n^2-n-1 příkladů záporných (n^2-n hypotetických vazeb mezi slovy a n-1 hypotetických vazeb od kořene mínus počet reálných vazeb, hrany vedoucí ke kořeni nenačítáme vůbec – jsou zbytečné).

Vlastnosti, které budou příklady obsahovat můžeme rozdělit do několika typů. Předně se bude jednat o některé morfologické vlastnosti (zjistíme je z morfologické roviny PDT). Ty budou zaznamenány pro obě slova z dvojice, jejich zápis bude vypadat následovně: $[(s_1,v_1) ... (s_1,v_n), (s_2,v_1) ... (s_2,v_n)]$, kde (s_x,v_y) je y-tá vlastnost x-tého slova. Každá z těchto vlastností bude nabývat hodnoty 0 nebo 1 – buď nenastává, nebo nastává. Tento postup lze použít proto, že počet morfologických vlastností a počet jejich hodnot je předem znám. V PDT je zachycováno 15 různých vlastností (viz příloha A), které dohromady mohou

nabývat maximálně 145 různých hodnot. Tyto různé hodnoty použijeme jako vlastnosti (typu 1/0) příkladů, takže pokud budeme chtít využít všechny morfologické vlastnosti obou slov, bude každý trénovací příklad obsahovat 2x 145 hodnot.

Další údaje, které můžeme k příkladům přidat, již nebudou jen údaje typu 1/0, ale budou moci nabývat různých hodnot. Jedná se například o údaje o poloze slov, tedy jejich relativní vzdálenost ve větě (myslíme vzdálenost slov od sebe), dále jejich vzájemnou polohu (které slovo stojí ve větě dříve) a samozřejmě i absolutní polohu ve větě přepočítanou do jednotného intervalu (např. do intervalu 1-100, tuto hodnotu budeme tedy počítat jako (pozice slova ve větě) / (počet slov věty) * 100).

Dále je možno k příkladům připojit hodnotu vyjadřující pravděpodobnost vazby mezi popisovanou dvojicí spočítanou na základě statistických údajů (viz oddíl 4.1). Zde je ovšem třeba dát si pozor na nedefinované hodnoty.

Samozřejmě je možné přidat nejrůznější další vlastnosti a jejich kombinace, použité vlastnosti budou podrobně popsány v 6. kapitole.

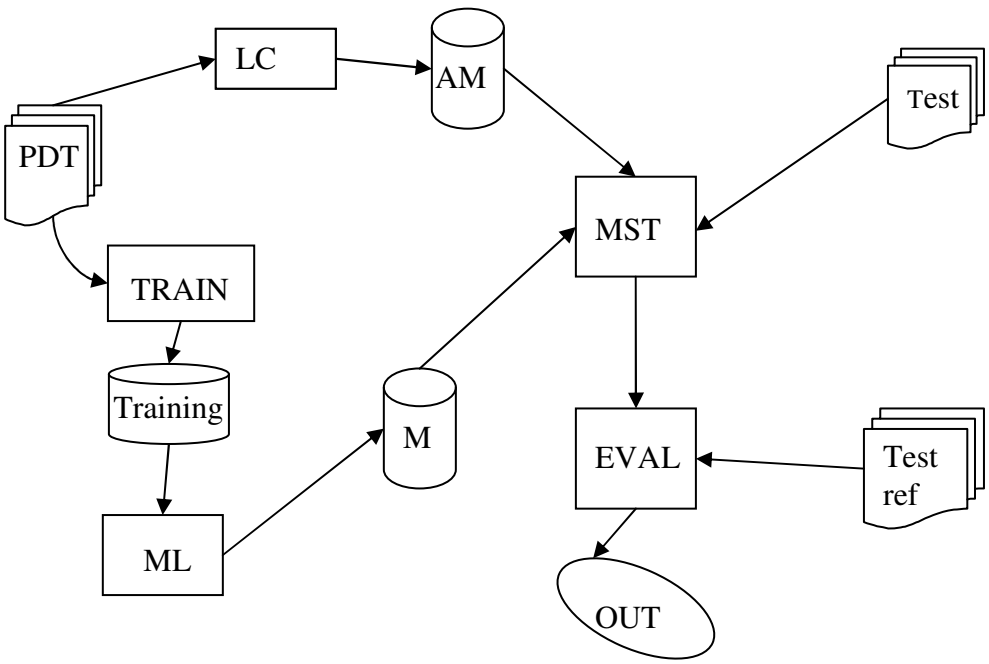
Kapitola 5

Parser

5.1 Struktura systému

Celková struktura parsovacího systému je modulární. Jádro tvoří algoritmus na hledání maximální kostry v orientovaném grafu (MST), další části pak řeší načítání trénovacích dat – modul LC (Load and Count) načítá statistické údaje pro pravděpodobnostní výpočty a modul TRAIN načítá trénovací data pro SVM. Součástí naší práce je také vyhodnocení úspěšnosti jednotlivých metod ohodnocování hran, proto předpokládáme zahrnutí modulu, který bude porovnávat podobnost výsledků na testovacích datech s výsledky připravenými člověkem – modul EVAL. Na *obrázku 7* je schematicky znázorněn celý systém.

Obrázek 7 – Modulární struktura parseru



5.2 Popis součástí

5.2.1 MST (Maximal spanning tree)

Tento modul obsahuje vlastní algoritmus na hledání maximální kostry v orientovaném grafu. Použijeme algoritmus Chu-Liu-Edmons popsaný v části 2.2.

Vstupem tohoto modulu jsou jednak testovací data z PDT a soubor s ohodnocením hran, který získáme jako výstup SVM pro příslušná testovací data, a nějaký hodnotící model získaný z trénovacích dat PDT. Pro samostatné a popřípadě kombinované použití pravděpodobnostních výpočtů (viz část 4.1) je dále vstupem soubor obsahující statistické údaje o výskytu slov a bigramů (výstup modulu LC).

Modul postupně načítá jednotlivé věty z testovacích dat, sestaví příslušný graf na základě vstupních souborů a použité metody ohodnocování hran a nalezne maximální kostru tohoto grafu. Tu pak zapíše do výstupního souboru (souborů) ve formátu PML.

5.2.2 LC (Load and count)

Vstupem tohoto modulu jsou trénovací data z PDT. Pro každou dvojici slov vyskytujících se spolu ve větě se načítají následující údaje: kolikrát se daná dvojice slov vyskytovala spolu ve větě a kolikrát byla mezi danou dvojicí slov vazba (samozřejmě uvažujeme v rámci celých trénovacích dat). Formát souboru je: ”slovo1_slovo2~X~A”, kde X je počet výskytů dvojice a A je rovno počtu vazeb mezi touto dvojicí.
Jelikož se následně klasická pravděpodobnost vazby v testovacích datech počítá jako počet výskytů lomeno počet vazeb, vynecháváme pro úsporu místa ty dvojice, jež ani jednou nebyla ve vazbě (0 / cokoliv je vždy 0).

Pro sestavení kontingenční tabulky (viz oddíl 4.1) slouží druhý výstupní soubor. Ten bude obsahovat seznam slov vyskytujících se v PDT a údaj o tom, kolikrát z daného slova nějaká vazba (hrana) vychází, a kolikrát je naopak dané slovo vázáno na některé jiné. Formát tohoto souboru je: “slovo~počet výstupních vazeb~počet vstupních vazeb“. Tento soubor by sám o sobě pro dopočítání kontingenční tabulky nestačil, ale spolu s prvním souborem už dává dostatečné množství informací pro výpočet tabulky pro jakoukoliv dvojici slov. Přesněji hodnota *a* kontingenční tabulky je zaznamenána přímo v prvním souboru (hodnota A), hodnota *b* lze dopočítat jako B-A, hodnota *c* jako C-A a d je doplněk do celkového počtu bigramů.

Pro efektivní implementaci tohoto modulu je použita hašovací tabulka, kde hašovacím klíčem je textový řetězec obsahující jednotlivé dvojice slov („slovo1_slovo2). Pro tyto klíče se ukládají příslušné hodnoty X a A (počet výskytů a počet vazeb). K načítání rozšířených statistických údajů (druhý výstupní soubor) se použije druhá hašovací tabulka, jejímiž klíči budou jednotlivá slova, ke kterým se budou ukládat příslušné hodnoty B a C.

5.2.3 TRAIN

Tato část se stará o načítání trénovacích dat pro SVM. Přesněji z trénovacích dat načte sadu vektorů ve formátu SVM určené pro trénování. Podruhé se tento modul použije při načítání příslušných testovacích dat, která jsou třeba pro zpětnou klasifikaci pomocí SVM. Je třeba dát pozor, aby obě sady vektorů byly stejného typu, tedy aby obsahovaly stejnou sadu sledovaných vlastností (fíčur).

Postup načítání je následující. Načítání zdrojových dat probíhá po větách. Nejprve se pro každé slovo S ve větě načtou údaje o všech dvojicích NODE_S.

Dále se postupuje stylem „každé slovo s každým“, tedy načtou se všechny možné bigramy (dvojice slov) z dané věty. Pak se pokračuje další větou.

Údaje, které se načítají pro jednotlivé bigramy, jsou popsány v oddílu 4.3, formát zápisu v oddílu 4.2 a příklady jsou uvedeny v kapitole 6.

5.2.4 EVAL (Evaluation)

Jako poslední uvádíme modul, který slouží ke konečnému vyhodnocení správnosti parsingu. Úkolem této části je porovnat závislostní stromy, které jsme získali z modulu MST (testovaná složka), se závislostními stromy, které byly pro stejná trénovací data sestavena člověkem (tedy příslušnou analytickou rovinou PDT; tuto složku označme jako vzorovou). Jelikož obě porovnávané složky jsou zapsány ve stejném formátu (PML), je činnost této části velmi jednoduchá. Skládá se pouze z načtení příslušných dat a jejich následným porovnáním.

Postupuje se opět po větách, což v tomto případě znamená po jednotlivých závislostních stromech. Oba porovnávané stromy se rozloží na jednotlivé dvojice slov. Každá taková dvojice samozřejmě představuje vazbu mezi těmito dvěma slovy. Aby při porovnávání nemohlo dojít k omylu (například pro věty, ve kterých se vyskytuje více stejných slov), reprezentujeme tyto dvojice slov jejich pořadím ve větě.

Všechny dvojice získané ze vzorové složky považujeme za správné, počítáme, kolik odpovídajících (a tedy správně určených) dvojic se nachází v testované složce. Tedy za správnou dvojici považujeme tu, která se nachází rovněž ve vzorové části.

Úspěšnost parsingu dané věty spočteme jako podíl počtu správně určených dvojic testovací složky a počtu všech dvojic z věty. Celkovou úspěšnost parsingu pak spočteme jako podíl počtu všech (ze všech vět) správně určených dvojic testovací složky a počtu všech dvojic.

5.3 Průběh parsingu

Předpokládáme, že máme k dispozici všechny výše popsané moduly a sadu trénovacích a testovacích dat zapsaných ve formátu PML. Dále máme připravenu sadu relevantních vlastností (fíčur), které budeme používat při trénování pomocí SVM. Budeme postupovat následovně:

Nejprve pomocí modulu LC načteme soubory se statistickými údaji pro odhady pravděpodobností. V této chvíli, pokud chceme provádět parsing jen na základě pravděpodobnostního ohodnocování hran, může již k němu přejít. V opačném případě přejdeme k modulu TRAIN. Načteme dvě sady příkladů pro SVM, které budou obsahovat shodné vlastnosti. První sadu, kterou načteme z trénovacích dat, použijeme pro trénování SVM. Po dokončení trénování využijeme druhou sadu příkladů (získanou z testovacích dat) a trénovací model, který jsme získali ze SVM, ke klasifikaci – opět pomocí SVM. Výsledkem bude soubor obsahující ohodnocení hran.

Nyní již máme připraveno vše pro samotný parsing, který provedeme pomocí modulu EDMONS na základě testovacích, dat souboru s ohodnoceními hran a případně souborů se statistickými údaji (pokud pravděpodobnostní hodnoty

nebyly zapracovány do dat pro SVM, lze je zde dodatečně zkombinovat se souborem obsahujícím ohodnocení hran; viz konkrétní příklady v kapitole 6).

Nakonec pomocí modulu EVAL vyhodnotíme úspěšnost daného testu parsingu.

Kapitola 6

Experimenty a jejich výsledky

6.0 Poznámka

Všechny zde uvedené testy byly provedeny na datech ”dtest” z PDT.

6.1 Pravděpodobnostní ohodnocování hran

Při použití ohodnocování hran pomocí klasické pravděpodobnosti (viz kapitola 4.1), se každé hraně znázorňující vazbu mezi slovy A a B přiřadí hodnota

P(A_B) = (Počet nalezených vazeb dvojice A_B / počet výskytů dané dvojice)

Samozřejmě P(A_B) je něco jiného, než P(B_A).

Pro odhad této pravděpodobnosti jsme zvolili dva různé přístupy. Při prvním jsme do vzorce dosadili jednotlivá slova tak, jak se vyskytovala ve větě. Druhou možností pak bylo použít místo slov jejich lemmata. Pro oba postupy bylo samozřejmě třeba provést zvlášť trénování (myslíme načtení souboru s pravděpodobnostmi (viz 5.2.2)), v obou případech jsme totiž sledovali rozdílné údaje. Přestože se může zdát použití lemmatizovaných tvarů méně přesné (což ostatně opravdu je), jeho výhodou je větší množství dvojic, pro něž můžeme pravděpodobnost vazby zjistit. Výsledky tomu také odpovídají:

Při použití slovních tvarů dosáhla celková úspěšnost parsingu *32%*, při použití lemmat *36%*.

Pro zlepšení těchto výsledků máme ještě další možnost – při načítání statistických údajů vynecháváme ty dvojice, které spolu ani jednou nebyly ve vazbě (viz 5.2.2). Potom ale za nedefinované hodnoty považujeme i ty, kterým by podle těchto údajů jednoznačně náležela nulová hodnota. Pro skutečně nedefinované hodnoty pak můžeme použít *vyhlazování* – přiřadíme jim nějakou malou nenulovou hodnotu. Použitím tohoto postupu jsme dosáhli těchto výsledků:
úspěšnost *34%* pro slovní tvary, *37%* pro lemmata. Za nedefinované pravděpodobnostní hodnoty jsme dosadili *0.05*. Přestože tento postup vykazuje o něco lepší výsledek, rozhodli jsme se ho v dále nepoužívat. Za cenu malého zlepšení totiž neúnosně (více než 50-ti násobně) narůstá soubor se statistickými údaji a tím i tím i hašovací tabulka, kterou používáme při jeho zpracování. V dalším textu tedy nebudeme nulové a nedefinované hodnoty rozlišovat (nedopouštíme se tím žádného velkého prohřešku – nulové hodnoty z velké části vycházejí nulové právě proto, že o nich nemáme dostatek údajů).

Použití některých dalších pravděpodobností je v příloze C, protože žádná z těchto hodnot nedosahuje úspěšnosti při klasické pravděpodobnosti, budeme dále používat jen klasickou pravděpodobnost.

6.2 Ohodnocování hran pomocí SVM

6.2.1 Sledované vlastnosti

V tomto oddíle budou pro jednotlivé testy rozhodující vlastnosti, které pro bigramy sledujeme (viz 4.3). Tyto vlastnosti můžeme rozdělit do následujících kategorií:

- *morfologické*
- *poziční*
- *hraniční*
- *pravděpodobnostní*

Morfologické vlastnosti jsou vlastnosti přímo zjistitelné z morfologické roviny PDT. Je jich celkem 145 (viz příloha A), nebudeme však nutně využívat vždy všechny. Morfologické vlastnosti samozřejmě sledujeme pro obě slova v každém bigramu.

Poziční vlastnosti zjistíme z polohy slov ve větě, zaznamenáváme tyto údaje:

- *relativní vzdálenost* – tedy vzdálenost slov v dané dvojici od sebe, vzdálenost slova od kořene dodefinujeme uměle:

  NODE_“.“ = 0
  NODE_sloveso = ABS(sloveso)
  NODE_(cokoliv jiného) = N

  N je počet slov věty, ABS(slovo) je absolutní pozice slova ve větě. Tato definice je jakousi heuristikou pro hrany vedoucí od kořene – v souladu s funkcí této vlastnosti zde platí, že nižší hodnota odpovídá větší pravděpodobnosti vazby.
- *absolutní vzdálenost* – pozice slova ve větě, přepočítána do jednotného intervalu (1-100), sledujeme pro obě slova dvojice
- *poloha* – zjišťujeme, které slovo bigramu stojí ve větě dříve, tuto vlastnost zapisujeme jako jednu hodnotu pro celý bigram, tedy 0, pokud první slovo bigramu se ve větě vyskytuje dříve, než druhé, 1 v opačném případě.

Hraniční vlastnost je vlastně jen jedna, teoreticky by ji bylo možno zařadit i do vlastností pozičních, nicméně zde sledujeme mimo polohy slov i rozdělení věty na celky. Vyjadřuje vzdálenost podle větných úseků – tedy počet čárek, které se vyskytují mezi oběma slovy bigramu. Kořen je přiřazen k prvnímu úseku věty, uměle se dodefinovává vazba NODE_“poslední slovo“ jako 0, protože pravděpodobnost této vazby je velmi vysoká (posledním slovem bývá tečka).

Pravděpodobnostní vlastnosti jsou popsány již výše (4.1, 6.1). Budeme používat převážně jen klasickou pravděpodobnost, případně doplněnou o *vyhlazování* nedefinovaných hodnot.

6.2.2 Testy bez pravděpodobností

V této části vynecháme pravděpodobnostní vlastnosti. Pro trénování SVM voláme příkazem: *"svm_light_learn [train] –z r –m 4000 [model]"*,
kde přepínač *–z r* zapíná regresi, přepínač *–m* určuje velikost bufferu (ovlivňuje rychlost výpočtu).
Jako trénovací data použijeme *omezená* trénovací data z PDT, konkrétně adresář Train-1. Takové omezení značně urychlí trénovací proces a přitom nutně neznamená zhoršení úspěšnosti – i tak získáme více než dostatečné množství trénovacích příkladů. Takto omezená trénovací data již budeme pro trénování SVM používat vždy, bez zvláštního upozornění.

Následuje několik testů:

Test1:
morfologické vlastnosti – údaje o slovním druhu a pádu
poziční – relativní vzdálenost a poloha
úspěšnost: 27%

Test2:
morfologické vlastnosti – údaje o slovním druhu, poddruhu a pádu
poziční – relativní vzdálenosti a poloha
úspěšnost: 28%

Oba výsledky jsou dost slabé. Problém je patrně v trénování SVM. V našich trénovacích datech značně převažují negativní příklady nad pozitivními. To vyplývá z podstaty dat – pro větu o n slovech máme v příslušném úplném grafu n^2-1 hran, ale jen n hran reprezentuje skutečné vazby. Poměr negativních příkladů ku kladným je tedy $\frac{n^2-n-1}{n}$, přesná hodnota bude záviset na průměrné délce vět v trénovacích datech.
Měřením na našich trénovacích datech jsme dospěli k poměru cca 23:1 ve prospěch negativních příkladů.

Ve všech následujících testech použijeme *vyrovnaná* trénovací data, tedy náhodně vypustíme některé (vlastně většinu) negativních příkladů tak, aby se poměr pozitivních a negativních příkladů vyrovnal.

Test3
morfologické vlastnosti – údaje o slovním druhu a pádu
poziční vlastnosti – relativní vzdálenost a poloha
úspěšnost: 34%

Test4
morfologické vlastnosti – údaje o slovním druhu, poddruhu a pádu
poziční vlastnosti – relativní vzdálenost a poloha
úspěšnost: 36%

Vyrovnáním pozitivních a negativních příkladů jsme tedy dosáhli znatelného zlepšení.

Test5
kompletní morfologické, poziční i hraniční vlastnosti
úspěšnost: 38%

I přes veškerou snahu se nám zatím nepodařilo získat právě nejlepší výsledky. Předpokládáme, že problém je ve správném použití SVM. V dalším testu tedy změníme systém načítání trénovacích dat, přesněji morfologických vlastností. Ty jsme až doposud načítali „lineárně“, tedy stylem:

$1{:}v_1\ 2{:}v_2\ \ldots\ n{:}v_n\ 1{:}w_1\ 2{:}w_2\ \ldots\ n{:}w_n$,

kde v_i je i-tá vlastnost prvního slova a w_i je i-tá vlastnost druhého slova.
Nyní zkusíme ty samé údaje zapsat „kvadraticky“, tedy následovně:

$1{:}(v_1\&w_1)\ 2{:}(v_1\&w_2)\ \ldots\ n{:}(v_1\&w_n)\ n{+}1{:}(v_2\&w_1)\ \ldots.\ \ldots.\ \ldots.\ n{*}n{:}(v_n\&w_n)$,

kde symbol & má význam logické konjunkce (v_i i w_i jsou vlastnosti typu 1/0 tedy true/false a tudíž je smysl této spojky zřejmý).
Význam tohoto zápisu spočívá v tom, že místo abychom oznamovali, že první slovo má nějaké vlastnosti a druhé slovo má nějaké vlastnosti, vyjadřujeme nyní nové vlastnosti pro celý bigram: dvojice má vlastnost *x*, pokud první slovo má vlastnost *a* a zároveň druhé slovo má vlastnost *b*.

Způsob zápisu ostatních (nemorfologických) vlastností ponecháme – tyto vlastnosti (vyjma absolutní pozice) již samy o sobě podávali informace o celém bigramu, ne jen o jeho částech.

Vzhledem ke kvadratickému nárůstu počtu zaznamenávaných vlastností, provedeme následující test pouze s morfologickými vlastnostmi „slovní druh“ a „pád“.

Test6
morfologické vlastnosti: slovní druh, pád – kvadratický zápis
poziční vlastnosti: všechny
hraniční vlastnost
úspěšnost 53%

Přidáním dalších morfologických vlastností jsme již žádné zlepšení nezaznamenali, spíše naopak, použitím všech morfologických vlastností úspěšnost dokonce poklesla (na cca 50%). Naopak velikost jednotlivých trénovacích příkladů stoupla rapidně. Proto zůstaneme u vlastností použitých v tomto testu.

6.2.3 Testy s pravděpodobnostmi

Nyní k trénovacím příkladům přidáme hodnoty klasické pravděpodobnosti – získáme ji na základě statistických údajů získaných modulem LOAD z celých trénovacích dat. Bigramům, pro které pravděpodobnost není definována (týká se

hlavně testovacích dat, tedy dvojic, které se v trénovacích datech nevyskytly) prozatím přiřazujeme nulovou hodnotu. Opět je také třeba rozlišit, zda pro odhad pravděpodobnosti použijeme údaje o slovních tvarech, nebo jejich lemmatech.

Test7
morfologické vlastnosti – údaje o slovním druhu, poddruhu a pádu
poziční vlastnosti – všechny
pravděpodobnostní vlastnosti – klasická pravděpodobnost (lemmatizované tvary)
úspěšnost: 42.4%

Výsledek tohoto testu vykazuje výrazné zhoršení oproti stejnému testu bez pravděpodobnostní hodnoty. To je nejspíše způsobeno nedefinovanými pravděpodobnostními hodnotami. Procházením trénovacích a testovacích dat jsme zjistili následující anomálii:
Pro záporné příklady je v trénovacích datech *poměr příkladů s nulovou pravděpodobností ku příkladům s nenulovou pravděpodobností* roven 5.1, stejná hodnota pro testovací data však vychází 5.97. U kladných příkladů je tento rozdíl ještě markantnější: u trénovacích příkladů vychází tento poměr nulový, kdyžto u testovacích dat je roven 1.22. Pro přehlednost jsou tyto údaje popsané v tabulce 1:

Tabulka 1

PX	TRÉNOVACÍ DATA	TESTOVACÍ DATA
KLADNÉ PŘÍKLADY	**0**	**1.22**
ZÁPORNÉ PŘÍKLADY	**5.1**	**5.97**

PX = Počet příkladů s nulovou pravděpodobností / počet příkladů s nenulovou pst

Důvod těchto rozdílů je zřejmý – statistické údaje použité pro odhad pravděpodobnosti hodnoty pocházejí z trénovacích dat – zde jsou tedy údaje o všech bigramech známé, testovací data však obsahují i bigramy, které se v trénovacích datech nevyskytly vůbec a tudíž se zde vyskytuje velké množství příkladů s nedefinovanou pravděpodobnostní hodnotou.

Problémem však je použití těchto údajů v SVM – trénovací model, který vznikne z trénovacích dat, neodpovídá testovacím datům, které ohodnocuje. Pokusíme se tuto situaci vyřešit – sledované hodnoty uměle dorovnáme. Toho lze docílit jednoduše tak, že v trénovacích datech určitou část příkladů s nenulovou pravděpodobnostní hodnotou tuto hodnotu vynulujeme. Druhou možností by bylo rozdělení trénovacích dat na dvě skupiny – jednu menší pro trénování SVM a druhou větší pro načítání statistických údajů. Tuto možnost však nepoužijeme, snížení množství dat pro načítání statistických údajů by mělo negativní vliv na celkovou efektivitu.

Budeme tedy postupovat tak, že upravíme trénovací data. U kladných příkladů vynulujeme 55% příkladů s nenulovými pravděpodobnostmi, u záporných příkladů 12.5% (Tyto hodnoty jsme dopočetli na základě měření počtu jednotlivých typů bigramů v trénovacích a testovacích datech). Tím pádem se sledovaná hodnota PX pro trénovací i testovací data vyrovná.

Jak ukazuje následující test, má tento postup příznivý efekt:

Test8
morfologické vlastnosti – údaje o slovním druhu, poddruhu a pádu
poziční vlastnosti – všechny
pravděpodobnostní vlastnosti – klasická pravděpodobnost (lemmatizované tvary); dorovnané (viz výše)
úspěšnost: 55.8%

V tomto okamžiku můžeme přejít k dalšímu řešení nedefinovaných pravděpodobnostních hodnot – vyhlazování (viz 6.1). Neznámým prvkům přiřazujeme nějakou jinou hodnotu. Vyzkoušíme dvě možnosti – hodnoty 1/N (N je počet slov věty, ze které bigram pochází) a hodnotu *4.2*, která je obecnou statistickou pravděpodobností, že daný bigram představuje vazbu. Tato hodnota je odvozena z počtu negativních a pozitivních příkladů v trénovacích datech, tedy 1/23 = 4.2 (viz výše).

V tabulce 2 jsou uvedeny výsledky pro jednotlivé postupy (u „dorovnaných" hodnot jsou upravená trénovací data ve shodě s postupem uvedeným výše, nedefinovaným hodnotám se přiřazuje hodnota uvedená v příslušném sloupci tabulky).

Tabulka 2 - Vyhlazování

Hodnoty	N/A –> 0	N/A –> 4.2	N/A –> 1/N
Klasické	42.4%	42.6%	42.3%
Dorovnané	55.8%	55.9%	56%

6.3 Kombinované testy

Další možností je zpracování pravděpodobnostních vlastností zvlášť (mimo SVM). Budeme postupovat následovně. SVM využijeme pro získání ohodnocení na základě morfologických, pozičních a hraničních vlastností (jako v části 6.2.2). Hodnoty klasické pravděpodobnosti pak spočteme zvlášť a pro ohodnocení hran použijeme nějakou lineární kombinaci těchto dvou hodnot. Tímto přístupem zajistíme, že práce SVM nebude negativně ovlivněna nedefinovanými pravděpodobnostními hodnotami. Bude však třeba nalézt správnou kombinaci hodnot a také zapracovat nedefinované pravděpodobnosti.

Test9
morfologické vlastnosti – kompletní údaje
poziční vlastnosti – všechny
hraniční vlastnost
kombinace s klasickou pravděpodobností: 0.8*HP + (10+HS)
úspěšnost: 51%

Test10
morfologické vlastnosti: slovní druh, pád – kvadratický zápis
poziční vlastnosti: všechny
hraniční vlastnost
kombinace s klasickou pravděpodobností: 0.6*HP + (10+HS)
úspěšnost 59%

HP je pravděpodobnostní hodnota, HS je hodnota hrany dle SVM, nedefinovaným hranám přiřazujeme HP = 0.

Vidíme, že tento postup dosahuje zatím nejlepších výsledků, je třeba však dát pozor na vhodnou kombinaci – již menší odchylka může celkový výsledek značně rozhodit.

Přesto však zlepšení oproti testu bez pravděpodobnosti není, zejména u kvadratického zápisu morfologických vlastností, tak veliké, jak by se dalo očekávat – klasická pravděpodobnost má sama o sobě úspěšnost 36%, SVM 53%. To, že nárůst úspěšnosti není větší, je patrně mimo jiné způsobeno tím, že množiny správně určených bigramů při obou postupech se z větší části překrývají.

6.4 Další postupy

Pro další zlepšení jsme již neměnili použité vlastnosti, pouze jsme optimalizovali trénovací proces SVM. Maximální dosažená úspěšnost při použití následujících parametrů trénování a kombinace s pravděpodobností založenou na slovních tvarech činí cca **61%.**

použité parametry SVM: –z r –m 4000 –w 0.01 –j 1.2 –q 30,
kde –w je epsilonová šířka "válce" pro regresi, –j nastavuje poměr trénovacích chyb mezi negativními a pozitivními příklady a –q je maximální velikost QP-podproblému [4]. Parametry –z a –r jsou popsány výše v textu. Optimální hodnoty parametrů byly nalezeny experimentální cestou.

Závěr

Pro použitelnost našeho parseru v praxi mluví rychlý algoritmus ($O(n^2)$) zpracování při předem zpracovaných trénovacích datech a připraveném ohodnocení hran.

Naopak problémem by mohlo být nutnost použití SVM pro klasifikaci všech zpracovávaných dat – znamená to, že nejprve musíme ze zpracovávaných dat načíst jednotlivé bigramy a jejich vlastnosti, pak je třeba využít SVM pro jejich klasifikaci a teprve potom můžeme přistoupit k vlastnímu parsingu. Samotné načtení testovacích dat pro SVM je už řádově složitější problém – z věty o n slovech potřebujeme n^2 bigramů a navíc pro každý bigram zjišťujeme, zda představuje reálnou vazbu (tedy zda jde o kladný či záporný příklad), složitost tohoto načítání je tedy již $O(n^3)$. A samotná klasifikace může být pro jiné, než lineární jádrové funkce, ještě složitější.

Modulárnost našeho parseru však umožňuje využití pouze některých částí – použijeme-li jen pravděpodobnostní ohodnocování, odpadne nám potřeba použití SVM a práce systému se urychlí (samozřejmě za cenu snížení efektivity).

Přestože výsledky, kterých jsme dosáhli, jsou zcela jistě překonány jinými metodami, domníváme se, že potenciál našeho postupu jsme zdaleka nevyčerpali. První mezí, na kterou narážíme, je zcela určitě velikost trénovacích dat, která hraje zásadní roli pro pravděpodobnostní ohodnocování hran. Pro současnou velikost trénovacích dat nelze pro značnou část bigramů z testovacích dat dopočítat pravděpodobnost vazby. Předpokládáme, že při zvětšení trénovacích dat by došlo i ke značnému zlepšení úspěšnosti parsingu.

Další možností pro zlepšení by zcela nepochybně bylo nalézt další vhodné vlastnosti bigramů použitelné pro trénování SVM a samozřejmě také nalezení optimální jádrové funkce. Otevřenost našeho systému také umožňuje použití libovolné jiné metody ohodnocování hran – ať už neuronové sítě, pokročilejší pravděpodobnostní hodnocení (viz [13]) nebo třeba již zmíněný algoritmus MIRA, se kterým McDonald dosáhl na českých datech úspěšnost přes 84% [7]. Náš systém může tedy sloužit i pro testování různých způsobů ohodnocování hran, aniž by bylo nutné jakkoliv zasahovat do jeho jádra (algoritmus maximální kostry), které je značně univerzální a samozřejmě také efektivní.

Seznam literatury

[1] [CHU 1965]
Y.J. Chu, T.H. Liu: *On the shortest arborescence of a directed graph.* Science Sinica, 14:1396-1400.

[2] [EDMONS]
J. Edmons: *Optimum branchings.* Journal of Research of the National Bureau of Standards, 71B:233-240.

[3] [GEORGIADIS 2003]
Leonidas Georgiadis: *Arborescence optimization problem solvable by Edmonds' algorithm.* Theoretical Computer Science 301, 2003

[4] [JOACHIMS 2002]
Thorsten Joachims: *Learning to Classify Text using Support Vector Machines (Dissertation).* Kluwer, 2002

[5] [KRAJÍČEK 2007]
Jiří Krajíček: *Multigramatiky a syntaktická analýza založená na nich (diplomová práce).* FIT VUT, Brno 2007

[6] [MANNING 1999]
Christopher R. Manning, Hinrich Schutze: *Foundation of Statistical Natural Language Processing.* The MIT Press, 1999

[7] [McDONALD ET AL.]
Ryan McDonald, Koby Crammer, Fernando Pereira: *Spanning Tree Methods for Discriminative Training of Dependency Parsers.* HLT-EMNLP, 2005

[8] [McDONALD ET AL.]
Ryan McDonald, Fernando Pereira, Kiril Rybanov, Jan Hajič: *Non-projective Dependency Parsing using Spanning Tree Algorithms.* HLT-EMNLP, 2005

[9] [PAJAS 2005]
Petr Pajas, Jan Štěpánek: *A generic XML-based format for structured linguistic annotation and its application to the Prague Dependency Treebank 2.0 (úfal/ckl technical report).* Matematicko-fyzikální fakulta Univerzity Karlovy, Praha 2005

[10] [PDT 2.0 2006]
Institute of Formal and Applied Linguistics: *The Prague Dependency Treebank, 2.0.* Matematicko-fyzikální fakulta Univerzity Karlovy, Praha 2006
http://ufal.mff.cuni.cz/pdt2.0/

[11] [PECINA 2006]
Pavel Pecina, Pavel Schlesinger: *Combining Association Measures for Collocation Extraction.* Institute of Formal and Applied Linguistic, Charles University, Prague 2006

[12] [VAPNIK 1995]
Vladimir N. Vapnik: *The Nature of Statistical Learning Theory.* Springer, 1995

[13] [ZEMAN 1997]
Daniel Zeman: *Pravděpodobnostní model významových zápisů vět (diplomová práce).* Matematicko-fyzikální fakulta Univerzity Karlovy, Praha 1995

Přílohy

Příloha A – morfologické vlastnosti PDT

V PML jsou morfologické vlastnosti zaznamenány pomocí značky <tag>. Tato značka obsahuje 15 pozic, které určují jednotlivé vlastnosti. Každé slovo nabývá právě jednu vlastnost z každé skupiny. Jejich popis podle pořadí pozic následuje (viz [10]).

1. Slovní druh

‘**A**’ – adjektivum; ‘**C**’ – číslovka; ‘**D**’ – příslovce; ‘**I**’ – citoslovce; ‘**J**’ – spojka; ‘**N**’ – substantivum; ‘**P**’ – zájmeno; ‘**V**’ – sloveso; ‘**R**’ – předložka; ‘**T**’ – částice; ‘**X**’ – neurčený slovní druh; ’**Z**’ – interpunkce, hranice věty

2. Slovní poddruh

‘**#**’ – hranice vety; ‘***%***’ – autorova signatura; ‘*****’ – slovo "krát"; ‘**,**’ – spojka podřadící; ‘**}**’ – římská číslice; ‘**:**’ – interpunkce všeobecně; ‘**=**’ – číslo zapsané číslicemi; ‘**?**’ – číslovka "kolik"; ‘**&**’ – nerozpoznaný slovní tvar; ‘**^**’ – spojka souřadící; ‘**4**‘ – zájmeno vztažné/tázací s adjektivním skloňováním (např. „jaký“, „který“, „čí“); ‘**5**‘ – zájmeno "on" ve tvarech po předložce ("něj", "něho"); ‘**6**‘ – zájmeno reflexivní "se" v dlouhých tvarech (sebe); ‘**7**’ – zájmeno reflexivní "se", "si", "ses", "sis"; ‘**8**’ – zájmeno přivlastňovací; ‘**9**’ – zájmeno vztažné; ‘**A**’ – adjektivum; ‘**B**’ – sloveso, tvar přítomného nebo budoucího času; ‘**C**’ – adjektivum, jmenný tvar; ‘**D**’ – zájmeno ukazovací; ‘**E**’ – zájmeno vztažné („což“); ‘**F**’ – součást předložky, která nestojí samostatně; ‘**G**’ – adjektivum odvozené od přítomného přechodníku; ‘**H**’ – krátké tvary osobních zájmen;’**I**’ – citoslovce; ‘**J**’ zájmeno vztažné „jenž“, „již“; ‘**K**’ – zájmeno tázací/vztažné; ‘**L**’ – zájmeno neurčité; ‘**M**’ – adjektivum odvozené od minulého přechodníku; ‘**N**’ – substantivum; ’**O**‘ – samostatně stojící zájmena („svůj“, „tentam“); ‘**P**’ – zájmeno osobní; ‘**Q**’ – zájmeno tázací/vztažné („co“, „cožpak“); ‘**R**’ – předložka (obecná, bez vokalizace); ‘**S**’ – zájmeno přivlastňovací; ‘**T**’ – částice; ‘**U**’ – adjektivum přivlastňovací; ‘**V**’ – předložka vokalizovaná („ve“, „ku“); ‘**W**’ – zájmena záporná; ‘**X**’ – slovní tvar, který byl rozpoznán morfologickou analýzou, ale značka chybí; ‘**Y**’ – zájmeno tázací/vztažné „co“ spojené s předložkou; ‘**Z**’ – zájmeno neurčité; ‘**a**‘ – číslovka neurčitá; ‘**b**’ – příslovce, které se neskloňuje, ani ho nelze negovat; ‘**c**’ – kondicionál slovesa „být“; ‘**d**’ – číslovka druhová s adjektivním skloňováním; ‘**e**’ – přechodník přítomný; ‘**f**’- infinitiv; ‘**g**’ – příslovce, které se skloňuje a časuje; ‘**h**’ – číslovka druhová; ‘**i**’ – sloveso ve tvaru rozkazovacího způsobu; ‘**j**’ – číslovka druhová větší nebo rovna 4 v substantivním postavení; ‘**k**’ – číslovka druhová větší nebo rovna 4 v adjektivním postavení, krátký tvar („čtvery“); ‘**l**‘ – číslovka základní s nesubstantivním skloňováním; ‘**m**’ – přechodník minulý; ‘**n**’ – číslovka základní větší nebo rovna 5; ‘**o**’ – číslovka násobná neurčitá; ‘**p**’ – příčestí činné; ‘**q**’ – archaické příčestí činné (zakončené „-t“); ‘**r**’ – číslovka řadová; ‘**s**’ – příčestí trpné; ‘**t**’ – archaický slovesný tvar přítomného a budoucího času (zakončení „-t“); ‘**u**’ – číslovka tázací násobná „kolikrát“; ‘**v**’ – číslovka násobná; ‘**w**’- číslovka neurčitá s adjektivním skloňováním; ‘**y**’ – zlomek zakončený na „-ina”; ‘**z**’ – číslovka tázací řadová „kolikátý“

3. Rod

‘**-**’ – neurčuje se; ‘**F**’ – feminum; ‘**N**’ – neutrum; ‘**H**’ – feminum nebo neutrum; ‚**I**’ – maskulinum inanimatum (rod mužský neživotný); ‘**M**‘ – maskulinum animatum (rod mužský životný); ‘**Q**’ – feminunum singuláru nebo neutrum plurálu (pouze u příčestí a jmenných adjektiv); ‘**T**’ – maskulinum inanimatum nebo femininum plurálu (pouze u příčestí a jmenných adjektiv); ‘**X**’ – libovolný rod (F/M/I/N); ‘**Y**’ – maskulinum (animatum nebo inanimatum); ‘**Z**’ – nikoliv femininum, tj. M/I/N

4. Číslo

‘**-**’ – neurčuje se; ‘**D**’ – duál (pouze 7. pád feminin); ‘**P**‘ – plurál (množné číslo); ‘**S**’ – singulár (jednotné číslo); ’**W**’ – pouze v kombinaci se jmenným rodem Q (singulár pro feminina, plurál pro neutra); ‘**X**’ – libovolné číslo (D/S/P)

5. Pád

‘**-**’ – neurčuje se; ‘**1**’ – nominativ (první pád); ‘**2**’ – genitiv (druhý pád); ‘**3**’ – dativ (třetí pád); ‘**4**’ – akuzativ (čtvrtý pád); ‘**5**’ – vokativ (pátý pád); ‘**6**’ – lokál (šestý pád); ‘**7**’ – instrumentál (sedmý pád); ‘**X**’ – libovolný pád

6. Přivlastňovací rod

‘**-**’ – neurčuje se; ‘**F**’ – femininum; ‘**M**’ – maskulinum animatum (pouze u adjektiv); ‘**X**’ – libovolný rod (M/I/F/N); ‘**Z**’ – nikoliv femininum, tj. M/I/N

7. Přivlastňovací číslo

‘**-**’ – neurčuje se; ‘**P**’ – plurál; ‘**S**’ – singulár; ‘**X**’ – libovolné číslo (S/P)

8. Osoba

‘**-**’ – neurčuje se; ‘**1**’ – první osoba; ‘**2**’ – druhá osoba; ‘**3**’ – třetí osoba; ‘**X**’ – libovolná osoba (1/2/3)

9. Čas

‘**-**’ – neurčuje se; ‘**F**’ – futurum (budoucí čas); ‘**P**’ – prézens (přítomný čas); ‘**R**’ – préteritum (minulý čas); ‘**H**’ – prézens nebo préteritum (P/R); ‘**X**’ – libovolný čas (F/R/P)

10. Stupeň

‘**-**’ – neurčuje se; ‘**1**’ – první stupeň; ‘**2**’ – druhý stupeň; ‘**3**‘ – třetí stupeň

11. Negace

‘**-**’ – neurčuje se; ‘**A**’ – afirmativ (bez negativní předpony „ne-“); ’**N**’ – negace (tvar s negativní předponou „ne-“)

12. Aktivum / Pasivum

‘**-**’ – neurčuje se; ‘**A**’ – aktivum; ‘**P**‘ – pasivum

13., 14. – Nevyužito

15. Varianta, stylový příznak apod.

‘**-**’ – základní tvar; ‘**1**’ – varianta k základnímu tvaru; ‘**2**’ – řídká, archaická nebo knižní varianta k základnímu tvaru; ‘**3**’ – velmi archaický tvar, též hovorový; ‘**4**’ – velmi archaický nebo knižní tvar, pouze spisovný; ‘**5**’ – hovorový tvar (ve veřejných projevech); ‘**6**’ – hovorový tvar (koncovka obecné češtiny); ‘**7**’ – hovorový tvar, varianta k 6; ‘**8**‘ – zkratky; ‘**9**’ – speciální použití (např. tvary zájmen po předložkách)

Příklad zápisu morfologických údajů pro slovo „rychlejší“:
<tag>AAFS1----2A----</tag>

Příloha B – uživatelská dokumentace

Následuje popis spouštění jednotlivých částí parseru a popis formátů vstupních a výstupních souborů. Pro použití předpokládáme PDT dekomprimovaný (v textové podobě), adresářová struktura je standardní – v hlavním adresáři PDT se nacházejí trénovací data – adresáře train1-train8 a testovací data – adresáře dtest a etest.
Na přiloženém CD se nacházejí jak všechny potřebné zdrojové kódy, tak jejich zkompilované verze (pro systém LINUX).

MST

Vstupní parametry:
-1 adresář s PDT (konkrétně s analytickou rovinou - amw; jednotlivé soubory v podadresářích train1-8, dtest a etest předpokládáme dekomprimovány!)
-T testovaný adresář (*dtest* nebo *etest*); *dtest* defaultně
-p primární statisticky soubor
-b sekundární statisticky soubor
-h soubor s ohodnocením hran (výsledek SVM)
-s skript s dalšími parametry
 formát skriptu:
  na začátku každé řádky se nachází písmeno označující vlastnost, následuje mezera a hodnota vlastnosti.
  - hodnota pro nedefinované pravděpodobnosti (míry): N [double hodnota]
  - typ pravděpodobnosti: P [int hodnota]
    hodnota: 1-13 podle použité pravděpodobnosti (míry), 1 defaultně - klasická pravděpodobnost, 2 a dále - míry v pořadí podle přílohy D
  - způsob ohodnocovaní hran: E [int hodnota]
    hodnota: 0-2; 2 - ohodnoceni pouze podle SVM, 1 - pouze pravděpodobnostní ohodnoceni, 0 - kombinace (defaultně)
  - poměr při kombinovaném ohodnocování: C [double hodnota]
    H1*C + H2; H1 je pravděpodobnostní ohodnoceni, H2 je hodnoceni dle SVM; default: 0.6

Výstup je směrován do adresáře s testovacími daty, ukládá se do souborů *.b – paralelně s daty *.a, která jsou anotována ručně. Umožňuje to jejich snadné porovnání. Jako ladící výstup (a kvůli možnosti přímo sledovat výsledky) jsou zároveň na standardní výstup vypisovány jednotlivé věty a seznam závislých dvojic výsledných závislostních stromů.

LC

Vstupní parametry:
-l umístění PDT
-A zapnutí/vypnutí komprimace primárního statistického souboru
- při zapnuté komprimaci se vynechávají údaje o bigramech, které spolu nikdy nebyly ve vazbě (default)
- hodnoty: *true/false* nebo *1/0*

-X nastaveni typu pravděpodobnosti - podle slovních tvarů (form) nebo podle slovních lemmat (lemma)
- hodnoty: *lemma/form*; default: *lemma*

Výstupní parametry:
-p primární statistický soubor
-b sekundární statistický soubor

TRAIN

Vstupní parametry:
-l adresář s PDT
-t typ dat (*train*, *dtest*, *etest*)
-p primární statistický soubor (pro výpočet pravděpodobností a asociačních měr)
-b sekundární statistický soubor (pro výpočet dalších asociačních měr)
-s skript obsahující vlastnosti, které se mají načítat + další parametry

formát skriptu:
na začátku každé řádky se nachází písmeno označující vlastnost, následuje mezera a hodnota vlastnosti (popřípadě seznam hodnot oddělených čárkou)
- morfologické vlastnosti: M [1,2,...,15]
- asociační míry: P [1,2,...,13]
- poziční vlastnosti: S [1,2,3]
- hraniční vlastnost: B 1

další parametry:
- kvadraticky zápis morfologických vlastnosti: Q 1
  -defaultně je nastaven lineární
- vyrovnání kladných a záporných příkladů: A [int hodnota x]
  -hodnota x označuje, ze každý x-tý záporný příklad má být ponechán (ostatní budou odstraněny); pro vyrovnaní použít A 22
- hodnota pro nedefinované pravděpodobnosti: N [double hodnota]
  -defaultně 0
- další parametry pro úpravu výpočtu "klasické" pravděpodobnosti
  -X [int hodnota]
  -Y [int hodnota]
  určuje, u kolika příkladu z 1000 s nenulovou pst bude pst vynulována
  X - kladné příklady, Y - záporné příklady
  použité hodnoty pro dorovnání (viz část 6.2.3, Test 8): X 550; Y 125;
  defaultně X i Y = 0

Výstupní parametr:
-o soubor pro SVM

EVAL

Vstupní soubory:
-1 umístění PDT
-T porovnávaný adresář (*dtest* nebo *etest*); defaultně *dtest*

Výstupem je informace předaná na standardní výstup (obrazovku) s údaji o úspěšnosti parsingu – procentuální úspěšnost a počet správně určených bigramů.

Formáty souborů:

- PDT (-1) – formát PML
- soubor s pravděpodobnostmi (-p)
  na první řádce je příznak typu dat – buď *#lemma* nebo *#form*. Dále obsahuje dvojice slov spolu s údaji o jejich výskytech, každá řádka má formát: ”slovo1_slovo2~X~A”, kde X je počet výskytů dvojice a A je rovno počtu vazeb mezi touto dvojicí. Dvojice s nulovou hodnotou A se vynechávají.
- sekundární soubor s pravděpodobnostmi (-b)
  na první řádce opět typ dat, dále obsahuje údaje o jednotlivých slovech, každá řádka má formát:
  “slovo~počet výstupních vazeb~počet vstupních vazeb“
- soubor s ohodnocením hran (-h)
  obsahuje ohodnocení všech hypotetických hran (myslíme možných vazeb mezi jednotlivými dvojicemi slov) v testovacích datech v jednoznačném pořadí (tak, jak byly načteny věty a vygenerovány bigramy). Formát souboru je jednoduchý – na každé řádce se nachází jedno reálné číslo.
- soubor pro SVM (-o)
  obsahuje soubor pro trénování SVM nebo pro klasifikaci – podle toho, jestli pochází z trénovacích nebo testovacích dat. Zapsán je ve formátu SVM (viz 4.2): “[c] [1:vlastnost_1] [2:vlastnost_2] ... [n:vlastnost_n]“,
  kde c je třída příkladu (buď 1 nebo -1) a jednotlivé vlastnosti jsou reálná čísla.

Použití parseru je popsáno v části 5.3.

Příloha C – programátorská dokumentace

V této části na rozdíl od uživatelské dokumentace nebudeme sledovat rozdělení podle jednotlivých modulů, ale spíše podle zdrojových souborů. Popíšeme účel jednotlivých procedur a zmíníme se o nejpodstatnějších datových strukturách.

Edmons.h
Tuto knihovnu využívá výhradně modul MST. Obsahuje vlastní algoritmus na hledání maximální kostry grafu (Chu-Liu-Edmons) – proceduru *int* Edmons(mat matrix)*. Struktura *mat* je nadefinována v téže knihovně a obsahuje vlastní graf (ve formátu matice sousednosti) a všechny potřebné pomocné proměnné – původní

počet vrcholů grafu, aktuální velikost grafu (během práce programu se mění), dále údaje o maximálních hodnotách v každém sloupci a každé řádce matice, seznam aktuálně platných vrcholů a další pomocné údaje.
Výstupem procedury je seznam hran představující výslednou maximální kostru, který je uložen v poli typu int následujícím způsobem: pro každé i je hrana (pole[i], i) součástí maximální kostry (to je možné díky tomu, že do každého vrcholu mimo kořene vstupuje právě jedna hrana).
Celá procedura pracuje rekurzivně podle popisu algoritmu v kapitole 2.
V knihovně jsou umístěny následující pomocné procedury:
void CountIndex(mat m) – přepočítává index platných vrcholů uložený ve struktuře *m* (pole, kde je označena platnost jednotlivých vrcholů: pole[i] = 1 znamená, že i-tý vrchol je platný; pole[i] = 0 neplatný vrchol) na seznam platných vrcholů. Výstup není zapotřebí, upravuje data přímo.
cyc Cycle(mat G) – hledá netriviální cyklus v grafu *G.maxcolumn*, který je aktuálním kandidátem na maximální kostru – v našem případě tedy v seznamu maximálních hodnot sloupců (ve shodě s popisem algoritmu – pro každý vrchol nás zajímá pouze maximální hrana, která do něj vstupuje). V tomto grafu je počet hran roven počtu vrcholů (bez kořene) a tedy nalezení cyklu probíhá v lineárním čase.
Výsledný cyklus je umístěn ve struktuře *cyc*, která kromě vlastního cyklu (seznam vrcholů) obsahuje údaj o délce a hodnotě cyklu.
*cyc Visit(int *G, int i, int c)* – pomocná procedura pro proceduru *Cycle*, provádí „návštěvu“ konkrétního vrcholu *i* v čase vstupu *c* a případně rekonstrukci nalezeného cyklu.
int Pred(cyc cycle, int pos) – Vrací předchůdce vrcholu *pos* v cyklu *cycle*.
void CONTRACT(mat matrix, cyc cycle) – Vstupem je celý graf ve formátu *mat* a cyklus. Procedura provádí kontrakci cyklu podle pravidel popsaných v kapitole 2. Výstup probíhá formou úprav grafu.

pdt_load.h
Knihovna určená pro načítání dat z PDT (ve formátu PML). Jsou zde nadefinovány následující struktury:
twain – obsahuje 2 slova (dvojici) a jejich polohy ve větě.
twaininfo – obsahuje slovo a dodatečnou informaci o něm (např. seznam morfologických vlastností)
combination – obsahuje celou větu – vektor *twain* – seznam všech dvojic slov, které jsou spolu ve vazbě a vektor *twaininfo* – rozšířené informace pro všechna slova ve větě.
Hlavní procedurou je *combination inx_PDT(ifstream *a, ifstream *w, int X).* Vstupem jsou ukazatele na dva streamy (potřebné údaje se načítají ze 2 souborů simultánně – **.a* a **.m* – jsou třeba údaje z morfologické i analytické roviny) a příznak, zda se načítají slova jako slovní tvary nebo jejich lemmata. Výstupem je úplná informace o jedné větě uložená ve struktuře *combination*. Pokud není zapotřebí rozšířených údajů o slovech, lze použít proceduru *in_PDT*.
Samotné načítání probíhá následovně – napřed se načte jedna věta z morfologické roviny a následně se načítá z analytické roviny příslušný závislostní strom – pro převod stromu na seznam dvojic je využit zásobník.
Pomocné funkce v této knihovně jsou:
bool contains(string line, string word) – Vstupem je řádka textu a slovo, procedura zjišťuje, zda řádka obsahuje slovo.

string remove_substring(string line, string S) – Vstupem je opět řádka textu a slovo S, vrací text, který se nachází mezi <S> a </S>
string load_id(string line) – Z řádky textu načte id vrcholu (předpokládá se, že ho řádka obsahuje).

file_system.h
Obsahuje vše potřebné pro procházení podadresářů PDT. Hlavní procedurou je *vector <string> dir(string adr, char* pdt)*. *Pdt* je umístění kořenového adresáře dat s PDT, *adr* upřesňuje, které podadresáře se budou procházet – ty, jejichž název obsahuje řetězec *adr*. Výstupem je vektor všech souborů bez přípony nacházejících se v procházených podadresářích (Hledá se podle souborů **.a*).
Z důvodu, že pro načítání dat z PDT obvykle používáme 2 soubory souběžně, zavádíme zde strukturu *file_name* obsahující názvy 2 souborů ve formátu char*.
Další pomocné procedury slouží k doplnění přípon k názvům souborů:
file_name name(vector<string>::iterator itt) přidává k názvu souboru *itt* přípony **a a *m*
a procedura
char input_name(vector<string>::iterator itt)* přidává příponu **.b*.

pdt_write.h
Obsahuje procedury potřebné pro zápis do formátu PML (využívá je modul MST). Samotný zápis provádí procedura *void write(vector <twainid> sezdv, string nodeid)*
sezdv obsahuje seznam dvojic a pořadí jejich slov ve větě, *nodeid* obsahuje označení kořene. Pro zápis seznamu dvojic do stromu ve formátu PML se využívá rekurzivní pomocná procedura *rek(vector <twainid> *sezdv, string actual, int space)*.
Zápis se provádí do ofstreamu *output*, který je deklarován v knihovně jako globální proměnná. Modul MST mu přiřazuje soubory shodných názvů (až na příponu – zde je přípona **.b*) se soubory, ze kterých pocházejí testovací data.
Procedura zapisuje pouze data o zpracovaných větách, hlavičky PML souborů zapisuje přímo modul MST.
Pomocné procedury jsou:
string indent(int poc) – vrací *poc* mezer ve stringu.
string a_(string line) – přepisuje první znak řádky na 'a'.

probab_load.h
Úkolem této knihovny je načtení souborů se statistickými daty (pro výpočet asociačních měr, viz kapitola 4.1 a příloha D) do hašovací tabulky. Samotná hašovací tabulka je implementována pomocí *stdext::hash_map*. Potřebné třídy – *HashString* a *HashStringCompare* jsou nadefinovány v knihovně *hash.h*. Pro použití je nutná ještě standardní knihovna *ext/hash_map*.
Pro načtení primárního statistického souboru slouží procedura
*void load(char *what, stdext::hash_map<string, p, HashString, HashStringCompare> *H, int *X)*
what obsahuje název vstupního souboru, *H* je hašovací tabulka a *X* je ukazatel na parametr značící použitý typ statistických dat – buď data o slovních tvarech nebo lemmatech.
Pro načtení sekundárního statistického souboru slouží obdobná procedura

*void loadb(char *whatb, stdext::hash_map<string, q, HashString, HashStringCompare> *S, int *X)*
V obou příkladech jsou klíči hašovací tabulky hodnoty typu string. Rozdíl je pouze v pojmenování prvků hašovací tabulky – v prvním případě se jedná o strukturu *p* obsahující prvky *count* a *binding* a v případě druhém o strukturu *q* s prvky *b* a *c*. Význam těchto hodnot zde podrobně popisovat nebudeme, odpovídá významu dat ze souborů, ze kterých jsou načítána (viz kapitola 5.2.2).

probab.h
Obsahuje výpočty jednotlivých asociačních měr (viz příloha D), potřebná data pocházejí z výše zmíněných hašovacích tabulek.

argumenty.h
Tato knihovna slouží k rozpoznání spouštěcích parametrů programu (všech čtyřech modulů) a k definování defaultních hodnot těchto parametrů. Hodnoty všech parametrů jsou zapsány do definované struktury *arg*, ze které si jednotlivé moduly berou potřebné parametry.
Vlastní rozpoznání parametrů provádí procedura *arg argumenty(int argc, char **argv)*

script.h
Slouží k načtení skriptů s rozšiřujícími parametry pro moduly TRAIN a MST. Hodnoty parametrů se zaznamenávají do struktur *fi* (parametry pro modul TRAIN) a *Efi* (pro MST). Jsou zde rovněž definovány defaultní hodnoty parametrů.
Pro načtení slouží příslušné procedury *fi load_script(char *s)* a *Efi load_Escript(char *s)*. Jejich vstupem je jméno skriptu, který mají načíst a výstupem jsou zjištěné parametry.
Pomocné procedury jsou:
void list(string line, bool v, const int N)* – z řádky *line* načítá hodnoty typu true/false do pole *v* o délce N
int value(string line) – načítá intovou hodnotu z řádky *line*.
double dvalue(string line) – načítá hodnotu typu double z řádky *line*.
Formát skriptu je podrobně popsán v příloze B, jednotlivé procedury tento formát předpokládají.

Dále popíšeme zdrojové kódy hlavních modulů.

MST – MST.cpp
Hlavní procedura (*main*) řeší všechny potřebné inicializace. Dále pro všechny zpracovávané soubory zapisuje hlavičku příslušného výstupního PML souboru a pro každou větu ze vstupního souboru spouští algoritmus na hledání maximální kostry (Edmons – knihovna Edmons.h). Matici sousednosti zpracovávaného grafu sestaví podle zadaných parametrů ohodnocování. Výslednou maximální kostru zapíše do výstupního souboru ve formátu PML (pomocí knihovny *pdt_write* – viz výše), zároveň vypisuje na standardní výstup výsledek jako seznam hran (dvojic slov) výsledné kostry.
Obsahuje pomocnou proceduru *vector <twaininfo> load_test(ifstream *s, int X)*, která se stará o vstup dat z PDT (na rozdíl od procedur z knihovny *pdt_load* tato procedura načítá pouze morfologickou rovinu, analytické se nedotýká).

TRAIN – TRAIN.cpp

Hlavní procedura se opět stará o inicializace a vytváří hlavní cyklus – pro všechny procházené zdrojové soubory a každou větu v nich volá proceduru
*void work_sentence(struct combination *K,struct fi *sc)*
Tato procedura provádí vlastní zápis vlastností jednotlivých bigramů do výstupního souboru.
Struktura *K* obsahuje údaje o větě, *sc* určuje načítané vlastnosti a způsob zápisu (vyrovnané pozitivní a negativní příklady, kvadratický zápis...).
Jako pomocné slouží procedury určené pro jednotlivé typy vlastností:
Procedura *vector <bool> atributes(string *v, bool *sc)* je určena pro načítání morfologických vlastností. Proměnná *v* obsahuje seznam morfologických vlastností ve formátu PML (viz příloha A), *sc* určuje, které vlastnosti se mají načíst. Výstupem je vector typu bool, který představuje bitovou mapu načítaných morfologických vlastností.
Pro načítání asociačních měr (pravděpodobnosti) slouží procedura
*vector <double> probability(string *a, string *b, int length, int binding, bool *sc, int X, int Y, double NA)*
Proměnné *a* a *b* obsahují slova bigramu, *length* je délka věty, *binding* určuje, zda daný bigram určuje reálnou vazbu (1 nebo -1), *sc* je seznam načítaných vlastností, *X*, *Y* slouží k dorovnání hodnot klasické pravděpodobnosti (viz 6.2.3) a *NA* je náhrada za nedefinované hodnoty. Pro výpočet konkrétních asociačních měr jsou využity funkce z knihovny *probab.h*
Další pomocnou procedurou je *vector <int> position(int i, int j, int sz,vector <bool> v, bool *sc)*, která se stará o načítání pozičních vlastností. Proměnné *i*, *j* vyjadřují polohy obou slov bigramu ve větě, *sz* je délka věty, *v* je seznam morfologických vlastností druhého slova bigramu (kvůli dodefinování vzdálenosti od kořene) a *sc* je opět seznam načítaných vlastností.
Poslední procedurou je *int bound(int i, int j, vector <float> hr, int size)*, která načítá hraniční vlastnost. Využívá výstup pomocné procedury *vector <float> bn(vector <twaininfo> sez)*, která je volána procedurou *work_sentence* na začátku zpracování každé věty a rozdělí větu na větné úseky (vektor *bn*).

LC – LOAD.cpp

Hlavní procedura řeší opět inicializace a vytváří hlavní cyklus – pro každou větu v každém souboru spouští proceduru *void work_sentence(vector <twain> *sentence)* – ta zaznamená údaje o všech bigramech ve větě do hašovacích tabulek *H* a *S* (v tomto modulu definovány globálně).
Hlavní procedura pak údaje z těchto tabulek uloží do souborů (primárního a sekundárního souboru se statistickými daty).

EVAL – EVAL.cpp

Nejjednodušší část celého systému. Pomocí knihovny *load_pdt* načítá ze všech procházených souborů paralelně 2 věty – jednu ze souboru s manuálně anotovanými daty a druhou ze souboru, který je výsledkem práce MST. O porovnání těchto vět se stará pomocná procedura
result compare(vector <twain> pdt, vector <twain> my)
Výsledek typu *result* obsahuje počet porovnávaných vazeb (rovno počtu slov věty) a počet správně určených vazeb v dané větě. Hlavní procedura pak tyto údaje pro všechny věty posčítá a na standardní výstup vypíše procentuální úspěšnost parsingu.

Příloha D – Asociační míry [11]

1. Joint probability:
   P(xy) 26.3%
2. Conditional probability:
   P(y|x) 30.5%
3. Reverse conditional probability:
   P(x|y) 24.3%
4. Pointwise mutual information:
   $\log \frac{P(xy)}{P(x*)P(*y)}$ 29%
5. Normalized expectation:
   $\frac{2f(xy)}{f(x*)+f(*y)}$ 26%
6. Mutual expectation:
   $\frac{2f(xy)}{f(x*)+f(*y)}.P(xy)$ 26.9%
7. Salience:
   $\log \frac{P(xy)^2}{P(x*)P(*y)}.\log f(xy)$ 9.1%

Některé asociační koeficienty:

8. Russel-Rao:
   $\frac{a}{a+b+c+d}$ 26.26%
9. Sokal-Michiner:
   $\frac{a+d}{a+b+c+d}$ 14.17%
10. Third-Sokal-Sneath:
    $\frac{b+c}{a+d}$ 6.45%
11. Jaccard:
    $\frac{a}{a+b+c}$ 21.95%
12. First Kulczynski:
    $\frac{a}{b+c}$ 21.26%

Příslušná kontingenční tabulka:

a = f(xy)	b = f(x¬y)	f(x∗)
c = f(¬xy)	d = f(¬x¬y)	f(¬x∗)
f(∗y)	f(∗¬y)	N

¬x znamená libovolné slovo mimo x, * zastupuje libovolné slovo a N je celkový počet bigramů. Další podrobnosti jsou uvedeny v [11].
Procentuální hodnoty značí úspěšnost parsingu na datech *dtest* při použití slovních lemmat.