VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

EXTRAKCE INFORMACÍ Z WIKIPEDIE

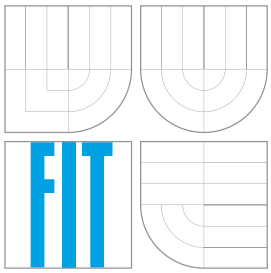
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

SERGEY PANOV

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

EXTRAKCE INFORMACÍ Z WIKIPEDIE

INFORMATION EXTRACTION FROM WIKIPEDIA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

SERGEY PANOV

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. RNDr. PAVEL SMRŽ, Ph.D.

BRNO 2016

Abstrakt

Cílem této práce byla implementace systému pro extrakci syntaktických struktur sloužící k vyjadření určitého typu vztahu. Pro extrakci jsem použil statistické zpracování vět potencialně vyjadřujících cílovy vztah. Na konci práce se podařilo dosáhnout 70% přesnosti při extrakci. Podobný systém by se dalo aplikovat pro zobecnění syntaktických struktur do tříd, které vyjadřují určitou sémantiku. Následně je možné získané třídy využít jako znalostní data pro NLP systémy.

Abstract

The purpose of this thesis was an implementation of system for extraction syntax structures which express certain type of relation. I used the statistical analysis of sentences which potentially signify specific connection. Eventually the 70% of validity of this thesis was achieved. This system could be used for generalization of syntax structures for classes that display decisive semantic. These classes could be used as an knowledge for NLP systems.

Klíčová slova

Extrakce informací, Vyhledávání informací, Extrakce vztahů, Zpracování přirozeného jazyka, Určování slovních druhů, SPARQL, Wikipedia, DBpedia, přesnost, úplnost, F-measure

Keywords

Information extraction, Information retrieval, Relation extraction, Natural language processing, Part-of-speech tagging, SPARQL, Wikipedia, DBpedia, precision, recall, F-measure

Citace

Sergey Panov: Extrakce informací z Wikipedie, bakalářská práce, Brno, FIT VUT v Brně, 2016

Extrakce informací z Wikipedie

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. RNDr. Ph.D. Pavla Smrže.

.....
Sergey Panov
16. května 2016

Poděkování

Tady bych chtěl vyjádřit svou vděčnost panu Doc. Pavlu Smržovi za jeho řady a doporučení během konzultací.

Obsah

Kapitola 1

Úvod

V dnešní době lidé mají přístup k obrovskému množství informací. Informace může být získaná z různých zdrojů například z obrázků, zvuků, textů a pod.. Jedním z nejpopulárnějších zdrojů informace je text. Nicméně množství textové informace je příliš velké a ne vždy je v lidských silách rychle dohledat požadovanou informaci. Právě pro účely vyhledávání určité textové infromace lidé začal přemýšlet o způsobech automatizaci tohoto procesu.

Hlavním cílem extrakce informace je najit určitou informaci ve vstupních zdrojích a poskytnout tuto informaci koncovému uživateli v čitelné a jednoznačné podobě.

Přestože extrahovat informaci můžeme z různých typů zdrojů (obrázky, videa a pod.), v této práce se soustředíme jenom na textové zdroje informace. Pro lepší pojetí téhle domény si na začátku definujeme základní pojmy se kterými setkáme v práce, také se probereme hlavní metriky které používají pro vyhodnocení kvality systémů. Seznamíme se s některými historickými ději které zahájili výzkum téhle domény a s moderními iniciativami které se zabývají zpracovaním textové informací. Také se podíváme na často používané techniky při extrakci a vytvoříme systém pro extrakci vztahů mezi entitami.

Dále definujeme pojem *nekonzistentní prvky* na Wikipedii. Navrhneme algoritmus pro detekci nekonzistentních prvků a implementujeme systém pro detekci těchto prvků.

Třetí části téhle práce bude seznamení se seznamy na Wikipedii. Podrobnejší náhled na strukturu DBpedii a návrh a implementace systému pro doplnění chybějící informace na Wikipedii.

Na závěr provedeme testovaní a vyhodnocení systémů.

Kapitola 2

Definice základních pojmů a seznamení se základními metodami pro extrakce informací

Jak už bylo zmíněno výše, lidé už dávno žačali přemýšlet o automatizaci vyhledávání určité informace v textu. Táhle kapitola se věnuje základním pojmům spojeným s vyhledáváním, některým základním technikam a metrikám vyhodnocení kvality vyhledávacích systémů. Tady si definujeme pojmy *„Extrakce informace (IE)“*, *„Vyhledávání informace (IR)“* a vysvetlíme si rozdil mezi nimi. Více se soustředíme na IE, protože pro naše účely role IE je podstatně významnější.

2.1 Vyhledávání informací IR

Vyhledávání informace (Information retrieval) - je proces vyhledávání materiálů (většinou textových dokumentů) které představují nestrukturovaný[1] *text obsahující určitou informaci v kolekci dostupných zdrojů. [3]*

Z definice můžeme usoudit, že hlavním cílem společností, které se zabývají IR je vývoj algoritmů pro co nejrychlejší nález dokumentů, souvisejících s dotazem uživatele. Klasickým příkladem systému pro IR je *Google Scholar*.

2.2 Extrakce informací IE

Extrakce informace (Information extraction) - je proces vyhledávání a extrakce strukturované infrormace v nestrukturovaným, nebo semi-strukturovaným[2] *textu. [6] [7] [11]*

Hlavním cílem IE je transformovat zdrojový text na strukturovaný výstupní text, címž zajistíme redukci zbytečné informací. Systém pro IE se snaží analyzovat části dokumentů, které obsahují relevantní informace. IE je užitečná pro libovolný typ nosičů, ze kterého potřebujeme extrahovat data (obrázky, videa a pod.).

Systémy pro IE používájí v různých doménách, kde je zapotřebí schopnost rychle a bez manuálních úsilí vyhledat důležitou informaci například:

[1] Nestrukturovaná informace (data) - informace, která nemá předefinovanou struktutu dat (není organizovaná v předefinované podobě)

[2] Semi-strukturovaná informace (data) - forma strukturované infrormace která nedodržuje přesný format zapisu, ale přesto obsahuje tagy nebo jiné označení pro rozdělení sématických elementů.

- Lékařští výzkumníci často potřebují vyfiltrovat z obrovského množství vědeckých článků a publikaci určitou informaci, kterou oni (lékařští výzkumníci) potřebují pro řesení konkretního problému. Pro nález specifické informace jednoduchý způsob vyhledáváni na základě klíčových slov nemusí poskytovat vyhovující výsledky, protože lékařské definice můžou mít několik způsobů vyjadření.

- Analytici v ministerstvě obrany prohlíží velké množství textové informací pro odhalení plánovaných teroristických dějů. Zatímco systémy pro IR dokážou relativně rychle dohledat články které popisují teroristické děje, systémy pro IE dokážou poskytnout více konkretní informaci.

- Finanční pracovníci často potřebují vyhledávát informaci o stavu trhu.

Ještě jednou úlohou kterou řeší IE je extrakce vztahů mezi dvěma (obecně více) entitami. Extrakce vztahů je úloha detekce a charakterizování sémantického vztahu mezi entitami v textu. Například mějme věty:

Stephen King was inspired by creations of Howard Phillips Lovecraft.
Francis Bacon was influenced by Picasso's "The Three Dancers"painting.

z těchto vět můžeme extrahovat vztahy:

$$\textit{Howard Phillips Lovecraft} \xrightarrow{inspired} \textit{Stephen King}$$
$$\textit{The Three Dancers} \xrightarrow{influenced} \textit{Francis Bacon}$$

2.3 Historie

Historie IE začíná v sedmdesátých letech minulého století. Prvním komerčním systémem pro analýzu informací byl JASPER, ten používala společnost Reuters pro analýzu finančních novin. Od této doby zájem o IE postupně narůstal. V roce 1987 byla organizována první konference MUC.[3]

MUC-1 bylá zkušební konferencí. Každá společnost, která se chtělá zůčastnít, měla právo samostatně definovat tvar zdrojových dat ze kterých pak extrahovala určitá data.

Ve srovnání s MUC-1, MUC-2 (1989) mněla přesně definované úkoly. Každá společnost získávala množinu vzorkových dat a měla za úkol prověst extrakci určitých struktur. Na MUC-2 byly definovány pojmy *úplnost (recall)* 2.4.1 a *přesnost (precision)* 2.4.1. Obě konference se zamířili na analýzu vojenských zpráv.

MUC-3 (1991) se zaměřila na analýzu teroristických dějů v Centralní a Jížní částech Ameriky na základě článků keteré poskytovala společnost Foreign Broadcast Information Service. Po analýze těchto článků bylo možné předpovědět potenciální místa dalších útoků a pod..

MUC-4 (1992) bylo stejné účely jako MUC-3, ale nabízela pokročilé techniky pro analýzu.

MUC-5 (1993) provědli v rámce programu TIPSTER, [4] a věnovala se problémům spojeným s vztahy mezi internacionálními společnostmi.

[3] Message Understanding Conferences - konference, sponzorované společnostmi NRAD, DARPA, NOSC. Hlavním cílem konferencí bylo povzbudit zájem vývojářů o tvorbu nových metod pro extrakci informací.

[4] TIPSTER - Americký program pro výzkum a vývoj v oblastech extrakci a vyhledávání informace.

MUC-6 (1995) se soustředila na analýze nestrukturovaného textu, identifikaci různých událostí a vyplňováním předpřipravených šablon informacemi o těchto událostech. Také se snažila povzbudit zájem o tvorbu systému pro extrakci informací který by byl přenosný a který by dokázal porozumět extrahované informaci.

Každá další MUC se inspirovala předchozími. [10]

2.4 Metriky vyhodnocení

Abychom dokázali porovnávat kvalitu různých systémů z oblastí IR a IE, potřebujeme mít metriky. Ideálně bychom chtěli aby zvolené metriky byly snadno aplikovatelné na různé systémy, abychom posléze mohli porovnávat do jaké míry se jednotlivé systémy liší mezi sebou. Pro splnění těchto požadávků potřebujeme maximalně výkonné ale zároveň jednoduché vyhodnocovací metriky.

Při definici metrik budeme vycházet z předpokladu, že každá metrika musí odpovídat následujícím požadavkům:

- Musí dosahovat své maximalní hodnoty
- Musí dosahovat své minimalní hodnoty
- Musí být monotónní
- Musí být jednoduchá na pochopení a intuitivní
- Musí být spolehlivá
- Při použití nesmí zatížit systém

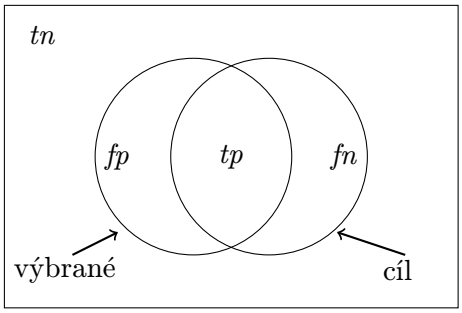
2.4.1 Přesnost a úplnost

V dnešní době mnohé systémy pracující s extrakcí a vyhledáváním informací jako metriky používají *přesnost (precision)*, *úplnost (recall)* a *F-measure*. Téhle metriky byly definované ma MUC-2 a do dnešní doby se stále používají. Dovedeme si představit situaci ve které máme množinu zdrojových dat ve které budeme provádět vyhledávání a množinu cílových dat která je podmnožinou zdrojových dat a kterou budeme chtít vyhledat. Po aplikaci IE nebo IR systému získáme množinu dat kterou chceme vyhledat. Tuto situaci znázorní obrázek 2.4.1, z tohoto obrázku je vidět, že množina zdrojových dat je $source = tn + fp + fn + tp$, množinou cílů je $target = tp + fn$ a množina výbraných dat je $selected = fp + tp$.

Tento obázek můžeme představit ve tvaru tabulky(2.1). Položky v téhle tabulce nám říkají, že systém dokázal správně odhalit a vybrat tp(true positives) entit[5], které bychom chtěli najít a ignoroval tn(true negatives) entit které nepatří množině cílů. Ale nedokázal odhalit fn(false negatives) entit které spadají do množiny cílů přestože výbral fp (false positives) entit které nepatří množině cílů.

1. *Přesnost (precision)* *- je vztah mezi počtem správně odhalených entit a celkovým počtem nálezů. Jinak řečeno: precision - je číslo udávající v kolikrát počet správných nálezů je menší než celkový počet nálezů. Z definici je zřejmé, že přesnost musí být kladném číslem v intervalu od nuly do jendé. [4] [1] [10]*

[5]Tady entita znamená článek nebo větu která obsahuje informaci ze zkoumaného zdroju.



Obrázek 2.1: Na obrázku *tp*-množina výbraných entit, která je podmnožinou cílů; *tn* - množina ignorovaných entit, která není podmnožinou cílů; *fp* - množina výbraných entit, která není podmnožinou cílů; *fn* - množina ignorovaných entit, která je podmnožinou cílů. [1]

	Aktualně	
Systém	cíl	¬cíl
výbrané	*tp*	*fp*
¬výbrané	*fn*	*tn*

Tabulka 2.1: Tabulková reprezentace 2.4.1

Z obrázku 2.4.1 a definici 1 můžeme odvodit vzorec 2.1:

$$precision = \frac{tp}{tp + fp} \tag{2.1}$$

2. *Úplnost (recall)* *- je vztah mezi počtem správně odhalených entit a celkovým počtem hledaných entit. Jinak řečeno: recall - je číslo udávající v kolikrát počet správných nálezů je menší než celkový počet správných entit. Z definici je zřejmé, že úplnost musí být kladné číslo v intervalu od nuly do jende. [1] [10]*

Z obrázku 2.4.1 a definici 2 můžeme odvodit vzorec 2.2:

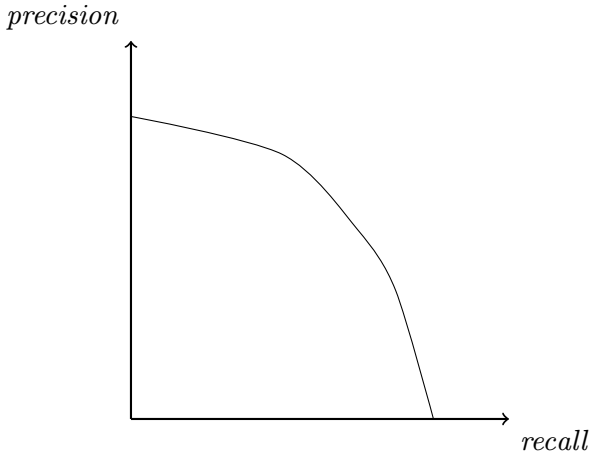
$$recall = \frac{tp}{tp + fn} \tag{2.2}$$

Z výše uvedených vzorců je očividně že čím více je hodnota *přesnost* tím méně hodnota *úplnosti* 2.2:

2.4.2 F-measure

Při použití systémů z oblasti IR a IE musíme často hledat kompromis mezi hodnotami *úplnosti* a *přesnosti*. V některých případech to nemá žádný smysl. V podobných případech můžeme použit mertiku *F-measure*, která kombinuje hodnoty *úplnosti* a *přesnosti*. Abychom vypočítali hodnotu *F-measure* musíme použít vzorec 2.3

$$F = \frac{(\beta^2 + 1)RP}{\beta^2 P + R} \tag{2.3}$$



Obrázek 2.2: Závislost *recall* na *precision*

Zde hodnota P určuje nam *precision*, hodnota R - *recall*. Hodnota β je faktorem, který určuje váhy *precision* a *recall*.

V většině případů jsou hodnoty P a R jsou vyvážené a můžeme použit vztah 2.4.

$$F = \frac{2PR}{R+P} \tag{2.4}$$

2.5 Techniky používané při extrakci informace

Táhle sekce popisuje základní techniky používané při IE. Pod pojmem „základní techniky“ rozumíme techniky, které jsou široce používané jako základ pro více sofistikované. V téhle práce se seznamíme se dvěma typy technik: „*techniky založené na syntaktických pravidlech*“ a „*techniky založené na Zpracování Přirozeného Jazyka (angl. NLP)*“.

Techniky založené na syntaktických pravídlech používají regulární výrazy pro provedení extrakce.

Techniky založené na NLP, na nižší úrovni zpracování, rozbijí text na tokeny. Rozdělení textu na tokeny dovoluje systému provádět podrobnejší analýzu kontextu na základě různých slovníků. Pod pojmem „*token*“ budeme rozumět slovo o kterém víme určitou informaci (např. jakym slovním druhem tohle slovo je).

2.5.1 Techniky založené na syntaktických pravidlech

Syntaktická pravidla popisují větu na nejnižší úrovní. Nejpopulárnejší způsob vyjadřit syntaktické pravidlo je regulární výraz. Část věty, která odpovídá výrazu, považována za hledanou informaci a poté extrahovaná. Je zřejmé, že při tomto přístupu není zapotřebí uvažovat žádná gramatická ani sémantická pravidla. Podobné přístupy se dají použit v různých případech, například při vyhledávání článků s nabídkou práce a v podobných případech kde informace má určitou strukturu. Obecně syntaktická pravidla můžou být interpretovaná pomocí konečného automatu (KA) a můžou být vygenerována automaticky z určitých struktur.

Zpracovaní regulárními výrazy

Regulární výrazy jsou široce používáne v systémech typu Unix jako vyhledávácí instrukce. Použití RV pro extrakci informací je stejné jako použití je pro vyhledávání v Unix. Jak bylo zmíněno, použití RV je vhodným řešením pro extrakci strukturované informace (např. datum, čas a pod.). Na rozdíl od více sofistikovaných technik, RV funguje velice rýchle protože jedinou podmínkou pro extrakci je shoda s vzorem.

Přestože se regulární výrazy dokážou rychle extrahovat části vět, které se shodují se vzorem, při vyhledávání se neuvažeje kontext který by mohl příspět k přesnějším výsledkům. Například výraz „`[0-9]+`“ reprezentuje posloupnost čísel, ale tento výraz není schopen správně odhalit posloupnost `100$` (res. nedokáže určit že se jedná o cenu). Tento nedostatek regularních výrazů může způsobit nežádoucí hodnoty *přesnosti* a *úplnosti*.

2.5.2 Techniky založené na NLP

Zpracování Přirozeného Jazyka (Natural Language Processing nebo NLP) - obor který zkoumá problémy analýzy textů (napsaných v podobě lidské řeči) počítačem. Techniky založené na NLP se spolehají na syntaktické a sémantické znalostí, které byly manualně vytvořeny lidmi pro speciální doménu. Systémy pro IE používají NLP pro předzpracování dokumentů. V této sekcí si podrobněji podíváme na jednu z nejduležitejších technik (Určování slovních druhů) a způsoby jak ji můžeme realizovat.

Určování slovních druhů (Part-of-speech tagging)

Určování slovních druhů (angl. Part-of-speech tagging, budeme používat zkratku POS tagging) je proces určování druhů jednotlivých slov v textu na základě definici slova a kontextu.

POS tagging je mnohem složitější proces, než si můžeme představit. Nestačí pouze vědět, že konkretní slovo je slovním druhem, protože v některých případech může slovo vyjadřovat zcelá odlišné druhy. A to není vzácný případ v mnohých jazycích (nás bude zajímat jenom angličtina), je velká část slov nejednoznačná. Například ve větě:

He decided to walk to the park.

walk (jít) reprezentuje sloveso v přítomném čase, ale v větě:

His walk was very short.

walk (procházka) je podstatným jménem.

Je známo, že v angličtině je devět základních slovních druhů: podstatné jméno, sloveso, determinátor (určitý a neurčitý), přídavné jméno, předložka, adverbium, spojka a citoslovce. Ale pro analýzu si nevystačíme s těmi devíti, potřebujeme se zanořit hlouběji do podtříd. Například podstatná jména můžeme rozdělit na množné číslo, přivlastňovací pád, jednotné číslo; slovesa můžou být v přítomném nebo minulém čase a pod..

Počivačový POS tagging rozlišuje 50 až 150 různých slovních druhů. Pro POS tagging existuje množství metod ale tadý se seznamíme s Rule-based tagging a Hidden Markov Model [2].

Zjednodušeně metody pro POS tagging můžeme rozdělit do dvou tříd: metody které nepotřebují učení s učitelem (unsupervised) a které potřebují (supervised).

Učení s učitelem a bez učitele

Jedním z hlavních rozdílu mezi metodami pro POS tagging je jejich příslušnost k třídam metod které potřebují učení s učitelem nebo které nepotřebují je. Třídě ve které metody nepotřebují učení konvenčně budeme říkat „první třída“, když to třídě kde metody potřebují učení budeme říkat „druhá třída“. Pro metody z první třídy je typické použití předpřipraveného „korpusu“. Pod pojmem „korpus“ můžeme rozumět text který už má označené slovní druhy nebo jiné informací, po analýze tohoto textu systém výtváří různe struktury které pak používá pro POS tagging. Na druhou stranu systémy z druhé třídy používají více sofistikované metody pro automatické přidělovaní slovních druhů.

Jedním z hlavních důvodu pro použití metod z první třídy je to že téhle metody jsou velice přenosné a dají se použit pro analýzu textů z různých domén, kdyžto metody z druhé třídy se dají požit jenom pro analýzu textů z konkrétní domény. Nicméně v případě existence kvalitně vytvořeného korpusu metody z druhé třídy ukazují podstatně lepší výsledky. Bohužel tvorba kvalitního korpusu je manuální prácí a není zaručená její 100% bezchybnost.

Rule-based tagging

Hlavní myšlenkou tohoto přístupu je určování slovního druhu na základě syntaktického kontextu věty. Pro lepší pojetí významu pojmu „syntaktický kontext“ si můžeme představit následující situaci: mějme slovo X druh kterého chceme určit, ve větě tomuto slovu předchází determinátor a po tomto slovu následuje podstatné jmého, z tohoto můžeme odvodit, že zkoumané slovo je přídavným jménem. Někdy se také dá použit morfematický rozbor slova například: když slovo končí na *-ing* a část slova před *-ing* je slovesem, můžeme předpokládat, že toto slovo je gerundium.

Většina systémů které používají tento přístup, než začít analyzovat text, potřebují trénovací data. Ačkoliv existují systémy které dokážou určit slovní druhy aniž by byly předem naučení, stejně potřebuji validaci výsledků z lidské strany. Klasický scénář podle kterého pracují tyto systémy je rozdělen do tři fází: v první fázi systém provádí POS tagging; dále uživatel provádí validaci výsledků; systém si zase zpracuje již opravený text a zapamatuje chyby, aby se v budoucnu nedopustil podobných chyb.

Stochastic tagging

Každé metodě která kombinuje frekvenci a pravděpodobnost při určování slovního druhu můžeme říkat stochasticka metoda. Nejjednodušší stochastické metody založené na pravděpodobností že konkretní slovo je konkretním slovním druhem. Při přiřazení tagu slovu systém uvažuje jenom informaci o tom jakým tagem bylo označeno stejné slovo v předcházejících případech.

Alternativou může být určování slovního druhu na základě pravděpodobnosti. Tomuto přístupu nekdy řikají *n-gram approach*, při tomto přístupu slovní druh cílového slova bude určen na základě slovních druhů n předcházejících slov. Typickým algoritmem pro realizaci tohoto chovaní je *Viterbiho algoritmus*.

Kombinaci dvou předcházejících přístupů získáme Skryté Markovovy modely (HMM). Pro HMM platí následující tvrzení: Každý skrytý tag stavu generuje slovo; každé slovo je nekorelované se všemi jinými slovy a jejích tagy; pravděpodobnost, že slovo je konkretním slovním druhem, vypočítává jenom na základě n předcházejících slov.

V dnešní době HMM je nejefektivnější přístup při POS tagging.

2.6 Moderní iniciativy zabývající se extrakcí informací

V předchozí sekci jsme se podíváli na základní techniky pro IE a příklady systémů, které používají téhle techniky. Tady se podíváme na moderní systémy pro IE.

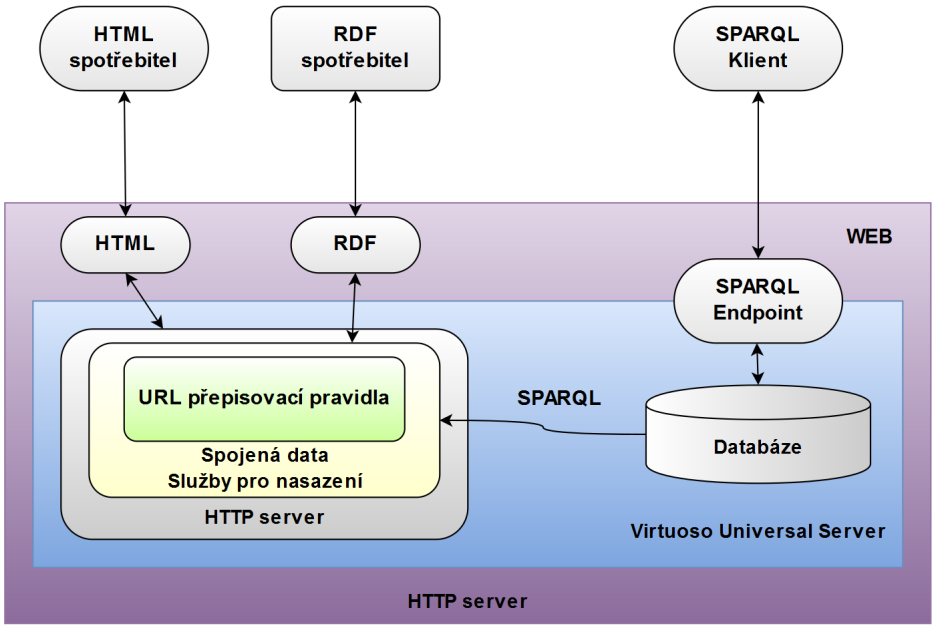
2.6.1 DBpedia

DBpedia je crowd-sourced komunita, zamířená na extrakci strukturované informací z Wikipedie (a jiných zdrojů) a publikování ji na Webu. DBpedia dovoluje uživatelům provádět sofistikované dotazy pro získání určité informací hledaní které klasickým způsobem (internet surfing) mohlo trvat delší dobu.

Anglická verze DBpedii obsahuje 4.58 milionů témat, z ních 4.22 jsou sdružené do ontologii, která obsahuje 1 445 000 lidí, 735 000 míst, 411 000 výtvarných dílů (z nich 123 000 muzikálních alb, 87 000 filmů, 19 000 video her), 241 000 organizací, 251 000 druhů živočichů a 6 000 druhů nemocí. Databáze přeložená na 125 různých jazyků, které dohromady pokrývají 38.3 milionů článků.

DBpedia má množství výhod před jinými existujícími databázemi: pokrývá množství domén; napojená na Wikipedii; je vícejazyčná; dovoluje uživatelům pokládat dotazy s poměrně složitou sémantikou (například "Poskytni mi informací o městech z New Jersey ve kterých žijí více něž 10 000 obyvatelů.").

Pro interakci s daty slouží OpenLink Virtuoso[6] na který se dá dotazovat jazykem SPARQL[7]. Architektura DBpedie je znázorněna na obrázku 2.6.1. [5]



Obrázek 2.3: Architektura DBpedii [5]

[6] OpenLink Virtuoso - moderní engine pro přístup k datum v databázi.
[7] SPARQL - dotazovací jazyk pomocí kterého se dá dotazovat na data uložene v RDF.

2.6.2 KnowItAll

KnowItAll je systém nezávislý na doméně pro automatickou extrakci informace z internetu. Pro vyhodnocení informací systém využívá statistické metody.

KnowItAll má k dispozici rozsáhlou ontologii a poměrně malý počet generických šablon založených na pravidlech, které používá pro tvorbu jiných pravidel pomocí kterých provádí extrakci informace pro každou třídu a vztah z ontologii. Systém se spoléhá na nezávislou na jazyce architekturu pro zpracování textu a doplnění ontologii různými fakty. KnowItAll navržen s podporou rozšiřování a výsokou průchodnosti.

Systém se skládá z několiká modulů, každý modul běži na samostatném vlákně a komunikuje s ostatními pomocí zpráv. Jako hlavní moduli lze uvést:

- **Extractor**: modul konkretizuje množinu extrakčních pravidel pro každou třídu a relaci z množiny obecných pravidel. Například mějme pravidlo *„noun_class1 such as list_of_nouns2“* indikuje, že každé podstatné jméno z seznamu *list_of_nouns2* je instanci třídy *noun_class1*. Toto pravidlo můžeme zde aplikovat při hledání měst např.: *We provide tours to cities such as Paris, Nice and Monte Carlo*. KnowItAll dokáže extrahovat města pro třídu *City*.

- **Search Engine Interface**: modul automaticky vytváří dotazy na základě pravidel. Každé pravidlo má související dotaz, vytvořený použitím klíčových slov z pravidla. Napřiklad pravidlo uvedené v předchozím bodě vynutí systém vyhledat stránky s větami, na které může být aplikované pravidlo. Po vyhledání Extractor extrahuje informaci použitim tohoto pravidla.

- **Assessor**: modul využívá statistiky, které říkají do jaké míry jsou předpoklady Extractoru správné. Assessor využívá PMI[8] mezi slovy a frázemi, podobný přístup využivají vyhledávací Web enginy. Například, předpokládejme že Extractor určil že „Liege“ je názevem města. Jestli PMI mezi „Liege“ a fráze „city of Liege“ je vysoký, můžeme předpokládat že „Liege“ patří třídě *City* a předpoklad Extractoru je platný.

- **Database**: modul uchovává informaci v databázi. Ten přístup má množství výhod, například data v databázi jsou perzistentní a databáze dovoluje rychlou aktualizaci a rychlý přístup k datům.

Podrobný popis systému není účelem této práce, více informaci je v [9] [8].

[8] Point Mutual Information (PMI) - je mírou asociace využivané v teorii informatiky a statiky.

Kapitola 3

Implementace systémů pro extrakci vztahů vybraného typu a identifikaci nekonzistentních prvků ve Wikipedii

3.1 Definice cíle

Velké množství projektů pro extrakci vztahů se snaží řešit úlohy definované ACE[1]. ACE zamířen na vývoj algoritmů pro extrakci vztahů mezi entitami. Během vývoju projekt definoval množinu druhů vztahů mezi entitami: **fyzické** (jedna entita je umístěná fizický blízko druhé), **personalní/socialní** (jedná entita je členem rodiny druhé entity) a pod..

Úlohou téhle práce bude vytvořit systém který, na základě analýzy určitých syntaktických konstrukcí ve větach, dokáže odhalit které výrazy můžou vyjadřovat sémantický vztah určitého typu.

3.2 Návrh řešení problému

3.2.1 Analýza vět

Vzhledem k tomu že program musí analyzovat anglickou verzi Wikipedie, nebylo by na škodu si uvědomit, že angličtina je analytickým jazykem. Zanmená to že každá věta zapsaná v angličtině má určitou strukturu. Zatím se soustředíme na větach které vyjadřují vztah, kterému konvenčně budeme říkat *„Ovlívnění mezi entitami“*. Něž začneme provádět extrakci musíme provést analýzu struktury vět, které můžou vyjadřovat tento vztah.

Po analýze zjistíme že nejčastejší struktury vět které vyjadřují tento vztah formálně můžeme popsat následovně:

- *Entity1* $\xrightarrow{verb\_in\_past\_tense}$ *Entity2*. V tomto případě *Entity1* ovlivnila *Entity2*, a způsob ovlivnění vyjedřen pomoci slovesa v minulém čase. *H. P. Lovecraft inspired Stephen King.*

[1] Automatic Content Extraction (ACE) – výzkumný program pro vývoj pokročilých technik pro extrakci informací.

- *Entity1* $\xleftarrow{verb\_in\_past\_tense}$ *Entity2*. Tento případ je zrcadlovým odrazem předcházejícího jenom tady *Entity2* ovlivnila *Entity1* a to ovlivnění vyjedřeno pomoci pasiva. *Stephen King was inspired by H. P. Lovecraft.*

- *verb_in_past_tense Entity*. Za předpokladu, že daná vět je na stránce o druhé entitě, a víme, že mezi entitami existuje vztah, můžeme předpokládat že daná věta vyjadřuje tento vztah. *He was inspired by H. P. Lovecraft.* Za předpokladu, že daná věta je na stránce o Stephenovi Kingovi, můžeme předpokládat, že se zájmeno *He* vztahuje k němu.

- *Entity verb_in_past_tense*. Obdobně jako v předcházejícím janom jiné umístění slovesa. *H. P. Lovecraft inspired him.*

Přestože existuje množství jiných způsobů kterými dá vyjadřit tento vztah, soustředíme se jenom na tyto.

3.2.2 Definice formatu dostupných dat

Vzhledem k tomu, že budeme chtít analyzovat vztahy mezi velkým počtem entit, budeme muset zpracovat velké množství stránek, v tomto případě přímé dotazování na „živou“ Wikipedii bude časově náročné. Naštěstí na školních serverech máme uložené části Wikipedie v MG4J souborech. MG4J soubory jsou korpusy získané ze stránek Wikipedii. Věty ze stránek jsou předzpracovány a rozdělené na tokeny. Každý tokem má v sobě množství informací, ale pro naš případ bude nejdůležitějším to, že z toho souboru dokážeme získat informaci o slovním druhu a základním tvaru slova.

Další informaci, kterou budeme potřebovat, bude množina dvojic, mezi kterými budeme chtít určit vztah. Budeme předpokládat, že mezi dvojicemi v jedné množině je stejný typ vztahu. V případě potřeby rozlišit dvou nebo více lidí se stejným jménem, vedle jména bude umístěno upřesnění, například jestli osoba je aktérem, spisovatelem nebo malířem.

3.2.3 Návrh funkcionality programu

Při návrhu funkcionality budeme vycházet z analýzy v 3.2.1. Abychom provedli extrakci vztahů musíme mít množinu vět kterou bude potřeba analyzovat. Věty na analýzu budeme brát ze stránek o entitach v MG4J souborech.

Po získání vět budeme předpokládat, že každá získaná věta vyjadřuje vztah. Z každé získané věty, na analýzu, budeme brát jenom části ve kterých může být klíčové slovo: v případě spoluvýskytu dvou entit v jedné větě – vezmeme část mezi entitami; v případě výskytu jenom jedné – část před entitou a po entitě.

V 3.2.1 jsme si všimli že pro vyjadření vztahu často používají slovesa v minulém čase, také si všimneme, že se pro vyjadření vztahu často používají stejná slova. Na základě toho navrhneme algoritmus pro vyhledávání klíčových lemmat 1.

Algoritmus 1 Vyhledávání klíčových lemmat

```
function getTargetLemmas(sentences)
    Lemmas = Array()                                          ▷ Úložiště lemmat
    for s in sentences do                                     ▷ Pro každou větu
        if both entities in s then
            PartBetween = getPartBetween(s)                   ▷ Část věty mezi entitami
            Lemmas.append(getPotentialLemmas(partBetween))    ▷ Přidat potenciální
lemma
        else
            PartBefore = getPartBefore(s)                     ▷ Část věty před entitou
            PartAfter = getPartAfter(s)                       ▷ Část věty po entitě
            Lemmas.append(getPotentialLemmas(PartBefore))
            Lemmas.append(getPotentialLemmas(PartAfter))
        end if
    end for
    MostFreqLemmas = getMostFreqLemmas(Lemmas)                ▷ Nejčástějí se vyskytující
lemma
    return MostFreqLemmas
end function
```

3.3 Volba programovacího paradigma

Jako programovací paradigma, zvolíme modulární programování. Toto programovací paradigma bude nejlíp odpovídat návrhu z 3.2.3 a přináší své výhody. Například při rozdělení programu na moduly:

1. Modul pro extrakci vět spojených s entitami *Sentences extractor*
2. Modul pro extrakci tokenů potenciálně vyjadřujicích vztah *Relations extractor*
3. Modul pro spočitání statistik výskytů jednotlivých tokenů *Statistic collector*

v případě změny formátu dat v zrojovém souboru s entitami nebo souboru MG4J stačí provest opravu (nahrazení) modulu *Sentence extractor* a program zůstane funkční. Nebo při použití jiného přístupu pro vyhledávání vztahů stačí nahradit modul *Relation extractor* a pod..

3.4 Volba programovacího jazyka

Jako programovací jazyk zvolíme *python*. *Python* vysokoúrovňový skriptovací programovací jazyk, orientovaný na vysokou výkonnost a čitelnost zdrojového kódu. Podporuje množství programovacích paradigmat, pro naše účely podpora modulárního programovacího paradigmatu je jedním z klíčových důvodů. Dalším stejně důležitým kritériem je to že *python* je nezávislým na operačním systému jazykem. Díky tomu výsledný program bude fungovat na všech operačních systémech, aniž bychom museli provádět prováděl změny v kódu, což dovoluje vyvíjet na libovolném OS. Třetím argumentem ve prospěch volby jazyka *python* je to že on poskytuje velice flexibilní mechanizmus pro práci s textem a drtivá většina operací bude prováděna právě nad řetězci.

Jako pomocný jazyk zvolime *Bash*. Vzhledem k tomu že primárním operačním systémem bude Linux, skripty v jazyce *Bash* zjednoduší spouštění programu.

3.5 Rozdělení programu na logické části

Jak už bylo zmíněno v 3.3, pro řešení úlohy využijeme modulárního programování. Program rozdělíme na čtyři moduly: *Main*, *Sentences extractor*, *Relations extractor*, *Statistic collector*. Program musí být spuštěn paralelně na všech serverech pro získání přístupu ke všem MG4J souborum.

3.5.1 Modul Main

Je vstupním bodem programu. Tento modul má nejjednoduší úlohu ze všech modulů – provést zpracování parametrů a předat řizení dalšímu modulu (*Sentences extractor*). Po převzetí rízení zpět, zapsat informací získanou od modulu *Relations extractor* 3.5.3 do serializačních souborů. Zápis do serializačních souborů je zapotřebí prože MG4J soubory uložené na různých serverech a pro získání maximalně přesné statistiky potřebujeme zpracovat vše věty získáné ze všech serverů.

3.5.2 Modul Sentences extractor

Modul přebírá řízení od modulu *Main*. Od něj získává cestu k souboru *source* (soubor obsahuje dvojice entit), cesty k MG4J souborům a informaci o tom jaký typ[2] entit je uložen v *source*. Po získání těchto druhů informací modul začne vyhledávání stránek spojených s jednou z entit. Po vyhledávání a následném zpracování, modul najde věty které můžou vyjadřovat vztah mezi entitami (předpokládá se že ve větě, která vyjadřuje vztah, se musí vyskytnout obě entity nebo entita o které není daná stránka). Po nálezu vět, modul předává řízení dalšímu modulu (*Relations extractor*).

3.5.3 Modul Relations extractor

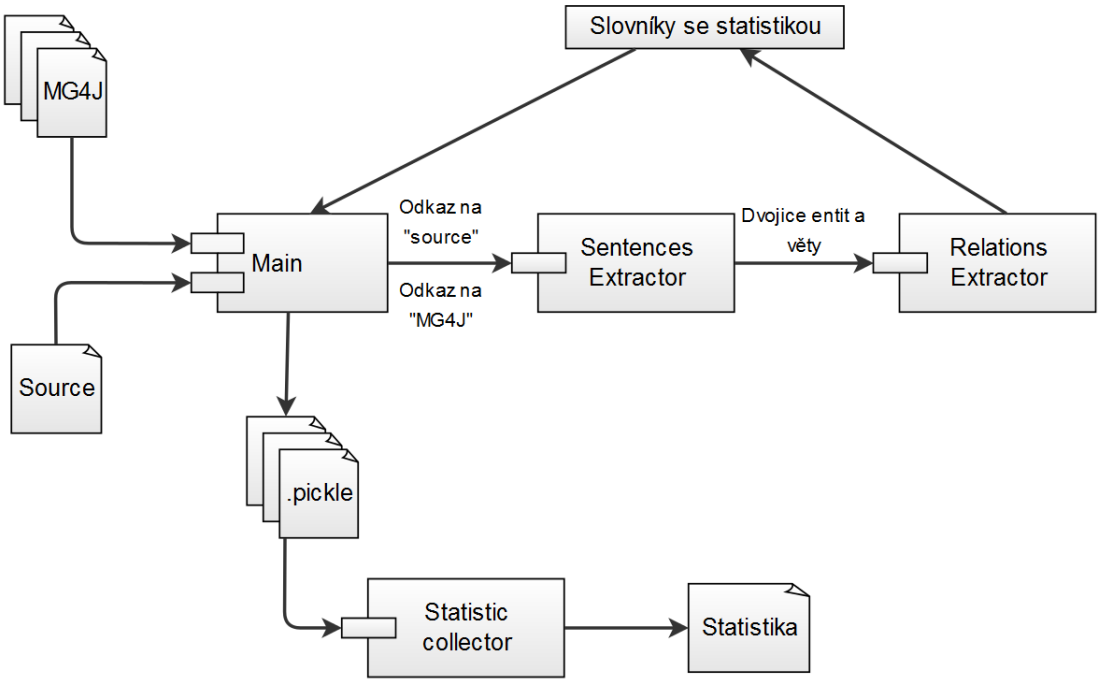
Od modulu *Sentences extractor* získá dvojice entit a věty které potenciálně vyjadřují vztah mezi entitami. Každou větu rozdělí na části 3.2.3. Z každé části extrahuje token který potenciálně vyjadřuje vztah a uloží informaci o počtu výskytů daného tokenu ve větach do slovníku. Po zpracování všech vět vrátí řízení modulu *Main* který zapíše tento slovník do serializačního souboru.

3.5.4 Modul Statistic collector

Po ukončení práce modulů *Main*, *Sentences extractor* a *Relations extractor* bude spušten modul *Statistic collector* který sebere informaci ze všech slovníků v serializačních souborech do jedneho slovníku, najde nejčastěji se vyskytující lemma a prohlásí je za lemma vyjadřující vztahy.

Archytektura programu znázorněna na obrázku 3.5.4.

[2]Logické entity rozdělujeme na dva typy: vlastní jmeno a jíné. Tohle rozdělení potřebujeme protože ve větě o osobě nemusí spolu vyskytovat jméno a příjmení ale může vyskytnout jenom příjmení či jméno, když to v případě že entita je nějakým dílem, název daného díla bude úplný.



Obrázek 3.1: Architektura programu pro extrakci vztahů.

3.6 Návrh a implementace systému pro identifikaci nekonzistentních prvků ve Wikipedii

3.6.1 Definice cíle

Wikipedia je rozsáhlá a editovat stránky může každý registrovaný uživatel. Při tvorbě článku uživatel může odkazovat na jiné relevantní články (například na články o městech, lidech a pod.). V případě že stránka, na kterou chce odkazovat autor, neexistuje Wikipedia zvýrazní tento odkaz červenou barvou. Pod pojmen „nekonzistentní prvek“ budeme rozumět právě odkazy které odkazují na články které zatím neexistují.

3.6.2 Návrh řešeni

Každá stránka na Wikipedii má unikátní název. Můžeme považovat název stránky za unikátní klíč. Prvním krokem pro identifikaci nekonzistentních odkazů bude sbírání unikátních klíčů všech střánek. Dalším krokem bude sbírání všech odkazů z každé stránky které odkazují na jiné stránky. Posledním, třetím, krokem bude vyhledávání klíčů z množivy získáne v druhem kroku v množině klíčů z prvního kroku. V případě úspěšného nálezu dojdeme k závěru o tom že dany odkaz je konzistentní jinak konzistentní není.

3.6.3 Zdroje informace

Jak bylo uvedeno výše, Wikipedia má velké množství stránek a dotazovaní na „živou“ Wikipedii bude časově náročné. Z toho vyplývá, že jako v případě s systémem pro extrakci vztahů by bylo mnohem rychlejší analyzovat lokalně uložená data. Naštěstí na školních

serverech máme staženou poslední verzi „Database backup dumps[3]“. Každá stránka zapsana ve tvaru „wikitextu[4]“. XML dump obsahuje různé druhy informací ale pro naše účely stačí vědet, že název stránky (unikatní klíč) je zabalen do tagů `<title>` a `</title>`, a to že odkaz na jinou stránku ve wikitextu zabalen do hranatých závorek `[[name]]` (tady `name` je název stránky který v XML dumpu zabalen do `<title>`).

3.6.4 Implementace systému

Jako programovací paradigma zase zvolíme modulární programování ze stejných důvodu popsaných 3.3. Jako programovací jazyky zase zvolíme *python* a *bash*. V tomto případě role jazyka *bash* bude podstatně větší než v případě systému pro extrakci vztahů. *Bash* poskytuje množství způsobů zpracovaní textu, vyhledávání v textu, porovnání jednotlivých řetězců a hlavně to dělá podstatně rychlej než jiné množství programovacích jazyků. Vzhledem k tomu že budeme zpracovávat dump soubor celé Wikipedie díky rýchlosti kterou poskytuje *bash* ušetřme čas.

Při návrhu architektury budeme vycházet z úvah popsaných v 3.6.2. Rozdělíme program na tři logické moduly pro každý krok.

- Modul **Title collector**. Úkolem tohoto modulu bude sebrat názvy všech stránek uložených v dumpu. Tento modul je napsaný v jazyce *bash*. Vzhledem k tomu že název stránky je zabalen do tagu `<title>`, stači použit jednoduchý regulární výraz pro nalezení všech názvů, což daný modul dělá.

- Modul **Links extractor**. Úkolem tohoto modulu bude vyhledat všechné odkazy na všech stránkach ve Wikipedii. Tento modul je napsaný v jazyce *python*. Při vyhledavání je vždy potřeba pamatovat na které stránce provadíme vyhledávání aby v případě nálezu nekonzistentního odkazu bylo snadno určit z jaké stránky je tento odkaz odkazován. Proto musíme zpracovat jednotlivé stránky jednu po druhé. Avšak zpracovaní souboru po jedné stránce bude příliš časově náročné ale zároveň chceme mít maximalně rychlý systém. Pro zrychlení běhu programu použijeme multithreading[5]. Každé vlákno zpracovává stránku a informaci o odkazech na stránce zapisuje do souboru.

- Molud **Red links detector**. Úkolem tohoto modulu bude v souboru který vytváří modul **Link extractor** najit odkazy které nejsou v souboru který vytváří modul **Title collector**. Tento modul stejně jako i **Title collector** napsaný v jazyce *bash*.

Postupným spouštěním těchto modulů získáme soubor s nekonzistentními odkazy.

[3] Database backup dumps - Kompletní kopie všech stránek z Wikipedie uložených ve formatu XML.
[4] Wiki markup (wikitext nebo wikicode) - format textu který se používá pro tvorbu stránek na Wikipedii.
[5] Multithreading – schopnost procesoru (CPU) spouštět více procesů nebo vláken současně.

Kapitola 4

Návrh a implementace systému pro automatické doplňovaní seznamů na Wikipedii

Díky svým uživatelům Wikipedia je rostoucí online encyklopedii. Měsičně na Wikipedii přibývá kolem 20 000 článků. Pro případy kde uživatel potřebuje získat informaci o všech instancích nejakého seznamu, při tom uživatel neví které konkretní instance patří do seznamu (např. uživatel potřebuje zjistit místo narození všech českých spisovatelů a nezná všechny české spisovatele), existují stránky s odkazy na téhle instance. Danou strukturu si můžeme představit jako strom ve kterém "kořenové"stránky mají odkazy na "listové"stránky které obsahují podrobnejší informaci o instanci. Vzhledem k tomu že větší část článků vytváří registrované uživatele může nastat případ kde uživatel z nějakých důvodů, po vytvoření článku, nepřidal odkaz na nově vytvořený článek do "kořenové"stránky.

4.1 Definice cílu

Naším cílem bude navrhnout a implementovat systém který dokáže detekovat chybějící data v seznamu, vyhledat je a doplnit ve spravném formátu který odpovídá danému seznamu. Vzhledem k tomu, že Wikipedia nedefinuje žádné přísné omezení pro seznamy a seznamy jsou obyčejnými stránkami které vytváří uživatelé, je potřeba zvolit nejaký typ seznamu, který budeme doplňovat. V této práci se budeme zabývat doplňováním seznamů sérií do sezon různých seriálů.

4.2 Tvar seznamů se sérií

Především provedeme analýzu struktury seznamů se sérií. Na Wikipedii se tyto seznamy nacházejí na stránkach s jménem „List of *name* episodes“, kde *name* je název seriálu. Informace o sériích v jednotlivých sezonách je uložená ve tvaru tabulky 4.2.

Tabulkový tvar je front-endem seznamů protože tabulku vidí návštěvník stránky. My potřebujeme analyzovat back-end (wikicode popisující stránku). Po otevření wikicodu zjistíme že tabulky nejsou přímo na stránce seznamu (List of *name* episodes), ale každá tabulka má vlastní stránku a je odkazovana ze stránky seznamu. Tím pádem při doplňování seznamu musíme doplňovat tabulku na stránce konkretní sezony, ale ne na stránce seznamu všech sezon.

No. overall	No. in season	Title	Directed by	Written by	Original air date	Prod. code	U.S. viewers (millions)
442	1	"Homer the Whopper"	Lance Kramer	Seth Rogen & Evan Goldberg	September 27, 2009	LABF13	8.31[27]
Comic Book Guy creates a new superhero called Everyman who takes powers from other superheroes. Homer is cast as the lead in the film adaptation. To get Homer into shape, the movie studio hires a celebrity fitness trainer, Lyle McCarthy to help him. Homer gets into great shape but his life falls apart when his fitness trainer quits and the movie bombs at the box office. *Guest Stars*: Seth Rogen, Matt Groening and Kevin Michael Richardson							
443	2	"Bart Gets a 'Z'"	Mark Kirkland	Matt Selman	October 4, 2009	LABF15	9.32[28]
Mrs. Krabappel is fired for drinking alcohol on the job (thanks to kids in her class spiking her coffee as revenge for confiscating their cell phones and BlackBerrys) and replaced with a hip, young teacher named Zachary Vaughn.							
444	3	"The Great Wife Hope"	Matthew Faughnan	Carolyn Omine	October 11, 2009	LABF16	7.5[29]
When the men and boys of Springfield become obsessed with Ultimate Fighting, Marge leads a protest against it when she catches Bart fighting in school — and ends up fighting the head of the Ultimate Fighting syndicate in order to have it banned. *Guest Star*: Chuck Liddell							

Obrázek 4.1: Část tabulky se seznamem sérií. Snímek pořizený ze stránky na Wikipedii.

4.3 Tvorba systému

4.3.1 Návrh algoritmu

Po obdržení názvu seriálu od uživatele je potřeba vyhledat stránku s poslední sezonou seriálu protože série můžou chybět jenom v poslední sezoně. Po získání stránky extrahujeme tabulku s seznamem a určíme kterou informaci je potřeba získat pro doplnění seznamu (které sloupce jsou v tabulce). Po zjištění které sloupce je potřeba vyplnit, získáme patřičnou informaci a vytvoříme tabulku s získanými daty. Nahradíme nově vytvořenou tabulkou starou. Tím získáme stránku s nově přidanými sériemi a pak provedeme aktualizaci stránky na Wikipedii.

4.3.2 Zdroje informace

Jako zdroj informaci ze kterého budeme brat data pro doplnění jednotlivých sloupců použijeme DBpediu. O DBpedii bylo zmíněno v 2.6.1 ale tady si probereme daný systém podrobneji.

DBpedia obsahuje velké množství dat sebraných z různých zdrojů (primárním zdrijem je Wikipedia). Pro nas nejdůležitějším faktorem při volbě DBpedii jako zdroje bude to že ona uchovává data ve strukturovaném tvaru. Množství domén o kterých DBpedia uchovává informaci označené tagy, každý tag má množinu vlastností které můžeme zjistit o instanci označené tagem. Například tagem *RadioStation* označené různé radio stanice a název konkretní stanici obsažen ve vlastnosti *rdfs:label*. Tito tagy jsou shromážděné do ontologii která

představuje acyklický graf, více informaci o tagech a ontologii DBpedii v [5].

Jako zdroj stránek které bude potřeba doplňovat použijeme dump Wikipedii ktery jsme používali pro implementaci systému na detekci nekonzistentních odkazů v 3.6. Vzhledem k tomu že seznam seriálů může být velkým použitit lokalního dumpu dokažeme vyhnout zbytečnému použiti internetu avšak nepřijdeme o data protože dump je „odrazem“ Wikipedii.

4.4 Implementace

Jako programovací paradigma zase použijeme modulární programování a jako programovací jazyk použijeme *python*. Zase při návrhu funkcionality jednotlívých modulů budeme vycházet z navrženého algoritmu 4.3.1. Každému kroku z algoritmu bude odpovídat modul.

4.5 Rozdělení systému na moduly

Prvním krokem bude obdržení názvu seriálu a vyhledávání stránky s poslední sezonou. Vzhledem k tomu, že dump obsahuje téměř 12.5 milionů stránek a chceme mít co nejrychlejší systém, použijeme multithreading. Pojmenujeme první modul **Page retrieval**. Rolí tohoto modulu bude vyhledat stránky sezon cílového seriálu. Proto paralelně spouští množství vláken ktere současně analyzují stránky a v případě že stránka je stránkou o sezoně zapíše ji do souboru. Může se zdát, že když potřebujeme obsah stránky jenom s poslední sezonou je zbytečně zapisovat stránky všech sezon, ale vzhledem k tomu že nevíme kolik sezon má daný seriál a vyhledování v dumpu je časově náročné kvůli množství stránek které je potřeba zpracovat, zápis všech stránek do souborů a následný výběr poslední sezony ušetři čas.

Druhým krokem bude vyhledávání na stránce tabulky s seznamem sérií a určování jakou informaci bude potřeba získat pro doplnění jednotlívých sloupců. Pojmenujeme tento modul **Table analyzer**. Tento modul je jednoduchým parserem. Na základě znalosti syntaxe pro tvorbu tabulek v jazyce wikicode, modul extrahuje informaci o sloupcích které je potřeba doplnit.

Třetím krokem je získání informace kterou budeme doplňovat do jednotlivých sloupců. Pojmenujeme tento modul **SPQRQL requester**. Tento modul bude provádět dotazování na DBpediu prostřednictvím jazyku SPARQL. Po získání informace o tom jaké sloupce je potřeba vyplnit a jakou sezonu jakého seriálu je potřeba doplňovat, modul vytvoři dotaz v jazyce SPARQL s sémantikou *„Poskytni informaci o určitých vlastnostech každé sérií která patří dané sezone daného seriálu“*. Po obdržení odpovědi od DBpidie modul vyfiltruje zbytečnou informaci a předá získaná data dalšímu modulu.

Čtvrtým krokem bude provedení vlastního doplňování získaných dat z DBpedie. Pojmenujeme tento modul **Page editor**. Cílem tohoto modulu bude vytvořit novou tabulku kde jednotlivé sloupce budou obsahovat informaci získanou z DBpedie. Dodržováním pravidel definovaných Wikipedií pro tvorbu tabulek, modul vytváří tabulku a vyplní ji získanými daty. Po doplnění tabulky modul nahrazuje starou tabulku nově vytvořenou a výslednou stránku zapisuje do souboru.

Posloupnost těchto kroků by mněla být řízena pomocným modulem. Pomocný modul pojmenujeme *Main*. Tento modul postupně provádí kroky 1 až 4.

Posledním, pátým krokem, bude samotná aktualizace stránky na Wikipedii. Pro provedení aktualizaci použijeme již existující nástroj *pywikibot*. Tento nástroj poskytuje množství různých funkcí pro práci s Wikipedií. Pomoci tohoto nástroje dokážeme nahradit starý obsah stránky novým.

Kapitola 5

Testování a vyhodnocení výsledků

V téhle kapitole provedeme testování programů a vyhodnotíme výsledky.

5.1 Systém pro extrakci vztahů

Testování budeme provádět na třech nezávislých množinách dvojic. Výsledky testování jsou uvedeny v tabulce 5.1. V tabulce sloupec **Počet vět** - počet extrahovaných vět z každého souboru; **Manualně extrahované konstrukce** - počet manualně extrahovaných konstrukcí ktere vyjadřují vztah; **Validní předpoklady** - počet validních konstrukcí extrahovaných systémem; **Nevalidní předpoklady** - počet navalidních konstrukcí extrahovaných systémem.

Počet vět	**Manualně extrahované konstrukce**	**Validní předpoklady**	**Nevalidní předpoklady**
38	22	10	4
79	43	18	8
55	31	16	7

Tabulka 5.1: Výsledky *„Ovlívnění mezi entitami“*

Z uvedených statistik dokážeme vypočitat hodnoty *přesnosti, úplnosti a F-measure*:

1. množina: $recall = \frac{10}{22} = 0.455$, $precision = \frac{10}{14} = 0.714$,
   $F\text{-}measure = \frac{2 \cdot 0.455 \cdot 0.714}{0.455 + 0.714} = 0.556$
2. množina: $recall = \frac{18}{43} = 0.419$, $precision = \frac{18}{26} = 0.692$,
   $F\text{-}measure = \frac{2 \cdot 0.419 \cdot 0.692}{0.419 + 0.692} = 0.522$
3. množina: $recall = \frac{16}{31} = 0.516$, $precision = \frac{16}{23} = 0.696$,
   $F\text{-}measure = \frac{2 \cdot 0.516 \cdot 0.696}{0.516 + 0.696} = 0.593$

Z výše uvedených výsledků můžeme vypočítat střední hodnoty: $recall = 0.463$, $precision = 0.701$, $F\text{-}measure = 0.557$. Z toho můžeme usodit o tom že slabostí tohoto systému je určení vět které vyjadřují vztah. Tohle způsobeno tím že výskyt alespoň jedne entity ve větě

je nepostačující prodminkou protože některé vztahy vyjádřené větami se složitějšími konstrukcemi. Pro zlepšení hodnoty *recall* je potřeba použit více sofistikovaný způsob určení vět. Avšak hodnota *precision* je relativně vysoká, takže výpočet frekvenci výskytu určitých syntaktických konstrukcí poskytuje docela přesné výsledky.

Výpočet časové náročnosti je poněkud složitější. Dobu běhu programu ovlivňují počet entit mezi kterými je potřeba určit vztah, zatíženost serverů a počet vět které je potřeba analyzovat. Přibližná doba vyhledavání a analýzy vět pro každý soubor byla 6 minut. Vzhledem k tomu že každý soubor obsahoval 6 dvojic na zpracovaní jedné dvojice je potřeba 1 minuta.

5.2 Systém pro identifikaci nekonzistentních odkazů

Pro ověřování funkcionality systému vytvoříme "zkrácenou"verzi dumpu. "Zkrácená"verze Wikipedie obsahuje jenom 6885 stránek, na všech stránkach je 1 018 524 odkazů. Pro vyhledání všech titulků stránek na "zkrácené"Wikipedii bylo potřeba 4 sekundy reálného času, z toho dokážeme vypočitan že pro nález a zápis jednoho titulku systém potřebuje jenom $6 \cdot 10^{-4}$ sekund. Pro vyhledání všech odkazů na stránkach a následný zápis je do souboru potřeba 29 sekund reálného času, z toho zase vypočítáme že pro vyhledání a zápis jedného odkazu potřebujeme $3 \cdot 10^{-5}$ sekund. Pro identifikaci jestli odkaz je konzistentní nebo není je potřeba $2 \cdot 10^{-2}$ sekundy.

5.3 List completer

Při testování budeme sledovat dva parametry: čas potřebný pro vyhledávání sezony a doplňování chybějící informace, shodu doplněné informací s očekávanou. Zase rýchlost vyhledávání bude záviset na zatíženosti serveru a počtu stránek sezon které potřeba vyhledat a zapsat do souborů. Vyhledávání 26 sezon a zápis je do souborů trvalo jednou hodinu při relativně zatíženém serveru. Následné dotayovaní na DBpediu a doplňování ziskané informací trvalo kolem jedne minuty. Doplněna informace odpovídá očokované avšak některou informaci nelze získat z DBpedii (například počet prohlížení) protože není na stránce o sezoně.

Kapitola 6

Závěr

Na závěr bych chtěl shrnout výsledký dané práce. Na začátku práce byly přesně definované úkoly, které bylo potřeba řešit, aby bylo možně tvrdit že práce byla úspěšně zvládnutá. Hlavní části této práce byl návrh a implementace systému pro statistické zpracování a následnou extrakci vztahů určitého typu mezi dvěma entitami. Právě této části bylo věnováno nejvíce času.

Zpracování přirozeného jazyka (NLP) je pro mě zcela novou oblastí. Než jsem začal vývoj systému, musel jsem se seznámit s základními pojmy, používanými v této oblasti, hlavními cíli společností, které vyvíjí systémy pro NLP, v jakých oblastech používají systémy pro NLP a jaké výhody téhle systémy přinášejí. Dalším krokem bylo seznámení s metrikami které se používají pro vyhodnocení kvality systémů a pro porovnaní systémů. Nejsložitější části této práce byl studium technik které se používají při NLP, pokročilé techniky (např. HMM) mají složitou matematickou bázi.

Další fáze při tvorbě systému byl výběr programovacího jazyka a programovacího paradigma protože správná volba pomůže co nejlépe řešit úlohu. Před návrhem architektury systému bylo potřeba prozkoumat zdroje dat (v našem případě MG4J soubory) a také syntaktické struktury anglického jazyka pro vyjadření vztahů určitého typu. Nejsložitější části při implementaci byla validace mezivýsledků (například mezivýsledkem modulu pro extrakci vět mají být věty s minimalně jednou entitou z dvojice). Také s ohledem na množství dat (v našem případě jde o celou Wikipedii) bylo potřeba přemýšlet o racionálním použití počitačových zdrojů (CPU, RAM a pod.). Testování a vyhodnocení výsledků také bylo poměrně složitou úlohou protože bylo potřeba manualně vyhledávat vztahy z velkého množství vět pro porovnání s výstupy systému.

Další části téhle práce byla implementace systému pro detekci nekonzistentních odkazů na Wikipedii. Při implementaci tohoto systému bylo potřeba si uvědomovat že zpracování cele Wikipedii je zdlouhavým procesem a pro zrychlení bylo potřeba použití multithreadingu, což mohlo přispět klasickým chybam jako uváznutí, data race a pod..

Poslední části práce byla implementace systému pro automatické doplňování chybějících dat do seznamů. Bohužel, vzhledem k absenci jakékoliv struktury, nelze realizovat univerzální algoritmus pro doplňování. Pro implementaci systému bylo potřeba se seznamit s syntaxi jazyka *wikicode* pro tvorbu tabulek. Také, vzhledem k tomu že zdrojem dat byla zvolena DBpedia, bylo potřeba nastudovat syntax jazyku SPARQL a porozumět struktuře která obsahuje informace (RDF). Pro zrychlení vyhledávání stránek sezon v dumpu bylo potřeba použit multithreading.

Na úplný závěr bych chtěl říct že byla provedená velká práce při studiu teoretickýho materiálů a při tvorbě systémů. Během tvorby bylo potřeba řešit množství problém spoje-

ných jak s pochopením teorii tak s aplikaci teoretických znalosti na praxi. Přesto všechné definované na začátku úkoly byly zvládnuté.

Literatura

[1] Christopher D. Manning, Hinrich Schütze: *Foundation of Statistical Natural Language Processing*. Massachusetts Institute of Technology, první vydání, 1999, ISBN 978-0262133609, 0262133601, 268 s.

[2] Christopher D. Manning, Hinrich Schütze: *Foundation of Statistical Natural Language Processing*. Massachusetts Institute of Technology, první vydání, 1999, ISBN 978-0262133609, 0262133601, 317-340 s.

[3] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze: *An Introduction to Information Retrieval*, kapitola 1 Boolean retrieval. Cambridge University Press, Duben 2009, str. 1.

[4] Diana Maynard, Wim Peters, Yaoyong Li: *Metrics for Evaluation of Ontology-based Information Extraction*. University of Sheffield, 2006.

[5] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patric van Kleef, Sören Auer, Christian Bizer: DBpedia - A Large-scale, Multilingual Knowledge Base Extracted From Wikipedia. *Semantic Web Journal*, 2015: s. 167–195.

[6] Klaus Zechner: A Literature Survey on Information Extraction and Text Summarization. Duben 1997.

[7] Line Eikvil: Information Extraction from world Wide Web. 1999, 6 s.

[8] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, Alexander Yates: Unsupervised named-entity extraction from the Web: An experimental study. Technická zpráva, University of Washington, Seattle, WA, Duben 2005.

[9] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, Alexander Yates: Web-Scale Information Extraction in KnowItAll: (preliminary results). 2004: s. 101 – 102.

[10] Ralph Grishman, Beth Sundheim: *Proceedings of the 16th conference on Computational linguistics - Volume 1*, kapitola Message Understanding Conference-6: a brief history. Association for Computational Linguistics Stroudsburg, 1996, s. 466–467.

[11] Raymond J. Mooney, Razvan Bunescu: *ACM SIGKDD Explorations Newsletter - Natural language processing and text mining*, kapitola Mining knowledge from text using information extraction. ACM New York, 2005, str. 3.

Příloha A

Obsah CD

A.1 Odevzdané soubory

- `relationsExtractor/` program pro extrakci vztahů
- `redLinksDetector/` program pro detekci nekonzistentních odkazů
- `listCompleter/` program pro doplňování seznamů na Wikipedii
- `plakat.pdf` plakat demonstrující výsledky práce
- `diploma/` adresář s zdrojovými souboru v formatu LaTeX
- `diploma_xpanov00.pdf` text práce.

Příloha B

Manual

B.1 Relation extractor

V adresáři `relationsExtractor/` jsou umístěné soubory:

- **main.py** soubor reprezentuje modul **Main** 3.5.1. Jako vstupní parametry modul očekává cestu k MG4J souboru – parametr `-d/--database`. Dalším parametrem musí být cesta k souboru s dvojicemi – parametr `-s/--source`. Na každém řádku souboru musí být dvě entity oddělené znakem tabulátoru. Další parametry mají upřesňovat typ entit: parametr `-l/--left` upřesňuje typ levé entity `-r/--right` – pravé. Typ může být bud' `name` pro lidí nebo `other` pro díla.
- **sentencesExtractor.py** reprezentuje modul **Sentences extractor** 3.5.2. Uživatel by neměl používat tento modul, řízení tohoto modulu zcela zajistí modul **Main**.
- **relationsExtractor.py** soubor reprezentuje modul **Relations extractor** 3.5.4. Řízení tohoto modulu také je zcela automatické, tento modul používá modul **Sentences extractor** a uživatel by neměl osobně využívat tento modul.
- **statisticCollector.py** soubor reprezentuje modul **Statistic collector** 3.5.4. Soubor neočekává žádný vstupní parametry. Modul předpokládá existenci adresáře `statistics` který obsahuje serializační soubory vytvořené modulem *Main*.

Také se v adresáře objeví pomocné soubory:

- **servers.txt** obsahuje seznam serverů s MG4J soubory.
- **persons, persons2** jsou příklady dvojic entit.
- **clr.sh** čistí adresář `statistics` pro případ opakovaného spouštění programu.
- **run.sh** skript zjednoduší spouštění programu pro analýzu každého MG4J souboru. Podobně modulu *Main*, na vstupu očekává parametry: `-s`, který je cestou k souboru s dvojicemi, a také typy entit v souboru `-r` a `-l`.

Program se spustí příkazem: `parallel-ssh -A -i -t 0 -h servers.txt -l loginUživatele "cd /cesta/k/adresáři/; bash run.sh -s source -l type -r type"` kde `loginUživatele` je login pro přihlášení na server, `source` je soborem s entitami a `type` je typem entit. Po ukončení běhu programu je potřeba ručně spustit **statisticCollector.py** příkazem `python3 statisticCollector.py`. Pro mazání serializačních souborů slouži **clr.sh**, spustí se příkazem `bash clr.sh`

B.2 Red links detector

V adresáři `redLinksDetector/` jsou umístěné soubory:

- **titlesCollector.sh** soubor reprezentuje modul **Title collector** 3.6.4. Jako vstupní parameter očekává cestu k wikidumpu (`-d`). Modul vyhledá vše stránky uložené v dumpu a zaznamená je do souboru **allArticles**.
- **linksExtractor.py** soubor reprezentuje modul **Links extractor** 3.6.4. Tento modul také očekává cestu k wikidumpu (`-d/--dump`). Vyhledává vše odkazy na všech stránkach a zaznamenává je do souboru **allLinks**.
- **redLinksDetector.sh** soubor reprezentuje modul **Red links detector** 3.6.4. Prohlíží odkazy z souboru **allLinks** a snáží se vyhledat je v souboru **allArticles**, v případě neúspěchu zapíše odkaz do souboru **reds**.
- **run.sh** soubor slouží pro automatizaci spuštění jednotlivých skriptů. Na vstupu očekává cestu k wikidumpu (**-d**). Skript postupně spouští jednotlivé moduly.

Program se spouští příkazem: `bash run.sh -d /cesta/k/wikidumpu`.

B.3 List completer

V adresáři `listCompleter/` jsou umístěné soubory:

- **pageRetrieval.py** reprezentuje modul **Page retrieval** 4.5. Na vstupu modul očekává cestu k wikidumpu (`-d/--dump`) a cestu k seznamu seriálů které je potřeba doplňovat (`-s/--source`). Po spuštění začne vyhledávání stránek sezon seriálů v dumpu a následné zapisování je do adresáře `initiate`. Také vytváří adresář **modified** do kterého budou zapsané doplněné stránky.
- **main.py** reprezentuje modul **Main** 4.5. Je řídícím modulem, na vstupu očekává jenom cestu k souboru ve kterém budou sepsané jednotlivé sezony které vyhledal modul **Page retrieval**. Dále tento modul provádí posloupnost kroků z 4.5.
- **tableAnalyzer.py** reprezentuje modul **Table analyzer** 4.5. Analyzuje tabulku s sérií v vrátí jednotlivé sloupce které je potřeba doplňovat.
- **sparqlRequester.py** reprezentuje modul **SPQRQL requester** 4.5. Modul vyhledá informace kterou budou doplněné jednotlivé řádky tabulky.
- **pageEditor.py** reprezentuje modul **Page editor**. Modul získává data kterými vyplní jednotlivé řádky tabulky nové tabulky kterou pak nahrazuje starou.
- **run.sh** skript zjednodušuje spouštění programu. Na vstupu, jako i **pageRetrieval.py**, očekává cesty k wikidumpu (`-d`) a souboru s seznamem seriálů (`-s`). Skript postupně spouští modul **Page retrieval**, pak vytváří soubor s seznamem sezon, který potřebuje **Main**, a spouští modul **Main** který zajišťuje analýzu, vyhledávání a doplňování řádků tabulky.

Program se spouští příkazem:
`bash run.sh -d /cesta/k/wikidumpu -s /cesta/k/seznamu/seriálů`