Předmluva

Devátý workshop **ITAT'09 – Informační Technologie – Aplikace a Teorie** se konal v Horském hotelu Kráľova studňa (http://www.kralova-studna.sk/), ve výšce 1300 metrů nad mořem v Nízkých Tatrách, Slovensko, od 25. do 29. září 2009.

Workshop ITAT (http://www.itat.cz/) je místem setkání vědců a odborníků zejména z České republiky a Slovenska pracujících v informatice. Oficiálním jazykem pro prezentace jsou čeština a slovenština. Letos z workshopu vydáváme dvě publikace – konferenční sborník s původními vědeckými pracemi v anglickém jazyce (11 příspěvků) a tento sborník.

Celkově bylo zasláno 51 příspěvků. Tento sborník obsahuje

- 14 původních vědeckých prací a
- 9 rozšířených abstraktů určených pro prezentaci formou posteru

Všechny práce byly recenzovány nejméně dvěma nezávislými recenzenty.

Tematicky je workshop široce zaměřen, zahrnuje všechny oblasti informatiky od teoretických základů informatiky a bezpečnosti, přes data a semantický web až po softwarové inženýrství. Velký důraz je kladen zejména na výměnu informací mezi účastníky. Workshop poskytuje také příležitost pro studenty k prvníveřejné prezentaci své práce a k debatám se zkušenějšími kolegy, velký prostor je proto vymezen pro neformální diskuse. Místo konání je tradičně alespoň 1000 m.n.m a mimo přímý dosah veřejné dopravy.

Workshop organizovali

- Ústav Informatiky Univerzity P.J. Šafárika, Košice
- Matematicko-fyzikální fakulta Univerzity Karlovy v Praze
- Ústav informatiky AV ČR Praha

Částečná podpora byla poskytnuta z projektů programu Informační Společnost Tematického Programu II Národního Programu Výzkumu ČR 1ET100300517 a institucionálního záměru MSM-0021620838.

Vřelé díky patří sponzorům - Profinit (http://www.profinit.eu/)

a CSKI (http://www.cski.cz/)

Filip Zavoral, Peter Vojtáš

Program Committee

Filip Zavoral, (Chair), *Charles University, Prague, CZ*
Gabriela Andrejková, *University of P.J. Šafárik, Košice, SK*
Mária Bieliková, *Slovak University of Technology, Bratislava, SK*
Leo Galamboš, *Czech Technical University, Prague, CZ*
Ladislav Hluchý, *Slovak Academy of Sciences, Bratislava, SK*
Tomáš Horváth, *University of P.J. Šafárik, Košice, SK*
Karel Ježek, *The University of West Bohemia, Plzeň, CZ*
Jozef Jirásek, *University of P.J. Šafárik, Košice, SK*
Jana Kohoutková, *Masaryk University, Brno, CZ*
Stanislav Krajči, *University of P.J. Šafárik, Košice, SK*
Věra Kůrková, *Institute of Computer Science, AS CR, Prague, CZ*
Markéta Lopatková, *Charles University, Prague, CZ*
Ján Paralič, *Technical University, Košice, SK*
Dana Pardubská, *Comenius University, Bratislava, SK*
Martin Plátek, *Charles University, Prague, CZ*
Jaroslav Pokorný, *Charles University, Prague, CZ*
Karel Richta, *Charles University, Prague, CZ*
Gabriel Semanišin, *University of P.J. Šafárik, Košice, SK*
Václav Snášel, *Technical University VŠB, Ostrava, CZ*
Vojtěch Svátek, *University of Economics, Prague, CZ*
Jiří Šíma, *Institute of Computer Science, AS CR, Prague, CZ*
Július Štuller, *Institute of Computer Science, AS CR, Prague, CZ*
Peter Vojtáš, *Charles University, Prague, CZ*
Jakub Yaghob, *Charles University, Prague, CZ*
Stanislav Žák, *Institute of Computer Science, AS CR, Prague, CZ*
Filip Železný, *Czech Technical University, Prague, CZ*

Organizing Committee

Tomáš Horváth, (chair), *University of P.J. Šafárik, Košice, SK*
Hanka Bílková, *Institute of Computer Science, AS CR, Prague, CZ*
Peter Gurský, *University of P.J. Šafárik, Košice, SK*
Róbert Novotný, *University of P.J. Šafárik, Košice, SK*
Jana Pribolová, *University of P.J. Šafárik, Košice, SK*
Veronika Vaneková, *University of P.J. Šafárik, Košice, SK*

Organization

ITAT 2009 Informační Technologie – Aplikace a Teorie byl organizován:

University of P.J. Šafárik, Košice, SK
Institute of Computer Science, AS CR, Prague, CZ
Faculty of Mathematics and Physics, Charles University in Prague, CZ
Slovak Society for Artificial Intelligence, SK

Table of Contents

ITAT'09

Information Technology – Applications and Theory

SCIENTIFIC PAPERS

Analýza a porovnanie metód zhlukovej analýzy*

Gabriela Andrejková and Miroslav Sukeľ

Ústav informatiky, Univerzita P. J. Šafárika, Košice
Gabriela.Andrejkova@upjs.sk, Miro.Sukel@gmail.com
WWW home page: http://ics.upjs.sk/~andrejkova

Abstract. *V práci sa zaoberáme metódami zhlukovej analýzy a zhlukovacích algoritmov. Je navrhnutý modifikovaný fuzzy c-means algoritmus. Pre hierarchické aglomeratívne zhlukovanie, metódu k-means a fuzzy c-means zhlukovanie sú zavedené základné kritériá pre ohodnotenie kvality výsledkov zhlukovania. Výsledky uvedených zhlukovacích algoritmov na klasifikáciu dát, získaných generovaním pri splnení stanovených podmienok, do zhlukov ukazujú, že modifikovaný fuzzy c-means algoritmus dosahuje najlepšie kvalitatívne výsledky, ale na úkor časovej zložitosti.*

1 Úvod

Množstvo informácií, ktoré ľudia získavajú každý deň a reprezentujú ich ako údaje dôležité pre ďalšie spracovanie, si vyžaduje tiež vhodné postupy na toto spracovanie. Prirodzeným prostriedkom - postupom, ktorý sa veľmi často využíva je klasifikácia, ktorá súvisí so zhlukovaním.

Zhlukovanie (klastrovanie, zhluková analýza) je metóda, ktorá sa používa na klasifikáciu skupín objektov. Jej cieľom je zatriediť množinu pozorovaných objektov do zhlukov (klastrov) tak, aby objekty zaradené do toho istého zhluku mali vyššiu mieru podobnosti ako objekty v ostatných zhlukoch.

Na hľadanie zhlukov existuje viacero algoritmov, ktoré je možné rozdeliť do dvoch skupín v závislosti od toho, či objekty sú zaraďované do vopred známych tried alebo triedy vznikajú v priebehu klasifikácie. Sú to:

- *algoritmy s dozorom (supervised algorithms)*, množiny vstupných vektorov dát ($\overline{x} \in \Re^k$, kde k je dimenzia vstupného priestoru) sa zobrazujú do konečnej množiny označení diskrétnych tried ($\{1, 2, \ldots, m\}$, kde m je celkový počet typov tried). Toto je modelované obyčajne pomocou nejakej matematickej funkcie $y = y(\overline{x}, \overline{w})$, kde $\overline{w}$ je vektor prispôsobiteľných parametrov; hodnoty týchto parametrov sa nastavujú pomocou induktívnych učiacich algoritmov;
- *algoritmy bez dozoru (unsupervised algorithms)*, ktoré sledujú skutočné zhlukovanie a pri ktorých je cieľom je separovať konečnú množinu dát do konečnej a diskretnej množiny "prirodzených" dátových zhlukov.

* Výskum je podporovaný VEGA grantom . 1/0035/09

Hľadanie metód na konštrukciu zhlukov má dlhú históriu, mohli by sme ísť až k Aristotelovi [3]. Referencie na metódy zhlukovania je možné nájsť napríklad v [4], [5], [6]. Veľmi dobrý prehľad poskytuje [2]. V [1] je možné nájsť modifikácie zhlukovacích algoritmov s dozorom.

Cieľom tohto článku je porovnať experimentálne výsledky získané použitím piatich známych algoritmov a jedného modifikovaného *c-means algoritmu*. Výsledky získané pomocou všetkých uvedených algoritmov v experimente na generovaných dátach vyhodnotiť použitím popísaných kritérií kvality.

2 Modelové zhlukovanie

Podľa spôsobu organizácie objektov je možná nasledujúca klasifikácia zhlukovacích algoritmov:

Hierarchické metódy organizujú analyzované objekty do systému zhlukov. Tento systém zhlukov pozostáva z navzájom rôznych neprázdnych podmnožín pôvodnej množiny. Má charakter postupností rozkladu množiny objektov, kde každý rozklad je zjemnením rozkladu predchádzajúceho (divízne metódy zhlukovania), respektíve nasledujúceho (aglomeratívne metódy zhlukovania).

a) *Aglomeratívne zhlukovacie metódy* – pri týchto metódach pristupujeme ku každému objektu ako k samostatnému zhluku a postupne vyberáme zhluky, ktorých vzájomná vzdialenosť je najmenšia, tieto potom zhlukujeme, pokiaľ nemáme všetky objekty zoskupené vo vhodnom počte zhlukov.
b) *Divízne zhlukovacie metódy* – pri tomto spôsobe sa uplatňuje opačný postup. Na začiatku je jeden zhluk, ktorý je tvorený všetkými objektami. Tento zhluk postupne rozdeľujeme na menšie. Pri rozdeľovaní zhluku sa pokúšame nájsť také nasledujúce rozdelenie, pre ktoré by platilo, že priemerná vzdialenosť objektov v zhluku, ktorý chceme vytvoriť je minimálna. Postup opakujeme, pokiaľ každý objekt netvorí samostatný zhluk.

Nehierarchické metódy - pri tomto prístupe vstupný priestor objektov rozdelíme do nehierarchického systému zhlukov; tento systém zhlukov pozostáva z neprázdnych a navzájom rôznych podmnožín množiny

vstupných objektov, v ktorom platí, že žiadna z nich nie je vlastnou podmnožinou inej a medzi jednotlivými podmnožinami neexistujú žiadne hierarchické vzťahy.

c) *Optimalizačné zhlukovacie metódy* – na základe vhodne zvoleného kritéria optimálnosti rozkladu hľadáme optimálny rozklad množiny objektov.
d) *Metóda analýzou modusov (pravdepodobnostný prístup)* – hodnotu náhodnej veličiny, ktorá má lokálne najväčší počet výskytov (lokálne maximum) budeme nazývať *modus*. Pri tejto metóde podľa polohy a existencie modusov frekvenčnej funkcie definujeme jednotlivé zhluky.

3 Metódy vyhodnocovania výsledkov zhlukovania

V závislosti od vstupných objektov alebo inicializácie môžu zhlukovacie algoritmy produkovať rôzne výstupy. Je potrebné definovať mieru, ktorá tieto výstupy ohodnotí a určí, ktorý model kvalitnejšie opisuje štruktúru dát. V nasledujúcej časti rozdelíme už spomenuté typy zhlukovania do dvoch skupín a pre každú skupinu na základe [7] uvedieme indexy, ktoré budú hodnotiť mieru kvality zhlukovacieho postupu. Taktiež predstavíme veličiny **kompaktnosť** a **separovanosť**, ktoré budeme brať do úvahy pri vyhodnocovaní výsledkov jednotlivých zhlukovacích metód. Ostrým (crispy) zhlukovaním budeme nazývať také rozdelenie vstupných objektov, pre ktoré každý zatriedený vstupný prvok bude patriť iba do jedného zhluku. Ak vstupné objekty budú priradené do viacerých zhlukov, vtedy budeme hovoriť o fuzzy zhlukovaní.

Budeme používať nasledujúce označenie: Nech $X = \{\overline{x_i};\ i = 1, \ldots, n\}$ je n-prvková množina objektov. C je konečná množina zhlukov, $|C| = m$, C_j označuje j-ty zhluk a jeho centrum je označené $\overline{c_j}$, počet prvkov v C_j je $|C_j| = m_j$. $C_c = \{\overline{c_j}; j = 1, \ldots, m\}$ je množina centier zhlukov. Objekty patriace do j-teho zhluku tvoria množinu $\{\overline{x_1^j}, \ldots, \overline{x_{m_j}^j}\}$. Označme u_{ij} príslušnosť prvku $\overline{x_i}$ do zhluku C_j.

3.1 Kritérium kvality ostrého zhlukovania

Nech α_j je priemerná vzdialenosť všetkých objektov v j-tom zhluku od jeho centra, d_{kj}^c je vzdialenosť centier k-teho a j-teho zhluku.

Davies – Bouldin (DB) index

Davies – Bouldinov index je definovaný ako

$$DB = \frac{1}{C} \sum_{m=1}^{C} \max_{i,j=1,\ldots,m, i \neq j} s_{ij}$$

pričom s_{kj} je miera podobnosti medzi zhlukmi C_k a C_j (s centrami $\overline{c_k}$ a $\overline{c_j}$) je definovaná tak, aby spĺňovala nasledujúce podmienky:

a) $s_{kj} \geq 0$,
b) $s_{kj} = s_{jk}$,
c) ak $\alpha_k = 0$ a $\alpha_j = 0$, tak aj $s_{kj} = 0$,
d) ak $\alpha_j > \alpha_k$ a $d_{kj}^c = d_{kl}^c$, tak $s_{kj} > s_{kl}$,
e) ak $\alpha_j = \alpha_k$ a $d_{kj}^c < d_{kl}^c$, tak $s_{kj} = s_{kl}$.

Miera podobnosti má nadobúdať nenulové kladné hodnoty a mala by spĺňať predchádzajúce podmienky. Budeme ju vyjadrovať ako

$$s_{kj} = \frac{\alpha_k + \alpha_j}{d_{kj}^c}.$$

Vzhľadom k tomu, že dobrý zhlukovací algoritmus by mal maximalizovať vzdialenosti medzi zhlukmi (separovanosť) a minimalizovať vzdialenosti medzi objektmi v zhlukoch (kompaktnosť), tak lepší zhlukovací algoritmus bude mať nižšiu hodnotu DB indexu.

3.2 Kritérium kvality fuzzy zhlukovania

Xie – Beni (XB) index

Tento index je vhodné použiť na ohodnotenie výsledkov zhlukovacích algoritmov, ktoré zatriedia každý prvok zo vstupnej množiny do viacerých zhlukov a každému prvku priradia hodnotu jeho príslušnosti k zhluku.
Fuzzy odchýlku d_{ij}^f pre vstupný prvok $\overline{x_i}$ a zhluk C_j definujeme ako vzdialenosť $\overline{x_i}$ od centra j-teho zhluku $\overline{c_j}$ ohodnotenú fuzzy príslušnosťou prvku $\overline{x_i}$ do zhluku C_j.

$$d_{ij}^f = u_{ij} ||\overline{x_i} - \overline{c_j}||$$

Pre zhluk C_j definujeme **odchýlku** φ_j **zhluku** C_j

$$\varphi_j = \sum_{i=1}^{n} (d_{ij}^f)^2$$

Označme $d_{\min}$ minimálnu vzdialenosť medzi centrami zhlukov

$$d_{\min} = \min_{k,l,k \neq l} ||\overline{c_k} - \overline{c_l}||$$

Na základe vyššie uvedeného definujeme hodnotu *XB indexu* ako

$$XB = \frac{\beta}{S}$$

pričom $\beta = \frac{\sum_{j=1}^{m} \varphi_j}{n}$ a $S = (d_{\min})^2$. Nižšie hodnoty β poukazujú na kompaktnejšie zhluky a vyššie hodnoty S na lepšie separované zhluky. Takže menšia hodnota XB indexu odráža lepšie oddelenie zhlukov.

3.3 Kompaktnosť a separovanosť

Ako už bolo uvedené, kompaktnosť predstavuje mieru vzdialeností medzi objektmi v zhlukoch a separovanosť mieru vzdialenosti medzi zhlukmi. Uvažujme n-prvkovú množinu k-rozmerných dát $X = \{\overline{x_i}; i = 1, \ldots, n\}$ a množinu centier zhlukov $C_c = \{\overline{c_j}; j = 1, \ldots, m\}$. Označme u_{ij} príslušnosť prvku $\overline{x_i}$ do zhluku C_j.

Kompaktnosť j-teho zhluku C_j definujeme ako

$$K_j = \frac{1}{|C_j|} \sum_{i=1}^{|C_j|} u_{ij}^2 ||\overline{x_i^j} - \overline{c_j}||$$

A celkovú kompaktnosť pre všetky zhluky ako

$$TK = \frac{1}{m} \sum_{j=1}^{m} K_j$$

Separovanosť k-teho zhluku bude vyjadrená nasledovne:

$$S_k = \min_{j, j \neq k} ||\overline{c_k} - \overline{c_j}||^2.$$

Ak uvažujeme o m zhlukov, tak celkovú separovanosť vyjadríme

$$TS = \frac{1}{m} \sum_{j=1}^{m} S_j$$

Podiel celkovej kompaktnosti a celkovej separovanosti (označme KZ) budeme používať ako kritérium kvality pri hľadaní najlepšieho rozdelenia do zhlukov.

$$KZ = \frac{TK}{TS}$$

Kompaktnosť, separovanosť a hodnotu prislúchajúceho indexu budeme považovať za mieru kvality výsledkov dosiahnutých pri použití zhlukovacích algoritmov.

4 Porovnanie zhlukovacích algoritmov

V experimente sme využili už spomínané zhlukovacie postupy ako hierarchické a aglomeratívne zhlukovanie, k-means a fuzzy c-means zhlukovanie. Tieto postupy sme upravili tak, aby pre ne v aplikácii bolo možné špecifikovať, či cieľom zhlukovania je získať čo najkvalitnejšie zhlukovanie, alebo zatriediť vstupné dáta do vopred stanoveného počtu zhlukov.

4.1 Charakterisktika použitých algoritmov

⋆ **Hierarchický aglomeratívny algoritmus s vopred určeným počtom zhlukov**
- reprezentuje hierarchické aglomeratívne zhlukovanie
- vstupom je n-rozmerná množina objektov, ktoré je potrebné zatriediť do vopred stanovených m zhlukov
- ako mieru vzdialeností sme použili Euklidovskú vzdialenosť
- vstupné prvky sa zatriedia do zhlukov, potom sa pre každý vytvorený zhluk určí jeho centrum a vypočíta sa kvalita zhlukovania

⋆ **Hierarchický aglomeratívny algoritmus**
- je podobný ako predchádzajúci algoritmus, ale zhlukovanie nekončí, ak sú dáta už zatriedené do zhlukov; v každej iterácii vytvoríme nový zhluk a kontrolujeme, aký vplyv má nové zatriedenie na kvalitu zhlukovania; ak sa kvalita zhlukovania v predchádzajúcich dvoch iteráciách zhoršila, vtedy je zhlukovanie ukončené a za výsledné rozdelenie budeme pokladať to s najlepšou kvalitou zhlukov

⋆ **K-means algoritmus**
- reprezentuje k-means zhlukovanie
- cieľom je zatriediť n-rozmernú množinu objektov do m zhlukov
- pri tomto algoritme sme použili parameter ε, ktorým budeme kontrolovať, či výsledok zhlukovania nezačne po určitom počte iterácií konvergovať, hodnotu ε sme nastavili na $0,01$
- zhlukovanie ukončíme, ak rozdiel kvality zhlukovania v dvoch po sebe nasledujúcich iteráciách je menší ako táto hodnota

⋆ **Fuzzy c-means algoritmus s vopred určeným počtom zhlukov**
- reprezentuje fuzzy c-means zhlukovú metódu
- vstupom je n-rozmerná množina objektov, ktoré je potrebné zatriediť do vopred stanovených m zhlukov, pričom každý vstupný objekt môže byť zaradený do viacerých zhlukov
- hodnota kritéria zastavenia iteračného procesu má v tomto prípade hodnotu $\varepsilon = 0.01$

⋆ **Fuzzy c-means algoritmus**
- zhlukovanie prebieha podobne ako v predchádzajúcom prípade, ale opäť kontrolujeme kvalitu zhlukovania; pri zhoršení kvality sa algoritmus zastaví a za výsledok budeme pokladať ten s najlepšou kvalitou

⋆ **Upravený fuzzy c-means algoritmus**

Tento algoritmus vychádza z fuzzy c-means algoritmu, ktorý je upravený tak, aby bolo možné pomocou hodnôt niektorých parametrov špecifikovať jeho správanie. Hlavným dôvodom vytvorenia tohoto algoritmu bolo umožniť sledovať, ako hodnoty parametrov ovplyvnia výsledok zhlukovania. Pre tento algoritmus je možné definovať hodnoty nasledujúcich parametrov:

a) *počet zhlukov* - počet zhlukov, do ktorých sa majú vstupné prvky zatriediť
b) *počet iterácií* - maximálny počet opakovaní algoritmu potrebný na získanie najlepšieho zatriedenia pre zadaný počet zhlukov

c) *fuzzifikácia* - index fuzzifikácie
d) *maximálny počet nezatriedených prvkov* - výsledkom algoritmu sú zhluky, do ktorých sú zatriedené vstupné prvky.

Niektoré prvky však algoritmus nedokázal zatriediť do žiadneho z vytvorených zhlukov, a tak sme ich označili ako *nezatriedené*. Postup sa opakuje, pokiaľ hodnota všetkých nezatriedených prvkov nebude menšia ako definovaná hranica.

Význam symbolov použitých v algoritme:

- Euklidovská vzdialenosť medzi i-tým prvkom vstupnej množiny a j-tým centrom,

$$d_{ij}^c = ||\overline{x_i} - \overline{c_j}|| \tag{1}$$

- hodnota príslušnosti i-teho prvku do j-teho zhluku - $u_{ij}, 0 \leq u_{ij} \leq 1, \sum_{i=1}^{m} u_{ij} = 1$, ktorá bude počítaná podľa

$$u_{ij} = \frac{1}{\sum_{k=1}^{m_j} (\frac{d_{ij}}{d_{kj}})^{2/(m-1)}} \tag{2}$$

Postup:

K1: *Inicializácia matice príslušnosti.*
Inicializácia matice príslušnosti U na náhodné hodnoty.

K2: *Určenie polohy centier v zhlukoch.*
Vypočet centier zhlukov podľa vzťahu

$$\overline{c_j} = \frac{\sum_{k=1}^{n} u_{kj}^r \overline{x_k^j}}{\sum_{k=1}^{m_j} u_{kj}^r},$$

pričom r je váhový exponent alebo index fuzzifikácie, $r \in \langle 1, \infty)$, a počet zhlukov bude daný hodnotou parametra *"počet zhlukov"*.

K3: Vypočet miery nepodobnosti medzi centrami a dátami použitím vzťahu

$$J(U, \overline{c_1}, \ldots, \overline{c_m}) = \sum_{i=1}^{m} \sum_{j=1}^{n} u_{ij}^r (d_{ij}^c)^2,$$

pričom hodnota príslušnosti je daná vzťahom (2) a vzdialenosť d_{ij}^c vzťahom (1).
Aktualizovať matice U^k a U^{k+1}, predstavujúce hodnoty príslušnosti v iteráciách nasledujúcich po sebe.
Hodnotu parametra *fuzzifikácia* využijeme pri výpočte hodnôt príslušností použitím (2).

K4: Pokiaľ nie je prekročený stanovený počet iterácií a ak norma rozdielu dvoch po sebe idúcich matíc hodnôt príslušností je väčšia ako hodnota ε, tak opakovať postup od K2.

K5: Výpočet je ukončený po vykonaní daného počtu iterácií alebo ak rozdiel jednotlivých príslušností bol menší ako hodnota kritéria zastavenia iteračného procesu. Vypočítať hodnoty určujúce kvalitu tohto zatriedenia.

K6: Z výsledku zhlukovania zistíme, aký počet daných prvkov nebolo možné zatriediť do žiadneho z vytvorených zhlukov. Ak je počet nezatriedených objektov väčší ako hodnota parametra *maximálny počet nezatriedených prvkov*, tak opakovať celý postup od K1.

4.2 Príprava dát

Vytvorili sme tri typy vstupných súborov:

- súbor obsahujúci malú testovaciu vzorku určený na testovanie aplikácie
- súbor s náhodnými dátami obsahuje reálne dáta, avšak bez informácie o optimálnom zatriedení
- súbor reprezentujúci vzorku dát spĺňajúcu zadané kritériá

Hlavným dôvodom vytvárania vstupných dát patriacich do tretej skupiny bola myšlienka vytvoriť vzorku, o ktorej by sme vedeli povedať, do koľkých zhlukov by mala byť zatriedená v optimálnom prípade. V experimente sme vytvorili všetky kombinácie nasledujúcich podmienok:

- počet parametrov pre vzorku v rozmedzí 1 až 3 reprezentuje, koľko rozmerný bude vstup
- počet dát pre každý parameter rovný 10, 100, 1000 až 10000 predstavuje veľkosť vzorky
- optimálny počet zhlukov po zatriedení 3 až 5 – počet zhlukov, do ktorých by mala byť táto vzorka zatriedená pri najlepšom rozdelení dát

Aby sme zaručili prehľadnosť vstupných vzoriek, pri vytváraní dát sme určili nasledujúce pravidlá:

- Pre každý parameter, ktorý mala vzorka spĺňať, sme určili interval, z ktorého sme generovali dáta. Tento interval sme vypočítali na základe stanoveného počtu parametrov a počtu dát vo vstupnom súbore. Definovali sme ho ako $\langle (p-1).d, pd \rangle, p = 1, \ldots, n$, kde n je počet parametrov a d je počet dát vo vstupnej vzorke.
- Takto získané intervaly sme rovnomerne rozdelili s ohľadom na stanovený počet zhlukov, ktoré by mal algoritmus vytvoriť pri optimálnom zatriedení.
- Tieto intervaly sme pre všetky zhluky zjemnili, aby sme zvýšili pravdepodobnosť, že generované dáta po zhlukovaní budú naozaj patriť do zhluku, pre ktorý boli vytvorené.
- Hodnoty *zjemnenia* sme nastavili na 1, 3 a 5 percent z intervalu určeného pre jeden parameter.

– Pre každý zhluk sme náhodne generovali dáta z intervalu určeného pre tento zhluk.

Príklad. Pre vstup s dvoma parametrami, počtom dát 1000 a troma zhlukmi sme uvažovali intervaly $\langle 0, 1000\rangle$ a $\langle 1000, 2000\rangle$. Interval určený pre každý parameter sme rozdelili na 3 časti, pričom sme uvažovali hodnoty zjemnenia 1%, 3% a 5%.

Poznámka: V experimente sme vytvorili nasledovné súbory s dátami:
Jednoduche – súbor obsahujúci vzorky s malým počtom dát.
Nerovnomerne – súbor s dátami s neznámou štruktúrou (nie je určené rozdelenie dát a ani počet zhlukov, do ktorých by mali byť dáta zadelené v najlepšom prípade) – na vyhodnocovanie zhlukovania získaného experimentom sme vzorky z tohto súboru nepoužili.
Rovnomerne x% – vytvorené pomocným programom – dáta v tomto súbore majú vopred známu štruktúru (počet dát, parametrov, zhlukov) – x predstavuje zjemnenie medzi zhlukmi (1%, 3% a 5%).

4.3 Výsledky porovnania algoritmov

Porovnaním všetkých výsledkov získaných v experimente sme dospeli k nasledujúcim záverom:

⋆ **HAC – Hierarchický aglomeratívny algoritmus**
  – Vo väčšine prípadov tento algoritmus zatriedil dáta zo vstupného súboru do správneho počtu zhlukov. U niektorých vstupoch bol počet očakávaných zhlukov nesprávny (Vzorka-P1-Z3-D100.data a Vzorka-P2-Z3-D100.data zo súboru Rovnomerne 3%) a tento algoritmus zatriedil takúto vzorku správne.
  – Algoritmus zatriedil všetky vstupné dáta do zhlukov bez vzniku nezatriedených prvkov, počas jednej iterácie a čas potrebný na zatriedenie bol nižší ako pri ostatných algoritmoch.

⋆ **KMEANS – K-means algoritmus**
  – Výsledky ovplyvňovala hodnota zjemnenia pre vstupné súbory. Pri menej jednoznačnej štruktúre zhlukov, kedy vo vstupnom súbore dáta neboli jednoznačne separované do zhlukov, tento algoritmus niekedy nezatriedil dáta do predpokladaného počtu zhlukov.
  – Tento algoritmus dosahoval najmenšiu časovú zložitosť spomedzi všetkých algoritmov, nevytváral žiadne "nezatriedené" dáta a správne zhluky vytvoril počas jednej iterácie.

⋆ **FCM – Fuzzy c-means algoritmus**
  – Vzhľadom na skutočnosť, že pri tomto zhlukovaní vstupné objekty mohli byť zaradené do viacerých zhlukov, čas potrebný na výpočet bol vyšší ako u predchádzajúcich algoritmov.
  – Niekedy bolo potrebných viacero iterácií na získanie správneho zatriedenia.
  – V porovnaní s ostatnými metódami až na výnimku správneho zatriedenia bol počet nezatriedených dát pri tomto algoritme vysoký.
  – So zvyšujúcim sa počtom parametrov, predpokladaných zhlukov a počtom dát vo vzorke rástla úmerne aj časová náročnosť tohto algoritmu a počet opakovaní. Pri vyššom počte dát a vyššom počte predpokladaných zhlukov vo vstupnej vzorke vzrástol počet opakovaní na viac ako 100, a preto niektoré výsledky boli označené ako neúspešné.

⋆ **UFCM – Upravený fuzzy c-means algoritmus**
  – Dosahuje najlepšie kvalitatívne výsledky, ale na úkor času potrebného na výpočet.
  – Podobne ako v predchádzajúcom prípade bol nutný vyšší počet opakovaní potrebných na získanie správneho zatriedenia a vznikol vyšší počet nezatriedených dát.
  – Najlepšia kvalita, ale aj najhoršie časové výsledky boli zapríčinené hodnotou fuzzifikácie, ktorá najviac spomedzi ostatných voliteľných parametrov ovplyvňovala kvalitu zhlukovania resp. výslednú hodnotu Xie - Beniho indexu.
  – Na základe pozorovaní parameter počet iterácií výrazne neovplyvňoval výsledok zhlukovania. Po stanovení dostačujúcej hodnoty nebolo nutné tento počet zvyšovať na získanie lepšieho zatriedenia. Naopak, keďže pri tomto algoritme sme zaznamenali vyšší počet nezatriedených dát, parameter *maximálny počet nezatriedených prvkov* s nižšou hodnotou mal za následok zvýšený počet celkových iterácií potrebných na získanie správneho zatriedenia.

Ukážka získaných výsledkov je uvedená v tabuľke 1.
Vysvetlenie k tabuľke 2:
Pri ohodnotení kvality zhlukovania pre každý vstupný súbor boli určené nasledujúce hodnoty: počet vytvorených zhlukov pre vstupnú vzorku bol vopred známy údaj o počte vytvorených zhlukov pri najlepšom výsledku zhlukovania.

Pre algoritmy zamerané na získanie najlepšieho zatriedenia vstupných dát sme preto porovnávali počet vytvorených zhlukov s predpokladaným počtom. Ak tieto hodnoty boli rovnaké, tak algoritmus sme označili ako OK, inak sme uviedli počet zhlukov vytvorených algoritmom. Pre algoritmy s možnosťou definovania počtu zhlukov sme túto hodnotu nastavovali od 2 do 7 zhlukov. Na základe porovnania prislúchajúcich hodnôt indexu kvality sme za výsledný počet zhlukov algoritmu považovali ten, ktorý dosahoval najlepšiu hodnotou indexu. Ak sa tento počet zhodoval s počtom predpokladaných zhlukov pre vstupnú vzorku, tak sme

Názov	VPZ	Komp.	Sep.	DB Index	CČZ[ms]
HACPP	2	197	2452	0.5118	3
HACPP	3	104	737	0.4402	4
HACPP	4	35	534	0.4095	4
HACPP	5	30	83	0.8037	6
HACPP	6	26	77	0.9077	4
HACPP	7	22	8	1.2200	7
HACNZ	4	35	534	0.4095	17
KMEANS	2	197	2452	0.5118	0
KMEANS	3	197	1493	0.4374	1
KMEANS	4	99	727	0.4512	0
KMEANS	5	35	356	0.3721	1
KMEANS	6	32	50	0.8716	1
KMEANS	7	32	0	0.7692	1
FCMPP	2	174	2614	0.0667	8
FCMPP	3	74	1199	0.0602	60
FCMPP	4	31	525	0.6200	15
FCMPP	5	25	61	0.6076	38
FCMPP	6	20	64	0.9306	54
FCMPP	7	18	0	0.9810	110
FCMNZ	4	32	526	0.0617	3924
UFCM	2	171	2675	0.0642	2
UFCM	3	73	1059	0.0694	107
UFCM	4	30	520	0.0590	3
UFCM	5	25	61	0.4589	7
UFCM	6	21	43	0.4987	15
UFCM	7	16	37	0.4517	1977

Table 1. Ukážka výsledkov pre vstupný súbor Vzorka_P2_Z4_D100.data. Predpokladaný počet zhlukov je 4, počet parametrov 2, VPZ – vytvorený počet zhlukov, CČZ - celková časová zložitosť.

tento algoritmus označili OK, inak sme uviedli počet zhlukov, pri ktorom algoritmus dosahoval najlepšiu hodnotu indexu kvality.

Označenie algoritmov v tabuľkách 1 a 2:
HAC PP - Hierarchický aglomeratívny zhlukovací algoritmus, určený počet zhlukov
HAC NZ – Hierarchický aglomeratívny zhlukovací algoritmus, najlepšie zatriedenie
KMEANS - H-means zhlukovací algoritmus
FCM PP - Fuzzy c-means zhlukovací algoritmus, určený počet zhlukov
FCM NZ – Fuzzy c-means zhlukovací algoritmus, najlepšie zatriedenie
UFCM - Upravený fuzzy c-means zhlukovací algoritmus.

5 Záver

Na základe získaných poznatkov bol navrhnutý upravený zhlukovací algoritmus, ktorý umožňuje bližšie špecifikovať niektoré parametre ovplyvňujúce výsledky samotného zhlukovania.

Názov vzorky	HAC PP	HAC NZ	KMEANS	FCM PP	FCM NZ	UFCM
P1_Z3_D100.dat	2	2	OK	2	2	2
P1_Z3_D1000.dat	OK	OK	OK	OK	OK	OK
P1_Z4_D100.dat	OK	OK	OK	OK	OK	OK
P1_Z4_D1000.dat	OK	OK	OK	OK	OK	OK
P1_Z5_D100.dat	OK	2	OK	OK	2	OK
P1_Z5_D1000.dat	OK	2	OK	OK	2	OK
P2_Z3_D100.dat	2	2	OK	OK	2	OK
P2_Z3_D1000.dat	OK	OK	OK	4	OK	OK
P2_Z4_D100.dat	OK	OK	5	OK	OK	OK
P2_Z4_D1000.dat	OK	OK	5	OK	OK	OK
P2_Z5_D100.dat	OK	OK	OK	OK	2	OK
P2_Z5_D1000.dat	OK	OK	OK	OK	OK	OK
P3_Z3_D100.dat	OK	4	OK	2	2	2
P3_Z3_D1000.dat	4	OK	2	OK	OK	OK
P3_Z4_D100.dat	OK	OK	OK	OK	OK	OK
P3_Z4_D1000.dat	OK	OK	5	OK	OK	OK
P3_Z5_D100.dat	OK	OK	OK	OK	OK	OK
P3_Z5_D1000.dat	OK	OK	OK	OK	OK	OK

Table 2. Ukážka výsledkov pre vstupné dáta so zjemnením s hodnotou 3%.

Uviedli sme kritéria kvality umožňujúce ohodnotiť výsledky zhlukovania získané pomocou skúmaných algoritmov. Poznatky sme využili v experimente, ktorý využíva spomínané zhlukovacie algoritmy na zatriedenie vstupných dát do zhlukov a umožňuje porovnať výsledky zhlukovania. Získané výsledky sú zhrnuté a porovnané vzhľadom k použitým zhlukovacím algoritmom a s ohľadom na kritériá kvality zhlukovania, vytvorený počet zhlukov a časovú zložitosť.

References

1. Y. Xia: *A global optimization method for semi-supervised clustering.* Data Minig Knowledge Disc **18**, 2009, 214–256.
2. R. Xu, D. Wunsch: *Survey of clustering algorithms.* IEEE Transaction on Neural Networks, 16, 3, 2005, 645–678.
3. P. Hansen, B. Jaumard: *Cluster analysis and mathematical programming.* Math. Programming, 79, 1997, 191–215.
4. M. Anderberg: *Cluster analysis for applications.* New York, Academic, 1973.
5. A. Jai, R. Dubes: *Algorithms for clustering data.* Englewood Cliffs, NJ: Prentice Hall, 1988.
6. A. Gordon: *Classification*, 2nd ed. London, U.K.: Chapman& Hall, 1999.
7. M. Halkidi, M. Bastistakis, M. Vazirgiannis: *Clustering validity checking methods: Part 2.* Athens University of Economics & Business, 2003.

Automatické přiřazování valenčních rámců a jejich slévání*

Eduard Bejček

Univerzita Karlova v Praze, Ústav formální a aplikované lingvistiky, Matematicko-fyzikální fakulta
Malostranské náměstí 25, 11800 Praha, Česká republika
bejcek@ufal.mff.cuni.cz

Abstrakt *Informace o valenci sloves je podstatná pro mnoho odvětví NLP. Existuje proto již několik valenčních slovníků. V tomto článku představíme dva z nich (VALLEX a PDT-VALLEX), které jsou k disposici v elektronické podobě a které mají společné východisko. Oba mají své přednosti a naším cílem je spojit je v jeden slovník.*
Máme k disposici data z korpusu PDT, kterým jsou ručně přiřazeny položky prvního ze slovníků. To nám pomůže provázat oba slovníky přes data využívajíce automatické identifikace (následované ruční prací s problémovými případy). Tímto poloautomatickým spojením dvou slovníků vznikne kvalitní lexikografický zdroj, který by jinak vyžadoval mnohem více lidské práce.
Průměrná úspěšnost namapování rámce z jednoho slovníku výběrem náhodného rámce z druhého je přibližně 60 %.
Článek se také zmiňuje o universálním formátu, ve kterém bude výhodné nová data ukládat odděleně od stávajících.

Úvod

Sloveso je v lingvistice tradičně chápáno jako centrum věty a jako takové tvoří kořen stromu syntaktické analýzy. Všechny ostatní části věty jsou pak vhodně zavěšeny na slovese, nebo na některém následníku slovesa ve stromové hierarchii. Dobrý valenční slovník tedy usnadňuje parsing věty, neboť nabízí možná slovesná doplnění – tedy sloty, které závisí na slovese a mohou být naplněny dalšími částmi věty. Další využití valenčního slovníku spočívá v jeho schopnosti rozlišení významu: rozpoznáním valenčních doplnění můžeme s pomocí valenčního slovníku určit, o který z významů slovesa se jedná, a použít tak například při překladu vhodný ekvivalent v cizím jazyce, či zpřesnit výsledky úlohy „information retrieval“ atp. Pokud se nám podaří využít stávajících valenčních slovníků a s vynaložením menšího úsilí je spojit v jeden, získáme lexikografický zdroj cenný pro mnohá odvětví NLP (Natural Language Processing).

Cílem tohoto článku je představit probíhající práce na automatickém slévání dvou valenčních slovníků. V první kapitole popíšeme oba slovníky a zmíníme jejich výhody. V druhé kapitole čtenáře seznámíme s korpusem PDT, který využijeme pro slévání, a zastavíme se u formátu, který navrhujeme pro uchování dat. Předposlední kapitola přináší stručný rozbor velikosti slovníků, které máme k disposici, a poslední představuje práci na slévání slovníků ve dvou fázích.

* Tato práce je podporována granty Grantové agentury Univerzity Karlovy č. 4200/2009 a Grantové agentury Akademie věd ČR č. 1ET100300517 a projektem MŠMT ČR LC536.

1 Valence sloves a valenční slovníky

Valencí slovesa[1] se označuje jeho schopnost vázat další konkrétní syntaktické prvky věty. Některé z nich jsou vyžadovány (*obligatorní doplnění*), jiné jen povoleny (*fakultativní doplnění*). Sloveso může mít více než jeden tzv. *valenční rámec*; různé rámce jednoho slovesa obvykle representují různé významy tohoto slovesa. Oba slovníky, kterými se zabýváme, vycházejí z teorie valence Funkčního generativního popisu (FGP) češtiny ([4,5,6]). Ten dále klasifikuje typy doplnění. Rozlišuje tak například posice *actor, paciens, adresát, efekt, překážka, směr „odkud“, čas „do kdy“* apod.

Příkladem může být třeba sloveso *čekat* a jeho tři valenční rámce:

1. Petr$_{\mathsf{ACT}}$ **čekal** od Martina$_{\mathsf{ORIG}}$ omluvu$_{\mathsf{PAT}}$.
2. Petr$_{\mathsf{ACT}}$ **čekal** na Martina$_{\mathsf{PAT}}$ s večeří$_{\mathsf{ACMP}}$.
3. Petr$_{\mathsf{ACT}}$ **čekal** Martinovi$_{\mathsf{BEN}}$ s dluhem$_{\mathsf{PAT}}$.

První má význam „očekávat“, druhý „odložit (nějakou činnost)“ a třetí „počkat (obvykle právě s dluhem)“ – zde tedy platí, že s různými valenčními rámci se liší i významy slovesa. U prvního jsou všechna tři doplnění obligatorní, u druhého nikoli (lze říci „Petr čekal na Martina.“, ale nikoli *„Petr čekal od Martina.“). Také předložky a pády jednotlivých doplnění jsou jasně určeny. (Tj. začíná-li věta „Petr čekal na...“ a následuje akusativ (tedy ne třeba „Petr čekal na nádraží“), je tím jednoznačně určen valenční rámec 2 a tedy použitý význam slovesa *čekat* (tj. „odložit“)).

Valenční vlastnosti se sloveso od slovesa liší a nelze je odvozovat obecnými pravidly; proto je nutné je popsat pro každé sloveso zvlášť. Cílem našeho projektu je nový valenční slovník, který bude obsahovat jednak rámce, které se vyskytovaly v obou slovnících, sloučené

[1] Valenci mohou mít také některá slovesná substantiva a adjektiva. Těmi se v tomto textu až na výjimky zabývat nebudeme.

do jednoho a také ostatní rámce upravené, aby odpovídaly požadavkům na rámce výsledného slovníku.

Nyní se seznámíme se těmito dvěma existujícími elektronickými valenčními slovníky. V mnohém se podobají, ale každý má oproti druhému i své přednosti.

1.1 VALLEX

Slovník VALLEX ([3]) se začal vytvářet v roce 2001. Stejně jako slovník PDT-VALLEX vychází z FGP (viz [7]). Popisuje valenční vlastnosti 2730 lexémů[2] (vidové dvojice jsou obvykle popisovány dohromady; počítaje je zvlášť, dostali bychom přibližně 4500 sloves).

Pro každý takový lexém jsou uvedeny všechny jeho známé valenční rámce. Každý rámec obsahuje obligatorní a fakultativní doplnění, způsob jejich povrchového vyjádření (předložka a pád, či infinitiv, nebo podřadná spojka pro celou vedlejší větu), glosu a příklad usnadňující pochopení významu či použití tohoto rámce a často další doplňující syntaktické informace (reflexivita, reciprocita, kontrola a syntaktickosémantická třída). V záhlaví lexému jsou vyjmenovány jeho vidové varianty (navíc u jednotlivých valenčních rámců mohou být potom některé z nich explicitně vyloučeny). Ukázka lexému *odpovídat* je na obrázku 1.

Mezi výhody VALLEXu patří:

Komplexnost pro sloveso. Slovesa, která jsou ve slovníku zahrnuta, jsou zpracována plně, neměl by chybět žádný rámec; zahrnuta byla i častá idiomatická použití.

Výběr sloves. Slovník pokrývá slovesná lemmata, která se vyskytují v českém textu nejčastěji.

Bohatost informací. Slovník poskytuje glosu, informaci o vidu, reflexivitě, atd.

Pečlivé ruční zpracování. Slovník vznikal dlouho, každý lexém byl zkontrolován více lidmi a porovnáván s jinými existujícími slovníky. Dá se teda rozhodně označit za spolehlivý zdroj.

1.2 PDT-VALLEX

PDT-VALLEX ([1]) začal vznikat zároveň s tektogramatickou rovinou PDT (viz níže) od roku 2000. Bylo potřeba mít valenční slovník, na nějž by mohly odkazovat všechna slovesa (a slovesná substantiva a adjektiva) z korpusu. Protože VALLEX ještě nebyl k dispozici, začaly se vyvíjet oba dva valenční slovníky paralelně.

Struktura (jak je vidět na obrázku 2 zobrazujícím opět sloveso *odpovídat*, tentokrát v PDT-VALLEXu) je podobná jako u VALLEXu. Opět je jedno lemma (tentorkát bez vidového protějšku) rozděleno na několik valenčních rámců. U každého jsou uvedeny charakteristiky jednotlivých doplnění (actor, paciens, atd. a pád, předložka, nebo spojka) a několik příkladů užití. Dále je uveden identifikátor rámce (pro provázání s PDT) a počet výskytů v PDT.

Výhody PDT-VALLEXu jsou zejména:

Anotace v datech. Velká deviza je skryta právě v jeho raison d'être: v jeho provázání s daty. Díky tomu jsou k disposici data pro kontrolu slovníku, pro trénování přiřazování rámců – a pro naši úlohu data, která napomohou provázání s jiným slovníkem.

Nejen slovesa. Slovník obsahuje kromě sloves (např. „Jan odpovídá Evě na dotaz“) také některá slovesná substantiva („odpovídání Jana Evě na dotaz“) a adjektiva („odpovídající na dotaz“).

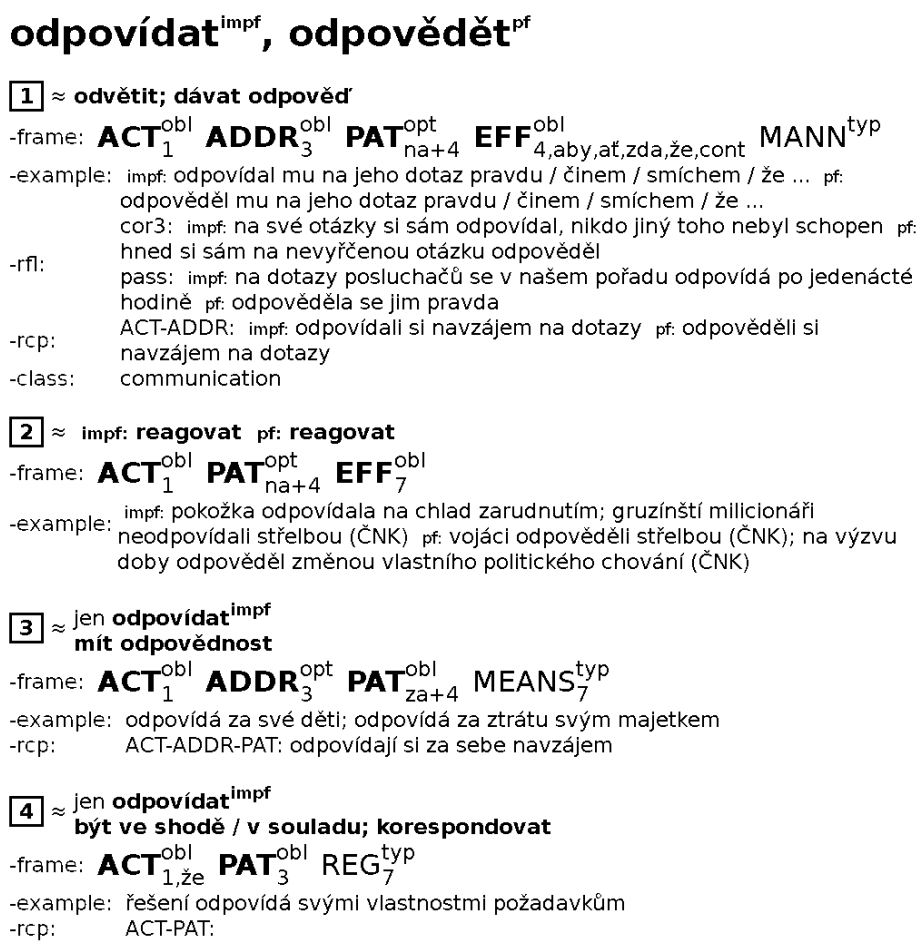
Frekvence. Ve slovníku je u každého valenčního rámce slovesa uvedena přibližná frekvence výskytu v textu. Ačkoli vzorek textu nebyl tak velký jako u VALLEXu, jde o čísla důležitá a unikátní, neboť z frekvence lexému se nedá usuzovat na frekvenci jednotlivých rámců a některý rámec nezařazený do VALLEXu může snadno být častější než jiný zařazený.

1.3 Odlišnosti

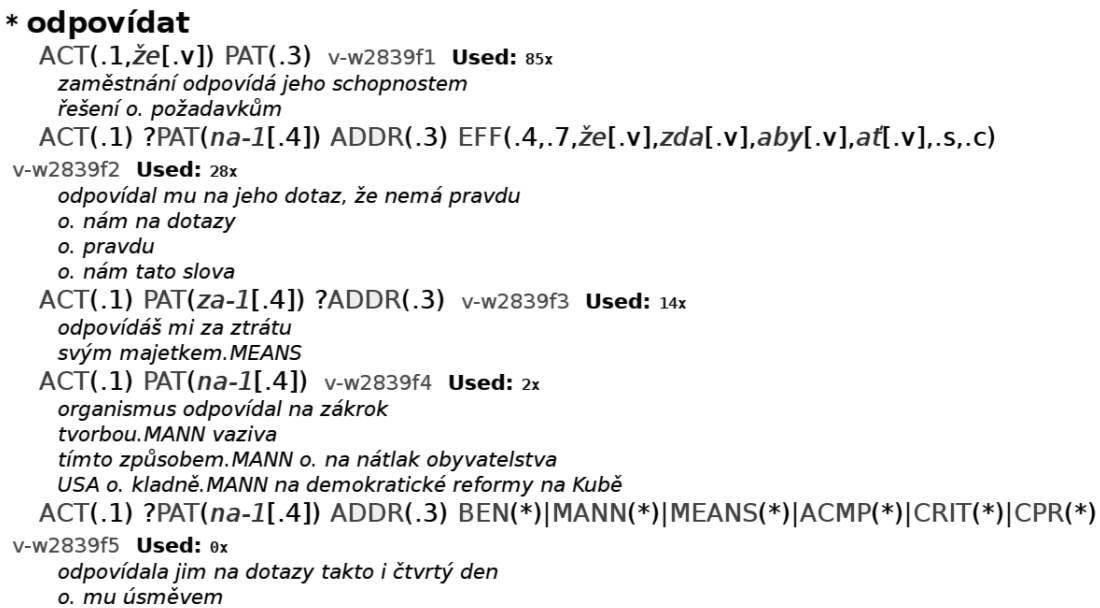
Na závěr kapitoly ještě projděme závažnější místa, v kterých se oba právě popsané slovníky liší.

1. Různé datové formáty. Oba slovníky jsou uloženy v XML, ale jejich struktura je zcela odlišná.
2. Jiný repertoár uvnitř rámce. Oba slovníky se trochu jinak vypořádávají se specifikací jednotlivých valenčních doplnění. Liší se mj. seznam spojek, které jsou použity k určení vedlejší věty ve valenci slovesa (PDT-VALLEX užívá navíc např. „jestli“), taktéž předložek (VALLEX užívá navíc např. „kolem“).
3. Různý přístup ke stejným rámcům. Slovníky vznikaly za jiných okolností a nemají tudíž sjednocenou metodologii. Tak se například stane, že věty „Dosáhl na něm slibu.“ a „Dosáhl svého.“ jsou v PDT-VALLEXu representovány dvěma rámci („dosáhnout na někom něčeho“ a frazém „dosáhnout svého“), kdežto ve VALLEXu jsou oba spojeny pod rámec jediný. Jiným příkladem je sloveso *napojit*, u něhož nastává homografie: sloveso nese jak význam „připojit, spojit“, tak také „dát napít“. V takovém případě je sloveso rozděleno na dva lexémy ve VALLEXu, ale ne v PDT-VALLEXu.

[2] Jsou to nejčastější česká slovesa vybraná podle jejich frekvence v Českém národním korupusu SYN2000: `http://ucnk.ff.cuni.cz`



Obr. 1. Sloveso *odpovídat* se čtyřmi valenčními rámci, jak je zachyceno ve slovníku VALLEX. (Obrázek pochází z webového rozhraní: `http://ufal.mff.cuni.cz/vallex/2.5/data/html/generated/alphabet/index.html`.)



Obr. 2. Sloveso *odpovídat* s pěti valenčními rámci, jak je zachyceno ve slovníku PDT-VALLEX. (Obrázek pochází z webového rozhraní ke slovníku: `http://ufal.mff.cuni.cz/pdt2.0/visual-data/pdt-vallex/0_vallex.html`.)

4. PDT-VALLEX zdaleka neobsahuje kompletní informaci pro jednotlivé lexémy. Ve slovníku jsou uvedeny jen ty rámce, které se vyskytly v datech, a pak některé další, které anotátoři chtěli uvést (ačkoli na ně v datech nenarazili). Nebylo cílem mít lexém zpracován v úplnosti. Navíc je sada informací u rámce o trochu chudší než u VALLEXu.
5. K VALLEXu neexistují žádná data, která by byla anotovaná odkazy na jeho rámce. Tato data musíme v našem projektu částečně vyrobit, abychom mohli přes data slévat některé odpovídající si rámce, nebo pak slévání evaluovat.

Body (1.) až (3.) pro nás jsou překážkou, kterou musíme při slévání překonat: (1.) je čistě technický problém, (2.) komplikuje automatické porovnání dvou shodných rámců (nebudou se jevit shodně, liší-li se metodologie jejich zachycení) a (3.) rovněž komplikuje automatickou proceduru a navíc vyžaduje lingvistické rozhodnutí, jak s takovými rámci zacházet při slévání. Naopak z bodů (4.) a (5.) plyne zisk po sloučení slovníků: výsledný slovník bude sjednocením obou současných a bude obsahovat odkazy do dat pro ty rámce z VALLEXu, pro které bude nalezen ekvivalent v PDT-VALLEXu.

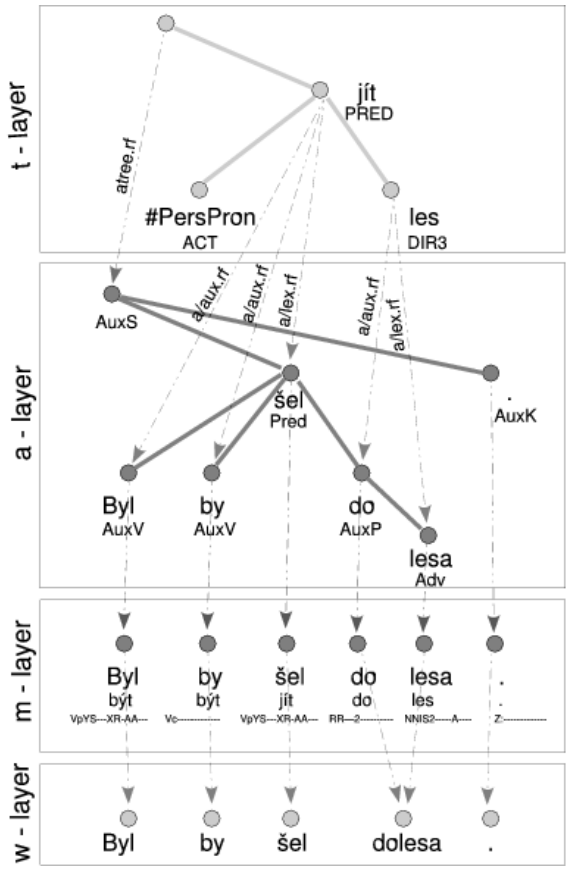
2 Pražský závislostní korpus

Třetím lingvistickým datovým zdrojem, který využijeme, je Pražský závislostní korpus (Prague Dependency Treebank, PDT, [2]). Tento korpus českých vět je v souladu s FGP členěn do tří rovin a každá obsahuje novou vrstvu ruční anotace. Nejníže je *m-rovina* s anotací morfologickou a s lemmatisací jednotlivých slov. Nad ní leží *a-rovina* se stromovou strukturou zachycující větnou syntaxi, větné členy ap. Nejvýše je *t-rovina* s hloubkovou syntaxí, která obsahuje už pouze významová slova a přiřazuje jim značky jako actor, paciens atd. Příklad na obrázku 3 to ukáže nejlépe.

PDT obsahuje 2 miliony slov, z toho více než 830 000 je obohaceno anotací na všech třech rovinách. Ze všech jevů, které PDT popisuje, je pro nás podstatná hlavně valence. Každé slovo v PDT, které má valenci (tedy všechna slovesa a některá substantiva, adjektiva, adverbia), obsahuje odkaz k příslušnému valenčnímu rámci v PDT-VALLEXu (pokud takový rámec v okamžiku anotace ještě neexistoval, byl vytvořen).

2.1 Prague Markup Language

Dovolíme si krátkou odbočku k formátu, ve kterém jsou data PDT uchovávána a zpracovávána. Nazývá



Obr. 3. Schematické znázornění všech tří rovin PDT a povrchové věty (*w-layer*) s překlepem: „Byl by šel dolesa.“ Uzly t-roviny nesou ještě mnoho další nezobrazené informace – kupříkladu uzel *jít* obsahuje odkaz `v-w1339f3` do PDT-VALLEXu.

se Prague Markup Language (PML, [8]) a je to aplikace XML. Uchovává anotaci zvlášť po jednotlivých rovinách, takže informace o jedné větě jsou vždy ve čtyřech souborech (viz rámečky obrázku 3) a ty jsou mezi sebou prolinkované.

Také my chceme uchovávat informaci o provázání slovníků s daty a mezi sebou jako *stand-off* anotaci.[3]Proto využijeme tzv. s-soubory (které jsou taktéž instancí PML), které budou obdobou nové roviny anotace. Jejich obecný formát nám umožní odkazovat do dat i do slovníků najednou.

3 Data

Pro náš projekt tedy využijeme dva valenční slovníky a korpus PDT, ve kterém je zanesena valenční informace z jednoho z nich.

[3] Stand-off anotace je koncept, podle něhož se dodatečná informace uchovává odděleně od původních dat. Ta tudíž nejsou nijak modifikována a lze s nimi volitelně pracovat buď spolu s anotací, nebo bez ní.

	lexémy	jedinečné	společné
VALLEX	4787	1618	3169
PDT-VALLEX	5510	2341	3169

Tab. 1. Počty lexémů v obou slovnících.

VALLEX obsahuje asi 4800 lexémů, PDT-VALLEX 5500[4] (viz tabulka 1). Průnik, tedy lexémy obsažené (v nějaké formě) v obou slovnících, tvoří více než polovinu každého slovníku.

Dále spočítáme přibližnou obtížnost úlohy. Baseline je obvykle úspěšnost nějaké triviální metody počítaná proti správným (*gold standard*) datům. K této úloze ale neexistují ručně anotovaná data pro VALLEX a tak nemůžeme stanovit chybovost nějaké triviální metody. Spočítáme tedy alespoň pravděpodobnost, že při náhodném mapování lexému PDT-VALLEXu na rámce VALLEXu vybereme správný. V průměru má lexém VALLEXu 2,5 rámce. Pravděpodobnost, že zvolím ten správný (průměrovaná přes celý VALLEX) vychází 0,6. To dává naději, že chytřejší postup může být úspěšný.

4 Postup slévání

Slévání probíhá ve dvou hlavních fázích. V první se některé valenční rámce podaří namapovat na odpovídající rámce druhého slovníku automaticky. Ve druhé části bude potřeba slít zbylé rámce ručně. Na následujících příkladech si ukážeme, kdy můžeme rámce sloučit automaticky.

Například sloveso *kazit* má jen jeden rámec v každém slovníku. Ačkoli tyto rámce nejsou úplně stejné (ve VALLEXu je bohatší), jsou kompatibilní (rámec PDT-VALLEXu je podmnožinou) a je možné je sloučit.

Sloveso *kandidovat* má sice rámce dva, ale v obou slovnících. Navíc jsou prakticky shodné, odhlédneme-li od odlišností slovníků samotných. Pokud je přepíšeme ve sjednocené notaci, vypadá první rámec („Petr kandiduje na poslance (do Parlamentu ČR).“) postupně v obou slovnících takto:

$\mathsf{ACT}^{\mathsf{obl}}_{\mathsf{nom}}\ \mathsf{PAT}^{\mathsf{opt}}_{\mathsf{na+acc}}$
$\mathsf{ACT}^{\mathsf{obl}}_{\mathsf{nom}}\ \mathsf{PAT}^{\mathsf{opt}}_{\mathsf{na+acc,\ za+acc}}\ \mathsf{DIR3}^{\mathsf{typ}}$

a druhý („Kandidovali Petra na europoslance (do Bruselu).“) takto:

$\mathsf{ACT}^{\mathsf{obl}}_{\mathsf{nom}}\ \mathsf{PAT}^{\mathsf{obl}}_{\mathsf{acc}}\ \mathsf{EFF}^{\mathsf{opt}}_{\mathsf{na+acc}}$
$\mathsf{ACT}^{\mathsf{obl}}_{\mathsf{nom}}\ \mathsf{PAT}^{\mathsf{obl}}_{\mathsf{acc}}\ \mathsf{EFF}^{\mathsf{opt}}_{\mathsf{na+acc,\ za+acc}}\ \mathsf{DIR3}^{\mathsf{typ}}$.

Rámce jsou si dostatečně podobné, abychom je mohli sloučit (zvláště když ve VALLEXu žádný další rámec není a nemůže dojít k záměně).

[4] To je počet pouze slovesných lexémů, dalších 4528 tvoří substantivní, adjektivní a adverbiální lexémy.

Ke komplikovanějšímu řešení nás nutí sloveso *rozvinout*, které má v PDT-VALLEXu tři zcela shodné rámce ($\mathsf{ACT}_{\mathsf{nom}}\ \mathsf{PAT}_{\mathsf{acc}}$) a je potřeba je namapovat na dva rámce VALLEXu. Tyto dva rámce se liší nepovinným $\mathsf{MEANS}_{\mathsf{inst}}$ – a toho zkusíme využít. Vyhledáme sloveso *rozvinout* v PDT, kde je u každého odkaz na jeden ze tří rámců PDT-VALLEXu. Pokud u některého z nich bude na slovese záviset také větný člen MEANS v instrumentálu, bude to náš kandidát.

Úspěšnost celého automatického postupu budeme testovat na ručně namapovaných rámcích a na datech, kterým vedle valenčních rámců PDT-VALLEXu přiřadíme ručně rámce z VALLEXu.

Výsledkem celého procesu pak například pro naše ukázkové heslo *odpovídat* budou odkazy

- od prvního rámce v PDT-VALLEXu (obr. 2) na 4 ve VALLEXu (obr. 1),
- od druhého rámce na 1,
- od třetího rámce na 3,
- od čtvrtého rámce na 2 a
- od pátého rámce opět na 1.

Celá procedura však má být použitelná i po vydání rozšířených versí slovníku. Proto musí umožňovat na závěr použít automatickou proceduru z první fáze a data z ruční druhé (ruční) fáze a provést celé slití slovníku automaticky.

Je tedy potřeba heuristika, která při novém slévání pozměněného rámce vezme v úvahu minulý výsledek a rozhodne, zda je změna dostatečně malá, aby mohlo sloučení proběhnout totožně. Pro nezměněné rámce bude výsledek stejný. Na ruční práci tedy zbydou pouze nové a zároveň problémové rámce a rámce příliš odlišné od předchozí verse slovníku. Pokud tento postup použijeme na původní nezměněné slovníky, musí být výsledný slovník totožný s ručně upraveným; to poslouží jako kontrola.

5 Závěr

Představili jsme projekt, který má propojením dvou valenčních slovníků získat nový kvalitní lexikografický zdroj. Z odhadované pravděpodobnost náhodného mapování lze usuzovat na použitelnou úspěšnost automatické procedury.

Reference

1. J. Hajič, J. Panevová, Z. Urešová, A. Bémová, V. Kolářová-Řezníčková, P. Pajas: *PDT-VALLEX: Creating a large-coverage valency lexicon for treebank annotation.* Proceedings of The Second Workshop on Treebanks and Linguistic Theories, 2003, 57–68.

2. J. Hajič, et al.: Prague Dependency Treebank 2.0. Linguistic Data Consortium, Philadelphia, PA, USA, 2006, URL: `http://ufal.mff.cuni.cz/pdt2.0`.
3. M. Lopatková, Z. Žabokrtský, V. Kettnerová: *Valenční slovník českých sloves*. Karolinum, Praha, 2008, 382 str. URL: `http://ufal.mff.cuni.cz/vallex/2.5`.
4. J. Panevová: *On verbal frames in functional generative description, Part I.* The Prague Bulletin of Mathematical Linguistics, **22**, 1974, 3–40.
5. J. Panevová: *On verbal frames in functional generative description, Part II.* The Prague Bulletin of Mathematical Linguistics, **23**, 1975, 17–52.
6. J. Panevová: *Valency frames and the meaning of the sentence.* The Prague School of Structural and Functional Linguistics, Amsterdam, Philadelphia, 1994, 223—243.
7. P. Sgall: *Generativní popis jazyka a česká deklinace.* Praha, Academia, 1967.
8. P. Pajas, J. Štěpánek: *A generic XML-based format for structured linguistic annotation and its application to Prague dependency treebank 2.0.* ÚFAL Technical Report **29**, Praha: MFF UK, 2005.

The Bobox project – a parallel native repository for semi-structured data and the semantic web*

David Bednárek, Jiří Dokulil, Jakub Yaghob, and Filip Zavoral

Faculty of Mathematics and Physics, Charles University in Prague
{bednarek,dokulil,yaghob,zavoral}@ksi.mff.cuni.cz

Abstract. *The state of the art in semi-structured data processing (and XML in particular) and Semantic web repositories corresponds each other: the non-scalability of pilot implementations, the inability of optimizations, and the cost of the fully native implementation. Although there are successful implementations in each of the approaches, none of the methods may be considered universal. The goal of the Bobox project is an implementation of a relational-like storage engine applicable both as a native XML database and as a semantic web repository. The main purpose of the engine is in experiments in both areas. The main stress is put to the performance of complex queries and transformations, and to the ability of parallel evaluation in particular.*

1 Introduction and motivation

Semi-structured data, and XML in particular, formed one of the hot topics in both academic and industrial research. In the area of XML data bases, indexing, and querying, the research resulted in a number of experimental as well as commercial implementations. Majority of the designs share the following principles:

- The physical representation of a XML collection is based either on binary XML [17] or on shredding XML data into relational form [14].
- In both cases, XML-specific indexing techniques are required – various forms of path or path/value indexes [7], interval encoding [13], or Dewey numbering [20] are used.
- Queries are translated to a form of algebra, developed from the relational algebra [10, 16, 19] by the addition of XML-specific operators, including structural, path [8], or twig joins [11].

Many implementations, in particular the commercial ones, were created as an extension to an existing relational DBMS [18, 14], others are built above an existing relational core [4, 9]. This approach allows reusing the solution of a number of collateral problems, including transaction handling and concurrency.

Despite the general success of the above-mentioned methods, there are still many areas open to new designs. In this project, we want to explore two problems:

- The application of the relational-like techniques of XQuery evaluation in a parallel environment.
- Interfacing between a XML stream [12] and a XML data base.

The two problems are related by the fact that a stream (pipe) is a natural way of interconnection in parallel environment when shared memory is not available. Furthermore, in many environments including the exploration of the web, a XML database is continuously fed by a XML stream instead of individual updates.

1.1 Applicability in the semantic web research

The semantic web initiative created a set of new applications of databases (in the broadest sense of this word). The new areas include the storage of the information gathered from the web by crawlers as well as the representation of the knowledge prepared with human assistance, including linguistic and ontological data. Last but not least, scientific research in the semantic web, social networks, and other areas that study the data gathered from the world-wide web, often produces huge collections of data that require appropriate storage technologies. The largest semantic collections of these days contain about 10^9 entries (facts, RDF triplets, etc.); however, significant growth is expected in the future.

There is already a number of implementations in the area of RDF repositories. The most common approaches are:

- A dedicated main-memory engine (e.g. [5]). This approach allows easy implementation and high performance; however, it is limited by the main memory available to a single process. Nowadays, high-end server hardware is required to process current data collections. Although the Moore's law offers the chance of multiplying the size twice in 18 months, this rate is probably too slow to accommodate the expansion of the Semantic web as expected by its proponents. This setting usually may be enhanced with the ability to retrieve

* This work was supported by the Grant Agency of the Czech Republic, grant number 201/09/0990 - XML Data Processing.

data from a persistent storage on demand, reducing the main memory requirements. However, such an approach is rarely efficient because of the absence of successful indexing strategy.
- Using an relational or object-relational DBMS [15]. The use of a RDBMS offers the experience gained in more than three decades of the relational DBMS development, in particular, their reliability, scalability, and cost-based optimization techniques. On the other hand, relational systems were designed with different applications in mind; consequently, some features like full transaction isolation or rollback ability are almost useless in a typical semantic web application. This unnecessary ballast then slows down insertion and other update operations. On the other hand, a typical semantic query, after translation to SQL, is a tough job for the relational query optimizer – the graph-like nature of semantic data often induces a long sequence of joins or even a transitive closure operation. Such queries were rare in traditional relational settings – although major vendors already noticed the changing structure of queries and improved their optimizers, some queries still run unacceptably slowly.
- A specialized native engine, like [6]. Although still based on relational operations, such an engine offers better query performance because its query optimizer is specialized to the particular semantic query language (e.g., SPARQL) and the compilation to SQL is bypassed. Moreover, specialized algebra operators may be added to support specific tasks. The main disadvantage of this approach is the costly implementation, although a number of components may be reused from relational systems.

The state of the art in Semantic web repositories corresponds to the situation in the area of XML processing, with a few more years of experience at the side of XML. In the XML world, the above-mentioned approaches may be identified too: There are in-memory implementations of XSLT and XQuery, implementations based on shredding XML into a relational DBMS, and native XML databases. The problems are also similar: The non-scalability of an in-memory engine, the inability of a RDBMS to optimize a query translated from a different language, and the cost of the fully native implementation. Nevertheless, there are successful XML implementations in all the three approaches. None of the methods may be considered universal – they perform differently in different areas of application.

1.2 Application area

This project is targeted at the application areas where the following criteria apply:

- The structure of the data cannot be matched to the models (like the E-R model) used in the area of relational databases.
- The schema of the data is variable over different sources, evolving in time, or unknown at all.
- The data often include recursive structures (XML documents, linguistic tree-banks) or graph-like connections (RDF).
- The repository is filled either by a continuous flow of data (e.g. by crawlers) or in large batches (e.g. RDF collections, experimental data); the incoming data replace existing data in a versioning manner. Individual record update is rare.
- Transaction control is reduced to batch-update / snapshot-read behavior; a delay in the propagation of updates is considered tolerable.
- XML-based formats are used in the exchange of data at both the batch-input and the query-output side.
- Incoming data may require transformation to fit to the required repository format.
- Sophisticated queries are placed that process or transform a large subset of the repository data. The performance of such queries may benefit from parallel evaluation.
- Queries are formulated in a domain-specific query language (XQuery, Sparql); the language is at least partially translatable to the relational algebra.

These criteria are met in many applications of the semantic web like RDF repositories; furthermore, there are also areas of XML where the reduced update / transaction capabilities may be tolerated.

1.3 The goal of the project

The goal of the Bobox project is an experimental implementation of a relational-like storage engine applicable both as a native XML database and as a semantic web repository. The main purpose of the engine is in experiments, in both the semantic web and XML areas. The main stress is put to the performance of complex queries and transformations, in particular, to the ability of parallel evaluation.

The implementation is intended to become the base for a number of experiments, including the evaluation of the storage engine on its own, the comparison of different structural join algorithms or shredding strategies etc. Besides these XML-specific goals, the application as an RDF repository will also be examined. Finally, the performance gains achieved by parallelization will be measured.

The experimental character allows to remove or weaken some requirements that would be otherwise considered essential, namely write-concurrency control, fault-tolerance, and maintenance support. On the other hand, extensibility, replaceability of all components, and decentralized design are required for future development.

The engine is based on the following layers:

- A query-language front-end compiles a domain-specific query language (XQuery, SPARQL) into a domain-independent relational-like intermediate language.
- The intermediate representation is statically analyzed and optimized.
- The optimized query is executed using a parallel engine in a multiprocessor and/or cluster arrangement.

The input to the engine is fed either from an input pipeline or from a partitioned persistent storage. The output is pushed to an output pipeline or stored back into the persistent storage. The input/output pipelines may consume/produce XML documents, including RDF, or other semantic web formats, including a feed from a web crawler. The persistent storage as well as the intermediate language is based on the relational paradigm and extended towards operations required in the XML/Web domains.

2 Evaluation engine

2.1 The architecture

The evaluation engine (Fig. 1) is based on a dynamically configured set of computing components, connected together by pipes. Each pipe carries a stream of relationally encoded data; each computing component performs a relational operation, structural join, or a combination of these. Conversely, some multiple-input structural join algorithms may be distributed over a set of components. The system is loosely synchronized, allowing out-of-order execution with respect to the canonical application language semantics.

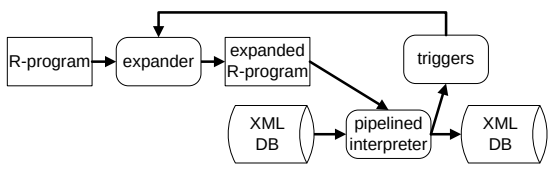


Fig. 1. Basic Bobox architecture.

The front-end generates a query evaluation plan and passes it to the back-end. The plan contains the desired structure of components and pipes (Fig. 3) together with information about structure of the records at each pipe, required buffer sizes, and parallelization plans. The computing components are then instantiated by the expander on the nodes where they should run and then connected with pipes. Depending on the actual locations of the nodes, the pipes can either be local or connect different nodes over the network.

When the instantiation of the plan is complete, the components are signaled to start the computation. The control is then handed to thread managers of the nodes and the computing components and pipes start to cooperatively evaluate the program in a multi-threaded distributed environment. Synchronization of the system is done only by the producer-consumer relationship between components connected by a pipe.

2.2 R-programs

One of the significant front-ends is the XQuery compiler. It works by translating a query to an *R-program* by a reversed evaluation approach [1]. Furthermore, static analysis methods [2] are implemented in the XQuery language front-end. The produced R-program is then easily translated into a Bobox query evaluation plan.

An *R-program* consists of a set of *R-functions*. The interior of each R-function is described by a directed graph of (extended) relational algebra operators and R-function calls. Each R-function receives one or more relations as its input arguments and produces one or more relations at its output.

Since the language of R-programs does not offer any programmatic structures like conditions or loops,

```
function main (m : (a : D, b : D))
  return (v : (a : D, b : D))
begin
  v := call[f](m, m, m); r := (m ∪ v);
end;

function f(z : (a : D, b : D),
  m : (a : D, b : D), x : (a : D, b : D))
  return (w : (a : D, b : D))
begin
  p := π[c/b](z); q := π[c/a](m);
  r := (p ⋈ q); s := δπ[a, b](r);
  t := (s \ x); u := (s ∪ x);
  v := call[f](t, m, u); w := (t ∪ v);
end;
```

Fig. 2. Example: An R-program to compute transitive closure.

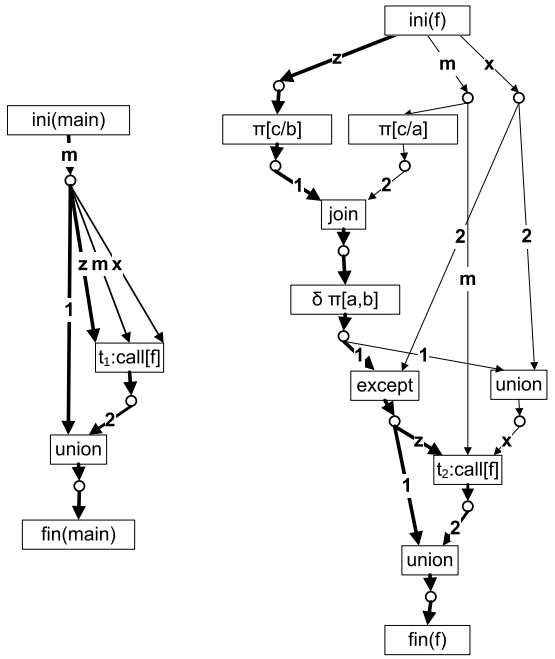


Fig. 3. Example: An R-program to compute transitive closure.

any recursive R-program would immediately fall into endless recursion. To give recursive R-programs their semantics, the notion of controlling argument is defined that allows to predict the output and to stop the recursion when the controlling arguments are empty. This mechanism corresponds to the case when recursion in an XQuery program is stopped by iterating over an empty set. Of course, termination of R-programs is not generally guaranteed just as the termination of XQuery programs is not.

The Fig. 2 shows an example of R-program in a textual form; the same R-program in a graphical form of a query plan is shown in the Fig. 3. This R-program computes the transitive closure m^+ of its argument m.

First, only the plan for the main function (left graph in the Fig. 2) is instantiated and the pipelined interpreter is started. While the evaluation is in progress, the t_1 component can receive data through one of the pipes. When this happens for the first time, the t_1 is replaced with instance of the plan for the body of the function (right graph in the Fig. 2). This new part contains t_2 which is replaced by the appropriate plan instance under the same circumstances.

2.3 RDF data

Most contemporary RDF query languages are table oriented. They can be implemented by the component/pipe system in a straightforward way. The expander used by R-programs is not used, but great number of computation components can be shared between the different language implementations.

Naturally, the front-end has to be reimplemented but it can use the same structure of query plans, which forms a well defined interface between the front- and back-ends. The number of back-end components used by more than one front-end is quite high and the managing code (expander, thread manager, etc.) is the same in both cases.

3 R-program optimization

In this section, we present the proposed methods and principles. More detailed description of R-Program optimizations is contained in [3].

Consider the sub-query *"return all persons which were employed on a given date"*.

```
declare function local:employed($P as xs:date)
{ fn:doc("company")//employee
    [ @hired lt $P and @fired gt $P]
};
<report>{
  for $D in fn:doc("history")//@date return
    <point date="{$D}"
      number="{fn:count(local:employed($D))}"/>
}</report>
```

Fig. 4. An XQuery function returning a sequence of nodes.

In XQuery, such a sub-query may be represented by a function. The function `employed` in the Fig. 4 is parametrized by the date `$P` and return a sequence of matching employees.

3.1 Naive evaluation

A naïve implementation calls the function `employed` for each date in the given history. Due to the nature of the condition placed on the `employee` nodes, value indexes on `@hired` and `@fired` can not reduce the number of scanned nodes significantly.

3.2 Optimized evaluation

To enlarge the opportunity to optimize, we suggest the following arrangement shown in the Fig. 5: the function `employed` is statically transformed so that it consumes all the values of the parameter `$P` at once. Consequently, the transformed function returns all the original return values in a single batch. In other words, the transformed function is called only once, instead of repeated calling in the naïve approach.

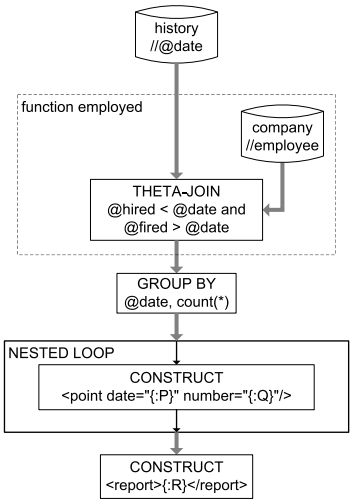


Fig. 5. Optimized Query Evaluation.

Bulk evaluation offers the ability to use more effective join techniques than nested-loop evaluation. In our example, a kind of *theta-join* is used to combine the set of parameter values with the set of employee nodes retrieved from a storage. To reduce cost, this theta-join may be implemented using repeated range scans on a sorted materialization of the left operand – this arrangement would never be possible in the naïve implementation. Figure 6 shows the corresponding query plan.

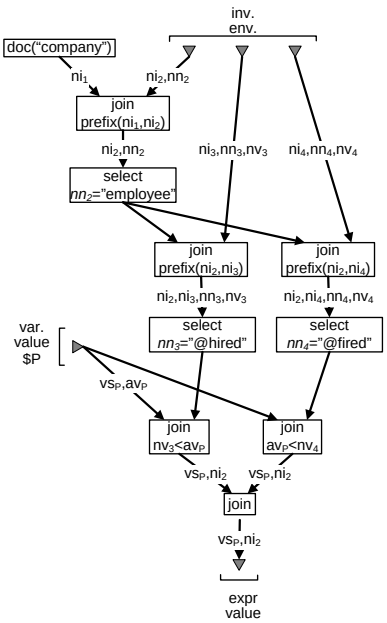


Fig. 6. Optimized Query Plan.

The concept of *view merging* is a form of procedure integration known from compiler construction. In the merged query, subsequent transformations are unaware of the original boundary of the view, allowing aggressive optimization (like join reordering) across the hidden boundary. Of course, the preservation of function boundaries reduces the maneuvering space of subsequent optimization.

3.3 Reverse optimized evaluation

The Fig. 7 shows an improved version of the execution schema. Assuming that the attribute `@date` has an ordered index, the theta-join was implemented by a repeated range scan over the index. The Fig. 7 depicts its query plan.

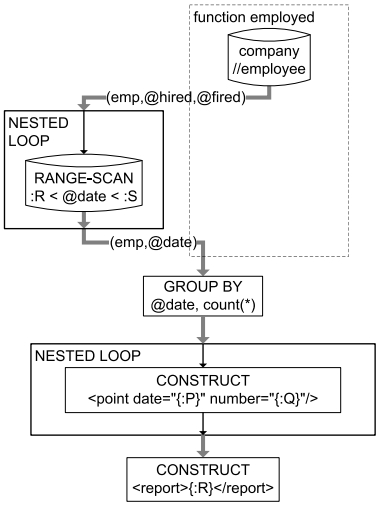


Fig. 7. Reverse Query Evaluation.

If the function `employed` were integrated into the surrounding query, the shift from the Fig. 5 to the Fig. 7 would be a relatively simple algebraic transformation. However, the bulk evaluation approach requires that the boundary of the function be still present. Therefore, such a transformation must be formalized as a transformation of the function interface.

The caller of the function is expected to perform a range-based theta-join of the original sequence of dates with the intervals generated by the function through the output parameter. The attributes not involved in the join (marked here as `emp`) are just passed around.

Although it may seem that we are pulling out all joins from the function, it is not true – only those join conditions that may be implemented with a particular physical access method are worthy of extraction.

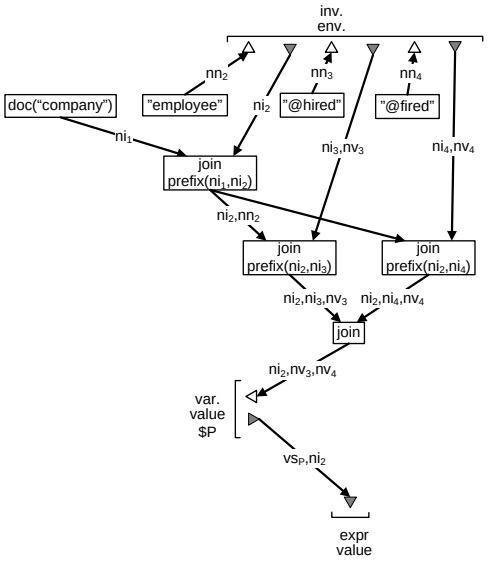


Fig. 8. Reverse Optimized Query Plan

Therefore, there is only a small number of transformations that may be applied to a function parameter.

The transformed function can no longer be evaluated in a simple call-return manner. Instead, the ability to pass the control to and from the function more than once is required. Moreover, the function must be able to retain their private data during the time the control is temporarily returned to its caller.

Pipelined execution is required to avoid unnecessary materialization of intermediate results. Therefore, a function may run effectively in parallel with its caller, returning the first data for output parameters sooner than the last data for input parameters arrive.

As illustrated in the example, the reversed data flow allows query reformulation in a way that is essentially equivalent to join reordering and similar techniques known from traditional RDBMS architectures. Such a replacement is necessary because traditional query rewrite methods are not directly applicable to R-programs due to the presence of functions and expression reuse.

3.4 Optimization architectures

The optimizations can be performed statically during a compile time or dynamically during the runtime. Intra-procedural optimization (denoted as *static rewriting*) and *local* plan selection may be applied as shown in the Fig. 9. The resulting physical plan is again in the form of a dag of algebra operators and function calls; thus it is again an R-program, albeit using a different set of operators.

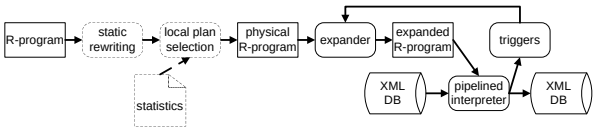


Fig. 9. Architecture with Static Optimization.

The effectiveness of intra-procedural cost-based optimization is limited because the cost of function calls and cardinality of their outputs is not known. This weakness may be addressed with the architecture depicted in the Fig. 10.

In each cycle, the expanded R-program may be optimized by rewriting and transformed using cost-based plan selection to a physical R-program. Since the original R-functions were integrated into a single function, the optimization is in fact inter-procedural. Of course, this phase may alter only the newly appended R-function body because the previously integrated code is already being executed. On the other hand, the plan selection may make use of cost and cardinality estimation computed throughout the whole integrated program. Therefore, it may produce better plans than in the case of local plan selection.

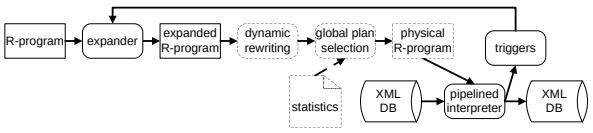


Fig. 10. Architecture with Dynamic Optimization.

The R-program can be considered to be a framework; during a query compilation it contain logical operators, during the transformation these logical operators are substituted by physical operators, the mapping is not necessarilly 1:1. Therefore the R-programs play the role of both logical and physical plan (in a form of a tree) in traditional query evaluation.

4 Conclusion and future work

In this paper we propose the architecture of the Bobox native repository useful for semi-structured data and the semantic web. The whole system is designed to run in a parallel and distributed environment. It is currently the only system capable of providing the specific features required to evaluate R-programs, most notably the run-time recursive plan expansion.

At the moment, we have an experimental implementation of key parts of the system. Their main pur-

pose is to test basic principles and to help us refine the interfaces between various components of the system. The consecutive experiments will be focused on the evaluation of the system and the optimization of parallel and distributed access and query evaluation.

References

1. D. Bednárek: *Output-driven XQuery evaluation.* In C. B. et al., editor, Intelligent Distributed Computing, Systems and Applications, Studies in Computational Intelligence, Springer-Verlag, 162, 2008, 55–64.
2. D. Bednárek: *Reducing temporary trees in XQuery.* In P. Atzeni, A. Caplinskas, and H. Jaakkola, editors, Advances in Databases and Information Systems, 12th East European Conference, ADBIS 2008, Pori, Finland, September 5-9, 2008. Proceedings, Lecture Notes in Computer Science, Springer, 5207, 2008, 30–45.
3. D. Bednárek: *Using r-programs in xquery evaluation.* In Technical Report. Department of Software Engineering, Charles University in Prague, 2009.
4. P.A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner: *MonetDB/XQuery: A fast XQuery processor powered by a relational engine.* In Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, IL, USA, June 2006.
5. J. Broekstra, A. Kampman, and F. v. Harmelen: *Sesame: A generic architecture for storing and querying rdf and rdf schema.* In ISWC'02: Proceedings of the First International Semantic Web Conference on The Semantic Web, London, UK, Springer-Verlag, 2002, 54–68.
6. M. Cai and M. Frank: *Rdfpeers: a scalable distributed rdf repository based on a structured peer-to-peer network.* In WWW '04: Proceedings of the 13th international conference on World Wide Web, New York, NY, USA, ACM, 2004, 650–657.
7. R. Goldman and J. Widom: *Dataguides: Enabling query formulation and optimization in semistructured databases.* In VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 1997, 436–445.
8. T. Grust, M. Keulen, and J. Teubner: *Staircase join: Teach a relational DBMS to watch its (axis) steps.* In In Proc. of the 29th Intl Conference on Very Large Databases (VLDB, 2003, 524–535.
9. T. Grust and J. Rittinger: *Jump through hoops to grok the loops Pathfinder's purely relational account of XQuery-style iteration semantics.* In Proceedings of the ACM SIGMOD/PODS 5th International Workshop on XQuery Implementation, Experience and Perspectives (XIME-P 2008), June 2008.
10. T. Grust and J. Teubner: *Relational Algebra: mother Tongue - XQuery: Fluent.* In V. Mihajlovic and D. Hiemstra, editors, First Twente Data Management Workshop (TDM 2004) on XML Databases and Information Retrieval, Enschede, The Netherlands, June 21, 2004, CTIT Workshop Proceedings Series, Centre for Telematics and Information Technology (CTIT), University of Twente, 2004, 9–16.
11. H. Jiang, H. Lu, and W. Wang: *Efficient processing of XML twig queries with OR-predicates.* In SIGMOD '04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, ACM, 2004, 59–70.
12. M. Li, M. Mani, and E.A. Rundensteiner: *Efficiently loading and processing XML streams.* In IDEAS '08: Proceedings of the 2008 international symposium on Database engineering and applications, New York, NY, USA, ACM, 2008, 59–67.
13. Q. Li and B. Moon: *Indexing and querying XML data for regular path expressions.* In VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 2001, 361–370.
14. Z.H. Liu, M. Krishnaprasad, and V. Arora: *Native XQuery processing in Oracle XMLDB.* In SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, ACM, 2005, 828–833.
15. A. Matono, T. Amagasa, M. Yoshikawa, and S. Uemura: *A path-based relational rdf database.* In ADC '05: Proceedings of the 16th Australasian Database Conference, Darlinghurst, Australia, Australia, Australian Computer Society, Inc., 2005, 95–103.
16. N. May, S. Helmer, and G. Moerkotte: *Strategies for query unnesting in XML databases.* ACM Trans. Database Syst., 31, 3, 2006, 968–1013.
17. M. Nicola and B. van der Linden: *Native XML support in DB2 universal database.* In VLDB '05: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB Endowment, 2005, 1164–1174.
18. S. Pal, I. Cseri, O. Seeliger, M. Rys, G. Schaller, W. Yu, D. Tomic, A. Baras, B. Berg, D. Churin, and E. Kogan: *XQuery implementation in a relational database system.* In VLDB '05: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB Endowment, 2005, 1175–1186.
19. C. Re, J. Simeon, and M. Fernandez: *A complete and efficient algebraic compiler for xquery.* Data Engineering, International Conference on, 0:14, 2006.
20. I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang: *Storing and querying ordered XML using a relational database system.* In SIGMOD '02: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, ACM, 2002, 204–215.

Využití techniky náhodného indexování v oblasti detekce plagiátů

Zdeněk Češka

Katedra informatiky a výpočetní techniky, Fakulta aplikovaných věd, Západočeská univerzita v Plzni
Univerzitní 22, 306 14 Plzeň, Česká republika
zceska@kiv.zcu.cz

Abstrakt. *Rostoucí snaha plagiovat cizí práce, především v oblasti školství, zapříčinila vývoj nových a lepších metod, které by těmto intrikám čelily. Tento článek rozvíjí myšlenku aplikace Latentní sémantické analýzy (LSA) v oblasti detekce plagiátů a navrhuje nová vylepšení. Hlavním diskutovaným předmětem je aplikace kompresní techniky tzv. náhodného indexování, která transformuje data do alternativního zmenšeného prostoru. Kromě toho se článek zabývá normalizací podobností mezi dokumenty a přináší novou asymetrickou normalizační formuli. Experimenty byly provedeny na manuálně vytvořeném korpusu českých plagiátů, který obsahuje 1500 dokumentů o politice. Dosažené výsledky indikují, že kompresní technika dokáže významně snížit časové požadavky pro LSA. Aplikací nové normalizační formule lze navíc dosáhnout i vyšší přesnosti detekce plagiátů při současně nižších časových požadavcích.*

1 Úvod

Zvyšující se zájem o určování autorství psaných dokumentů vede k vývoji nových pokročilých metod, které jsou schopny automaticky detekovat případy plagiátorství. Tento problém je navíc umocněn množstvím volně dostupných dokumentů na Internetu, pojednávajících o různorodých tématech. Cílem metod pro detekci plagiátů je objektivně posoudit rozličné zdroje a nalézt ty, které byly nějakým způsobem plagiovány. Ačkoli současné moderní metody dávají dobré výsledky, stále je vyžadováno konečné lidské rozhodnutí o tom, co lze považovat za plagiát. Současné metody slouží především jako vodítko a významným způsobem šetří lidský čas.

Tento článek je zaměřen na zlepšení výsledků metody SVDPLAG 2, která je založena na Latentní sémantické analýze (LSA), viz 8. Pro extrakci latentní sémantiky z textu se využívá matematická metoda Singulární hodnotové dekompozice (SVD - 1). Klíčové příznaky, které jsou touto metodou zkoumány, představují fráze obsažené v textových dokumentech. Jak popisuje článek 2, fráze jsou reprezentovány slovními N-gramy, které se postupně analyzují a extrahují z předzpracovaného textu.

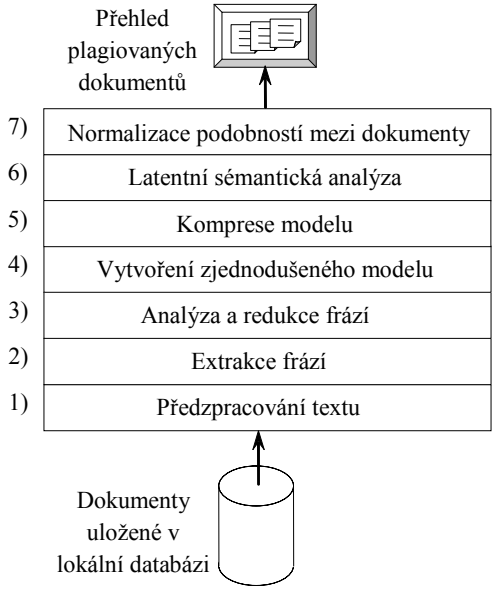
SVDPLAG má odlišný přístup k analýze textu oproti jiným metodám, pracujícím pouze s kosinovou mírou vektoru, která obsahuje počty výskytů jednotlivých slov, viz 12, podobně systém SCAM 13. Rovněž metody založené na prostém průniku společných slovních N-gramů, jako je systém FERRET 9, nedosahují vyžadovaných výsledků. Bližší popis různých přístupů a jejich souhrn lze nalézt např. v článcích 3 a 10. SVDPLAG jde v tomto ohledu cestou rozsáhlých statistických výpočtů v rámci LSA, jež provádí analýzu všech dokumentů současně. Podávané výsledky jsou proto podstatně vyšší než u ostatních metod. Nevýhodou jsou vyšší časové požadavky, které jsou hlavním předmětem řešení tohoto článku.

Další text v tomto článku je organizován následovně. Sekce 2 navrhuje užití kompresní techniky založené na náhodném indexování, společně s novou optimalizovanou normalizační formulí pro výpočet podobností mezi dokumenty. Sekce 3 prezentuje výsledky navržených modifikací na metodě SVDPLAG. Závěrečná diskuse dosažených výsledků je podána v sekci 4.

2 Navrhovaná vylepšení SVDPLAG metody

Princip této metody 2 je založen na LSA, kde se jako jádro pro extrakci sémantických vztahů využívá matematická metoda SVD. Na základě toho je též odvozen název metody pro detekci plagiátů SVDPLAG.

Obr. 1 prezentuje jednotlivé vrstvy zpracování, které byly detailně popsány v článku 2. Jediná významná modifikace spočívá v přidání vrstvy pro kompresi modelu, podstatně urychlující výpočet následujícího LSA. Rovněž byla provedena drobná úprava v sedmé vrstvě, zahrnující novou asymetrickou normalizační formuli. Provedené modifikace jsou popsány v následujícím textu.



Obr. 1. Vrstvy zpracování SVDPLAG metody.

2.1 Komprese modelu metodou náhodného indexování

Navržený model fráze x dokument v článku 2 se při zpracování rozsáhlých kolekcí potýká s velkými rozměry matice A. Nechť A je n x m obdélníková matice složená

z n vektorů $[A_1, A_2, \ldots, A_n]$, kde vektor A_i představuje fráze obsažené v dokumentu i. Vektor A_i se skládá z m prvků $a_{i,j}$, kde každý prvek je váženou frekvencí výskytu fráze j v dokumentu i. Možné řešení problému rozměrné matice A skýtá kompresní technika, která transformuje A do alternativního prostoru obsahující přibližně stejnou informaci, s využitím menšího počtu dimenzí.

Kanerva a kol. navrhli techniku *náhodného indexování* 7, která využívá rozložení prvků v řídké matici 6. Tato technika umožňuje podstatně zmenšit jeden z rozměrů matice A. Kanerva a kol. aplikovali tento postup na matici kódující vztahy slovo-dokument. Zmenšená matice byla následně použita pro výpočet podobnosti slov s pomocí LSA. S ohledem na detekci plagiátů je matice A, představující model výskytu frází ve zkoumaných dokumentech, extrémně řídká. V tomto ohledu lze techniku náhodného indexování dobře uplatnit.

Transformaci původní matice A o rozměru m x n do nového komprimovaného prostoru A' s rozměrem m' x n naznačuje rovnice (1).

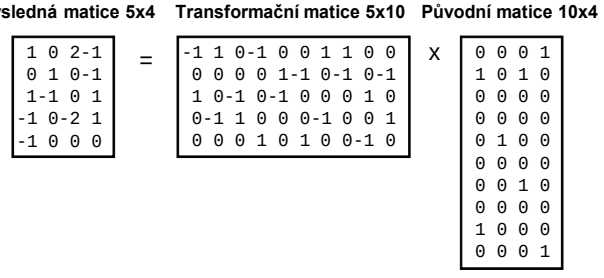
$$A' = T \times A \tag{1}$$

T je v tomto případě transformační matice m' x m složená z m indexových vektorů $[T_1, T_2, \ldots, T_{m'}]$, kde každý indexový vektor T_i obsahuje o náhodně umístěných 1 a -1. Kromě toho jsou všechny indexové vektory vzájemně lineárně nezávislé. Počet náhodně umístěných 1 a -1 musí splňovat podmínku $o << m'$. Aplikací všech těchto kriterií získáme P-unitární matici T, která je pravděpodobnostní a splňuje podmínku (2).

$$\frac{T^T \times T}{2 \cdot o} \approx I \tag{2}$$

Výsledná matice A' tudíž obsahuje dobrou aproximaci informace obsažené v původní matici A.

Obr. 2 prezentuje náhodné indexování na příkladu. Transformační matice T byla vytvořena dle stanovených kritérií, kde o = 1. Původní matice A je velmi řídká, odpovídající situaci výskytu frází v dokumentech, kde řádky představují fráze a sloupečky jsou dokumenty. Transformací získáme matici A', jejíž počet řádek byl podstatně zredukován v porovnání s A. Fráze jsou nyní namapovány na indexové vektory. Obsažená informace v novém prostoru je nicméně stále přibližně stejná, přinejmenším pro náš účel detekce plagiátů.



Obr. 2. Příklad náhodného indexování.

2.2 Normalizace podobností mezi dokumenty

Normalizací podobností mezi dokumenty je nutné se zabývat z důvodu redukčního procesu ve třetí vrstvě, který odstraňuje méně významné fráze. Více informací o tomto problému lze nalézt v článku 2, kde byla rovněž uvedena formule pro tzv. symetrickou (SYM) normalizaci, viz (3). V této formuli $sim_{SVD}(R, S)$ představuje podobnost mezi dokumenty R a S vypočtenou na základě SVD procesu, $ph_{orig}(D)$ označuje množinu frází obsažených v dokumentu D před redukcí a $ph_{red}(D)$ je množina frází po redukci.

$$sim_{SYM}(R,S) = sim_{SVD}(R,S) \cdot \sqrt{\frac{|ph_{red}(R)|}{|ph_{orig}(R)|} \cdot \frac{|ph_{red}(S)|}{|ph_{orig}(S)|}} \tag{3}$$

Nevýhodou symetrické normalizace je nerelevantní hodnocení párů dokumentů s velmi rozdílnými velikostmi, kde jeden je podmnožinou druhého. Formule (4) řeší tento problém výběrem menší z dvou množin frází před redukcí. Tuto normalizaci proto nazýváme jako asymetrickou (ASYM). K odlišení těchto dvou modifikací, označujeme SVDPLAG jako $\text{SVDPLAG}_{\text{SYM}}$ nebo $\text{SVDPLAG}_{\text{ASYM}}$.

$$sim_{ASYM}(R,S) = sim_{SVD}(R,S) \cdot \frac{\sqrt{|ph_{red}(R)| \cdot |ph_{red}(S)|}}{\min\left(|ph_{orig}(R)|, |ph_{orig}(S)|\right)} \tag{4}$$

Dále zavádíme práh $\tau \in \langle 0,1 \rangle$, který představuje minimální stupeň plagiátorství. Pokud je výsledná podobnost mezi dokumenty R a S větší než τ, jsou oba z dokumentů považovány za plagiované, viz (5). Podobnostní míra sim může být v tomto případě zastoupena jak symetrickou, tak asymetrickou variantou.

$$plagiarized(R,S) = \begin{cases} true & if \quad sim(R,S) \geq \tau \\ false & if \quad sim(R,S) < \tau \end{cases} \tag{5}$$

3 Experimenty

3.1 Testovací data

Veškeré experimenty v tomto článku byly provedeny na korpusu 1500 plagiovaných dokumentů o politice psaných v českém jazyce. Celkově se tento korpus skládá z 550 dokumentů, které byly manuálně plagiovány studenty. K tomuto účelu bylo z ČTK korpusu 4, ročník 1999, vybráno 350 zpráv o politice, použitých jako podklad pro vytvoření plagiátů. Zbylých 600 dokumentů bylo vybráno ze stejného zdroje jako nezávislé zpráv na stejné téma, sloužící jako kontrola.

Pro vytvoření 550 plagiovaných dokumentů z 350 zdrojových dokumentů, byli studenti pověřeni kombinovat dva a více náhodně vybraných dokumentů. Výsledkem je, že každý dokument má odlišný stupeň podobnosti se zdrojovým dokumentem.

Při vytváření plagiátů byly uvažovány následující pravidla:

1. Zkopíruj několik odstavců z vybraných dokumentů
2. Smaž okolo 20% vět z nově vytvořeného dokumentu
3. Smaž okolo 10% slov s uvážením smysluplnosti vět
4. Zaměň okolo 20% vět z různých odstavců

5. Přeformuluj okolo 10% vět, přidáním nových myšlenek do textu
6. Pro zajištění smysluplnosti textu mohou být vložena nebo modifikována některá slova v textu

Pro speciální účely byl vytvořen zmenšený korpus 500 dokumentů, který je pouze zmenšenou verzí původního korpusu. Celkově tento zmenšený korpus obsahuje 170 plagiovaných dokumentů, 173 zdrojových dokumentů z ČTK a 157 nezávislých zpráv o politice jakožto kontrola.

3.2 Testovací prostředí

Veškeré experimenty byly provedeny na Intel Core 2 Duo E6600, 4 GB RAM a operačním systému Windows Server 2003 R2 v 64-bitovém režimu. Naše experimentální prostředí bylo vyvinuto v .NET Framework 3.5 s využitím Extreme Optimization Numerical Libraries v3.1 5. Pro efektivní měření časových požadavků byl umožněn běh pouze jednoho vlákna.

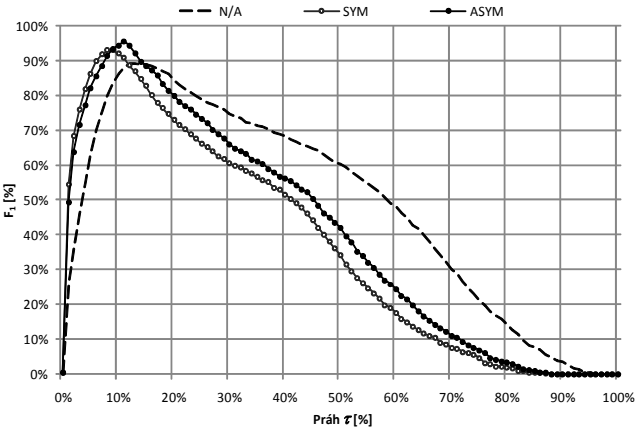
3.3 Užité metriky

K porovnání naměřených výsledků zavádíme standardní míru přesnosti p a úplnosti r dle 11. Dále zavádíme míru F_1, která kombinuje přesnost a úplnost do harmonické střední hodnoty

$$F_1 = \frac{2 \cdot p \cdot r}{p + r}. \qquad (6)$$

3.4 Vliv normalizace podobností mezi dokumenty

Obr. 3 prezentuje rozdíl mezi symetrickou (SYM) a asymetrickou (ASYM) normalizací, kde je zachycena závislost míry F_1 na prahu τ. Asymetrická normalizace získává významnou výhodu pro dokumenty nestejné délky, kde jeden je podmnožinou druhého, což se odráží na výsledku 95,68% F_1 oproti symetrické normalizaci 93,43% F_1.



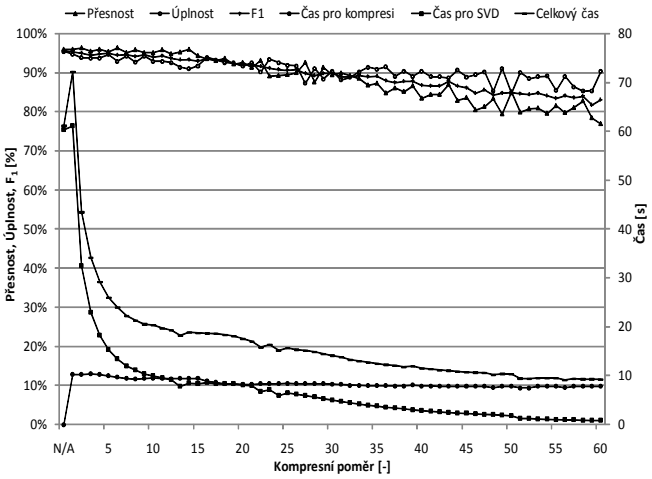
Obr. 3. Závislost míry F_1 na prahu τ pro metodu SVDPLAG (obě varianty symetrická i asymetrická normalizace) s využitím 4-gramů jako příznaků.

Na obrázku je rovněž zachycena křivka bez aplikace normalizační formule „N/A“. V tomto případě dosahuje F_1 pouhých 89,34%, což naznačuje důležitost normalizačního procesu, je-li ve třetí vrstvě aktivován redukční proces odstraňující méně významné fráze. Z důvodu významné výhody asymetrické normalizace jsou následující testy provedeny pouze na této variantě.

3.5 Vliv náhodného indexování

Technika komprese příznaků je klíčovou součástí SVDPLAG metody, která podstatným způsobem snižuje časové i paměťové požadavky pro SVD a umožňuje zpracování rozsáhlých dat.

Obr. 4 zachycuje chování pro různé kompresní poměry na plném korpusu 1500 dokumentů. Přesnost a úplnost naznačují maximální výchylky způsobené náhodným indexováním. Ze statistického pohledu je efekt takový, že pokles v přesnosti vyvolá nárůst v úplnosti. Vlastní míra F_1, která je střední harmonickou hodnotou mezi přesností a úplností, pak pouze zvolna klesá s rostoucími kompresní poměry.



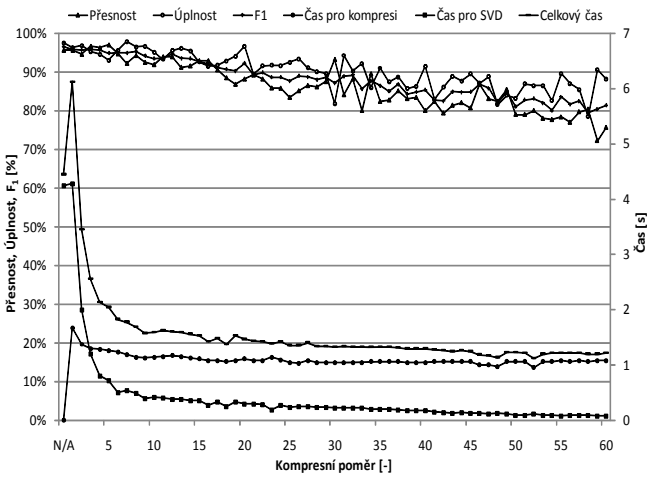
Obr. 4. Vliv kompresního poměru na míru F_1 a výpočetní čas metody SVDPLAG$_{ASYM}$. Experiment byl proveden na plném korpusu 1500 dokumentů.

Situace, kdy není komprese aplikována, je v obrázku označena „N/A“. V tomto případě SVD vyžaduje k běhu 60,34 vteřin. Aktivace vlastní komprese vyžaduje dodatečných 10,20 vteřin pro překódování matice, nicméně vyšší kompresní poměry významně snižují čas pro SVD. Kupříkladu kompresní poměr 1:10 snižuje výpočetní čas SVD na 10,02 vteřin, při současném poklesu času pro vlastní kompresi na 9,48 vteřin. Výsledný čas pro transformaci matice do alternativního prostoru společně s SVD procesem je 19,50 vteřin, což je třikrát nižší čas v porovnání s původními 60,34 vteřinami. Po uplatnění kompresního poměru 1:10 klesá F_1 míra na 94,70% z původních 95,68% (bez komprese pro variantu s asymetrickou normalizací). Vyšší kompresní poměry přináší další snížení časových požadavků, avšak rovněž větší výchylky v přesnosti a úplnosti, vedoucí k výraznějšímu poklesu F_1 míry.

Obr. 5 prezentuje chování na zmenšeném korpusu 500 dokumentů. Technika náhodného indexování dosahuje statisticky lepších výsledků pro rozměrnější data, což můžeme odvodit porovnáním s předchozím obrázkem pro plný korpus 1500 dokumentů. Pro menší objemy dat lze očekávat větší výchylky v přesnosti i úplnosti a rovněž

prudší pokles F_1. Naopak u rozsáhlejších dat je evidentní podstatně hladší průběh, který dovoluje užití vyšších kompresních poměrů.

V obou experimentech byl počet náhodně umístěných 1 a -1 stanoven na 10, viz Sekce 2.1. Tímto je splněna podmínka, že počet náhodně umístěných prvků musí být podstatně menší než rozměr dimenze m' po kompresi. Během experimentů jsme vyzkoušeli široký počet náhodně umístěných čísel, avšak nepodařilo se nám odhalit žádný významný vliv na přesnost ani úplnost.



Obr. 5. Vliv kompresního poměru na míru F_1 a výpočetní čas metody SVDPLAG$_{ASYM}$. Experiment byl proveden na zmenšeném korpusu 500 dokumentů.

4 Závěr

Tento článek představil techniku náhodného indexování a její aplikaci v oblasti detekce plagiátů psaného textu, konkrétně na metodě SVDPLAG. Tato metoda je založena na Latentní sémantické analýze (LSA), využívající matematickou metodu Singulární hodnotové dekompozice (SVD) pro extrakci latentní sémantiky z analyzovaného textu. Extremní řídkost zkoumané matice, kódující vztahy fráze-dokument, dovoluje její kompresi technikou náhodného indexování. Tato technika transformuje původní matici do alternativního prostoru. Tímto postupem lze významně urychlit zpracování rozsáhlých datových kolekcí.

Rovněž byla navržena nová asymetrická normalizace podobností mezi dokumenty, která výrazným způsobem zlepšuje ohodnocení dokumentů nestejné délky, kde jeden je podmnožinou druhého.

Normalizace	Nastavení	Práh τ [%]	F_1 [%]	Celkový čas [s]
SYM	k.p. = n/a	9,3	93,43	60,34
ASYM	k.p. = n/a	11,0	95,68	60,34
ASYM	k.p. = 10	10,8	94,70	19,50

Tab. 1. Přehled dosažených výsledků pro SVDPLAG.

Tab. 1 shrnuje dosažené výsledky pro vlastní kompresní techniku i normalizaci. Jak můžeme vidět, symetrická (SYM) normalizace dosahuje pouhých 93,43% F_1, kdežto asymetrická (ASYM) 95,68% F_1. Aplikací techniky náhodného indexování s kompresním poměrem k.p. = 10 klesá F_1 na 94,70%, nicméně současně se podstatně zrychluje SVD výpočet. V našem případě klesají časové požadavky pro kompresi a SVD proces na jednu třetinu, z 60,34 vteřin na 19,50.

Navržené úpravy vylepšují původní SVDPLAG metodu jak po stránce vyšší F_1 míry, tak po stránce nižších časových požadavků. Dalšího snížení časových požadavků by bylo možné dosáhnout paralelním zpracováním.

Poděkování

Tato práce byla částečně podporována z prostředků Národního Programu Výzkumu II, projekt 2C06009 (COT-SEWing).

Reference

1. M. Berry, S. Dumais, G. O'Brein: *Using linear algebra for intelligent information retrieval.* SIAM Review, Society for Industrial and Applied Mathematics, Philadelphia, USA, 37, 4, 1995, 573-595, ISSN 0036-1445.
2. Z. Ceska: *Využití moderních přístupů pro detekci plagiátů,* Proceedings of the ITAT 2008, Information Technologies – Applications and Theory, Hrebienok, Slovakia, 2008, 23-26, ISBN 978-80-969184-8-5.
3. P. Clough: *Plagiarism in natural and programming languages: An overview of current tools and technologies.* Internal Report CS-00-05, Department of Computer Science, University of Sheffield, 2000.
4. ČTK, Czech News Agency: Political and General News Service. URL: http://www.ctk.cz/sluzby/databaze/ dokumentacni/
5. Extreme Optimization, „Numerical Libraries for .NET Professional 3.1 64 bit", poslední změna 24.11.2008. URL: http://www.extremeoptimization.com/
6. P. Kanerva: *Sparse distributed memory*. The MIT Press, 2nd rev. edition, 1988, ISBN 0-262-11132-2.
7. P. Kanerva, J. Kristoferson, A. Holst: *Random indexing of text samples for latent semantic analysis*. Proceedings of the 22nd Annual Conference of the Cognitive Science Society, 2000, 103-106.
8. T. Landauer, P. Foltz, D. Laham: *An introduction to latent semantic analysis*. Discourse Processes, 25, 1998, 259-284.
9. P. Lane, C. Lyon, J. Malcolm: *Demonstration of the ferret plagiarism detektor*., Proceedings of the 2nd International Plagiarism Conference, Newcastle, 2006.
10. H. Maurer, F. Kappe, B. Zaka: *Plagiarism – a survey*. Journal of Universal Computer Science, 12, 8, 2006, 1050-1084.
11. C. J. van Rijsbergen: *Information retrieval*. Butterworth-Heinemann, 2nd rev. edition, 1979. ISBN 0-408-70929-4.
12. P. Runeson, M. Alexanderson, O. Nyholm: *Detection of duplicate defect reports using natural language processing.* Proceedings of the IEEE 29th International Conference on Software Engineering, 2007, 499-510.
13. N. Shivakumar, H. Garcia-Molina: *SCAM: A copy detection mechanism for digital documents*. Proceedings of 2nd International Conference in Theory and Practice of Digital Libraries, Austin, 1995.

Mandatory access control for small office and home environment*

Jaroslav Janáček

Department of Computer Science, Faculty of Mathematics, Physics and Informatics
Comenius University, Mlynská dolina, 842 48 Bratislava, Slovakia
janacek@dcs.fmph.uniba.sk

Abstract. *Personal computers are often used in small office and home environment for a wide range of purposes – from general web browsing and e-mail processing to processing data that are sensitive regarding their confidentiality and/or integrity. Commonly used general purpose operating systems for personal computers implement discretionary access control that is designed to allow the owner of an object to specify which users are authorized to perform individual operations on the object, but that provides no protection against access by malicious applications running on behalf of the object's owner. We present a security model that combines the ideas and benefits of mandatory access control used in classified information processing systems with the requirements resulting from the typical small office and home computer use. The model is based on a simple two-dimensional data classification scheme taking into account both confidentiality and integrity independently, and it reduces the level of trust that has to be given to the applications that need to pass information in the normally forbidden direction.*

1 Introduction

Personal computers are used for a wide range of purposes – from general web browsing and e-mail processing to processing sensitive data in applications such as Internet banking, electronic signature creation and verification, processing sensitive personal and business data, and others. Applications often contain programming errors that can be abused to execute arbitrary code in place of the application by providing specially crafted input to the application. Therefore, applications that process data originating from untrusted sources cannot be trusted not to perform some malicious activities.

Consider, for example, an e-mail reading program containing a programming error that can be abused to execute code included in a specially crafted e-mail message. When such an e-mail message is received, the code included in it is executed as a part of the e-mail reading application's process, and it can perform whatever the e-mail reading application is allowed to by the operating system.

While larger organizations can dedicate some computers to the sensitive data processing and prevent them from communicating with untrusted external systems, it can be hardly expected in the small office or home environment. It these environments, a single computer with a single operating system is usually used for the whole range of purposes.

Current common desktop operating systems used in the small office and home environments (such as Microsoft Windows XP Professional, Vista or various distributions of Linux) provide discretionary access control mechanisms that allow the users to protect their files against access and/or modification by other users[1]. It is based on the following principle:

- Every filesystem object (such as a file or a directory) is owned by a user or a group of users – its *owner*.
- Every process (*subject*) runs on behalf of a user.
- The owner of an object can specify a discretionary access control list for the object. Each entry in the access control list specifies the permitted operations for processes running on behalf of an identified user (or a group of users).

If a user can perform an operation on a file containing a sensitive piece of information, any process running on behalf of the user can perform the operation on the file. If, for example, a user runs an e-mail reading application that can be tricked to execute a piece of code included in a specially crafted message, an attacker can gain access to all files that the user has access to – including files containing sensitive information.

The insufficiency of the discretionary access control in the common operating systems has motivated the work on integrating other controls. In the Linux world, the most significant projects are Linux Security Modules (LSM)[4] and SELinux[5]. LSM provides a framework in the Linux kernel to integrate a security module with the kernel. The kernel then calls the security module's functions whenever it needs to make an access control decision.

SELinux is a security module (using LSM) implementing Domain and Type Enforcement (DTE) [3, 6] for Linux. DTE associates a *domain* with each subject and a *type* with each object, and a policy specifies

* This paper was supported by the grant VEGA 1/0266/09.

[1] Another common operating system – Microsoft Windows Home edition – does not provide this access control mechanism.

which operations can be performed by a subject in the domain D on an object of type T. SELinux (DTE in general) can be used to enforce a wide range of security policies but the rules may become rather complex. While it is a great tool for securing systems with well-defined security requirements (such as servers), it is difficult to prepare a general policy for desktops where the users' requirements differ significantly [7].

In the Windows world, Windows Vista comes with two significant security enhancements – User Account Control (UAC) [8] and Mandatory Integrity Control (MIC) [9]. UAC deals with the fact that, while it is a bad security practice, many (or most?) users use accounts with the administrator's privileges. Any malicious code executed on behalf of such user has the full privileges on the system. When UAC is enabled, Windows Vista prompts the user using a pop-up window when a program actually needs to use the administrator's privileges.

MIC allows subjects and objects to be assigned an *integrity level* and prevents subjects from modifying objects with a higher integrity level than that of the subject. In fact, it implements one of the rules of the Biba model (see sec. 3). It may be also configured to prevent subjects from reading from objects with a higher level, i.e. it optionally implements one of the rules of the Bell-LaPadula model (see sec. 3).

We present a security model suitable for a desktop operating system for the small office and home environment that deals with the protection of confidentiality and integrity of data against malicious applications.

2 Security needs of typical data

In this section we will consider several classes of applications typically used in the small office and home environment, and specify the security requirements for the data they process.

2.1 Classes of typical applications

General Internet access One of the typical classes of applications is the class of the applications used to access Internet resources. It includes web-browsers, e-mail clients, and various other communication systems (e.g. ICQ[2], IRC[3], VoIP[4], video-conferencing systems, ...). The common feature of these applications is that they communicate with external systems that cannot be trusted to protect the confidentiality of the data transferred to them, nor can they be trusted not to send *malicious data* back to our system. By the term *malicious data* we mean data that have been prepared in a way to abuse a weakness of the application receiving them. Due to programming errors, the applications in this class can often be abused to perform malicious activities by providing specially prepared data. Therefore, even if the application itself is believed not do anything unwanted, its behaviour may be changed, by processing malicious data, in an unpredictable way.

It has to be assumed that the applications of this class:

- export any data available to them to the Internet,
- import (potentially) malicious data from the Internet, and therefore, their output has to be considered (potentially) malicious too,
- may become malicious by processing the malicious input.

Local applications The class of local applications contains the applications that are used to process data stored in a local filesystem. These applications generally do not need network access to perform their tasks. The typical examples are text processors, spreadsheets, presentation software, graphic editors, ...

Local applications are used to process data with varying requirements regarding the confidentiality and integrity protection. If they process malicious data, they may become malicious due to programming errors.

Sensitive web access Web browsers are often used to access remote services that process data requiring confidentiality and/or integrity protection. A typical example is an Internet-banking system. It provides access to financial information; it allows the user to submit transaction orders to the bank, etc. It also processes authentication data (e.g. passwords). All such data may be considered confidential by the user, and therefore, are to be adequately protected. The confidentiality and the integrity of the data during their transmission is usually protected by means of cryptography. Cryptography is usually also used to provide authentication of the remote system. But the data is also to be protected while stored in the memory or in a file on the local computer. Consider an instance of a web browser used for general Internet access. It may have processed some malicious data, a and therefore, it may have become a malicious application exporting everything to an attacker. If the instance of the web browser is later used to access an Internet banking system, all the confidential information may leak.

[2] I Seek You – an instant messaging system for sending online and offline messages between users.

[3] Internet Relay Chat – a real-time Internet chat system designed mainly for group communication in discussion forums – channels.

[4] Voice over IP – digital voice communication using IP networks.

Digital signature and data encryption Digital signature creation applications, as well as data decryption applications need access to a private key. Digital signature verification applications, as well as data encryption applications need access to a public key.

The private key is a very sensitive piece of information the confidentiality of which has to be protected. The integrity of the private key has to be protected as well because its modification can lead not only to the loss of ability to create correct digital signatures or to decrypt data, but also to the leak of information that is sufficient to compute the corresponding private key in certain cases.

The public key requires no confidentiality protection, but it does require integrity protection. If attackers were able to modify the public key used to verify a digital signature, they would be able to create a digitally signed document that would pass the signature verification process. In the case of encryption, if the public key were modified by an attacker, the attacker would be able to decrypt the encrypted data instead of the intended receiver.

The encrypted output of a data encryption application may be transmitted via communication channels that do not provide confidentiality protection even if the confidentiality of the original data is to be protected. The output of a data decryption application may also require confidentiality protection.

2.2 Data classification scheme

We can conclude, from the previous subsection, that the need of confidentiality protection and the need of integrity protection are independent on each other. Some data need integrity protection while they can be disclosed to the public, some data need both, some need none. We will, therefore, use a two-dimensional classification scheme for the data consisting of the *confidentiality level* and the *integrity level*.

As far as the confidentiality is concerned, we can classify the data into three basic categories:

- public data,
- normal data – *C-normal*,
- data that are sensitive regarding their confidentiality – *C-sensitive*.

The public data require no confidentiality protection. They may be freely transmitted via communication channels and/or to remote systems that provide no confidentiality protection. An example of public data is the data downloaded from public Internet.

The normal data are to be protected by means of discretionary access control against unauthorized reading by other users than the owner of the data.

The C-sensitive data are the data that their owner (a user) wishes to remain unreadable to the others regardless of the software the user uses, and even if the users makes some mistakes (such as setting wrong access rights for discretionary access control). Examples of C-sensitive data are private and secret keys, passwords for Internet banking, etc.

As far as the integrity (or trustworthiness) of data is concerned, we can also classify the data into three basic categories:

- potentially malicious data,
- normal data – *I-normal*,
- data that are sensitive regarding their integrity – *I-sensitive*.

The requirements of the integrity protection of data is tightly coupled to the trustworthiness of the data. The trustworthiness of data can be thought of as a metric of how reliable the data are. If some data can be modified by anyone, they cannot be trusted not to contain wrong or malicious information. If some data are to be relied on, their integrity has to be protected.

The potentially malicious data require no integrity protection, and can neither be trusted to contain valid information, nor can be trusted not to contain malicious content.

The normal data is to be protected by means of discretionary access control against unauthorized modification by other users that the owner of the data.

The I-sensitive data are the data that their owner wishes to remain unmodified by the others regardless of the software the user uses, and even if the users makes some mistakes. The I-sensitive data are to be modifiable only under special conditions upon their owner's request. A special category of I-sensitive data is the category of the shared system files such as the programs, the libraries, various system-wide configuration files, the user database, ... Some of these files may be modifiable by the designated system administrator, some of them should be even more restricted.

The number of the confidentiality and integrity categories may be higher in real systems. The three levels described above have a general and easy to understand meaning, and we will use them in this chapter to explain the ideas of our model. We will not, however, restrict the number levels to any fixed value.

2.3 Classification of typical data

Having specified the classification scheme we can reconsider the typical applications and specify the typical classification of the data they process.

General Internet access The data originating from the general Internet have to be considered potentially malicious. No data other than public should be readable by the applications of this class.

Local applications The applications of this class are used to process data with different classification levels. In order to protect confidentiality of data, these applications may not be allowed to pass information from an object with a higher confidentiality level to an object with a lower confidentiality level. In order to protect integrity of data, they may not be allowed to pass information from an object with a lower integrity level to an object with a higher integrity level.

Sensitive web access The applications of this class communicate with external systems that can be trusted (to some extent) to protect confidentiality of the transmitted information, and also not to provide potentially malicious data. The data received from these external systems may also require confidentiality protection at the C-normal (or even C-sensitive) level.

Digital signature and data encryption The private key should be classified as C-sensitive and I-sensitive. The public keys should be classified as I-sensitive and may be public. The integrity level of the output of the signature verification application may be higher than the integrity level of the corresponding input if the signatory is trustworthy and the signature verification succeeds. The confidentiality level of the output of the data encryption application may be lower than that of the input because the confidentiality is protected by encryption. The confidentiality level of the decrypted output of the decryption application should be often higher than that of the encrypted input as the confidentiality protection by encryption is removed.

3 Security model

A common approach to ensuring the confidentiality and/or the integrity of information in systems that deal with data classified into several confidentiality and/or integrity levels, is to define an information flow policy, and then to enforce the policy. In order to enforce an information flow policy, subjects are divided into two categories – trusted and untrusted. A trusted subject is a subject that is trusted to enforce the information flow policy (with exceptions) by itself; an untrusted subject is a subject that is not trusted to enforce the policy by itself, and therefore the policy has to be enforced on the subject's operations by the system.

A typical information flow policy protecting confidentiality (e.g. one based on Bell-LaPadula model[1]) states that a subject operating at a confidentiality level C_S may only read from an object with a confidentiality level C_{O_r} if $C_S \geq C_{O_r}$, and may only write to an object with a confidentiality level C_{O_w} if $C_S \leq C_{O_w}$. If a subject is to be able to read from a more confidential object, and to write to a less confidential object, it has to be a trusted subject.

A typical information flow policy protecting integrity (e.g. one based on Biba model[2]) states that a subject operating at an integrity level I_S may only read from an object with an integrity level I_{O_r} if $I_S \leq I_{O_r}$, and may only write to an object with an integrity level I_{O_w} if $I_S \geq I_{O_w}$. Only a trusted subject can read from an object with a lower integrity level, and write to an object with a higher integrity level.

The requirements of the Bell-LaPadula model provide a guarantee that no information can be passed from a more confidential object to a less confidential object by untrusted subjects. The requirements of the Biba model, on the other hand, provide a guarantee that no information can be passed from an object with a lower integrity level to an object with a higher integrity level.

While these models are efficient in the traditional classified information processing systems, they are not very suitable for the small office and home environment. The problem is that many of the applications we considered in the previous section would have to be trusted subjects, but they often cannot be fully trusted.

For example, a digital signature verification application may be used to read a potentially malicious input, and to produce an I-normal copy of the input if the signature is correct. It would, therefore, have to be a trusted subject allowed to read from the potentially malicious level and to write to the I-normal level. If it contained exploitable errors, it could be abused to pass other information from potentially malicious to I-normal objects.

Another good example is the signature creation application. It needs to be able to read from C-sensitive private key and write to a lower confidentiality level. It would need to be a trusted subject, and then it would be allowed to pass information from C-sensitive objects to objects at lower confidentiality levels.

To minimize the level of trust that has to be given to the applications, we divide the subjects into three categories:

- untrusted subjects,
- partially trusted subjects, and
- trusted subjects.

An untrusted subject is a subject that is not trusted to enforce the information flow policy. It is assumed to perform any operations on any objects unless it is prevented from doing so by the operating system.

A trusted subject is a subject that is trusted to enforce the information flow policy by itself. A trusted subject may be used to perform tasks than require violation of the policy under conditions that are verified by the trusted subject. A trusted subject can, therefore, be used to implement an exception to the policy.

A partially trusted subject is a subject that is trusted to enforce the information flow policy regarding a specific set of objects, but not trusted to enforce the information flow policy regarding any other objects. In other words, a trusted subject is

- trusted not to transfer information from a defined set of objects (designated inputs) at a higher confidentiality level to a defined set of objects (designated outputs) at a lower confidentiality level in a way other than the intended one, and
- trusted not to transfer information from a defined set of objects (designated inputs) at a lower integrity level to a defined set of objects (designated outputs) at a higher integrity level in a way other than the intended one, but
- not trusted not to transfer information between any other objects.

The sets of designated inputs and outputs regarding confidentiality are distinct from the sets regarding integrity. Any of the sets may be empty. A partially trusted subject, like a trusted one, can be used to implement an exception to the policy, because it can violate the policy (and it is trusted to do it only in an intended way).

The most important difference between trusted and partially trusted subjects is in the level of trust. While trusted subjects are completely trusted to behave correctly, partially trusted subjects are only trusted not to abuse the possibility of the information flow violating the policy between a defined set of input objects and a defined set of output objects.

3.1 Formal definition of the information flow policy

Let $\mathbf{C} = \{0, 1, \ldots, c_{\max}\}$ be the set of confidentiality levels, $\mathbf{I} = \{0, 1, \ldots, i_{\max}\}$ be the set of integrity levels, and $\mathbf{L}$ be a finite set of possible labels for objects, $0 \in \mathbf{L}$ being the default label used for objects without an explicitly assigned label. The labels will be used to define the sets of designated inputs and outputs for partially trusted subjects. Let $\mathbf{C}$ and $\mathbf{I}$ be ordered so that 0 is the least sensitive level and $c_{\max}$ and $i_{\max}$ are the most sensitive levels.
Let each object O have the following attributes:

- $C_O \in \mathbf{C}$ – the confidentiality level of the object,
- $I_O \in \mathbf{I}$ – the integrity level of the object,
- $L_O \in \mathbf{L}$ – the label assigned to the object.

Let each subject S have the following attributes:

- $CR_S \in \mathbf{C}$ – the highest confidentiality level the subject can normally read from,
- $CW_S \in \mathbf{C}$ – the lowest confidentiality level the subject can normally write to,
- $CRL_S \in \mathbf{C}$ – the highest confidentiality level of a specially labelled object that the subject can read from,
- $CWL_S \in \mathbf{C}$ – the lowest confidentiality level of a specially labelled object that the subject can write to,
- $CRLS_S \subseteq \mathbf{L}$ – the set of labels of the objects that the subject can read from as a partially trusted subject,
- $CWLS_S \subseteq \mathbf{L}$ – the set of labels of the objects that the subject can write to as a partially trusted subject,
- $IR_S \in \mathbf{I}$ – the lowest integrity level the subject can normally read from,
- $IW_S \in \mathbf{I}$ – the highest integrity level the subject can normally write to,
- $IRL_S \in \mathbf{I}$ – the lowest integrity level of a specially labelled object that the subject can read from,
- $IWL_S \in \mathbf{I}$ – the highest integrity level of a specially labelled object that the subject can write to,
- $IRLS_S \subseteq \mathbf{L}$ – the set of labels of the objects that the subject can read from as a partially trusted subject,
- $IWLS_S \subseteq \mathbf{L}$ – the set of labels of the objects that the subject can write to as a partially trusted subject.

A subject S can read from an object O if $\mathbf{read}(S, O)$ is true, where

$$\begin{aligned}&\mathbf{read}(S,O) \iff \\ &\quad [CR_S \geq C_O \vee (CRL_S \geq C_O \wedge L_O \in CRLS_S)] \\ &\wedge\, [IR_S \leq I_O \vee (IRL_S \leq I_O \wedge L_O \in IRLS_S)]\end{aligned}$$

A subject S may write to an object O if $\mathbf{write}(S, O)$ is true, where

$$\begin{aligned}&\mathbf{write}(S,O) \iff \\ &\quad [CW_S \leq C_O \vee (CWL_S \leq C_O \wedge L_O \in CWLS_S)] \\ &\wedge\, [IW_S \geq I_O \vee (IWL_S \geq I_O \wedge L_O \in IWLS_S)]\end{aligned}$$

Each untrusted subject S must satisfy the following conditions:

$$\begin{gathered}CW_S = CWL_S \geq CR_S = CRL_S \\ IW_S = IWL_S \leq IR_S = IRL_S \\ CWLS_S = CRLS_S = IWLS_S = IRLS_S = \emptyset\end{gathered}$$

Each partially trusted subject S must satisfy the following conditions:

$$CW_S \geq CR_S$$
$$CW_S \geq CRL_S$$
$$CWL_S \geq CR_S$$
$$IW_S \leq IR_S$$
$$IW_S \leq IRL_S$$
$$IWL_S \leq IR_S$$

As far as only untrusted subjects are concerned, the $\mathbf{read}(S, O)$ and $\mathbf{write}(S, O)$ predicates reduce to a combination of the rules of Bell-LaPadula and Biba models. Trusted subjects are given upper boundaries on the confidentiality level for reading and on the integrity level for writing, and lower boundaries on the confidentiality level for writing and on the integrity level for reading. Partially trusted subjects are allowed to pass information from their designated inputs to their designated outputs, but are very limited otherwise.

3.2 Security properties of the information flow policy

We will state some security properties of the information flow policy defined above in the formal way. The theorems can be proven although the proofs are out of the scope of this paper (due to their length). They are based on deriving contradictions if the theorems did not hold. First, we will formally define the flow of information within the system.

Definition 1. *Let* $\mathbf{S}$ *be a set of subjects and* $\mathbf{O}$ *be a set of objects. Let* $O_0, O_{out} \in \mathbf{O}$ *be any two objects. We say that a policy allows an information flow from* O_0 *to* O_{out} *within the system* $(\mathbf{S}, \mathbf{O})$ *and denote it as* $\boldsymbol{flow}(\mathbf{S}, \mathbf{O}, O_0, O_{out})$ *if there exists a finite sequence of pairs* $(S_1, O_1), (S_2, O_2), \ldots, (S_n, O_n)$ *where* $\forall i \in \{1, \ldots, n\} : O_i \in \mathbf{O} \wedge S_i \in \mathbf{S}$ *such that*

$$\forall i \in \{1, \ldots, n\} : \boldsymbol{read}(S_i, O_{i-1}) \wedge \boldsymbol{write}(S_i, O_i)$$
$$\text{and} \quad O_n = O_{out}$$

where $\boldsymbol{read}(S, O)$ *and* $\boldsymbol{write}(S, O)$ *are the functions of the policy determining whether the subject* S *can read from, or write to the object* O*.*

Using the formal definition of flow, we can precisely define what we mean by an information leak – a violation of the confidentiality protection requirements.

Definition 2. *Let* $\mathbf{S}$ *be a set of subjects and* $\mathbf{O}$ *be a set of objects. We say that a policy allows an information leak within the system* $(\mathbf{S}, \mathbf{O})$ *and denote it as* $\boldsymbol{leak}(\mathbf{S}, \mathbf{O})$ *if*

$$\exists O_a, O_b \in \mathbf{O} : C_{O_a} > C_{O_b} \wedge \boldsymbol{flow}(\mathbf{S}, \mathbf{O}, O_a, O_b)$$

We can also define the precise meaning of a violation of the integrity protection requirements – information spoiling.

Definition 3. *Let* $\mathbf{S}$ *be a set of subjects and* $\mathbf{O}$ *be a set of objects. We say that a policy allows information spoiling within the system* $(\mathbf{S}, \mathbf{O})$ *and denote it as* $\boldsymbol{spoil}(\mathbf{S}, \mathbf{O})$ *if*

$$\exists O_a, O_b \in \mathbf{O} : I_{O_a} < I_{O_b} \wedge \boldsymbol{flow}(\mathbf{S}, \mathbf{O}, O_a, O_b)$$

Having the definitions, we can state the basic security properties using the following theorems. The first two theorems deal with the case when there are only untrusted subjects. In such case, the information flow policy guarantees that no information from a more confidential object can end up in a less confidential object, and that no information from an object with a lower integrity level can influence any object with a higher integrity level.

Theorem 1. *If* $\mathbf{S}$ *is a set of* untrusted *subjects and* $\mathbf{O}$ *is a set of objects, our policy does not allow any information leak within the system* $(\mathbf{S}, \mathbf{O})$*.*

Theorem 2. *If* $\mathbf{S}$ *is a set of* untrusted *subjects and* $\mathbf{O}$ *is a set of objects, our policy does not allow any information spoiling within the system* $(\mathbf{S}, \mathbf{O})$*.*

Another two theorems deal with the case when there may be some partially trusted subjects as well. The first of these theorems says that in order to pass information from an object with a higher confidentiality level to an object with a lower confidentiality level using only untrusted and partially trusted subjects, each subject that passes information from an object with a higher confidentiality level to an object with a lower confidentiality level, must be a partially trusted subject and it must be passing the information from its specially labelled input to its specially labelled output. Assuming that no partially trusted subject passes information from its special inputs to its special outputs in an unintended way, any information leak allowed by the policy within a system without trusted subjects is intended.

Theorem 3. *Let* $\mathbf{S}$ *be a set of* untrusted *and/or* partially trusted *subjects and let* $\mathbf{O}$ *be a set of objects. Let* $O_0, O_{out} \in \mathbf{O}$ *be two objects such that* $C_{O_0} > C_{O_{out}}$ *and* $\boldsymbol{flow}(\mathbf{S}, \mathbf{O}, O_0, O_{out})$*. For every finite sequence of pairs* $(S_1, O_1), \ldots, (S_n, O_n)$ *such that* $\forall i \in \{1, \ldots, n\} : S_i \in \mathbf{S} \wedge O_i \in \mathbf{O} \wedge \boldsymbol{read}(S_i, O_{i-1}) \wedge \boldsymbol{write}(S_i, O_i) \wedge O_n = O_{out}$ *for each pair* (S_j, O_j) *such that* $C_{O_{j-1}} > C_{O_j}$*:*

$$S_j \text{ is a partially trusted subject}$$
$$\text{and } L_{O_{j-1}} \in CRLS_{S_j}$$
$$\text{and } C_{O_{j-1}} \leq CRL_{S_j}$$
$$\text{and } L_{O_j} \in CWLS_{S_j}$$
$$\text{and } C_{O_j} \geq CWL_{S_j}$$

The last theorem says, that in order to pass information from an object with a lower integrity level to an object with a higher integrity level using only untrusted and partially trusted subjects, each subject that passes information from an object with a lower integrity level to an object with a higher integrity level, must be a partially trusted subject and it must be passing the information from its specially labelled input to its specially labelled output. Assuming that no partially trusted subject passes information from its special inputs to its special outputs in an unintended way, any information spoiling allowed by the policy within a system without trusted subjects is intended.

Theorem 4. *Let* $\mathbf{S}$ *be a set of* untrusted *and/or* partially trusted *subjects and let* $\mathbf{O}$ *be a set of objects. Let* $O_0, O_{out} \in \mathbf{O}$ *be two objects such that* $I_{O_0} < I_{O_{out}}$ *and* $\boldsymbol{flow}(\mathbf{S}, \mathbf{O}, O_0, O_{out})$*. For every finite sequence of pairs* $(S_1, O_1), \ldots, (S_n, O_n)$ *such that* $\forall i \in \{1, \ldots, n\} : S_i \in \mathbf{S} \wedge O_i \in \mathbf{O} \wedge \boldsymbol{read}(S_i, O_{i-1}) \wedge \boldsymbol{write}(S_i, O_i) \wedge O_n = O_{out}$ *for each pair* (S_j, O_j) *such that* $I_{O_{j-1}} < I_{O_j}$

$$
\begin{aligned}
&S_j \text{ is a partially trusted subject} \\
&\text{and } L_{O_{j-1}} \in IRLS_{S_j} \\
&\text{and } I_{O_{j-1}} \geq IRL_{S_j} \\
&\text{and } L_{O_j} \in IWLS_{S_j} \\
&\text{and } I_{O_j} \leq IWL_{S_j}
\end{aligned}
$$

3.3 Usage examples

We will now show how our security model can be used to protect the typical data mentioned in some of the examples from the previous section.

General Internet access The applications of this class should be run as untrusted subjects with $CR_S = CW_S = 0$ and $IR_S = IW_S = 0$ (assuming the communication objects to access the Internet are classified as public and potentially malicious). Whatever the application does, it will be unable to export any non-public data to the Internet, and it will be unable to modify anything other than potentially malicious objects. The effect of a malicious code being executed as the result of an exploited bug in the application will remain limited.

Local applications The applications of this class should be run as untrusted subjects with the confidentiality and integrity levels corresponding to the files they are to process. They will be unable to cause any information leak or information spoiling.

Sensitive web access The applications of this class may be run either as untrusted subjects at the appropriate confidentiality and integrity levels (most probably C-normal, I-normal), or as partially trusted subjects. In the former case, the communication objects to access the trusted external system would be classified at the appropriate levels (e.g. C-normal, I-normal).

An example of the latter case follows. The subject S representing the application that needs to transfer C-normal data between the local system and a trusted remote system would have the following attributes:

$$
\begin{aligned}
&CR_S = 0 \\
&CW_S = 1 \\
&CRL_S = 1 \\
&CRLS_S = \{L_{local}\} \\
&CWL_S = 0 \\
&CWLS_S = \{L_{ext}\} \\
&IR_S = IW_S = IRL_S = IWL_S = 0 \\
&IRLS_S = IWLS_S = \emptyset
\end{aligned}
$$

The communication object to access the trusted remote system would be classified as public and potentially malicious, but labelled with the label L_{ext}. The local objects at the C-normal level that are to be transferable to the remote system would be labelled with the label L_{local}. This way, the application would be able to read the selected C-normal data (and any public data) and would be able to send data to the designated remote system (and to write data to any C-normal or higher and potentially malicious file).

Digital signature creation The digital signature creation application should be run as a partially trusted subject S with the following attributes:

$$
\begin{aligned}
&CR_S = 1 \\
&CW_S = 2 \\
&CRL_S = 2 \\
&CRLS_S = \{L_{privatekey}\} \\
&CWL_S = 1 \\
&CWLS_S = \{L_{signed}\} \\
&IR_S = IW_S = IRL_S = IWL_S = 1 \\
&IRLS_S = IWLS_S = \emptyset
\end{aligned}
$$

The private key would be C-sensitive, I-sensitive with the label $L_{privatekey}$, the files to sign would be C-normal or lower and I-normal or higher, and the file for output would be C-normal, labelled with L_{signed} and I-normal or lower. This way the application must be trusted only not to reveal the private key through the signed output in order to prevent information leak.

Digital signature verification The digital signature verification application used to increase the integrity level when the verification is successful should be run as a partially trusted subject S with the following attributes:

$$CR_S = CW_S = CRL_S = CWL_S = 1$$
$$CRLS_S = CWLS_S = \emptyset$$
$$IR_S = 1$$
$$IW_S = 0$$
$$IRL_S = 0$$
$$IWL_S = 1$$
$$IRLS_S = \{L_{toverify}\}$$
$$IWLS_S = \{L_{verified}\}$$

The public key would be public and I-sensitive, the signed files to verify could be potentially malicious labelled with $L_{toverify}$ and the output would be I-normal and labelled with $L_{verified}$. The application would be unable modify any I-normal file except for its output, and it could not be influenced by any potentially malicious data except the data to verify.

Data encryption The data encryption application using asymmetric cryptography to encrypt C-normal data to produce public encrypted output would be run as a partially trusted subject S with the following attributes:

$$CR_S = 0$$
$$CW_S = 1$$
$$CRL_S = 1$$
$$CWL_S = 0$$
$$CRLS_S = \{L_{toencrypt}\}$$
$$CWLS_S = \{L_{encrypted}\}$$
$$IR_S = IW_S = IRL_S = IWL_S = 1$$
$$IRLS_S = IWLS_S = \emptyset$$

The C-normal data to encrypt would be labelled with $L_{toencrypt}$ and the encrypted public output would be labelled $L_{encrypted}$. The application would have to be trusted only not to reveal the original data through its designated output.

4 Conclusions

The presented security model supplements the traditional discretionary access control by providing protection of confidentiality and integrity of sensitive data against malicious applications running on behalf of the owner of the data. Unlike MIC in Windows Vista, it deals with the confidentiality and integrity aspects separately, and it provides provable security properties similar to those of Biba and Bell-LaPadula models. When extended to cover other operations (e.g. creation and deletion of objects, object attributes' manipulation, interaction between subject, etc.) and implemented, it will allow a user to run applications processing sensitive data alongside potentially malicious applications while assuring the user that the malicious applications cannot interfere with the sensitive data, whether regarding the confidentiality or the integrity aspect.

As far as the feasibility of implementation is concerned, there are at least two ways to implement our security model in Linux operating system. One approach is to utilize LSM and write a security module that would associate the model's attributes with the subjects and objects, and that would make the access control decisions according to our policy. Another approach is to utilize SELinux, define domains and types corresponding to the needed combinations of the model's attributes, and write DTE policy rules that enforce our policy. The advantage of the latter approach is that our policy can be combined with other policies that can be enforced using SELinux, but it may be more complicated than the former approach.

References

1. D.E. Bell, L.J. La Padula: *Secure computer system: unified exposition and multics interpretation.* Technical report, 1976.
2. H.F. Tipton, M. Krause (editors): *Information security management handbook*, 5th edition, CRC Press LLC, 2004, ISBN 0-8493-1997-8.
3. L. Badger, D.F. Sterne, D.L. Sherman, K.M. Walker, S.A. Haghighat: *Practical domain and type enforcement for UNIX.* In Proceedings of the 1995 IEEE Symposium on Security and Privacy.
4. Linux Security Modules, `http://lsm.immunix.org/`.
5. Security Enhanced Linux, `http://www.nsa.gov/selinux/`.
6. Domain and Type Enforcement, `http://www.cs.wm.edu/ ~hallyn/dte/`.
7. Fedora SELinux Project – Discussion of Policies, `http://fedoraproject.org/wiki/SELinux/Policies`.
8. Mark Russinovich: Inside Windows Vista User Account Control, `http://technet.microsoft.com/en-us/magazine/2007.06.uac.aspx`
9. Steve Riley: Mandatory integrity control in Windows Vista, `http://blogs.technet.com/steriley/archive/2006/07/21/442870.aspx`

Epizodická paměť inteligentních virtuálních agentů

Rudolf Kadlec and Cyril Brom
Matematicko-Fyzikální fakulta
Univerzita Karlova v Praze
Univerzita Karlova, Malostranské nám. 25, 118 00 Praha 1, Česká Republika
rudolf.kadlec@gmail.com, brom@ksvi.mff.cuni.cz

Abstract. *Epizodickou pamětí nazýváme souhrn procesů zodpovědných za zapamatovávání, uchovávání a vybavování událostí, jichž byl člověk osobně účasten. Současní inteligentní virtuální agenti, jakými jsou například postavy z počítačových her, většinou epizodickou paměti vybaveni nejsou. Pro agenta může ale epizodická paměť představovat důležitý zdroj informací a pomáhat mu například učit se ze svých chyb nebo interpretovat pozorovaný stav světa. Účelem tohoto článku je seznámit čtenáře s naším přístupem k zkoumání epizodické paměti agentů. Nejprve se budeme zabývat samotnou definicí epizodické paměti, poté prozkoumáme její možné aplikace v agentních systémech a nakonec budeme prezentovat plán našeho budoucího výzkumu, především odhalování motivací agentů.*

1 Úvod

Považujeme za samozřejmé, že lidé jsou schopni si zapamatovat svoji předešlou zkušenost, tuto zkušenost mohou zprostředkovávat ostatním skrze jazyk, umí navázat v činnostech přerušených naléhavějšími úkoly, dovedou odhadovat motivace ostatních lidí a na základě toho předpovídat jejich budoucí akce. Jakkoliv jsou tyto schopnosti přirozené u lidí, v oblasti umělé inteligence nebyly tyto analogické problémy stále uspokojivě vyřešeny.

Epizodická paměť (EP) [23] může být společným jmenovatelem řešení všech výše zmíněných problémů. Epizodická paměť je z psychologického hlediska pamětí na události (epizody) z vlastního života. Ústřední je vlastní prožitek a tudíž i možné subjektivní hodnocení pozorované situace. Příkladem krátké epizody uložené v epizodické paměti může být: „minulý víkend mě na chatě poškrábala sousedova kočka“. Epizodická paměť v sobě sdružuje mnoho modalit, např.: místo, čas, motivaci, emoce a další.

Psychologie odděluje epizodickou paměť od sémantické paměti, obsahující obecné znalosti o světě. Zdá se, že tyto systémy jsou částečně disociovány i neurobiologicky, zároveň jsou ale na této úrovni i částečně provázané; konkrétní detaily interakce mechanismů obou systémů jsou zatím neznámé (např. [2]). Hlavní rozdíl mezi oběma systémy je v tom, že v případě epizodické paměti jsou informace vždy vázané k vlastní osobě a jejím prožitkům, zato v případě sémantické paměti považujeme zpracovávané údaje za neosobní – objektivní fakta o světě (např. “auto má 4 kola”) [9]. Jednodušší obdobou epizodické paměti jsou pravděpodobně vybaveny i některé zvířecí druhy [6].

Stejným účelům, kterým slouží u lidí, by EP mohla sloužit i u virtuálních společníků nebo postav z virtuálního vypravěčství [19] či u počítačem řízených postav z videoher. Tyto entity jsou souhrnně nazývány inteligentní virtuální agenti (IVA). Schopnost zapamatovat si minulé události a plausibilně zapomínat jejich detaily by mohla přispět ke zlepšení interakce IVA s člověkem tím, že zvýší věrohodnost chování IVA. Zároveň tyto mechanismy mohou být i inženýrskou nutností pro IVA běžící po dlouhé časové úseky v komplexních virtuálních světech.

Cílem tohoto článku je seznámit čtenáře s naším programem výzkumu EP. Ohniskem našeho zájmu je klastrování sekvencí akcí do smysluplných epizod a zároveň předvídání akcí IVA i lidí. Jak se ukáže, oba problémy jsou provázané. Hledání koherentních podposloupností akcí tvořících samostatné epizody umožní IVA lépe interpretovat chování agentů v jeho okolí. Anticipace jejich akcí mu pak může poskytnout kompetiční výhodu. Ve zbytku článku budeme podrobněji diskutovat oba problémy, námi navrhovanou metodiku evaluace modelů řešících tyto problémy a v neposlední řadě softwarové prostředí virtuálního světa v kterém budou IVA vtěleni.

V následující sekci rozebíráme současný stav poznání EP z perspektivy psychologie a neurověd. Sekce 3 představuje současné výpočetní modely EP. Sekce 4 diskutuje navrhované řešení problémů anticipace a klastrování epizod. Sekce 5 navrhuje metodologii validace modelů ze sekce 4. Sekce 6 popisuje virtuální prostředí, které hodláme použít pro naše experimenty.

2 Základní poznatky z neurovědy a psychologie

Faktické poznatky o episodické paměti přináší neurověda a psychologie. Neurobiologie například ukazuje na zásadní roli hippokampu v zapamatovávání si nových epizod i deklarativních faktů [20, 21, 7] a v navigaci a orientaci v prostoru obecně [15]. Neurobiologie i psychologie přináší poznatky o částečné disociaci (mimo jiné) episodické a sémantické paměti (např. [22, 2]), což má i klinický význam.

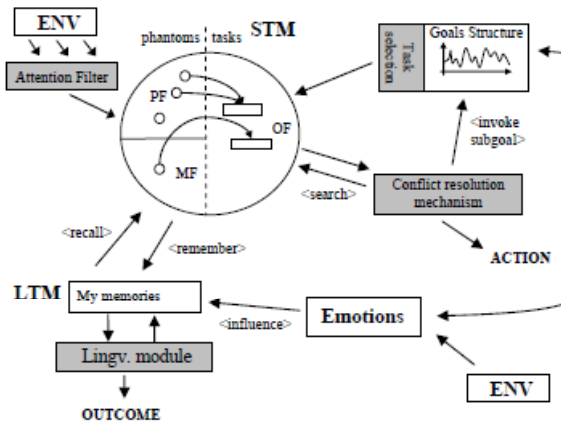
Zajímavé poznatky pro naši práci přináší ve své rešerši vnímání struktury událostí Zacks a Tversky [24]. Ti chápou EP jako aparát pro predikci budoucího stavu agenta a jeho okolí, zároveň definují jako místa předělu mezi dvěma epizodami ty chvíle, kdy predikce selháva. Kromě toho poukazují na analogii ve vnímání epizod a objektů. Epizody i objekty se mohou skládat z menších podcelků, které umíme samostatně identifikovat. Mají tedy hierarchickou strukturu. Objekty jsou vymezeny prostorem, který zabírají, zatímco epizody jsou vymezeny časem, ve kterém trvají. Tato analogie vede k myšlence inspirovat se při identifikaci epizod algoritmy pro klasifikaci objektů v obrazových datech. Předěly mezi epizodami jsou chápány jako místa, ve kterých selhává predikce dalšího stavu, tedy místa nesoucí informaci důležitou pro další vývoj světa.

3 Existující výpočetní modely EP

Implementace modelu EP může být základem pro tvorbu mnoha dalších funkčních celků IVA. Namátkou jmenujme vysvětlování chování ostatních agentů, zodpovídání dotazů týkajících se minulých událostí, navazování v přerušených činnostech, offline učení a další (podrobnější přehled přináší [14]).

Nejjednodušším výpočetním modelem EP je prostý log obsahující všechny události, které se ve světě staly, spolu s dodatečnými informacemi jako je například vnitřní stav agenta (jeho motivace, emoce atd.). Nevýhodou tohoto přístupu je paměťová náročnost rostoucí lineárně s časem simulace a z toho vyplývající vysoká náročnost vyhledávacích algoritmů. Tento přístup je pro oblast IVA nedostačující. IVA jsou často nasazováni v aplikacích, kde interagují s lidmi. Jejich řídící program proto běží v reálném čase. Z toho plynou vysoké nároky na rychlost odpovědi. Jedním z požadavků na EP je tedy tvorba vhodných indexů umožňujících toto vyhledávání. Oblastí, jejíž důležitost pravděpodobně ještě stoupne s příchodem komplexních virtuálních prostředí, ve kterých budou IVA simulovány nepřetržitě po delší časové intervaly (dny, týdny), je plausibilní zapomínání, které můžeme interpretovat jako ztrátovou formu komprese dat specifické povahy.

Ideálním stavem by bylo vytvoření generického modelu EP, který by měl předpoklady v sobě postihnout všechny výše zmíněné funkce EP. Někteří autoři dokonce pracují přímo na tomto cíli [22]. Ponecháme-li stranou otázku, zda může být takový celistvý model vůbec vytvořen (polemiku na toto téma vede např. [3]), najdeme mnoho modelů zabývajících se dílčími podproblémy jeho tvorby.



Obr 1. Architektura modelu Peškové: STM – krátkodobá paměť, LTM – dlouhodobé paměť, ENV – okolní prostředí, PF – percepční pole, MF – záznamy z dlouhodobé paměti, OF – motivace agenta. Obrázek převzat z [4].

Tyto modely jsou často vytvářené na míru určité aplikaci, bez ambicí na přenositelnost do jiných prostředí. Příkladem budiž EP pro výukového agenta Steva [18] nebo pro agenty z aplikace určené k prevenci šikany na školách FearNot! [1]. EP zde slouží pro potřeby zhodnocení jednotlivých fází tréninku.

Náš současný přístup se zaměřuje na vytváření modelů řešících jednotlivé podproblémy univerzální EP a jejich následnou vzájemnou integraci. Ústředním modelem pro tyto snahy je model Peškové [4], jehož architektura je na obr. 1. Ústřední komponentou tohoto modelu je krátkodobá paměť (STM), která sjednocuje tři rozdílné zdroje informací. Vstupují do ní aktuálně pozorované předměty, které prošly filtrem vnímání (oblast paměti pro ně vyhrazená se nazývá percepční pole (PF)), dále pak současné motivace agenta (OF) a informace o předmětech z dlouhodobé paměti projikující se do paměťového pole (MF). Pokud obsah krátkodobé paměti umožní splnění nějaké motivace, bude její otisk uložen do dlouhodobé paměti (LTM). Rychlost zapomínání obsahu LTM ovlivňuje emoční modul. Vzpomínky asociované se silnou emocí jsou zapomínány pomaleji, než ty emočně neutrální. Obsahem LTM jsou informace o předešlé aktivitě agenta. Záznam agentových činností v sobě navíc zohledňuje jejich hierarchickou strukturu. Model Peškové používá AND/OR stromy pro popis rozhodovacího systému agenta a zároveň jejich upravenou formu využívá i pro záznam vzpomínek o agentově činnosti. Vnitřní uzly stromu odpovídají kompozitním činnostem, listy pak atomickým akcím, které mohou být ještě parametrizovány svými zdroji (např. *zalij(KVĚTINA_24, KONEV_1)*). Omezením tohoto modelu je například strojově přesná, neplauzibilní reprezentace času nebo nehierarchická reprezentace míst. Tyto problémy se snaží řešit například Burkert [5], přidávající plausibilní reprezentaci času, nebo Korenko zavádějící hierarchické vnímání prostoru [12]. Dalším problémem modelu je, že uvažuje pouze historii samotného agenta a nevytváří vzpomínky o dalších IVA potažmo o lidmi ovládaných avatarech přítomných v prostředí. Přidání této funkcionality je hlavním cílem naší práce. Cestu k jeho dosažení ukazuje následující kapitola.

4 Sledování ostatních IVA – cíl výzkumu

V prostředích simulujících skutečný svět agent vidí pouze atomické akce prováděné dalšími IVA, nezná stav jejich vnitřní, nepozorovatelných proměnných a proto neví, jaká je motivace sledovaného chování a které akce patří kterým cílům. Cílem naší práce je z této omezené informace – sekvence atomických akcí – odvodit smysluplnou posloupnost akcí a zjistit jejich motivaci.

Ve výzkumu EP se chceme zaměřit především na:

1. odhalovaní předělů mezi epizodami
2. zjišťování motivací agentů

Experimenty, kde lidé měli za úkol určovat dějové předěly ve videosekvencích, ukázaly, že se na označení míst předělů víceméně shodnou [13]. Má tedy smysl hledat algoritmy, které by nám umožnily epizody jako celky identifikovat. Již zmiňovaná analogie mezi strukturou epizod a objektů nás vede k myšlence použití upravených algoritmů pro identifikaci objektů v obrazových datech. Samozřejmě v příznakovém prostoru rozšířeném o čas.

Otázky, které si klademe, jsou: a) zda bude pro úspěšnou klasifikaci epizod nutno použít metodu

předpokládající určitý model světa, nebo b) jestli půjde použít obecnější přístup pracující pouze na základě statistických charakteristik dat, např. samoorganizaci. Znalost předělů mezi epizodami přímo přispěje k řešení druhého cíle.

Druhou oblastí je zjišťování motivace jednání IVA i lidí ovládajících avatary na základě pozorování jejich akcí. To nám umožní:

1. automaticky anotovat sekvence akcí jejich odhadovanou motivací a tím lépe vysvětlit chování ostatních IVA
2. odhadovat budoucí akce na základě předpokládané motivace a již pozorovaných akcí, tato znalost může být pro IVA kompetiční výhodou

Všimněme si, že předpovídání budoucích akcí je klíčové pro určování předělů mezi epizodami a naopak. Předěly můžeme definovat jako místa, která neodpovídala našemu očekávání [24], kde selhala predikce. Naopak pokud budeme mít sekvence akcí rozdělené na funkčně koherentní celky, bude snazší odvodit jejich účel, protože úsudek nebudou ovlivňovat akce z minulých epizod, které už aktuální chování agenta pravděpodobně tak silně neovlivňují. Jak je vidět, obě úlohy jsou navzájem provázané, vyřešení jedné napomůže řešení druhé.

Při určování motivací už model světa zcela jistě zapotřebí bude. My navrhujeme za model vzít AND/OR stromy dostupné agentovu rozhodovacímu systému. To znamená, že agent bude interpretovat chování ostatních IVA skrze svou vlastní znalost prostředí. Úlohu můžeme obecně formulovat jako zjišťování podmíněné pravděpodobnosti, že pozorovaná sekvence akcí je výsledkem exekuce určitého AND/OR stromu. Hlavním úkolem je nalézt kódování sekvence akcí maximálně usnadňující hledání této pravděpodobnosti. Otázek, kterými se chceme zabývat, je několik. Zaprvé nás zajímá, zda je nutné uvažovat tempo – časové rozestupy v provádění jednotlivých akcí. Druhou otázkou je, zda lze pro potřeby rozpoznávání motivací sekvenci akcí v nějakém smyslu normalizovat, například uvažovat pouze relativní časové uspořádání. Nebo, zda lze dokonce přijmout zjednodušení až sekvence na úroveň množiny akcí.

Pokud už IVA bude znát nejpravděpodobnější strom určující právě agentovo chování, může jej použít pro mentální simulaci – přehrání všech akcí určených tímto stromem až do pozorovaného stavu a následně předpovědět další nejpravděpodobnější akci. Alternativní přístup je použít znalost aktuálního stromu jako abstrakci shrnující minulý stav systému a zjišťovat další nejpravděpodobnější akci jako výsledek stochastického systému daného dvěma stavy – identifikátorem nejpravděpodobnějšího AND/OR stromu a poslední provedenou akcí. Představme si to na následujícím příkladu. Jakou akci chce asi provést pozorovaný agent, který se právě blíží k zařízení rychlého občerstvení a jeho pravděpodobná motivace je *HLAD?* Nejspíše to bude jiná akce než když bude agentovou motivací *POULIČNÍ_PROTEST*.

5 Validace modelu

Výsledný model je třeba validovat. Validace našeho modelu může probíhat ve dvou krocích. V první fázi navrhujeme testovat model na posuzování akcí dalších IVA. Důvodem je zejména technická přístupnosti tohoto řešení. Ve druhé fázi by pak bylo dobré použít data získaná z interakce lidí ve virtuálních prostředích. Očekáváme, že se nám alespoň v některých případech podaří odhadovat cíle sledované IVA resp. lidmi.

Primárním účelem IVA je imitovat lidské chování. Kritériem „správnosti“ této imitace je, že se chování bude blížit tomu, jak by se v daných situacích chovali skuteční lidé. Proto mohou experimenty s IVA sloužit jako předstupeň experimentů na datech pocházejících od lidí. Výhodou IVA je, že nám přímo poskytují motivace svého chování. Tím máme všechna data nutná pro validaci našeho modelu. Podobnou metodologii použil při testování modelu tvorby časových konceptů Burkert [5]. Slabinou tohoto přístupu je umělá povaha testovaných dat - data závisí na programu pozorovaných IVA a není zaručeno, že člověk by se choval obdobně. Výhodou je možnost spouštění experimentů v rozdílných parametrizacích prostředí a tím systematicky zkoumat vlastnosti našeho modelu. Domníváme se proto, že v první fázi vývoje modelu EP má tato metodologie i přes zmíněnou nevýhodu své opodstatnění.

Druhým krokem je otestovat vlastnosti našeho modelu na datech zachycujících chování skutečných lidí. Sběr takovýchto dat v reálném světě je netriviální záležitostí vyžadující mnoho času a prostředků. Naopak ve virtuálním prostředí je řada problémů s ním spojených vyřešena. S minimálními náklady lze zaznamenávat akce vykonané avatary, které ovládají lidé. První takový zatím neveřejný korpus vzniká pro doménu restauračního stravování [16]. Nevýhodou takovýchto dat je samozřejmě nepřítomnost motivací v logu akcí, tato informace musí být dodána zdlouhavou ruční anotací.

6 Virtuální prostředí

Pro validaci modelu je zapotředí použit dostatečně komplexní simulátor imitující reálný svět. Vhodným kandidátem může být upravené prostředí komerční hry Unreal Tournament 2004 [8] (UT2004), které se nám již osvědčilo v minulosti například při evoluci chování IVA [11].

Pro vytváření řídící logiky IVA vtělených v prostředí UT2004 lze použít softwarového prostředí Pogamut [10]. Rozšíření Pogamut GRID [17] pak umožňuje paralelní spouštění experimentů na clusteru počítačů a tím výrazně urychluje statistickou validaci modelů. Komplexnost prostředí ukazuje obr. 2 zobrazující pohled na civilní městečko simulované v UT2004 z ptačí perspektivy. Vnitřní prstenec budov má vymodelované i interiéry, vnější budovy mají pouze fasádu.

Symbolická reprezentace vjemů zprostředkovávaná platformou Pogamut poskytuje vzhledem k východiskům našeho modelu vhodnou úroveň abstrakce pro další zpracování. IVA pracuje přímo s vjemy typu „agent A snědl jablko J“ a nemusí tudíž tyto koncepty odvozovat z jejich subsymbolických reprezentací, jak je tomu například při zpracování videa.

Obr. 2. Ukázka virtuálního prostředí ve hře Unreal Tournament 2004, upravená verze mapy DM-UnrealVille.

Naším cílem je vytvořit za pomocí platformy Pogamut inteligentního virtuálního agenta řízeného AND/OR stromy a vybaveného diskutovaným modelem EP. V tomto navazujeme na Burkerta [5], který již v platformě Pogamut IVA řízeného AND/OR stromy implementoval.

7 Závěr

Cílem tohoto článku bylo přiblížit čtenáři problematiku EP a představit náš konkrétní výzkumný program, který hodláme v dalších letech realizovat. Cílem našeho snažení je vytvořit model schopný na základě pozorování akcí odhadovat motivace dalších aktérů v prostředí, předpovídat jejich budoucí akce a klastrovat jejich počínání do smysluplných celků. V článku jsme diskutovali, jak by bylo možné těchto cílů dosáhnout, navrhli metodiku validace zkoumaných modelů a představili konkrétní softwarové prostředí, ve kterém budou modely implementovány.

Poděkování

Tato práce byla podpořena grantem GAČR 201/09/H057, grantem GAUK č. 21809 a částečně také projektem „Information Society“ 1ET100300517 a výzkumným záměrem MŠMT MSM0021620838.

Reference

1. R.S. Aylett, S. Louchart, J. Dias, A. Paiva, M. Vala: *FearNot! – an experiment in emergent narrative.* In: Proceedings of Intelligent Virtual Agents, LNAI 3661, Springer, 2005, 305–316.
2. A. Baddley, (eds.): *Episodic Memory: New Directions in Research*. Oxford University Press, 2001.
3. C. Brom, J. Lukavsky: *Towards virtual characters with a full episodic memory II: the episodic memory strikes back.* In: Proc. Empathic Agents, AAMAS Workshop, 2009, 1-9.
4. C. Brom, K. Pešková, J. Lukavský: *What does your actor remember? Towards characters with a full episodic memory*. Proc. ICVS, LNCS 4871, Springer, 2007, 89-101.
5. O. Burkert: *Connectionist model of episodic memory for virtual humans*. Msc Thesis, Charles University in Prague, 2009.
6. N.S. Clayton, A. Dickinson: *Episodic-like memory during cache recovery by scrub jaws*. Nature 395, 1998, 272-274.
7. H. Eichenbaum: *Hippocampus: cognitive processes and neural representations that underlie declarative memory.* In: Neuron, 44, 2004, 109-120.
8. Epic Games: Unreal Tournament 2004: http://www.unrealtournament.com, [26. 5. 2009]
9. M.W. Eysenck: *Cognitive Psychology: a Student's handbook.* Psychology Press, New York, 2005.
10. J. Gemrot, R. Kadlec, M. Bida, O. Burkert, R. Pibil, J. Havlicek, L. Zemcak, J. Simlovic, R. Vansa, M. Stolba, C. Brom: *Pogamut 3 can assist developers in building AI for their videogame agents.* In: Proc. Agents for Games and Simulations, AAMAS workshop, 2009, 144-148.
11. J. Gemrot, R Kadlec, P. Vidnerová, C. Brom: *Evoluce chování virtuálních postav v 3D hrách*. In: Proceedings of ITAT, 2008, Slovak Republic.
12. T. Korenko, C. Brom, J. Lukavský: Tech rep. 5/2008, KSVI, MFF UK .
13. D. Newtson: *Attribution and the unit of perception of ongoing behavior*. Journal of Personality and Social Psychology, 2, 8, 1973, 28-38.
14. A. Nuxoll: *Enhancing intelligent agents with episodic memory*. Ph.D. diss. The University of Michigan, 2007.
15. J. O'Keefe, L. Nadel: *Hippocampus as a Cognitive Map.* Oxford, UK: Claredon Press, 1978.
16. Orkin & Roy: *Automatic learning and generation of social behavior from collective human gameplay.* Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2009.
17. Pogamut GRID: http://artemis.ms.mff.cuni.cz/pogamut/, Sub-projects. [26. 5. 2009]
18. J. Rickel, W.L. Johnson: *Animated agents for procedural training in virtual reality: perception, cognition, and motor control.* In: App. Artificial Intelligence, 13, 1999.
19. K. Ryokai, C. Vaucelle, J. Cassell: *Virtual peers as partners in storytellingand literacy learning*. In: Journal of Computer Assisted Learning, 19, 2, 2003, 195 – 208.
20. W.B. Scoville, B. Milner: *Loss of recent memory after bilateral hippocampal lesions*. J. Neurol. Neurosurg. Psychiatry, 20, 1957, 11-21.
21. L.R. Squire: *Memory and the hippocampus: a synthesis from findings with rats, monkeys, and humans*. In: Psychol Review 99, 1992, 195-231.
22. G.D. Tecuci: *A generic memory module for events.* Ph.D. Dissertation, The University of Texas at Austin, 2007.
23. E. Tulving: *Elements of Episodic Memory.* Oxford: Claredon Press, 1983.
24. J.M. Zacks, B. Tversky: *Event structure in perception and conception*. Psychological Bulletin, 127, 2001, 3-21.

Experimentálne overenie didaktickej účinnosti a časovej náročnosti adaptívnych hypermediálnych systémov

Jozef Kapusta and Michal Munk

Katedra informatiky, Fakulta prírodných vied, Univerzita Konštantína Filozofa v Nitre
Tr. A. Hlinku 1, 949 74 Nitra, Slovensko

Abstrakt *Problematika adaptácie vzdelávacieho prostredia s využitím adaptívnych hypermediálnych systémov /AHS/ v sebe zahŕňa nielen potrebu implementácie týchto systémov, návrh vhodných štruktúr pre riešenie problémov adaptivity, ale aj zhodnotenie e-learningu, pedagogicko-psychologické aspekty tvorby materiálov pre podporu výučby, osnovanie učiva, efektivitu podania problematiky a pod. Autori článku na základe poznatkov a vedomostí z uvedenej oblasti uskutočnili experiment zameraný na kvantitatívne vyhodnotenie výsledkov možností aplikácie AHS v informatických predmetoch na Katedre informatiky FPV UKF Nitra.*
Získané výsledky z experimentu overili didaktickú účinnosť e-learningových kurzov, zostavených pomocou adaptívnych hypermédiálnych systémov, časovú efektívnosť týchto systémov, ako aj výber najvýhodnejšej formy adaptácie. V experimente sa porovnávala technika adaptívnej anotácie odkazov a technika priameho adaptívneho vedenia. Dôležitým zistením, vyplývajúcim z výsledkov získaných z uskutočneného experimentu bolo, že technika priameho vedenia bola v porovnaní s ďalšími technikami najmenej časovo efektívna, avšak jej didaktická účinnosť bola najvyššia.

1 Úvod

S čoraz väčším prienikom internetu do oblasti medziľudskej komunikácie sa v niektorých oblastiach začal prejavovať jeho všeobecný charakter. Hypermediálnym materiálom čoraz častejšie chýba personalizácia, priblíženie sa a prispôsobenie zobrazovaných informácií individuálnym potrebám používateľa. Tento nedostatok sa pokúša v súčasnosti odstrániť aplikovanie adaptívnych hypermediálnych systémov (ďalej AHS) do hypertextových dokumentov. Tento špecifický typ aplikácií kombinuje hypermédia, techniky modelovania používateľov a určitú formu umelej inteligencie, ktorá prispôsobuje formu, ale tiež aj obsah hypermediálnych dokumentov na mieru každému používateľovi.

Väčšina súčasných projektov AHS je zameraná hlavne na výučbu a prezentáciu informácií vo vzdelávaní. Problematika AHS vo vzdelávaní v sebe zahŕňa nielen ich technickú časť, t.j. potrebu vyriešenia a implementácie AHS, návrh adaptivity a štruktúry AHS, ale aj didaktickú, ktorá hľadá nové možnosti v podpore výučby s IKT metódou e-learningu a pedagogicko-psychologickú súvisiacu s tvorbou materiálov pre podporu výučby, hľadaním vhodnej osnovy učiva, efektívneho podania študovanej problematiky a pod.

Prispôsobovanie v AHS je založené na vedomostiach o obsahu jednotlivých výučbových stránok, väzbách medzi nimi a predpokladoch o vedomostiach, preferenciách a ďalších charakteristikách študenta [1]. Podstatnou činnosťou pri tvorbe AHS je práve vytvorenie obsahu a tiež získanie a reprezentácia vedomostí o prispôsobovaní. V súčasnosti sa využívajú najmä prístupy založené na explicitnej reprezentácii vedomostí prostredníctvom pravidiel [2].

2 Adaptívna podpora navigácie

AHS sú založené na dvoch skupinách techník prispôsobovania: adaptívna prezentácia a adaptívna podpora navigácie. Hlavným spôsobom adaptívnej prezentácie je adaptívna prezentácia textu. Do adaptívnej prezentácie preto patria techniky alternatívneho zobrazovania fragmentov stránok alebo celých stránok, alternatívne zobrazenie obrázkov, alternatívny text, rozbaľovací text a iné [3]. Uskutočnený experiment sa venuje dvom technikám adaptívnej podpory navigácie, z toho dôvodu sa budeme v ďalšom texte venovať hlavne týmto technikám.

Adaptívna podpora navigácie spočíva v ovplyvňovaní cesty používateľa v informačnom priestore. Pri tejto technike vyhodnocuje adaptačné jadro systému vhodnosť každého zobrazovaného odkazu pre používateľa a predkladá mu výsledok, na základe ktorého ovplyvňuje cestu používateľa v systéme dokumentov. Toto ovplyvňovanie môže byť direktívne v tom zmysle, že systém znemožní cesty, ktoré nie sú pre používateľa v danom kontexte „vhodné“ alebo nedirektívne, kedy rôznymi prostriedkami používateľského rozhrania systém prezentuje používateľovi odporúčané (alebo neodporúčané) cesty v informačnom priestore. Pri nedirektívnom prístupe systém odkazy iba usporiada podľa dôležitosti, resp. inak odlíši dôležitý odkaz [1].

Na realizáciu uvedených metód navigácie v informačnom obsahu, či už pri direktívnom alebo nedirektívnom prístupe, sa používajú najmä tieto techniky:

- **priame vedenie**: AHS vedie používateľa v informačnom priestore, t.j. vyberá najvhodnejšie koncepty a fragmenty im priradené. Realizuje sa pomocou typického tlačidla „Ďalej“,
- **usporiadanie odkazov**: odkazy na ďalšie stránky sa hierarchicky usporiadajú podľa vhodnosti,
- **anotácia odkazov**: adaptívny systém označuje odkazy „vhodné“ pre používateľa [4],
- **skrývanie odkazov**: odkazy, ktoré vedú k neodporúčaným informáciám sa skryjú. Skrývanie možno realizovať niekoľkými formami: odkaz sa nezobrazí (zobrazí sa iba text odkazu), odkaz sa blokuje (spôsob prezentácie závisí od kombinácie nezobrazenia odkazu a anotácie odkazu), alebo odkaz sa zruší z prezentácie,
- **generovanie odkazov**: AHS dynamicky generuje nové odkazy (napr. objavuje súvislosti medzi jednotlivými konceptmi),
- **adaptácia máp**: AHS na základe modelu používateľa a/alebo modelu prostredia dynamicky vytvára mapu domény (grafická prezentácia navigácie) [5][6] [7].

3 Realizované experimenty s podobným zameraním

Pri analýze článkov z dostupných digitálnych knižníc sme našli niekoľko experimentov zameraných na využitie AHS vo vzdelávaní a na porovnanie adaptívnych techník. Medzi prvými bol experiment, ktorý realizoval M. Specht v roku 1998 na Institute for Applied Information Technology v Sankt Augustin v Nemecku [8]. Experiment bol realizovaný na vzorke 85 študentov, ktorí boli rozdelení do nasledovných skupín:

- bez adaptácie,
- adaptívna anotácia liniek (technika semafóru) – statickým linkám bol počas práce so systémom priradený, pomocou techniky semafor, príznak pre odporúčanie (zelený kruh) resp. neodporúčanie (červený kruh) odkazu,
- skrývanie liniek – linky neboli zobrazené, postupnou prácou sa odkrývali,
- skrývanie a adaptívna anotácia liniek – linky sa postupnou prácou odkrývali a zároveň sa technikou semaforu zobrazovala ich vhodnosť pre študenta.

Zaujímavosťou je, že sám M. Specht priznáva, že boli realizované dva experimenty, pričom jeden sa skončil neúspechom, pretože viacero študentov nedokončilo záverečný test. V úspešne realizovanom experimente sa kombinácia adaptívnej anotácie a skrývania liniek javila ako najlepšia kombinácia pre získavanie nových vedomostí, napriek tomu, že nebol zistený štatisticky významný rozdiel medzi touto technikou a ostatnými adaptívnymi technikami použitými v experimente. Autor dospel k záveru, že kombinácia viacerých foriem adaptácie dokáže študentom vytvoriť dobré a hlavne motivačné študijné prostredie, ktoré ich núti k štúdiu. Ako najzaujímavejšie techniky študenti označili adaptívne odporúčanie liniek a kombináciu odporúčania liniek a skrývania liniek.

Podobné výsledky mal i experiment, ktorý bol realizovaný v roku 1998. K. Höök a M. Svensson sa v ňom zamerali nielen na porovnanie rozdielov medzi výučbou s podporou AHS a výučbou bez adaptívnej podpory, ale aj na možnosti adaptivity výučby v prvom ročníku štúdia v porovnaní s ostatnými ročníkmi [9]. Výsledky experimentu ukázali významný vzťah medzi predchádzajúcimi vedomosťami študentov a adaptívnou anotáciou študijných materiálov. Výskumom sa zistilo, že študenti s dobrými vstupnými vedomosťami z predkladanej študijnej problematiky skôr preferovali adaptívnu techniku odporúčania odkazov, oproti študentom so slabšími vedomosťami, ktorí skôr preferovali techniku priameho vedenia.

Širšie porovnanie priniesol experiment realizovaný na University of Sydney. V ňom J. Eklund a K. Sinclair [10] sledovali rozdiely vo výsledkoch študentov, ktorí študovali rozdelení do nasledovných skupín podľa podpory adaptívnych techník:

- bez adaptácie,
- adaptívna anotácia liniek (technika semaforu),
- blokovanie liniek – linky boli zobrazené, ale nebola možnosť ich aktivovať,
- skrývanie liniek – linky neboli zobrazené, postupnou prácou sa odkrývali.

Tento experiment, zameraný na zisťovanie úspešnosti rôznych foriem adaptívnej anotácie vo výučbe, bol realizovaný pomocou systému Interbook. Výskum potvrdil štatisticky významný rozdiel v úrovni vedomostí u študentov študujúcich pomocou všetkých troch adaptívnych techník v porovnaní so študentmi bez adaptívnej podpory. Medzi skupinami s podporou AHS však nebol zistený štatisticky významný rozdiel.

4 Metodika experimentu

Experiment je zameraný na oblasť vysokoškolského vzdelávania. Potrebuje študent vysokoškolského štúdia pri vzdelávaní personalizáciu obsahu a výklad učiva a je vôbec vhodné individualizovanie učiva na vysokej škole? Podľa nášho názoru je otázne, či je efektívnejšie prekladať študentom vysokoškolského štúdia učivo pomocou AHS alebo „klasickou“ e-learningovou formou. Tiež v prípade AHS je potrebné nájsť najefektívnejšiu techniku adaptácie.

V našom experimente sme sa zamerali na tematickú oblasť „Programovanie internetových aplikácií“ v kombinovanej forme bakalárskeho štúdia v odbore Aplikovaná informatika.

Stanovili sme si nasledujúce výskumné otázky:

- Sú e-learningové kurzy, zostavené pomocou adaptívnych hypermédiálnych systémov, didakticky účinné?
- Sú e-learningové kurzy, zostavené pomocou adaptívnych hypermédiálnych systémov efektívne z hľadiska času potrebného na prebranie daného učiva?
- Ktorá adaptívna technika je pre študentov najvhodnejšia?

Pre overenie uvedených otázok sme stanovili nasledovnú postupnosť krokov riešenia:

1 Vytvorenie kontrolných a experimentálnych skupín.

2 Vytvorenie meracích procedúr a analýza spoľahlivosti:
   - odhad reliability použitých didaktických testov,
   - identifikácia podozrivých úloh.

3 Realizácia experimentálneho plánu:
   - realizácia pretestu v skúmaných skupinách,
   - podrobenie experimentálnych skupín intervencii,
   - realizácia posttestu v skúmaných skupinách.

4 Porozumenie údajom:
   - výpočet popisných charakteristík a intervalov spoľahlivosti,
   - vizualizácia štatistík,
   - stanovenie nulových štatistických hypotéz.

5 Overovanie validity použitých štatistických metód:
   - overenie predpokladov použitia analýzy rozptylu a kovariancie.

6 Analýza údajov a interpretácia výsledkov:
   - analýza rozptylu a kovariancie.

5 Použité technológie

Pre experiment bolo zvolené prostredie LMS Moodle. Tento manažovací vzdelávací systém bol vybraný nielen z dôvodu jeho implemtácie na našej univerzite ako univerzitného systému pre e-learning a elektronickú podporu štúdia, ale aj širokého používania v akademickej

oblasti pri riadení výučby. Okrem viacerých dostupných aktivít v systéme, ktoré sme použili pre tvorbu e-kurzu chceme vyzdvihnúť modul Prednáška, ktorý sme použili na tvorbu lekcie pre priame vedenie študentov a modul iLMS, ktorý sme pre potreby experimentu implementovali do LMS Moodle. Tento modul bol použitý ako adaptívny systém pre odporúčanie odkazov.

5.1 Modul iLMS – modul pre odporúčanie odkazov

Pre odporúčanie odkazov bol použitý iLMS modul, ktorý umožňuje na základe metadát a definovania závislostí odporúčať používateľovi odkazy. Tieto odkazy odporúča pomocou štyroch značiek: značka pre odporúčaný odkaz, značka pre „neutrálny“ odkaz, značka odkazu pre odporúčanie resp. neodporúčanie tohto odkazu sa systém nevedel na základe metadát rozhodnúť a značka pre odkaz, ktorý systém neodporúča študentovi. Modul iLMS je doplnok do systému Moodle. Z technickej stránky modul obsahuje nový adaptívny formát kurzu (formát dopĺňa tradičné formáty kurzov: tématický a týždňový) a niekoľko blokov pre tvorbu adaptívneho obsahu v LMS Moodle.

Obr.1. Vzhľad obsahu kurzu v systéme pre odporúčanie odkazov.

Modul vyvinul Gert Sauerstein ako súčasť svojej diplomovej práce „KI-Ansätze zur Lerner-Adaption in Lern-Management-Systemen“ na Technische Universität Ilmenau (Ilmenau, Germany). Ako autor vo svojej práci tvrdí, modul bol vyvíjaný na základe myšlienok P. Brusilovskeho. Modul po každom ponúknutom odkaze od študenta požaduje ohodnotenie vhodnosti a obsahovej relevantnosti daného odkazu. Zaujímavosťou tohto modulu je, že zahŕňa do adaptačného mechanizmu náladu, resp. aktuálnu motiváciu študenta. Tento atribút získa pomocou bloku v systéme Moodle, v ktorom môže študent vyznačiť svoju aktuálnu náladu.

Nevýhodou modulu iLMS je, že tento modul sa momentálne nevyvíja, a tiež, že sa dá implementovať iba do LMS Moodle 1.9+. Žiaľ asi najväčším problémov modulu je, že je iba prototypom určeným iba pre „jednoduché“ kurzy.

5.2 Aktivita Prednáška – modul pre priame vedenie študenta

Prednáška je typ aktivity, ktorá zaujímavou a flexibilnou formou sprístupní študentom vzdelávací materiál. Pozostáva z viacerých stránok textu, tzv. kariet, ktoré môže byť doplnené napríklad o obrázky alebo hypertextové odkazy. Každá karta je ukončená otázkou a študent má k dispozícii niekoľko možných odpovedí. Ak študent odpovedal na kontrolnú otázku správne, môže postúpiť v štúdiu na ďalšiu stránku (kartu). Ak odpovedal nesprávne, je vrátený na predchádzajúcu stránku, aby si problematiku doštudoval. Spôsob navigácie v rámci „Prednášky“ závisí od nastavenia parametrov tejto aktivity.

Prednáška je univerzálnym modulom, ktorý si tvorca kurzu môže plne prispôsobiť svojim predstavám. Môže sa rozhodnúť pre lineárny prechod jednotlivými stránkami prednášky, pričom každú ukončí kontrolnou otázkou. Efektívnejšie, ale zároveň náročnejšie je, ak sa tvorca kurzu rozhodne rozdeliť jednotlivé stránky Prednášky do viacerých paralelných vetiev a prinúti tak študenta k podrobnému skúmaniu popisovanej problematiky. Vetvenie sa realizuje práve kontrolnými otázkami na konci každej stránky. Po správnej odpovedi pokračuje napríklad študent na ďalšej stránke, ale ak odpovie nesprávne, môže byť presmerovaný na ľubovoľnú inú stránku Prednášky. Učiteľ má možnosť jednotlivé odpovede hodnotiť, prípadne komentovať. Vďaka týmto možnostiam patrí Prednáška z hľadiska jej vytvorenia k najnáročnejším modulom, ktoré LMS Moodle obsahuje [11].

6 Použité metódy overenia výsledkov experimentu

6.1 Analýza rozptylu

Analýza rozptylu jednoduchého triedenia (1-way ANOVA) je najjednoduchšou formou analýzy rozptylu, ktorá skúma závislosť kvantitatívnej premennej od kvalitatívnej (faktora). Cieľom analýzy rozptylu je odhaliť, či rozdiely priemerov jednotlivých skupín (podľa úrovne faktora) sú štatisticky významné (premenná závisí od faktora) alebo iba náhodné (premenná nezávisí od faktora). Významná hodnota F štatistiky vedie k zamietnutiu nulovej hypotézy o rovnosti priemerov, a tým k prijatiu alternatívnej hypotézy, ktorá tvrdí, že priemer aspoň jednej skupiny je rôzny od ostatných.

Model jednofaktorovej analýzy rozptylu:

$$Y_{ij} = \mu + \alpha_i + e_{ij}, \qquad (1)$$

kde μ je priemer, α_i je príspevok (efekt) *i*-tej úrovne faktora *A* a e_{ij} je rezíduum. Parametre μ, α_i, i = 1,2,...,I sú neznáme a e_{ij}, i = 1,2,...,I, j = 1,2,...,n_i sú nezávislé veličiny s rozdelením $N(0, \sigma^2)$.
Zaujíma nás jedna nulová hypotéza:

$$H0_A: \alpha_1 = ... = \alpha_I = 0 \text{ (všetky efekty sú nulové)}$$

Rovnocenný model jednofaktorovej analýzy rozptylu:

$$Y_{ij} = \mu_i + e_{ij}, \qquad (2)$$

kde μ_i je priemer v *i*-tej úrovne faktora A.

Zaujíma nás jedna nulová hypotéza:

$H0_A$: $\mu_1 = ... = \mu_I$ (priemery skupín sú rovnaké)

Obidva modely a hypotézy sú úplne rovnocenné a vedú k rovnakým vzorcom .

Multivariačná/viacrozmerná analýza rozptylu (MANOVA) je rozšírenie analýzy rozptylu na dve a viac závislých premenných *Y*. MANOVA porovnáva vektory priemerov (Y_1, Y_2,..) skupín podľa úrovne faktora. MANOVA na rozdiel od ANOVY nepoužíva jedinú testovaciu štatistiku, ale štyri.

6.2 Analýza kovariancie

Analýza kovariancie (ANCOVA) spája prvky analýzy rozptylu a viacrozmernej regresie. Do modelu analýzy rozptylu, ktorý skúma závislosť intenzívnej/kvantitatívnej premennej na nominálnych faktoroch sa pridá jeden alebo viac kvantitatívnych faktorov – kovariátov (kovariančných premenných).

Lineárny regresný model pre analýzu kovariancie v prípade dvoch skupín a jednej kovariančnej premennej *X*:

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 z_i + \beta_3 x_i z_i, \qquad (3)$$

kde *Z* označuje hodnotu indikátorovej premennej. Tá nadobúda hodnoty 0 alebo 1 podľa toho, či meranie patrí jednej alebo druhej skupine. Koeficient β_2 odráža pôsobenie nominálneho faktora. Koeficient β_1 hodnotí vplyv kovariančnej premennej. Koeficient β_3 zastupuje prítomnosť interakcie medzi kovariátom a nominálnym faktorom.

Rovnocenný model analýzy kovariancie s jedným nominálnym faktorom a jedným kovariátom:

$$Y_{ij} = \mu + \alpha_i + \beta(x_{ij} - \bar{x}) + e_{ij}, \qquad (4)$$

kde μ je priemer, α_i je príspevok i - tej úrovne faktora *A* a e_{ij} je rezíduum. Parametre μ, α_i, i = 1,2,...,*I* sú neznáme a e_{ij}, i = 1,2,...,*I*, j = 1,2,...,n_i sú nezávislé veličiny s rozdelením $N(0, \sigma^2)$.

Ak je vzťah medzi *X* a *Y* významný, potom model analýzy kovariancie vysvetľuje viac variability premennej *Y* ako model analýzy rozptylu. Analýza kovariancie testuje či upravené priemery skupín sú rozdielne. Priemery sú upravené tak, ako keby vo všetkých skupinách bola rovnaká (priemerná) hodnota intenzívneho/kvantitatívneho faktora.

Podobne ako sa ANOVA pridaním kvantitatívneho faktora rozšíri na ANCOVA, je možné rozšíriť aj MANOVA na MANCOVA.

7 Výsledky

Experiment bol realizovaný v letnom semestri 2007/2008 na Katedre informatiky. Experimentu sa zúčastnilo 75 študentov druhého ročníka bakalárskeho štúdia odboru Aplikovaná informatika.

7.1 Rozdelenie do skupín

Počas semestra sme v predmete „Programovanie internetových aplikácií“ sledovali štyri skupiny študentov, ktoré boli vytvorené štandardným zadelením do skupín. Študenti boli rozdelení na nasledovné skupiny:

1. Bez podpory LMS Moodle (*Unsupported*) – skupina, v ktorej výučba prebielaha klasickou F2F formou,
2. s podporou „štandardného“ e-kurzu (*Non-Adaptation*) - skupina, ktorá bola vedená pomocou blended learning v LMS Moodle,
3. s podporou modulu pre priame vedenie študenta (*Direct Guidance*) – skupina, ktorá študovala e-kurz s využitím modulu pre priame vedenie,
4. s podporou adaptívneho systému pre odporúčanie odkazov (*Links Annotation*) – skupina, v ktorej bol nasadený adaptívny systém iLMS.

Group	N	skupina	pretest	pôsobenie	posttest
Unsupported	21	kontrolná	✔	kontrola	✔
Non-Adaptation	17	kontrolná	✔	kontrola	✔
Direct Guidance	19	experimentálna	✔	ošetrenie	✔
Links Annotation	18	experimentálna	✔	ošetrenie	✔

Tab. 1. Experimentálny plán.

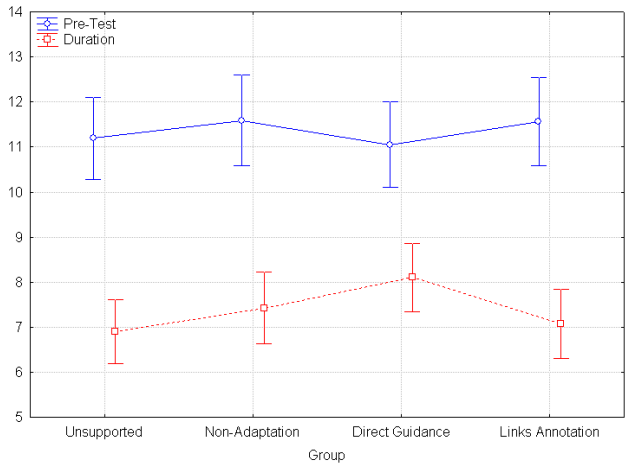
Pre takéto zatriedenie do skupín sme sa rozhodli z dôvodu, že zvolené skupiny majú už vytvorené svoje virtuálne triedy v rámci systému. Študenti bez podpory LMS a študujúci s podporou klasického e-kurzu tvoria kontrolné skupiny a študenti s podporou modulu priameho vedenia a adaptívneho systému pre odporúčanie odkazov tvoria experimentálne skupiny.

Kurzy vo všetkých troch skupinách mali jednotnú štruktúru aj obsah. Rozdiel v jednotlivých kurzoch bol v rozdielnej forme predkladania obsahu výučbového meteriálu. Učivo tematickej oblasti bolo rozdelené do malých častí, tak aby bola časť pri klasickom rozlíšení obrazovky vysoká na max. 1,5 zobraziteľnej plochy. Po každej časti nasledovala kontrolná otázka. Napriek tomu, že adaptívne systémy poskytujú možnosti pre vytvorenie oveľa zaujímavejšej štruktúry kurzu, takáto jednoduchá štruktúra kurzu bola použitá zámerne, aby sme „obohacovaním“ adaptívnych kurzov neskresľovali výsledky, resp. neznevýhodňovali skupinu s podporou e-kurzu bez adaptivity.

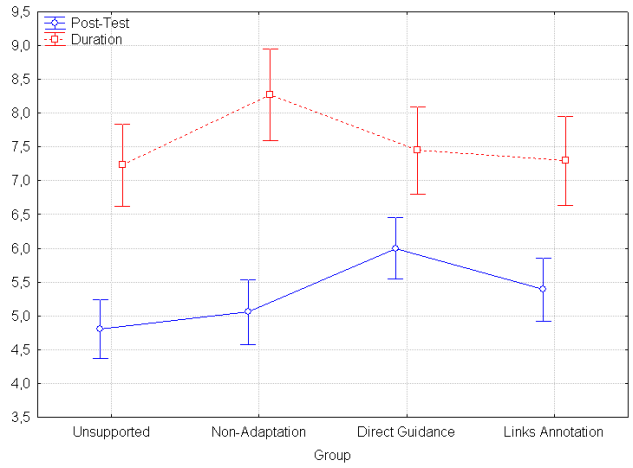
7.2 Analýza pretestu

Skupiny sa na prvý pohľad javia ako rovnocenné – všetky skupiny absolvovali rovnaké predmety spolu u rovnakých vyučujúcich. Rozdelenie skupín z pohľadu veku a pohlavia je tiež rovnocenné. Tento fakt sme štatisticky overili pomocou pretestu. Pretest bol realizovaný formou vstupného testu overujúceho základné vedomosti potrebné na zvládnutie preberanej problematiky. Vo vstupnom teste, ktorý pozostával z 15 otázok, sme sledovali skóre a čas, za ktorý jednotlivý študenti test vypracovali.

Nasledujúci graf vizualizuje výsledky MANOVA. Zobrazuje bodový a intervalový odhad priemeru skóre pretestu (*Pre-Test*) a času potrebnom na jeho vypracovanie (*Duration*).



Obr. 2. Graf priemeru a intervalu spoľahlivosti pre premenné Pre-Test a Duration.



Obr. 3. Graf priemeru a intervalu spoľahlivosti pre premenné Post-Test a Duration.

Na základe výsledkov MANOVA nezamietame nulovú hypotézu, tvrdiacu, že rozdiel v skóre pretestu a času potrebnom na jeho spracovanie medzi skupinami, nie je štatisticky významný, t.j. vektor závislých premenných (*Pre-Test, Duration*) nie je závislý na faktore *Group*. Čím sa nám potvrdil predpoklad o rovnocennosti skupín a nebolo potrebné pristupovať k randomizácii.

7.3 Analýza posttestu

Po ukončení štúdia sme vyhodnotili znalosti študentov pomocou záverečného testu. Záverečný test pozostával zo siedmych úloh. Tieto boli zamerané na kontrolu zvládnutia jednotlivých tematických oblastí predmetu.

	Test	Value	F	Effect df	Error df	p
Intercept	Wilks	0,01530	2252,214	2	70	0,000000
	Pillai's	0,98470	2252,214	2	70	0,000000
	Hotelling	64,34898	2252,214	2	70	0,000000
	Roy's	64,34898	2252,214	2	70	0,000000
Group	Wilks	0,75439	3,531	6	140	0,002762
	Pillai's	0,25999	3,536	6	142	0,002715
	Hotelling	0,30652	3,525	6	138	0,002817
	Roy's	0,21985	5,203	3	71	0,002637

Tab. 2. MANOVA, Multivariačné testy významnosti pre Post-Test.

Na základe výsledkov MANOVA (tab. 2) zamietame nulovú hypotézu s 99% spoľahlivosťou, t.j. vektor závislých premenných (*Post-Test, Duration*) je závislý na faktore *Group*.

	Degr. of Freedom	Post-Test SS	Post-Test MS	Post-Test F	Post-Test p
Intercept	1	2105,347	2105,347	2121,571	0,000000
Group	3	15,490	5,163	5,203	0,002637
Error	71	70,457	0,992		
Total	74	85,947			

Tab. 3 ANOVA, Univariačné výsledky pre Post-Test.

Z jednorozmerných výsledkov analýzy rozptylu (tab. 3) zamietame nulovú hypotézu s 99% spoľahlivosťou v prípade skóre posttestu, t. j. závislá premenná *Post-Test* je závislá na faktore *Group*. Naopak rozdiely v prípade času potrebnom na spracovanie posttestu sa nepreukázali. Nasledujúci graf vizualizuje výsledky ANOVA/MANOVA.

Po zamietnutí nulovej hypotézy v prípade posttestu, nás zaujíma, ktoré dvojice sa významne líšia.

	Group	{1} 4,8095	{2} 5,0588	{3} 6	{4} 5,3889
1	Unsupported		0,884979	0,002562	0,308703
2	Non-Adaptation	0,884979		0,036747	0,769176
3	Direct Guidance	0,002562	0,036747		0,263381
4	Links Annotation	0,308703	0,769176	0,263381	

Tab. 4 Tukeyov HSD test (pre nerovnakú početnosť) pre Post-Test

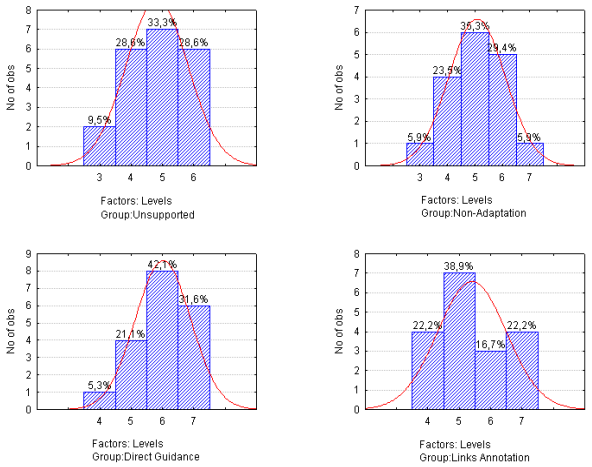
Štatisticky významné rozdiely sa preukázali medzi *Direct Guidance* a *Unsupported* ($p<0,01$) a medzi *Direct Guidance* a *Non-Adaptation* ($p<0,05$) v prospech *Direct Guidance*. Naopak zaujímavým zistením je, že medzi *Links Annotation* a ostanými úrovňami faktora *Group* neboli zistené štatisticky významné rozdiely.

Okrem štatisticky významných rozdielov je i na základe popisnej štatistiky (Obr. 3) vidieť, že najlepšie záverečný test zvládli skupiny, v ktorých boli použité AHS, t.j. skupiny *Direct Guidance* a *Links Annotation*. Z metód adaptácie to bola samozrejme skupina so systémom priameho vedenia používateľa, ktorej významné rozdiely sme preukázali. Tento fakt umocňuje aj skutočnosť, že práve táto skupina mala vo vstupnom teste (Obr. 2) výsledky najslabšie.

Aby sme nezniżovali silu štatistických testov overili sme predpoklady validity analýzy rozptylu. Konkrétne uvádzame diagnostiku predpokladov validity modelu analýzy rozptylu pre závislú premennú *Post-Test* vzhľadom na preukázané štatistické rozdiely.

Nasledujúci graf zobrazuje rozdelenie závislej premennej *Post-Test* v jednotlivých skupinách podľa úrovne faktora *Group*. Rozdelenie v jednotlivých skupinách je preložené očakávaným normálnym rozdelením. Z grafu sú zrejmé mierne odchýlky od normality. Z tohto dôvodu na overenie predpokladu rovnosti rozptylov použijeme neparametrický Leveneov test.

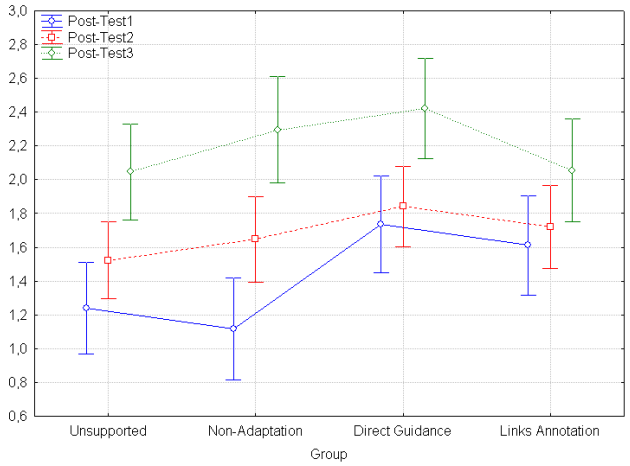
Test je štatisticky nevýznamný ($p>0,05$), t.j. hypotéza o rovnosti rozptylov sa nezamieta. Môžeme konštatovať, že k porušeniu predpokladu nedošlo. Pre úplnosť výsledkov uvádzame graf (Obr. 5), v ktorom sú znázornené výsledky záverečného testu pre jednotlivé tematické oblasti (*Post-Test1, Post-Test2, Post-Test3*).



Obr. 4 Kategorizovaný histogram pre Post-Test x Group.

	MS Effect	MS Error	F	p
Post-Test	0,261519	0,331973	0,787772	0,504699

Tab. 5 Leveneov test pre Post-Test.

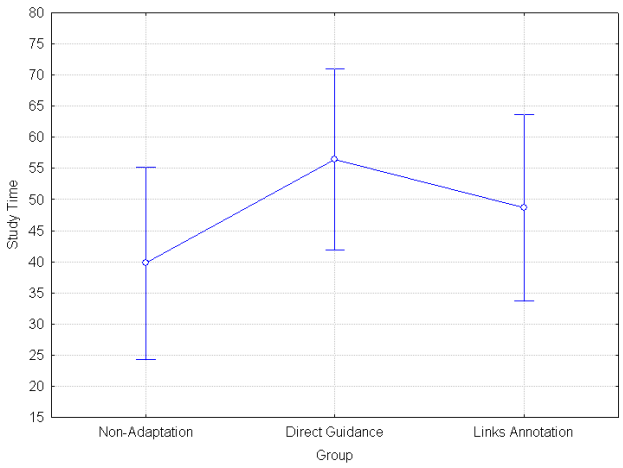


Obr. 5. Graf priemeru a intervalu spoľahlivosti pre parciálne výsledky premennej Post-Test.

Aj z parciálneho pohľadu na výsledky posttestu je vidieť lepšie skóre v prospech metódy priameho vedenia.

7.4 Analýza študijného času

Ďalšou oblasťou experimentu bolo zisťovanie času potrebného na preštudovanie danej problematiky. Systém Moodle, v ktorom boli adaptívne systémy implementované v sebe obsahuje mechanizmus pre vytváranie logovacieho súboru. V nasledujúcom grafe je zobraný prehľad koľko času venovali študenti v rámci jednotlivých skupín preštudovaniu danej problematiky. Z dostupných časových informácii boli vybrané iba hodnoty „čistého“ štúdia, t.j. v čase nie sú zahrnuté intervaly kedy sa používateľ prihlásil do systému, „náhodné“ prezeranie kurzu, resp. klikanie na aktivity ako fórum, slovník a pod. V hodnotách času sa odrážajú iba hodnoty kedy študent pracoval na jednotlivých lekciách zameraných na konkrétnu tematickú oblasť. Údaje v grafe sú v minútach. V grafe samozrejme chýba skupina bez podpory, v ktorej sme nemali mechanizmus na zisťovanie relevantných časových údajov.

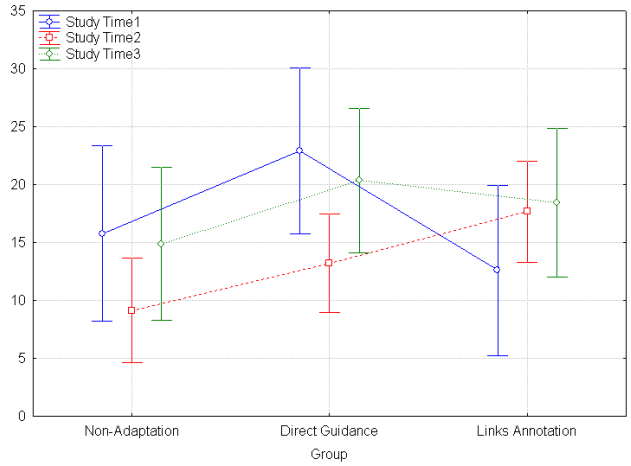


Obr. 6. Graf priemeru a intervalu spoľahlivosti pre premennú Study Time.

Na základe výsledkov ANOVA nezamietame nulovú hypotézu, tvrdiacu, že rozdiel v skóre študijného času medzi skupinami, nie je štatisticky významný, t.j. závislých premenná *Study Time* nie je závislá na faktore *Group*.

Napriek tomu, že sa nepreukázali štatisticky významné rozdiely, výsledky pri zakomponovaní času na štúdium sú prekvapivé. Môžeme konštatovať, že študenti strávili pri použití AHS viac času štúdiom ako pri klasickom e-kurze. Práve didakticky najúčinnejšia metóda adaptácie - priame vedenie (skupina *Direct Guidance*) bola z hľadiska času na štúdium najmenej efektívna. Vysvetľujeme si to tým, že kurz, ktorý vedie študenta pomocou priameho vedenia dokáže oveľa viac študenta zaujať a motivovať v ďalšom štúdiu.

Pre úplnosť výsledkov uvádzame graf (Obr. 7), v ktorom je znázornený čas potrebný na prebranie daného tematického celku (*Study Time1, Study Time2, Study Time3*) v rámci sledovaných skupín. Z týchto parciálnych údajov o čase štúdia vidíme ešte väčšie rozdiely v použití jednotlivých metód pri vedení študenta.



Obr. 7 Graf priemeru a intervalu spoľahlivosti pre parciálne výsledky premennej Study Time.

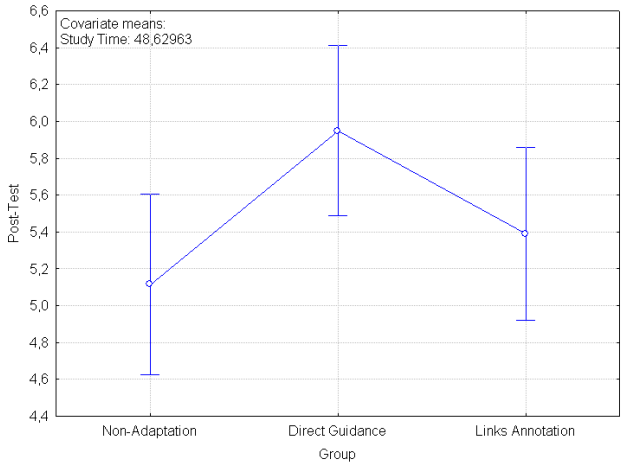
Na základe predchádzajúcich výsledkov sme sa rozhodli do modelu zahrnúť ďalší faktor – *Study time*. Nasledujúcou analýzou testujeme, či upravené priemery skupín sú rozdielne. Priemery sú upravené, tak ako keby vo všetkých skupinách bola rovnaká (priemerná) hodnota faktora *Study time*.

	SS	Degr. of Freedom	MS	F	p
Intercept	416,0901	1	416,0901	423,4300	0,000000
Study Time	2,0857	1	2,0857	2,1225	0,151404
Group	6,2916	2	3,1458	3,2013	0,049179
Error	49,1333	50	0,9827		

Tab. 6 ANCOVA, Univariačné testy významnosti pre Post-Test.

Z tabuľky (Tab. 6) vidíme, že vzťah medzi *Post-Test* a *Study Time* je štatisticky nevýznamný (p>0,05), t. j. dĺžka času potrebného na štúdium neovplyvnila výsledky posttestu. Zamietame nulovú hypotézu s 95% spoľahlivosťou, tvrdiacu, že rozdiel v skóre posttestu medzi skupinami, nie je štatisticky významný, t.j. závislá premenná *Post-Test* je závislá na faktore *Group*.

Nasledujúci graf vizualizuje výsledky ANCOVA.



Obr. 8 Graf priemeru a intervalu spoľahlivosti pre premennú Post-Test s kontrolou premennej Study Time.

Po zamietnutí nulovej hypotézy nás zaujíma, ktoré dvojice sa významne líšia.

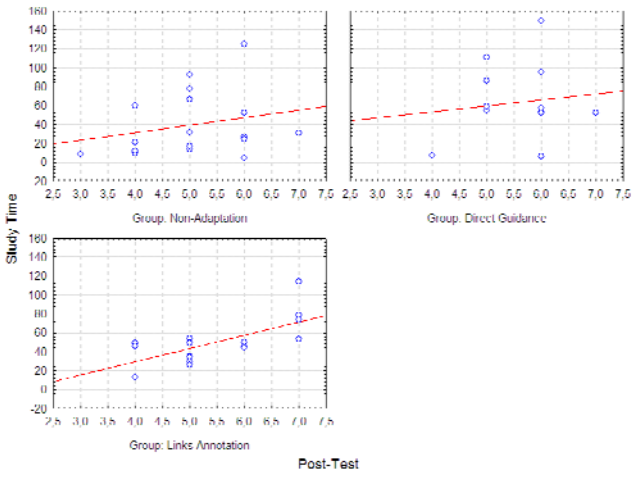
	Group	{1} 5,0588	{2} 6,0000	{3} 5,3889
1	Non-Adaptation		0,021271	0,598655
2	Direct Guidance	0,021271		0,164350
3	Links Annotation	0,598655	0,164350	

Tab. 7 Tukeyov HSD test (pre nerovnakú početnosť) pre Post-Test s kontrolou premennej Study Time

Štatisticky významné rozdiely sa preukázali medzi *Direct Guidance* a *Non-Adaptation* (p<0,05) v prospech *Direct Guidance*.

Obmedzujúcou podmienkou použitia analýzy kovariancie je rovnosť regresných koeficientov vo všetkých skupinách. Táto podmienka je veľmi často porušená, čo limituje použitie analýzy kovariancie. Významná interakcia kvantitatívneho a nominálneho faktora znamená porušenie tejto podmienky. Potom rôzne priemery skupín závisia od kvantitatívneho faktora – kovariančnej premennej. Najjednoduchším spôsobom ako overiť túto podmienku je skontrolovať kategorizovaný bodový graf (obr. 9), pre každú úroveň faktora.

V tomto prípade je podmienka splnená, vzhľadom na to, že priamky majú približne rovnaký sklon (obr. 9), t.j. regresný koeficient je približne rovnaký v obidvoch skupinách.



Obr. 9 Kategorizovaný bodový graf Post-Test x Study Time.

8 Diskusia

Aby sme neznižovali silu štatistických testov overili sme ich validitu a na získanie dát sme použili reliabilné a validné meracie procedúry. Proces merania je predpokladom získania dát. Aby meranie bolo kvalitné, musí byť meracia procedúra spoľahlivá. Cieľom úvodnej fázy realizovaného experimentu, ktorá nebola popísaná v tomto článku, bolo posúdenie reliability didaktických testov a identifikovanie podozrivých úloh. Pri skúmaní ďalších problémov, musíme zaručiť, že dokážeme odhadnúť vplyv kvality meracej procedúry na naše výsledky experimentu.

Výsledky analýzy spolahlivosti pretestu nám ukázali, že ani jedna z úloh výrazne neznižovala celkovú spolahlivosť testov, ktorá dosahovala v oboch testov takmer hodnotu 0,8 v prípade metódy rozdelenia testu na dve polovice. V prípade posttestu sme identifikovali podozrivé úlohy, ktoré znižovali celovú spolahlivosť testu. Táto identifikácia podozrivých úloh bola vykonaná na základe výsledkov posttestu u študentov študujúcich klasickou formou v zimnom semestri 2007/2008. Po prepracovaní spomínaných úloh sa výrazne zvýšila celková spolahlivosť testu (Cronbach's alpha: 0,627). Aplikáciou analýzy spoľahlivosti sme získali spoľahlivé experimentálne data, prostredníctvom ktorých sme mohli overiť naše výskumné otázky.

Na riešenie výskumného problému sme použili dve metódy, analýzu rozptylu a analýzu kovariancie, kde analýza rozptylu je jednoduchšia a nevyžaduje predpoklad o zhode regresie v jednotlivých skupinách. Na druhej strane je interpretácia menej validná, keď existujú medzi skupinami rozdiely v kontrolovanej premennej. Podobne ako v iných situáciách sa odporúča uskutočniť obidva spôsoby analýzy a porovnať ich výsledky. V našom prípade sú výsledky zhodné, máme dôvod ich považovať za robustné.

Predchádzajúcou analýzou uvedenou v kapitole 7 sme overili didaktickú účinnosť použitia adaptívnych hypermediálnych systémov vo vysokoškolskom vzdelávaní. Vychádzajúc z výsledkov analýzy sme preukázali, že e-learningové kurzy zostavené na základe poznatkov z AHS, sú didakticky účinné v predmetoch zameraných na programovanie. Pri analýze adaptívnych techník sme štatisticky preukázali, že nasadenie systému s priamym vedením malo pozitívny vplyv na výsledky záverečného testu u študentov študujúcich s touto podporou. Didaktická účinnosť e-kurzu s podporou techniky priameho vedenia sa potvrdila aj pri zohľadnení rozdielnej dĺžky času potrebného na štúdium jednotlivých tematických celkov. Pri technike adaptívneho odporúčania odkazov, napriek tomu, že sa nepotvrdil šta-

tisticky významný rozdiel, je vidieť na základe výsledkov popisnej štatistiky (obr. 3) lepšie výsledky v záverečnom teste. Z adaptívnych techník sa ako najvhodnejšia z hľadiska didaktickej účinnosti javí práve technika priameho vedenia.

Zaujímavé boli výsledky pri overovaní časovej efektivity e-kurzov. Pôvodne sme chceli experimentom dokázať, že použitie AHS je nielen didakticky účinné ale aj časovo efektívnejšie ako klasický e-kurz. Z výsledkov experimentu sme nepreukázali, že niektorá technika je pre študenta časovo náročnejšia, avšak z výsledkov popisnej štatistiky sme zistili rozdiely v čase v prospech neadaptívnych metód. Prekvapivo môžeme v našich podmienkach konštatovať, že študenti strávili pri použití AHS viac času štúdiom ako pri klasickom e-kurze. Časovo najnáročnejšia je práve technika priameho vedenia. Podľa nášho názoru to zapríčinila nielen zaujímavá forma predkladania preberanej problematiky, ale aj samotná technika, ktorá zabraňuje študentovi povrchnému štúdiu, t.j. aktivuje ho povinnými otázkami, ktoré musí správne zodpovedať, inak mu nedovolí pokračovať v štúdiu.

Napriek komplikovanosti tvorby a implementácie AHS, je predpoklad, že problémy autorov a pedagógov neodradia od ďalšieho využívania tejto progresívnej technológie. Aj keď ideálny systém, ktorý sa úplne priblíži každému študentovi je určite utópiou, pomocou dobre navrhnutých AHS sa dokonalej personalizácii vyučovania môžeme priblížiť.

References

1. P. Brusilovsky: *Adaptive Hypermedia.* User Modeling and User-Adapted Interaction, 11, 2001, 87-110.
2. M. Bureš, A. Morávek, I. Jelínek: *Nová generace webových technologií*. 1.VOX a.s. – Nakladatelství. 2005 Praha.
3. P. Brusilovsky: *Developing adaptive educational hypermedia systems: From design models to authoring tools.* In: Authoring Tools for Advanced Technology Learning Environments. T. Murray, S. Blessing, and S. Ainsworth (Eds.), Kluwer Academic Publisher, 2003.
4. P. De Bra, L. Calvi: *AHA: A generic adaptive hypermedia system*. In: Proc. of the 2nd Workshop on Adaptive Hypertext and Hypermedia, Pittsburg 1998., 5–12.
5. M. Bieliková, P. Šaloun: *Adaptivní webové systémy pro vzdělávání*. In DIVAI 2007: Zborníka príspevkov, FPV UKF Nitra 2007, 125-129.
6. C. Kaplan et. al.: *Adaptive hypertext navigation based on user goals and context.* In Adaptive Hypertext and Hypermedia, Peter Brusilovsky (ed.), Kluwer academic publishers, Dordrecht, 1998.
7. F. Paterno, C. Mancini: *Designing web interfaces adaptable to different types of use*. In: Proc. of the Workshop Museums and the Web, Pittsburgh 1999.
8. M. Specht: *Empirical evaluation of adaptive annotation in hypermedia.*. In: ED-MEDIA & ED-TELECOM98. T. Ottman and I. Tomek. Charlottesville, 1998, AACE, 2, 1327-1332.
9. K. Höök, M. Svensson: *Evaluating adaptive navigation support*. In: Exploring Navigation; Towards a framework for design and evaluation of navigation in electronic space. Swedish Institute of Computer Science, 1998, 141-151.
10. J. Eklund, K. Sinclair: *An empirical appraisal of the effectiveness of adaptive interfaces for instructional systems.* In: Educational Technology & Society 3., 2000.
11. G. Švejda a kol.: *Vybrané kapitoly z tvorby e-learningových kurzov*. Nitra : UKF, 2006, 136 s.

Získávání paralelních textů z webu*

Hana Klempová[1], Michal Novák[1], Peter Fabian[1], Jan Ehrenberger[2], and Ondřej Bojar[1]

[1] Charles University in Prague, Institute of Formal and Applied Linguistics (ÚFAL)
hana.klempova@seznam.cz, mixnov@gmail.com, macbeth8@gmail.com, bojar@ufal.mff.cuni.cz
[2] Czech Technical University, FJFI
lankvil@seznam.cz

Abstrakt *Příspěvek se zaměřuje na vytváření paralelního česko-anglického korpusu pro účely strojového překladu vyhledáváním a stahováním paralelních textů z Internetu. Navrhujeme a vyhodnocujeme několik vlastních metod pro nalezení kandidátských webů, identifikaci jazyka stránek a především párování získaných dokumentů.*

1 Úvod

Texty dostupné elektronicky ve více jazycích, tzv. paralelní texty, představují velmi cenný zdroj dat pro tvorbu překladových slovníků nebo pro překladatele. Nenahraditelné jsou pro systémy (statistického) strojového překladu.

Práce [1] popisuje druhé vydání paralelního česko-anglického korpusu CzEng a uvádí vliv velikosti a typu použitých paralelních dat na kvalitu strojového překladu. Nejcennější je získat texty v daném oboru, který chceme překládat, jak se však ukazuje, i jakékoli texty mimo obor mohou kvalitu výstupu zlepšit.

Cílem této práce je navrhnout a otestovat metodu automatického získávání paralelního korpusu z webu. Proces vytváření korpusu sestává z následujících fází: hledání kandidátských webů, které pravděpodobně obsahují identické texty ve více jazycích (viz oddíl 2), procházení a stahovaní stránek (textů) z kandidátských webů (oddíl 3), automatické identifikace jazyka jednotlivých stránek (oddíl 4) a ze závěrečného párovaní stažených textů (oddíl 5).

Článek v příslušných oddílech přináší jak vyhodnocení úspěšnosti nalezení paralelních webů, tak i vyhodnocení párovaní nalezených dokumentů.

2 Hledání kandidátských webů

Prvním problémem, kterému čelíme při získávání paralelních korpusů z internetu je problém nalezení vhodných kandidátských stránek, které potenciálně obsahují odkaz na svůj překlad, nebo jsou samy vícejazyčné či jinak paralelní. Výběr vhodné metody je důležitý, protože ovlivňuje jednak množství získaných výsledků, ale také počet stránek, které bude nutné prohledat a mezi kterými se budou hledat páry. V této fázi upřednostňujeme přesnost (precision) nalezených výsledků před jejich množstvím (recall), které by však obsahovalo jen málo relevantních dokumentů a jehož zpracování by mohlo trvat neúměrně dlouhou dobu.

2.1 Metody hledání

Dosavadní články pojednávající o získávání paralelních korpusů se příliš touto fázi nezabývají, hledání kandidátských webů je většinou řešeno jednoduchým dotazem do webového vyhledávače, např. v práci [7] je použito vyhledávače Altavista. Hlubší rozbor relevantních dotazů do vyhledávačů je učiněn v práci [8]. My jsme použili právě tuto metodu v mírně zjednodušené verzi. Metoda spočívá v kladení vhodně zvolených ručně vytvořených dotazů na největší internetové vyhledávače – Yahoo.com a Google.com – s cílem hledat stránky v jednom jazyce obsahující odkaz na jinou jazykovou mutaci. Původní Saint-Amandova metoda spočívá v automatickém nalezení názvů jazyků na základě jejich zkratek („cs“ a „en“) a jejich křížové kombinaci s doménovými jmény typickými pro stránky v daném jazyce („.cz“ pro české stránky, „.uk“ ap. pro anglické stránky). Pro pár čeština-angličtina byly tedy vyrobeny dotazy ve tvarech: „english site:cz“, „česky site:uk“, „česky site:nz“, „čeština site:au“ a jejich různé jazykově zkřížené kombinace. Na rozdíl od Saint-Amanda jsme se rozhodli doplnit hledání výrazů „česky“ o název jazyka „čeština“ a prohledávání těchto výrazů jsme rozšířili také na všeobecné domény com, net a org. Při hledání je možné též využít složitějších dotazů používajících direktivy `filetype`, `inanchor`, `intext`, `inurl` (např. „česky inurl:lang=en“ pro stránky jejichž URL obsahuje náznak, že je stránka v angličtině), či se zkusit spolehnout na rozpoznávaní jazyků vyhledávače Google (direktiva `lang`). Posledně jmenovanou možnost jsme se po zběžném prohlédnutí nepříliš spolehlivých výsledků rozhodli zatím nepoužít, její pozdější bližší prozkoumání je však pravděpodobné.

Při použití databází vyhledávačů jako vstupní brány ke zdánlivě nekonečnému množství dat na webu

* Práce na tomto projektu je podporována granty FP7-ICT-2007-3-231720 (EuroMatrix Plus) a MSM 0021620838.

narážíme na vestavěné limity dotazovacích rozhraní. Při použití veřejného API k vyhledávači Yahoo.com je možné položit maximálně 5 000 dotazů denně; co je však horší, maximálně je možno se dostat k 1 000 výsledkům na jeden dotaz. Například na dotaz „english site:cz“ Yahoo.com zahlásí 40 milionů nalezených dokumentů, můžeme se však dostat jen k tisíci z nich. Ještě o něco horší je situace s vyhledávačem Google, který sice hlásí nalezení téměř 70 milionů dokumentů, ale při 614. výsledku vyhodnotí další položky jako podobné již zobrazeným a nevypíše je. Při použití klasického přístupu přes internetový prohlížeč, nebo pomocí SOAP API platí omezení na 1 000 výsledků na jeden dotaz obdobně jako u Yahoo, celkový počet povolených dotazů je však ještě menší. Mezi další možnosti, jak extrahovat z vyhledávačů více výsledků, patří použití různých triků s restrikcí pomocí data, či zaregistrování se pro použití nového AJAXového API.

K dalším možnostem hledání vhodných kandidátů patří implementace vlastního robota, který bude stránky procházet a hledat mezi nimi vícejazyčné weby. Tento postup je zevrubně popsán v [3].

2.2 Vyhodnocení vhodnosti nalezených webů

Pomocí výše popsaných jednoduchých metod se nám povedlo získat pomocí vyhledávače Yahoo cca. 8 500 URL adres a pomocí Google cca 6 400 URL adres. Z nich jsme na ruční evaluaci vybrali náhodně 2 %, co činí 171, resp. 129 stránek. U stránek jsme kontrolovali, zda obsahují odkaz na paralelní texty; kromě toho jsme taky hodnotili celou doménu (druhého řádu) příslušnou k nalezené stránce. Přehled klasifikace stránek a domén uvádějí tabulky 1 a 2.

Jak je vidět v tabulce 3, z vyhledávače Yahoo je paralelních 35 % stránek, z Google je to jenom 15 %, celkově paralelních je 26 % stránek. Bez jakékoliv paralelní informace je v případě Yahoo 43 % stránek, v případě Google až 57 % stránek, což je v celkovém počtu 49 % stránek. Obtížně použitelných nebo zcela nepoužitelných stránek je tedy po sečtení necelých 75 %.

n	obsahuje jenom jednu jazykovou verzi a v jejím okolí jsme nenašli žádnou stopu po paralelní verzi
x	obsahuje dva jazyky, jsou však oba přítomny v téže stránce
f	se vyskytuje ve vícero jazykových mutacích, každá z mutací má jiný obsah (typický příklad: Wikipedie)
l	sice obsahuje přepínač jazyků, obě verze však mají totožný obsah (tedy česká i anglická verze obsahují tentýž český text)
m	má překlad do jiných jazykových verzí, ale jenom strojový, většinou Google Translate
s	sice paralelní, ale strukturně se překlady značně liší
p	paralelní stránka

Tab. 1. Klasifikace stránek při ručním hodnocení.

ugc	uživatelsky tvořený obsah – typicky přeložené jen části obecného rozhraní stránek, např. blogy, vlastní youtube kanály a pod.
p/f	část domény je paralelní, část obsahuje pro různé jazyky odlišné texty
p/l	část domény je paralelní, část jednojazyčná, tvářící se, že obsahuje jazyků více
p/n	část domény je paralelní, část jednojazyčná
p	paralelní stránka

Tab. 2. Klasifikace domén při ručním hodnocení.

Vyhledávač	**Zastoupení typů stránek**						
	n	x	m	f	l	s	p
Yahoo	42,7	4,1	0,6	11,7	5,3	0,6	35,1
Google	57,4	0,8	0,8	11,6	6,2	8,5	14,7
Celkem	49,0	2,7	0,7	11,7	5,7	4,0	26,3

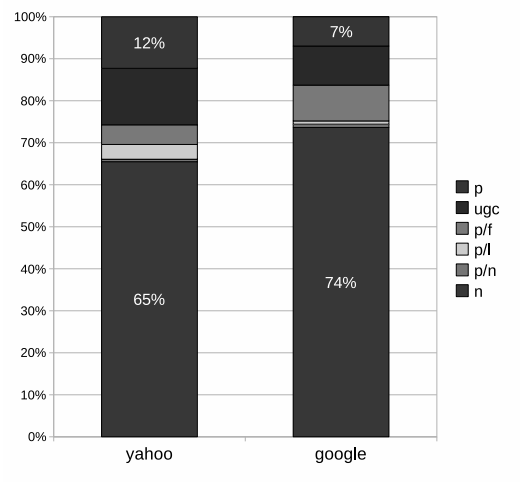
Tab. 3. Hodnocení stránek podle vyhledávačů (hodnoty v procentech).

Z grafu na obr. 1, který obsahuje hodnocení domén druhého řádu, je patrné, že poměr čistě paralelních domén v celém hodnoceném souboru domén, příslušným k náhodně vybraným stránkám, je v případě vyhledávače Yahoo 12 %, v případě Google cca 7 %, celkově tato skupina webů tvoří 10 % za všech domén v našem testovacím vzorku. Je však potřeba si uvědomit, že i části domén hodnocených p/f, p/l a p/n jsou vhodnými kandidáty, protože jisté jejich části jsou též paralelní. Domény kompletně a částečně paralelní tvoří více než 19 % kontrolovaných domén.

Prezentované výsledky pokládáme za poměrně slibné, pro větší reprezentativnost by bylo ovšem vhodné ručně ohodnotit větší skupinu stránek a ověřit též anotátorskou shodu více anotátorů. Je možné, že další zlepšení úspěšnosti by přinesly sofistikovaněji kladené dotazy na vyhledávače nebo specializované metody hledání paralelních stránek pomocí webových robotů.

3 Stažení a očištění textů

Abychom získali co nejvíce paralelních textů, nalezené stránky pokládáme pouze za doporučené startovní body a pomocí standardní UNIXové utility `wget` s přepínačem `-m` stáhneme všechny dokumenty z dané domény dosažitelné rekurzivně přes odkazy z počáteční stránky. Tato metoda je časově i prostorově poměrně náročná. Větší domény mohou obsahovat množství



Obr. 1. Hodnocení domén druhého rádu podle vyhledávačů.

archivního obsahu a jejich stahování, i když nejsou třeba vůbec paralelní, tak může celý proces významně zpomalit a zaplnit disk. Navíc je pak nutné tyto stránky zkoumat alespoň pomocí rozpoznávače jazyků pro zjištění, zda obsahují češtinu a angličtinu (případně jiné jazyky, o které se zajímáme). Na druhou stranu můžeme mít poměrně velkou jistotu, že pokud se v dané doméně nějaké paralelní stránky nacházejí, tato metoda na ně narazí a stáhne je.

Sofistikovanějšího postupu je využito v práci [8] – z nalezené stránky se pokusí dostat na její paralelní verzi (verzi v jiném jazyce, v originální práci se vyhledává více jazykových verzí najednou). Pokud se mu to povede, vytvoří z paralelních stránek stromové struktury na základě HTML a přiřadí tak k sobě i odkazy na další navzájem paralelní stránky. Pomocí zarovnaných odkazů je daná doména přezkoumána a nalezené paralelní stránky uloženy. Tento přístup ušetří mnoho místa a času, na druhou stranu je zde možnost, že paralelní stránky nebudou nalezeny, a nevyužijeme tak celý potenciál dané domény.

Metoda zvolená pro očištění textů od nadbytečných HTML značek byla jednoduchá – námi implementovaný odstraňovač tagů se ukázal jako nejvhodnější řešení, jelikož jiná řešení (např. pomocí PERLovského modulu `HTML::Parser`) často odstranila i text.

4 Identifikace jazyka

Stažené a očištěné texty je nutné rozdělit do skupin podle jazyků, ve kterých jsou tyto texty napsány. Fáze párování (viz oddíl 5) funguje jenom za předpokladu, že texty jsou správně faktorizovány podle jazyka.

Tab. 4. Úspěšnost identifikátoru jazyka na anglických a českých textech o velikosti 200 znaků (100 pokusů) a 400 znaků (50 pokusů).

Jazyk	**Úspěšnost dle velikosti vzorku**	
	200 znaků	400 znaků
angličtina	95 %	98 %
čeština	97 %	100 %

Identifikaci jazyka lze provádět např. statistickými metodami založenými na porovnání frekvencí výskytu znakových n-tic (n-gramů) v textu s frekvencemi v trénovacích datech, pro které je známo na jakém jazyku byly natrénovány (modely). Metoda popsaná v [8] trénuje tyto modely pro velkou množinu jazyků z dat internetové encyklopedie Wikipedia. Četnosti jsou zaznamenávány na plném textu očištěném od HTML, z důvodu jazykové nezávislosti se neprovádí žádná tokenizace (dělení na slova) ani další jiné úpravy textu, které by při identifikaci mohly pomoci. N-gramová metoda byla popsána i dřív v práci [4], kde jsou navrženy také možnosti vylepšení prostřednictvím Markovových řetězců nebo Bayesovských rozhodovacích metod. Tento přístup však v sobě skrývá problém malé reprezentativnosti natrénovaného modelu. Kdyby se totiž jako česká trénovací data vybral text, který pojednává o Praze, mohlo by se stát, že mezi nejčastějšími trigramy bude „Pra“ nebo „ Pr“, přičemž obecně jsou v češtině nejčastější trigramy „ch “, „ ní“ nebo „ př“.

Zmíněný problém řeší větší množství trénovacích dat. Čím více dat, tím reprezentativnější je model. Náš systém však používá identifikátor jazyka, na jehož natrénování postačí menší objem textu. Princip této metody spočívá v tom, že četnosti n-gramů se nepočítají na celém textu, t.j. na všech slovních výskytech, ale jen na slovních typech. Text se převede na množinu použitých slov (slovních forem) a na této množině se spočte n-gramová statistika. Tento postup vyžaduje tokenizaci textu, což v našem případě (čeština a angličtina) není obtížná úloha. Pro jazyky jako čínština nebo japonština je použití popsaného postupu problematičtější.

Konkrétněji k modelům našeho identifikátoru jazyka:

- Byly natrénovány modely pro 21 evropských jazyků, včetně těch, které by mohly být eventuelně zaměněny s češtinou nebo angličtinou.
- Na natrénování byly použity různě dlouhé texty z Wikipedie, nejkratší obsahoval cca 50 tis. znaků.
- Text byl rozsekán na všech neabecedních znacích, čímž se jednak odstranila všechna čísla a speciální znaky, a jednak se tím text tokenizoval.

– Každé slovo bylo ohraničeno z obou stran závorkami, abychom odlišili prefixové a sufixové n-gramy, tj. n-gramy na počátku a konci slov.
– Výsledný model pro daný jazyk je seznam 100 nejčastějších trigramů na slovních typech, t.j. každé slovo započteno jenom jednou.

Samotná identifikace probíhá tak, že se stejným způsobem spočte model z textu, jehož jazyk se snažíme zjistit (testovaný text/model). Testovaný model je porovnán s každým z natrénovaných modelů a je vybrán takový jazyk, jehož model je nejpodobnější testovanému modelu. Pro míru podobnosti mohou být použity všechny metriky pro určení vzdálenosti mezi vektory (např. Euklidova). My jsme použili míru vyjádřenou vzorcem

$$p = 1 - \frac{|l_1 - t_1| + ... + |l_n - t_n|}{2} \qquad (1)$$

kde $l_1...l_n$ jsou relativní frekvence jednotlivých trigramů v natrénovaném modelu a $t_1...t_n$ jsou relativní frekvence stejných trigramů v testovaném modelu.

Úspěšnost identifikátoru jsme testovali na českých a anglických textech z internetových novin. V prvním testu jsme texty rozdělili na 100 částí o 200 znacích, v druhém testu na 50 částí o 400 znacích. Výsledky ukázaly, že už na takto malých kusech textu náš identifikátor jazyka dosahuje úspěšnosti nad 95 % a s rostoucí velikostí textu roste úspěšnost identifikace. Podrobnější výsledky jsou v tabulce 4.

5 Párování dokumentů

V dosavadních pracích byly popsány různé metody párování dokumentů. Nejrychlejší jsou metody založené na metainformacích stažených stránek, např. párování na základě podobnosti URL adres [7,2,3]. Takovou metodou je i párování na základě porovnání délek dokumentů (v [7] a [2]), které je však vhodnější pro párování vět.

Paralelní dokumenty mívají často stejnou nebo podobnou strukturu HTML značek, čehož využívají strukturní metody, např. využitím linearizovaného HTML v [7] nebo porovnáním stromové struktury v [8].

Nejpomalejší, ale zároveň nejúčinnější jsou metody založené na obsahové podobnosti dokumentů. Pro příbuzné jazyky nebo jazyky s některými slovy příbuznými je možné použít metodu, která pomocí Levensteinovy vzdálenosti určuje podobnost textů nebo záchytných slov (použito v [7]). Záchytná slova jsou zadána slovníkem, mít kompletní slovník však není potřeba. Práce [2] také používá tuto metodu, jelikož však zkoumá velmi odlišné jazyky (angličtina, čínština), je tato metoda ekvivalentní strukturní metodě linearizace HTML.

Konečně musíme zmínit i metody, využívající slovník při měření obsahové podobnosti. Tyto metody mohou text reprezentovat jako vektor četností jednotlivých typů slov a porovnávat podobnost vektorů [10] nebo párovat kandidátské texty slova na slovo a podobnost měřit mírou spárování [7]. Práce [10] používá i druhou metodu a navíc řeší specifika čínštiny.

Naše metody pro nalezení paralelních textů v množině českých a anglických textů byly navrženy nezávisle a implementovány v aplikaci WebCorpus, podrobněji viz [6]. Využíváme kombinaci jednoduché strukturní metody a slovníkové obsahové metody. Algoritmus lze rozdělit do několika základních kroků:

1. Zmapování struktury souboru (pozice prázdných a neprázdných řádků).
2. Segmentace a tokenizace souborů (na každém řádku souboru právě jedna věta, slova a interpunkční znaménka jsou oddělena mezerou).
3. Výběr n nejčastějších slov pro každý soubor.
4. Přeložení anglických slov pomocí slovníku.
5. Vyhledání základních tvarů pro česká slova.
6. Párování souborů jednou z několika implementovaných metod.

Kroky 1 až 5 provádíme dávkově předem, viz následující oddíl. Jádro párování (krok 6) pak popisuje oddíl 5.2.

5.1 Zpracování souborů

Prvním krokem algoritmu je zmapování struktury souboru. Výsledkem této operace je binární vektor, kde na i-té pozici je 0, právě když je i-tý řádek souboru prázdný, a 1 v opačném případě (viz též obrázek 2). Motivací k této metodě bylo pozorování, že stránky, které jsou si navzájem překladem, jsou velmi často stejně členěné. Nespornou výhodou této možnosti porovnání souborů je její časová nenáročnost.

Následujícím krokem předzpracování je segmentace a tokenizace textu. V této části dochází k restrukturalizaci textu tak, aby na každém řádku byla právě jedna věta a všechna slova a interpunkční znaménka byla oddělena mezerou. K tokenizaci textu byl použit String::Tokenizer a následná segmentace probíhá pomocí regulárních výrazů. Největším úskalím je v této úloze rozeznat, kdy je tečka použita jako součást zkratky a kdy ukončuje větu.

Následující zpracování souborů se liší podle toho, zda se jedná o anglický či český text. V případě anglických souborů se nejprve nalezne n nejčastějších slov (default je 80). Tyto výrazy se přeloží ve slovníku

a odfiltrují se slova, která příliš nevypovídají o významu daného textu (např. některá zájmena či předložky).

V případě českých souborů se také nejprve nalezne n nejčastějších slov, místo překladů však ke slovům jen nalezneme základní tvary pro odstranění morfologické bohatosti. K tomu používáme námi implementovaný modul, který pracuje s morfologickým slovníkem z práce [5][3].

5.2 Hodnocení dvojic textů

Hledání paralelních textů probíhá vždy samostatně v rámci každého staženého webu.

Pro ohodnocení vzájemné paralelnosti dvojic souborů si program vytvoří dvě matice. V první z nich, nazvěme ji matice S, je vyjádřena podobnost souborů na základě binární charakteristiky struktury staženého souboru. Technicky se jedná o porovnání dvou řetězců pomocí editační (Levenshteinovy) vzdálenosti implementované v modulu `String::Similarity`, který vrací hodnotu v intervalu <0,1> vyjadřující míru jejich podobnosti. Ilustrace je na obrázku 2.

```
mountain hotel JARMILKA - animation services -
Liberec



 mountain hotel JARMILKA



                home
                accommodation
                restaurant

  Liberec

  The town of Liberec
 The town Liberec is considered to be a
significant administrative, cultural, industrial
and sports centre not only of the

Jizera Mountains but also of the whole region. It
lies in a valley between the hills of the Jizera
Mountains and the Jested Ridge.



  The ZOO Liberec



  The ZOO Liberec
```

1100001000111011111011000100001

Obr. 2. Ukázkový text stránky po odstranění HTML značek a jeho binární charakteristika struktury (znak 0 na místě prázdného řádku, znak 1 na místě neprázdného řádku). Pokud bychom k tomuto anglickému souboru měli paralelní český s touto binární charakteristikou 1100001<u>1</u>00111<u>0</u>0111011100<u>1</u>10000<u>01</u> (odlišné znaky jsou podtržené), vyšla by pomocí String::Similarity míra podobnosti 0.89.

[3] `http://ufal.mff.cuni.cz/pdt/ /Morphology_and_Tagging/Morphology/index.html`

	Slovo	Četnost	Index D	Index H
27	liberec	16	27	27
26	město	8	26	26
25	rok	6	23	25
24	centr	6	23	25
23	století	6	23	25
22	jenž	5	21	22
21	zahrada	5	21	22
20	hora	4	19	20
19	m2	4	19	20
18	ZOO	3	13	18
17	animační	3	13	18
	...			
14	aquapark	3	13	18
13	služba	3	13	18
12	radnice	2	3	12
11	jizerský	2	3	12
	...			
4	hodně	2	3	12
3	daleko	2	3	12
2	lázeň	1	1	2
1	kulturní	1	1	2

Tab. 5. Ukázka výpočtu indexu D a H pro daná slova a jejich četnosti.

Na základě nejčastějších slov souboru je vytvořena matice L. Porovnání probíhá nejprve vždy tak, že pro každé slovo z n nejčastějších slov českého souboru hledáme ekvivalent mezi překlady n nejčastějších slov anglického souboru. Vedle n nejčastějších slov si také pamatujeme jejich četnosti. Nabízí se tedy otázka, jak s informací o četnosti při hodnocení dvojice textů naložit. Označme c četnost slova v českém souboru a e četnost shodného slova v anglickém souboru. Potom existuje hned několik možností, jak bodové ohodnocení určit:

1. První a nejjednodušší možností je četnosti ignorovat a za každé shodné slovo ohodnotit dvojici souborů po jednom bodě.
2. Druhou variantou je poměr součtu četností a dvojnásobku jejich rozdílu:

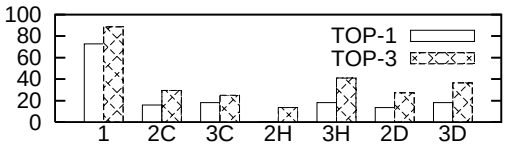
$$\frac{c+e}{2*|c-e+1|} \tag{2}$$

   Tímto způsobem se nejvíce ocení slova, která jsou velmi častá v obou textech.
3. Dále se samozřejmě nabízí klasický aritmetický průměr četností:

$$\frac{c+e}{2} \tag{3}$$

Další variantou je dosadit na místo skutečné četnosti (označme C) námi navržené indexy D a H, které vyjadřují dolní a horní index slova při uspořádání dle četnosti, příklad viz tabulka 5. Motivací k výpočtu indexů D a H nám byla snaha co nejvíce rozlišit bodové hodnocení dvojice textů v případech shody v hodně častých a naopak málo častých slovech.

K tomu, abychom mohli vybrat nejefektivnější z popsaných metod, bylo nutné provést experiment.



Obr. 3. Procento správně spárovaných dvojic dokumentů při použití jednotlivých metrik.

Sadu testovacích dat tvořilo 44 dvojic českých a anglických textů, které si jsou překladem a byly staženy ze tří menších webů. Všechny soubory prošly analýzou, která našla n nejčastějších slov včetně jejich četností. Poté byly všechny české a anglické dvojice porovnány pomocí všech implementovaných metrik. Cílem bylo nalézt tu z nich, která nejlépe ohodnocuje dvojice textů, které si jsou překladem. Nejprve jsme tedy sledovali, v kolika případech bude správná dvojice danou metrikou hodnocena nejvýše (TOP-1). Poté jsme uvažovali, zda je správná dvojice alespoň mezi třemi nejlépe hodnocenými páry (TOP-3). Výsledky obou testů zachycuje obrázek 3. Číslo určuje testovanou metriku dle výše uvedeného seznamu, písmeno dosazenou četnost, resp. index, například metrika 3C znamená výpočet aritmetického průměru z původních četností.

Z grafu jasně plyne, že nejlépe hodnotila správné dvojice textů první nejjednodušší metrika, která přidělí dvojici souborů jeden bod za každé shodné slovo bez ohledu na jeho četnost. Tato metrika měla úspěšnost 32 ze 44 (tj. 72.7 %) v prvním testu a 39 ze 44 (tj. 88.6 %) ve druhém testu. Jak je patrné z grafu, zbylé metriky nedosáhly ani na padesátiprocentní úspěšnost. To může být způsobeno hned několika důvody:

- Překlady většiny webových stránek nejsou příliš přesné a doslovné.
- Může dojít k nepřesnosti v průběhu překládání anglických slov např. pokud hledaný výraz slovník vůbec neobsahuje a tím pádem se nepřeloží nebo v množině překladů není ten tvar, který potřebujeme.
- Také zpracování českých výrazů není vždy stoprocentní a může se stát, že nejsou úspěšně nalezeny všechny základní tvary. Navíc v případě některých slov nelze jejich základní tvar bez ohledu na kontext určit jednoznačně, např. slovo *má* může být tvarem slova *mít* ale také *moje* nebo *můj*. Aplikace v tomto případě pracuje pouze s prvním nalezeným základním tvarem.

5.3 Spárování souborů

Pro finální párování souborů jsme se rozhodli implementovat celkem sedm metod, které různě kombinují výše popsané matice S a L. Navíc se opíráme o výstup nástroje hunalign [9], který zkoumanou dvojici textů zkusí spárovat na úrovni vět a určí kvalitu přiřazení. Kvalita párování po větách je samozřejmě velmi dobrým ukazatelem paralelnosti dané dvojice dokumentů, je však ve srovnání s předešlými metodami výrazně výpočetně náročnější.

Jednotlivé kombinace metod jsou pojmenované takto: *str_lex_multi, str_lex_fix, str_lex_full, str_fix, lex_fix, str_full, lex_full*. A zde je základní návod, jak se v nich orientovat:

Pokud je v názvu metody obsaženo **str**, znamená to, že se pracuje s maticí S, která hodnotí dvojice souborů na základě struktury textu ihned po odstranění HTML značek.

Pokud je v názvu metody slovo **lex**, bere se v úvahu matice L, která hodnotí dvojice souborů na základě shodného výskytu významových slov.

Pokud jméno metody začíná **str_lex**, pracuje metoda s oběma úhly pohledu a hledá takové přiřazení, které je dobře hodnocené oběma metodami. Pro každý český soubor se berou v úvahu tři nejlépe hodnocené anglické soubory z obou porovnání. Označme 1.l nejlépe hodnocený anglický soubor v daném řádku matice L, 1.s nejlépe hodnocený anglický soubor v daném řádku matice S atd. Potom se k danému českému souboru hledá nejvhodnější anglický tak, aby byl mezi třemi nejlépe hodnocenými pomocí obou metod. Pokud se takový najde, je s daným českým souborem zpracován hunalignem, který vrátí kvalitu přiřazení. Parametrem lze určit nejnižší možnou hodnotu, aby se mohla dvojice prohlásit za paralelní (defaultně je nastavena na -1). Shodnost anglických souborů obdržených metodou S a L se testuje v tomto sledu:

1) 1.s – 1.l	4) 2.s – 2.l	7) 3.s – 2.l
2) 2.s – 1.l	5) 3.s – 1.l	8) 2.s – 3.l
3) 1.s – 2.l	6) 1.s – 3.l	9) 3.s – 3.l

Dá se tedy říci, že lehce favorizována je metoda L (vysoce hodnocené soubory metodou L se testují dříve), která by z logiky věci měla mít skutečně větší vypovídací hodnotu.

Slovo **multi** v názvu metody znamená, že pro každý český soubor najdu první vyhovující anglický a ten mu přiřadím. První vyhovující anglický soubor je takový, který první ve výše zmíněném pořadí vytváří s daným českým souborem paralelní korpus o minimálně hraniční kvalitě. V této variantě se může stát, že jeden anglický soubor bude přiřazen k více českým souborům.

Pokud je v názvu metody obsaženo **fix**, znamená to, že pro již přiřazené anglické soubory si pamatuji, s jakou kvalitou byly přiřazeny. Pokud je k danému českému souboru nalezen anglický soubor splňující základní podmínky (tj. je mezi třemi nejlepšími v me-

	EN1	EN2	EN3	EN4	EN5	EN6
CZ1	**.99**	**.98**	**.97**	.81	.89	.80
CZ2	**.97**	**.98**	.81	.69	**.94**	.72
CZ3	**.96**	.90	**.98**	**.99**	.67	.78
CZ4	.90	**.93**	**.95**	**.98**	.81	.80
CZ5	.90	**.98**	.82	.87	**.97**	**.96**
CZ6	.76	**.93**	.87	.90	**.95**	**.98**

Tab. 6. Ukázka matice S. Tučně jsou vyznačeny tři nejlepší páry každého řádku.

	EN1	EN2	EN3	EN4	EN5	EN6
CZ1	**18**	**20**	4	**15**	5	9
CZ2	**15**	**17**	3	3	**13**	4
CZ3	2	5	**18**	**19**	**10**	4
CZ4	**12**	**13**	6	**22**	3	6
CZ5	9	**18**	4	7	**14**	**17**
CZ6	5	**14**	4	8	**18**	**17**

Tab. 7. Ukázka matice L.

todě S nebo v metodě L nebo v obou a spárování je nad hraniční kvalitou), ale přiřazení by dosahovalo nižší kvality, než s kterou byl již anglický soubor přiřazen, potom se toto přiřazení neprovede a hledání pokračuje dál. Naopak, pokud nastane situace, že by anglický soubor mohl být přiřazen s vyšší kvalitou, toto přiřazení se provede a v předešlém běhu programu vytvořená dvojice souborů se rozváže. Pro český soubor z rozvázané dvojice se hledá vhodný anglický protějšek znovu. V této metodě je tedy každý soubor přiřazen nejvýše jednou.

Poslední možnou zkratkou obsaženou v názvu metody je **full**. V tomto případě se postupuje podobně jako v tom předešlém, ale je snahou nacházet opravdu nejlepší přiřazení. To znamená, že pro každý český soubor se otestují všechny anglické soubory, které přicházejí v úvahu a z nich se vybere ten nejlepší (hledání je ovšem hladové a nezaručuje nejlepší párování ze všech možných). Tato varianta je tedy výpočetně nejnáročnější, protože se testuje největší počet přiřazení, měla by však vést ke kvalitnějšímu výsledku. Také v této metodě je každý soubor přiřazen nejvýše jednou.

Pro lepší pochopení v obrázku 4 uvádíme práci metody str_lex_fix nad množinou českých textů CZ1 až CZ6 a množinou anglických textů EN1 až EN6, s maticemi S a L uvedenými v tabulkách 6 a 7.

5.4 Složitost a vyhodnocení

Označme n_c počet českých a n_e počet anglických souborů ve zpracovávaném webu. Potom časová náročnost vytvoření matic L a S je $O(n_c * n_e)$. Časová náročnost hledání paralelních textů je $O(n_c * 3h)$, kde h je časová náročnost spárování dvou souborů pomocí nástroje hunalign (každý český soubor zkoušíme spárovat s maximálně třemi anglickými v každé z implementovaných metod).

Všechny implementované metody jsme podrobili testu. Experiment proběhl na množině 74 dvojic čes-

1. stav na zásobníku: (CZ1, CZ2, CZ3, CZ4, CZ5, CZ26)
   - na vrcholu zásobníku je prvek CZ1
   - tři nejlepší kandidáti matice S: EN1, EN2, EN3
   - tři nejlepší kandidáti matice L: EN2, EN1, EN4
   - jako první zkusíme spárovat CZ1 s EN2, nechť je kvalita spárování h(CZ1,EN2)=0.7
   - soubor EN2 zatím nebyl spárován, tedy dvojice CZ1, EN2 vyhovuje
2. stav na zásobníku: (CZ2, CZ3, CZ4, CZ5, CZ6)
   - na vrcholu zásobníku je prvek CZ2
   - tři nejlepší kandidáti matice S: EN2, EN1, EN5
   - tři nejlepší kandidáti matice L: EN2, EN1, EN5
   - h(CZ2,EN2)=0.9 > 0.7
   - zruší se dvojice CZ1,EN2 a vytvoří se nová dvojice CZ2,EN2
3. stav na zásobníku: (CZ1, CZ3, CZ4, CZ5, CZ6)
   - S: EN1, EN2, EN3; L: EN2, EN1, EN4
   - h(CZ1,EN2)=0.7 < 0.9, h(CZ1,EN1)=0.8
   - vytvoří se nová dvojice CZ1,EN1
4. stav na zásobníku: (CZ3, CZ4, CZ5, CZ6)
   - S: EN4, EN3, EN1; L: EN4, EN3, EN5
   - h(CZ3,EN4)=0.5 ⇒ nová dvojice CZ3,EN4
5. stav na zásobníku: (CZ4, CZ5, CZ6)
   - S: EN4, EN3, EN2; L: EN4, EN2, EN1
   - h(CZ4,EN4)=0.6 > 0.5 ⇒ nová dvojice CZ4,EN4
6. stav na zásobníku: (CZ3, CZ5, CZ6)
   - S: EN4, EN3, EN1; L: EN4, EN3, EN5
   - h(CZ3,EN4)=0.5 < 0.6
   - h(CZ3,EN3)=0.8 ⇒ nová dvojice CZ3,EN3
7. stav na zásobníku: (CZ5, CZ6)
   - S: EN2, EN5, EN6; L: EN2, EN6, EN5
   - h(CZ5,EN2)=0.3 < 0.9
   - h(CZ5,EN6)=0.6 ⇒ nová dvojice CZ5,EN6
8. stav na zásobníku: (CZ6)
   - S: EN6, EN5, EN2; L: EN5, EN6, EN2
   - h(CZ6,EN5)=0.6 ⇒ nová dvojice CZ6,EN5

Obr. 4. Postup metody **str_lex_fix** nad maticí S a L z tabulek 6 a 7.

kých a anglických textů (celkem tedy 148 souborů). Postupně jsme zkusili soubory spárovat pomocí všech sedmi metod. Ke zpracování výsledků jsme se rozhodli použít metodu precision-recall. V tomto konkrétním případě true positive=správně vybraná dvojice textů, true negative=správně nevybraná dvojice, false positive=špatně vybraná dvojice a konečně false negative=špatně nevybraná dvojice (metody vybírají ty dvojice, které považují za vzájemné překlady). Jelikož na vstupu má metoda 74 českých a 74 anglických souborů, existuje celkem 74 x 74 = 5 476 dvojic souborů. Z toho 74 dvojic tvoří překlady a zbylých 5 402 dvojic k sobě nepatří.

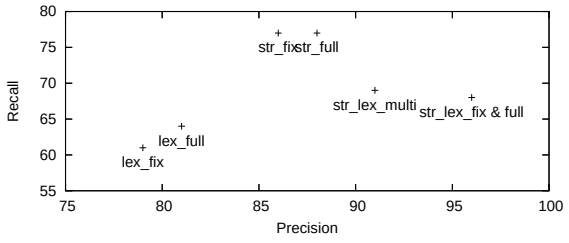
První testovanou metodou byla str_lex_multi. Tato metoda našla 51 správných dvojic a 5 špatných. Hledané hodnoty precision a recall se tedy spočítají takto:

$$precision = \frac{tp}{tp + fp} = \frac{51}{51 + 5} = 0.91 \qquad (4)$$

$$recall = \frac{tp}{tp + fn} = \frac{51}{51 + 23} = 0.69 \qquad (5)$$

Stejný výpočet probíhá i s výsledky zbylých metod. Obdržené hodnoty zobrazuje graf v obrázku 5.

Na základě grafu lze udělat tato pozorování:



Obr. 5. Výsledek testu metod párování souborů.

- Kombinace porovnání souborů na základě struktury a nejčastějších slov přinesla nejpřesnější výsledky, ovšem ne nejvyšší recall.
- Dá se říci, že metody full dosahují v průměru lepších výsledků než metody fix.
- Z našeho testu vyšla metoda str lépe než metoda lex. Tento výsledek se ale odvíjí od charakteru vstupních webů, a proto ho dle našeho názoru nelze příliš zobecňovat.
- Metoda str může být zajímavá v případě, kdy klademe důraz na časovou optimalizaci úlohy, neboť je v porovnání s metodou lex výrazně rychlejší.

6 Celkové vyhodnocení

Na závěr jsme se rozhodli otestovat, nakolik úspěšné bude nalezení vhodných párů dokumentů na náhodně vybraných 5 odkazech s nejvyšším hodnocením – „p“ pro stránku i doménu. Počáteční odhodlání testovat několik náhodně vybraných domén z každé skupiny hodnocení jsme museli opustit kvůli časové i prostorové náročností stahovaní celých domén. Nechali jsme si tedy stáhnout 5 náhodně vybraných domén a na nich pustit párovací algoritmus s výchozími možnostmi nastavení párování, jak je uvedeno výše.

Na daném testovacím vzorku bylo správně spárováno 69 stránek ze 75 nalezených párů, co představuje přesnost (precision) 92 %. Malý korpus vytvořený z těchto 5 domén obsahuje 2 400 paralelních vět, 51 700 českých a 55 200 anglických slov. Jak se tedy ukázalo podle kvality získaných URL a úspěšnosti párování, výsledný korpus je vytvořen ze zhruba 10 % nalezených URL, přičemž každá z nich přispěje průměrně 12 paralelními stránkami. Je však nutno podotknout, že tato statistika byla vytvořena jen na malých datech, a proto ji není možno považovat za zcela spolehlivou. Rozsáhlejší testování získaných dat bude předmětem dalších pokusů.

7 Závěr

Práce shrnuje metody automatického sběru paralelních korpusů z webu a testuje je na páru jazyků čeština-angličtina. Navržené metody automatického rozpoznání jazyků a párování stažených stránek dosahují uspokojivých výsledků, i když další možnosti zlepšení lze uvažovat, např. s použitím metod strojového učení.

Ukazuje se, že pro získaní paralelního korpusu je dnes limitujícím faktorem spíše schopnost nalézt dostatečné množství paralelních webů. Vyhodnocení námi použitých metod ukázalo, že přibližně deset procent nalezených URL adres je použitelných pro další zpracování. Dalším prohledáním domén získaných z URL se dá i při relativně malém množství počátečních adres nalézt velké množství paralelních dokumentů. Problémem však často zůstávají vestavěná omezení ve webových vyhledávačích, o něž se naše metoda opírá. V další práci se tedy zaměříme zejména na vylepšení vyhledávání paralelních webů.

Reference

1. O. Bojar, M. Janíček, Z. Žabokrtský, P. Češka, and P. Beňa: *CzEng 0.7: parallel corpus with community-supplied translations.* In Proc. of LREC, Marrakech, Morocco, May 2008. ELRA.
2. J. Chen and J.-Y. Nie: *Automatic construction of parallel English-Chinese corpus for cross-language information retrieval.* In Proc. of ANLP, Seattle, Washington, 2000, 21–28.
3. J. Chen, R. Chau, and C.-H. Yeh: *Discovering parallel text from the world wide web.* In The Australasian Workshop on Data Mining and Web Intelligence (DMWI-2004), Dunedin, New Zealand, 2003.
4. T. Dunning: *Statistical identification of language.* Computing Research Laboratory Technical Memo MCCS 94-273, New Mexico State University, Las Cruces, New Mexico, 1994.
5. J. Hajič: *Disambiguation of rich inflection - computational morphology of Czech.* Prague Karolinum, Charles University Press, volume I., 2001, 334 pp.
6. H. Klempová: *Nástroj pro sběr paralelních textů z webu.* Bakalářská práce, Matematicko-fyzikální faktulka, Univerzita Karlova v Praze, 2009.
7. P. Resnik and N. A. Smith: *The web as a parallel corpus.* In Computational Linguistics, 29, 3, September 2003, 349–380.
8. H. Saint-Amand. *Gathering a parallel corpus from the web.* Master's Thesis, Universität des Saarlandes, Saarbrücken, Germany, September 2008.
9. D. Varga, L. Németh, P. Halácsy, A. Kornai, V. Trón, and V. Nagy: *Parallel corpora for medium density languages.* In Proc. of RANLP, Borovets, Bulgaria, 2005, 590–596.
10. C. C. Yang and K.W. Li: *Automatic construction of English/Chinese parallel corpora.* In Journal of the American Society for Information Science and Technology, 2003, 730–742.

Phalanger IntelliSense: Syntactic and semantic prediction

Jakub Míšek, Daniel Balaš and Filip Zavoral
Department of Software Engineering
Charles University in Prague, Czech Republic

Abstract. *In the context of computer programming, the importance of computer assistance is being understood by many developer communities. Developers are e.g. using the same well known expressions or searching method signatures in library documentations. Code sense or IntelliSense methods make most of these actions unnecessary because they serve the available useful information directly to the programmer in a completely automated way. Recently, with the increased focus of the industry on dynamic languages a problem emerges - the complete knowledge on the source code is postponed until the runtime, since there may be ambiguous semantics in the code fragment.*
As a part of the Phalanger project the methods for syntax and semantic analysis of the dynamic code were designed, especially targeted for the PHP programming language. These methods produce a list of valid possibilities which can be then used on a specified position in the source code; such as declarations, variables and function parameters. This collected information can be also used to a fine-grained syntax highlighting.

1 Introduction

Many programmers still use simple text editors to write their programs. They are either used to these editors or they don't have any other, better available development environment for their programming language. It's then hard to avoid syntax errors or typing errors. Furthermore keeping the names of all classes, class members, namespaces, API functions in mind and remembering the names of variables visible in all current scopes is almost impossible. They must browse the source code and all the documentation manually to figure out function parameters and function return value types. They also have to look for comments in the source code and some remarks in SDK and API documentations to check additional information about used functions, classes and other objects.

This is not a major concern when writing small applications developed by small teams of programmers or maybe even by one single programmer. On the other hand large development teams typically produce huge amounts of various software modules with a lot of large documentation files.

Here is a place for an intelligent module used by the source code editor. Using the source code it would process every documentation file and every source code file. It would then suggest the useful information to the programmer in the time they need it.

Additionally, more helpful features can be implemented. For example an intelligent source code editor that understands the programming language can suggest the possible symbols that can be typed. It is also able to replace the typing errors automatically with the correct expression. When the available information is ambiguous it is able to suggest a list of possible options.

Another practical issue is the possibility to navigate through the project's source code files. An intelligent editor knows the structure of the source code. It allows the programmer to navigate to a specific declaration. It can also display declarations within the current file, allowing the programmer to quickly jump wherever they need. It's obvious that working in this environment has to make programming faster and more reliable.

As a part of the Phalanger project 13 we have implemented language integration of PHP dynamic language into the Microsoft Visual Studio environment. This implementation supports most of the common IntelliSense features. This environment already contains an editor with a very good language integration framework, so we designed the *language integration* for PHP and dynamic languages in general.

Since PHP is a dynamic language, there are no static declarations. There can be different variables, classes or class members every time the same source code runs. No declaration has a definite meaning until the runtime and until the declaration is used. However IntelliSense works with type information which in dynamic languages is missing until runtime. Another issue with some of the dynamic languages is the dynamicity of the source file inclusion statement. The compiler doesn't know which source file will be inserted in the place of the inclusion statement. These are issues for the language integration – all successive alternatives on every code fragment have to be populated.

The main contribution of this paper is a description of dynamic language source code analysis and information gathering in a well-arranged way. Unlike the static languages, the analysis of dynamic languages has to estimate the types and possible values from the code fragment semantics. These methods also take into account non-static and non-typed implicit and explicit declarations, so that the resulting suggestions cover most of the developer's common tasks.

2 Integration architecture

The mechanism of code sense, called *language integration*, has to respond on several events. The user is supposed to get help while he is typing or when he moves the mouse cursor over an unknown symbol. These are the basic events which should be followed by the corresponding response. There can be other events like when the source code editor needs to redraw the text it asks the *language integration* for the color of the words. All the events are raised by the source code editor and are handled by the *language integration.*

Source code editor requests the *language integration* to parse the source code every time it was changed and as

a result the *language integration* returns a list of tokens. The tokens have some specified properties. The main properties are the position in the source code, token identifier and the color of the token. Other properties are flags used by the editor to determine possible actions which are supported by this token. These flags enable the language integration to enable or disable the future events on each token.

After parsing the editor can send specific requests. The most important are requests for the list of available declarations (which can be typed at the cursor position) and requests for text information of the token under the mouse cursor, called the tooltip. The first one is called every time the user inserts a character, which is a part of the token with a flag implying that this token wants a list of available declarations. The second one is called when the user moves the mouse cursor over a token with a flag specifying that this token is able to offer textual information.

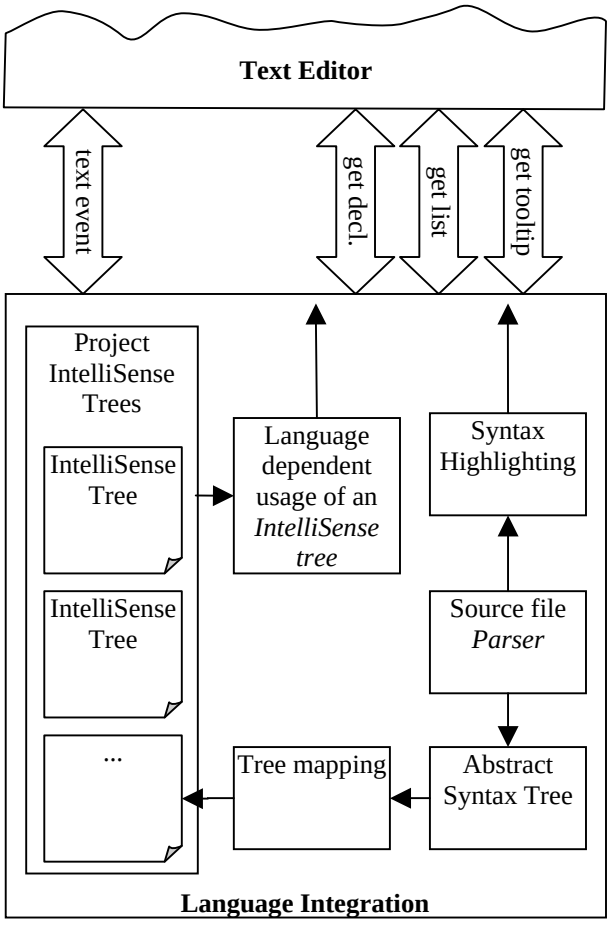


Fig. 1. Language integration architecture.

In this place the intelligence of the *language integration* comes into place. The *language integration* has to manage its own information about every source file in the project, its own in-memory database of symbols. This database is used to find a declaration or a list of declarations corresponding to specified token, where it can represent a finished or an unfinished symbol. Therefore the *language integration* stores the source code in its own *abstract syntax tree* (*IntelliSense tree*) which is built for all files placed in or referenced by the application being edited.

The tree contains all declarations and definitions organized in the same scope structure as they are in its source code. It is optimized for searching for specific declarations visible from the scope in the source code. It is important to note that while the trees of the source code files are being built, the information about every declaration is gathered subsequently using semantic analysis. Every assignment or other usage is caught and forwarded to the appropriate declaration for analysis (the creation of the trees is described in section 3.3).

The *IntelliSense tree* used by the *language integration* is designed to be universal. It can be created from language specific syntax trees or other sources by mapping them into the *IntelliSense tree*. Then the *language integration* works with these trees so it doesn't see any differences between original languages, it accesses the trees in a same unified way. The usage of these trees depends on the source code where the IntelliSense is used, so the specific usages for specific language syntaxes have to be defined there.

Tree of every source file is cached in the memory and its last version is available all the time. This is important because after the user has typed something into the source code, the source code is mostly no longer syntax-valid and therefore a new updated IntelliSene tree can’t be created directly. Then the cached one has to be used and only the node positions should be modified to reflect the new token positions.

3 Code analysis

The main part of the *language integration* mechanism is the ability to manage information about the source code, typically gathered using semantic analysis of the syntax trees. It then has to provide methods for getting the source code structure, defined symbols (declarations) and related information about them.

This symbol information should contain common entities of object oriented languages and data resulting from the semantic analysis too:

Symbol name - Symbol text identifier as it is used in the source code (declared identifier name, e.g. variable name, type name, function name etc.). In dynamic languages it might be a result of an expression evaluation. The name is needed for identifying the symbol and is defined by the user.

Symbol type - There can be many different symbols, each describing different language entity. Fortunately there is some well known set of entity types in programming languages, such as *function*, *variable*, *class*, *namespace* etc. These types are then important to identify symbols more closely. So every symbol should contain its type identifier.

Location - The symbol position is need - a span location in the source code and also its position in the context of other symbols. It's used to enable us easily navigate to it. So there have to be a line number and column number range, and all the symbols have to be stored in a hierarchical structure same as it is in the syntax tree.

Variable type - IntelliSense is practically based on the knowledge of variables type. Hence the symbols describing

variables have to contain their type if it's known and if it could be resolved. Moreover in dynamic languages one variable can carry different type instance in different parts of the code and in different program executions. Also the variable can be initiated dynamically, and its type can be known only at runtime. Therefore the symbol should contain some list of possible types resolved within semantic analysis. But in general it cannot be resolved definitely in dynamic languages; even single program execution won't help in some cases.

Possible values - In dynamic languages sometimes the value of variable has to be known to determine another variable type. Hence it has to be possible to try to evaluate a variable and remember the result. This can be used then for next processes, such as analyzing of indirect variable usages etc.

Description - Text description helps the user to identify the symbol. Also it can provide related information about it. The symbol info may be obtained from user documentary comments, online documentation or other sources. Also related texts can be attached as well as the symbol file location.

Function signature - In case of function declaration, IntelliSense requires its parameters. So the symbol describing a function should provide a list of parameters. They can be described in the same way as regular symbol is - everyone should contain name, type, possible values and text description.

Object members - In object oriented languages, single symbols (especially type declarations and its instances) contain object members (properties and methods). In case of variables it implies from its type. In dynamic languages the type need not to be known, but possible members of a variable can be gathered using semantic analysis. In general any symbol can have members (e.g. class members, function return value members, namespace members like inner declarations etc.).

Extensibility - In the future other data as a result of analyses might be added. The information management (in-memory database of symbols) must have the ability to be extended and must be allowed to store the custom data. Particular symbol types may require their own memory space for additional related information, such as base type of class declaration or return expressions of the function, used for next analyses.

Therefore, in order to make this mechanism usable in general, the structure of a typical source code should be stored in a tree. Actually the tree structure produced by parsers and used by compilers called *abstract syntax tree* (AST) 12 is very flexible and satisfies most of the specified requirements. With some modifications it can be used for gathering information about the source code to be usable for implementing IntelliSense features.

For the purposes of IntelliSense we've designed the universal *IntelliSense tree* which is derived from the notion of *AST*. Classic AST doesn't describe all the information needed by the IntelliSense, like single node descriptions, suggested types or other data used for declaration analysis. IntelliSense trees work then like in-memory databases of symbols, defined by the user in the source code as well as defined by the language specifications. Database of native language symbols is stored in the same way as the user-defined symbols.

Having the specified ASTs as a product of the source code parsing, it is possible to define mappings from these types of AST to the *IntelliSense tree*. All IntelliSense operations use the *IntelliSense tree* to obtain the information about the source code. The tree gathers the information from the original AST and other sources, such as other ASTs, documentation files or user comments. Thanks to this method it is possible to use more source files written in different languages within the one *language integration* mechanism. When a language is integrated with other languages, like PHP is integrated with its extensions or C# with .Net assemblies and XML documentation, the *language integration* then does not see any difference – Any source file can be mapped to this universal *IntelliSense tree* by providing appropriate mapping. Once the tree is built, additional information is added to the tree by subsequent processing (semantic analysis) of the tree or its source data.

3.1 IntelliSense tree architecture

The tree described below is a data structure for managing information about the symbols defined in the source file. In addition to that this in-memory database of symbols stores the results of semantic analysis. Then it offers processed data in a unified way, adapted for usage in IntelliSense implementations.

The structure of the tree has to reflect the structure of the source code. The nodes represent single language entities, such as function bodies, class scopes and general code scopes, which are defined in the source code. They are not initialized until the information about them is requested. Thus the tree is built lazily starting from the root scope, to save computing time. This laziness saves time and memory, only the needed parts of the tree have to been built, the other parts are not touched.

There are two base types of nodes in the tree, describing two main languages entities:

Scope node - The first one is the *scope node*, also used as a root of the tree. It works like a container for other *scope nodes* and *declaration nodes*. Typically it will correspond to some code block which contains other blocks and symbols declaration.

Declaration node - The second one, the *declaration node*, can specify any implicitly or explicitly declared symbol in the source code, such as function declaration, class declaration, namespace, variable, constant, function parameter or even a language keyword. Also a *declaration node* can be internally linked to a *scope node*, as the class declaration with the scope containing member' declarations.

All the nodes among other things also contain their current location span in the source code, because it enables us to find the tree node representing the code scope at specified position and reversely to navigate to the specified declaration location in the source code.

In general the IntelliSense tree describes the source code file in the same way as the syntax tree does. Hence it would be possible to define mapping from ASTs or another source file interpretation to the IntelliSense tree. Note that some elements of typical AST (e.g. function calls) do not need to be used in the process of mapping, but they could be used in the next processes of semantic analysis; so these AST elements should be just remembered in the corresponding scope node.

In addition to this, tree nodes provide methods for language integration support, such as selecting local declarations, getting symbol description or listing object members; therefore their single implementations have to be able to perform analysis of the tree to gather required data. These analyses are only performed when the data are requested.

3.1.1 Scope node

The scope node works like a container for other scope nodes and declarations. It corresponds practically to block of code in the source file. Also it's used as the root node of IntelliSense tree. See Tab. 1 for the base scope node member properties.

Scope property	*Description*
Position	Location span of the scope in the source code string.
Declarations	List of declared symbols (declarations) in the scope.
Scopes	List of other code scopes written within this scope
Inclusions	List of included file names is remembered. IntelliSense trees representing listed files are then used to obtain additional local declarations.
Used namespaces	List of namespaces used in the scope. Their declarations are implicitly visible in the current scope and its child scopes.
Parent scope	Reference to the scope containing this scope, it's empty in case of the root scope.
File name	Path to the file where this scope is placed.
Scope type	Specified scope type, such as code block, function body, class body or namespace body. Not important.

Tab. 1. Scope node specification.

Additionally there are properties important for implementing features of the language service, needed for identifying the scope node and other information used mainly for determining a list of locally reachable declarations:

File name - A path to the source file, where the scope is located, is used for identifying particular declarations. It can be served to the user within the symbols description. Practically the file name value is equal to the file name value of parent scope node if exists. Only the root scope node has to be able to store the file name, other nodes just point to this value.

Used namespaces - A list of namespaces used by the code in this scope (and child scopes) is used for gathering locally reachable declarations. Used namespaces are typically named within the "uses" or "import" statements.

Inclusions - In any code scope other source files can be included, typically using the "include" statement. Therefore the list of included files has to be stored in the scope. Contrariwise the root scope can be included by other scopes (because the root scope represents the whole tree which describes the source file). Then the scope has to be able to get a list of scopes where it is being included. These lists are used for gathering possible declarations visible in the scope, because the declarations from the included files and from the parent scopes are mostly visible too. For resolving the inclusions see 0.

This scope node specification represents an abstract object. Then the node derivations have to be implemented for the specific scopes and specific sources, such as various syntax trees etc. Particular implementations then manage the mapping from the source and semantic analysis of the source. For example the PHP global scope (root scope of the PHP source code) and also the generic PHP code scope have to be implemented. Both of them will use the PHP syntax tree as the source of their child scopes and declarations and as the source of next analyses.

3.1.2 Declaration node

The base type of declaration node is more complex (see Tab. 2 for its interface specification) and it's the main part of the IntelliSense tree, as in-memory database of symbols. It is designed to describe symbols declared in the source code and to manage various information about them. Especially the symbols important for IntelliSense implementations should be supported, such as variables, constants, class declarations, functions or namespaces; declared implicitly or even explicitly.

Therefore this node contains not only the symbol name and text description, but also the information for specific symbol type must be present; such as possible object members or function signature. There are also many properties which describe the symbol type, possible values of variable, object instance members, object static members, symbols in a namespace, array items or indirectly used symbols. This is solved using following members of the base declaration node:

Name - The name of the symbol, as it is used in the source code.

Description - Known text information about the declaration (symbol description) can be taken from various meta-data such as comments near the declaration or the online documentation; the file name and position where the declaration is placed can be appended too. Finally it contains all the available information about the declaration. The *language integration* can then offer this information to the user.

Symbol type - The declaration is also described by its *symbol type*; it describes what specified node declares. There are many symbol types like variable, constant, function, namespace, class and keyword. It is used by the *IntelliSense integration* to select appropriate symbols in the specified context. It is also used for displaying an appropriate icon next to the symbol name, when a declaration list is populated.

Visibility - Especially objects within a class declaration have defined a visibility (access modifier). It tells the compiler (and in this case the language integration) in what code scopes and how the symbol is reachable. Typical access modifiers are public, protected, private, static and constant. The resulting *visibility* flag can be their combination.

Function signature - In case of function, all its signatures have to be stored. This information has to be then available through the IntelliSense as the method parameters lists. Also function parameters can be used during the analysis of the function body code, to declare local variables from parameters and to determine their possible type or values.

Declaration node property	*Property description*	
Position	Location span in the source code. Used for navigating to the declaration.	
Name	The text name of the symbol.	
Analyzers	List of symbol analyzers. Primarily used for obtaining members and values.	
Parent scope	Reference to the scope node where the declaration is placed originally.	
Symbol type	The type of the declaration, describes what the symbol means.	
	Type	A class or structuralized type.
	Variable	A variable declaration.
	Constant	A constant declaration.
	Namespace	Namespace container.
	Function	Function declaration.
	Keyword	A language keyword.
Description	Text description of the symbol.	
Visibility	Declaration visibility flags, e.g. *public*, *protected*, *private* or *static*	
Function signature	List of signatures used for function call assistance; every signature defines a list of parameters with parameter name, type declaration and parameter description.	

Tab. 2. Declaration node specification.

Analyzers - The information about every symbol can be obtained from many various expressions. Symbol values, its type members or dynamically added members imply from single symbol usages. The declaration node then has to collect these usages, analyze them and offer the results (object members, static members, possible values and array items declarations). Therefore an analyzer object is created for every symbol usage. There must be different implementations of analyzer for different symbol usages, e.g. assignment, add operation, function call etc. These analyzers are then saved within the declaration node. Whenever its members or values are needed, all the requested results from analyzers are unified and returned. This separation saves CPU time and memory, because a single analyzer can be shared by many declaration nodes and the analysis is performed only if some information is requested. For the analyzer object description see 3.1.3 and **Tab. 3**. Analyzer generalizes the type resolving problem. They are used instead of types to obtain symbol information by subsequent processing of the code, even dynamically added object members are then caught and they are able to be displayed by IntelliSense.

Most of the properties might be described as an expression dynamically and the particular declaration node implementation (an analyzer) has to analyze them and try to evaluate them on demand. That's an issue with dynamic languages, where even the declarations name can be described as an expression which's value is known definitely only at runtime (see also 60).

Using the analyzers, the declaration node may offer a list of possible values. It's used when the node describes a variable. These values are important for resolving dynamic indirect usages and indirect function calls. The values are expressed as expressions which have to be evaluated to obtain the result. These results are used then as another declaration name at runtime or they can be used as a part of symbol description. See 3.2 how the expression evaluation can be bypassed and why.

3.1.3 Analyzers

The declaration node needs to manage its possible object members and its possible values. In static languages the members are typically obtained through the object type. But in dynamic languages they can be collected from various sources; the *analyzers* are used for that. The *analyzer* is a helper abstract object (**Tab. 3**) which offers a list of object members (in a form of declaration nodes) and a list of string values, using the analysis of a part of the source code (typically specific expression) and the knowledge of the rest of the analyzed code.

Analyzer property	*Property description*
Values	List of possible text values.
Object members	Object instance member declarations. (e.g. after "->")
Static members	Static member declarations. (e.g. after "::")
Namespace members	Declarations in the namespace. (e.g. after ":::" in Phalanger)
Indirect declarations	Declarations used from indirect usages of this object.
Array items	Analyzers of array items.

Tab. 3. An analyzer interface.

Every *declaration node* contains the list of *analyzers* and every *analyzer* gathers required data from custom source, such as an AST expression node or .NET type object. There are many possible sources and therefore one or more analyzers are used within one declaration node. For every symbol usage, an analyzer is created and added into its declaration node. Also one analyzer, which describes specific expression, can be shared by more declaration nodes. It saves memory and CPU time; in addition to that it partially solves the type resolving problem through the tracking of a variable value.

An analyzer is a certain replacement for the object type. It manages only object properties, such as its members and its possible values. Every symbol is described by set of analyzers, where each one gathers data from different source. Properties of the object are then able to be extended by adding a new analyzer into the set. Analyzers are reusable, it's then possible to inherit or copy properties of another object by referencing its existing analyzer.

3.2 Dynamic information

With dynamic languages it's necessary to perform semantic analysis to collect information about each declaration from the whole source code semantics. Therefore even the particular expressions have to be processed and used for declaring new or modifying existing declarations. These declarations are stored in the IntelliSense tree, where they can be easily found or created and modified.

The analyzers are then used for expanding existing declaration with new information. If there are new object members or variable values used in the source code, a new custom analyzer containing these extensions is added into the appropriate declaration node.

Also a declaration can be declared or used in an expression as its evaluation. Even the compiler doesn't know the result of every expression, so it cannot know all the declarations and calls that will be possible to execute. But the language integration has to offer all possibilities. Therefore it has to guess what can be possibly written into the source code.

The only exact way how to list all the possibilities is to execute the source code. But the execution has usually many side effects, and every time the program runs it can produce different results depending on the current hardware, time, files, random generators etc. Hence making the execution is not a good idea.

The particular expressions have to be bypassed somehow to get an approximate result and meaning. Some simple expressions can be actually executed, other ones can match predefined patterns and then semi-evaluated. The rest of the expressions can be ignored with the awareness of incomplete information stored in the IntelliSense tree.

3.2.1 Type resolving

IntelliSense is practically based on types. Among other things in dependence on symbol type (e.g. variable, class or function return value) it offers members placed within this type. But in dynamic languages one symbol can represent instances of more types in different program executions and different locations. In addition to that the symbol can be extended by adding new member declarations dynamically besides the types members. The single type then is not sufficient for getting all possible symbol members. See 3.1.2 and 3.1.3 how the symbol members are managed.

Hence the language integration must watch for particular symbol usages. Most important are all the assignment expressions where the symbol gets a new value. The type resolving is based on this; within the assigning all the properties of the right value are added (without duplicities) into the declaration node describing the left value. The single properties are expressed within the analyzer objects, they can be then simply referenced and added into the declaration node. Just note that right value must be resolved first before using its analyzers. This mechanism works even for dynamically added members, because usage of a non-existing member extends the symbol properties with a custom analyzer. It is then referenced within the assignment operation with all the other analyzers.

This way the symbol properties (especially its members) are collected by tracking the symbol assignments recursively, up until its possible values source (e.g. "new" expression, literal or some irresolvable expression). The symbol is then described by all the analyzers which could be resolved. They can then be used to obtain all the possible object members.

In case of function calls, the return statements must be found in the function definition body. Then the expressions that are specified within these statements are used. The expressions are analyzed and the resulting analyzer objects are used for describing the function call symbol.

3.2.2 Expression patterns

Every pattern defines a small sub-tree of the *AST* (expression without values) with the semi-evaluation code which returns an *analyzer* (3.1.3). This *analyzer* can then contain approximate results for the given expression in its values list or it can offer a list of object members – this depends on the specific pattern implementation.

Method	*Returns*	*Description*
IsMatch	True of False	Checks if the given AST matches the pattern
GetAnalyzer	Analyzer	Uses own semi-execution routine and creates an analyzer with results

Tab. 4. The *expression pattern* interface.

The expression patterns (**Tab. 4**) are defined explicitly and their usage is very flexible and works well. For a specific language there are numerous frequently used expressions. These can be recognized by the *language integration* using the *expression patterns* and then they can be semi-evaluated. Thanks to this mechanism, safe expressions with no side effects can be processed and analyzed.

It is then sufficient to define patterns e.g. for indirect usage, simple expression evaluation and other basic dynamic and non-dynamic operations; see Tab. 5 for an example. Expressions which match the defined patterns are transformed to the specific analyzer by the *language integration*. Then the *analyzers* can be used to obtain a result value or expression object members.

Pattern	*Result analyzer*
Variable use	Copy of the variable declaration analyzers.
Literal	Literal value.
new TypeName	TypeName declaration public and static members.
Indirect Variable use	Members of declarations with the name of the variable values.

Tab. 5. Patterns and their semi-execution procedure**.**

3.2.3 Dynamic members

In dynamic languages it is usually possible to add new members to an object during runtime. Wherever an object member is used in the source code, it has to be checked whether it is an existing member of the parent object declaration or not. If not, this dynamic member has to be added into the list of object members.

The *custom analyzers* are used for these purposes. A new *custom analyzer* with this new dynamic member declaration is created and added into the parent object declaration. As a result the origin declaration members are mixed with other members. In this case the dynamically used member is added to the list of current object members.

3.3 Mapping into IntelliSense tree

Creation of the *IntelliSense tree* depends on the original language. The source has to be analyzed and then particular language elements are converted into the tree nodes. The typical process consists of the source code parsing and creation of the *AST*. The visitor pattern is used on the *AST* and source code elements are converted into the corresponding implementations of *IntelliSense tree* nodes (see Tab. 6 for simplified mapping). This process is re-executed whenever the source is changed.

The tree can be created from other sources, like .NET assemblies. In this case, objects from the assembly are mapped into the *declaration nodes*, which are organized in the structure of the namespaces. Whatever the sources are, the *IntelliSense tree* implementation unifies them. Once built, additional information is added to the tree by subsequent processing (semantic analysis).

Just note that in the dynamic language, even the code scopes and single expressions have to be processed, because they could contain some new implicit declaration or they could define some known symbol extension. In non-dynamic languages the process is much simpler, because only the parts of code which can actually contain some explicit declarations can be used.

The process of mapping doesn't need to be done immediately. In fact a specific tree node can remember a reference to the origin source element and the mapping can be done on-demand and not before the content of the tree node is used. Also the process can be easily parallelized by mapping different tree nodes into different threads.

3.4 Linking trees

It is possible to use symbols declared in other source files referenced from the source code. Therefore the *scope nodes* contain a list of IntelliSense trees which are included.

AST	*Mapped IntelliSense Tree*
Source code - file name - file content	create Scope node - file name = source file name - parent scope is empty Process the file contents, put created declarations and scopes here
Function - name - signature - code scope	create Declaration node - name = function name - type = Function - function signature - analyzer of the function - function description from comments
	create Scope node - parent scope file name - function body code scope
Variable use - name - usage	find or create declaration node with specified name. Declaration node - name = variable name using patterns - type = Variable
Class declaration - name - members	create Declaration node - name = class name - type = Type - analyzer = class analyzer offering its members
Assignment - A = B	B' = GetAnalyzer of B A' = declaration node of A Add analyzer from B' to A'

Tab. 6. Mapping of an *AST* into *IntelliSense Tree*

The included file name must be resolved to obtain the right tree. This file name is a generic expression within the include statement, which can only be fully evaluated during runtime in dynamic languages. Therefore the expression has to be semi-evaluated using patterns (see 3.2.2). Fortunately, in most cases, the target file name is expressed as a simple string literal, but otherwise the semi-evaluation might fail; then the corresponding tree is not linked with the scope and the declarations within the included file are not available.

The list of visible declarations for the specified scope can be collected recursively from the current scope, parent scope and from trees which are included in the current scope. Because the declarations located in the including scopes are also visible in the included files then also the scopes which include the current scope should be used. This is needed to obtain every possible declaration (see 4.1).

3.5 Tree cache

The *IntelliSense tree* of every source file is stored in the global memory cache. Then unchanged source codes need not be reparsed and the last version of its tree can be used.

Each source file has its own tree built by the *language integration.* This tree is rebuilt every time a source file is changed. Hence the *language integration* is trying to process source codes which are not necessarily syntax valid. This occurs when the source code is just being

modified by the user. This means that *IntelliSense tree* can't be created directly. But the tree is needed even for non-syntax valid source codes to keep the IntelliSense working.

For these purposes the last cached tree of the processed source file is used, but with some modifications. When the source code is being edited, the source positions of the scopes and declarations in the cached tree do not reflect their current positions in the source code. Hence with every added or removed line or character in the source code, the positions of the tree nodes have to be moved in the same way by the same amount of characters or lines. Let us note that only the positions located further from the change must be corrected. When some part of the source code is removed, tree nodes in this area should be removed too. Then this modified tree can be used even for the changed syntax invalid source file. Only the new code added within the syntax invalid code is not described here.

4 IntelliSense implementation

The *IntelliSense tree* provides structuralized source code scopes and declarations with additional information. That's used for obtaining a list of available declarations in the specified code scope. Optionally this list can be filtered to get one or more specified declarations which match a specific word. It can be used to get the additional information about that word as well.

4.1 Local declarations

A list of visible local declarations depends on the current language. In most cases the declarations can be collected from the current code scope and parent code scopes subsequantly. In addition to that included trees root scope and optionally scopes which include the current scope might be used too. The last one allows the user to work with declarations which may be visible, if the source code file is executed as an inclusion.

If the language supports the namespaces, all of *IntelliSense trees* in the project should be processed to find namespace declarations imported in the current code scope. Then also the declarations in these namespaces are listed as local visible declarations. A typical selection process is depictured on Fig. 2.

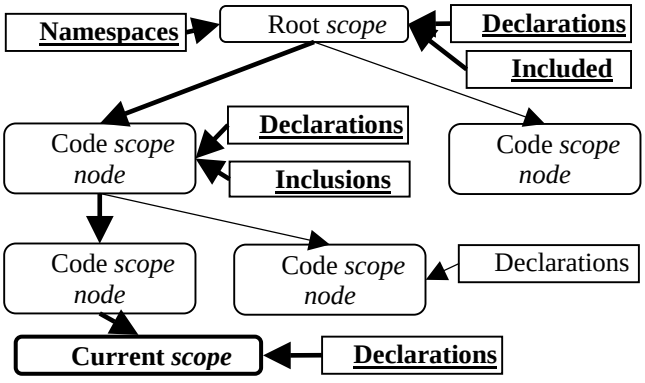


Fig. 2. Local declaration example.

4.2 Expression declarations

To enable functionalities of the IntelliSense, such as word completion and tooltips, the *language integration* needs to find declarations which correspond to particular expressions. Not all expressions have to be recognized; only variable usages, type usages or function calls are important. The issue with these expressions is that they can be used as a part of some expression chain, optionally with array indexers and indirect usages.

First of all the expression has to be parsed and transformed into some kind of a unified data structure. Typically this is an expression under the cursor and it is transformed into a part of the language-specific *AST*, which is the appropriate unified data structure describing the syntax of the expression with the already defined visitor functionality.

The created *sub-AST* describes the way how to obtain the matching declarations from the *IntelliSense tree*. A visitor can be used on the AST. Then all the particular AST elements have their own selection method in the visitor. For the root expression of the chain, all local declarations matching the AST element will be selected. Similarly for the chain member expressions, member declarations of the previously selected declarations matching the AST element will be selected and will replace the previous ones. Table 7 contains an example of such processing.

For an array or indirect access it's a very similar procedure. The previously selected declarations are used and their analyzers with lists of array analyzers or indirect declarations will replace the current results.

Declarations remaining on the end of the process correspond to the expression.

$A[123+456]->C		
Variable $A	Indexer […]	Member ->C
Local declarations with the name "A" and type Variable	Select all array analyzers from *"$A"*	Select all object members with the name equals "C" from *"$A[…]"*

Tab. 7. Example of an expression chain and its declaration nodes.

4.3 Word completion

An automatic completion of the words is basically the same process as the process of finding expression declarations. When an expression is being written the part written so far is parsed to obtain the syntactic structure (*sub-AST*). This structure is then used to obtain a declaration list of elements, which match the symbol under the cursor (the syntactic substructure). All the declarations that are superset of the symbol are listed. They don't necessarily have to be the same. If there is no expression to process, meaning that user wants to auto-complete an empty space, the list of all local declarations can be offered to display all the possibilities which could be used.

The resulting list of declarations is offered to the user. In case of only one item in the list, the word before the cursor can be auto-completed.

4.4 Special cases

The *language integration* should modify the offered lists by the current code context. The behavior of the modifications depends on the language.

For example the "new" keyword before the expression can be detected and the offered list should be filtered to display only type declarations. Also some special keywords can be added into the list.

5 Related Work

An IntelliSense is becoming a common part of modern source code editors. There are working commercial solutions even for dynamic languages. Also low-level solutions for own languages could be used.

5.1 VS.Php

VS.Php 4 is the commercial PHP language integration for the Visual Studio 2005 and 2008. At present it offers the IntelliSense and syntax highlighting for PHP, HTML, JavaScript, CSS and XML. Also the integration is able to use third party software for debugging the PHP source code.

It's a complete commercial solution for developers. However, this product does not have the dynamic operations resolved. It ignores object dynamic members and also it does not handle indirect calls. The developer cannot easily implicitly declare new variable, because anything the developer types is changed into something already declared.

Finally the file inclusions are ignored completely. The language integration just offers everything what is explicitly declared in the whole project folder.

5.2 Microsoft DLR

Microsoft DLR 5 is an open source project enabling compilation of custom dynamic languages. Basically all the implementation leans on building the *DLR expression* from the source code. In the future once the building of that expression will be managed it will also automatically serve simple *language integration* solution with text highlighting and IntelliSense capabilities.

5.3 JavaScript IntelliSense

Microsoft Visual Studio has an integrated support for JavaScript dynamic language 6 which tries to resolve the types of variables. The integration supports IntelliSense and offers even the dynamic object members and also all the current HTML entities are included in complete DOM model.

5.4 Non-dictionary predictive text

Text oriented methods without a dictionary work for any source code or generally for any text. These techniques 81011 do not know the specific language. Only words from existing source files are used. Then while writing the text the editor just offers the list of known words which contain the word under the cursor. A set of characters or regular expressions have to be defined only to let the editor know what the simple word could be.

These simple methods are very fast; they should help and speed up the process of programming. Also it is language independent and it's easy to implement. But when using this solution the programmer has no more information about the source code and particular declarations. The editor lacks any intelligence so it offers also meaningless words combinations.

6 Conclusion and future work

This paper describes methods used for gathering information from the source code, so they can be presented in a well-arranged way. Originally it was targeted on the PHP dynamic language, but the data structures are designed to support both static and dynamic languages in general and to integrate them.

As a part of the work we have implemented the language integration for Microsoft Visual Studio, called Phalanger IntelliSense, where the presented techniques are used. The integration processes the PHP language and integrates it with the .Net static assemblies. The text prediction and source code analysis works well even for non static expressions without a need of execution.

Future enhancements of the Phalanger IntelliSense should work with more well known expressions. Also the language specific methods, such as mapping functions, could be programmed in a declarative way, so it would be easy to integrate more languages quickly.

References

1. Visual Studio SDK: Language Services, http://msdn.microsoft.com/en-us/library/bb165099(VS.80).aspx
2. IronPython project, http://www.codeplex.com/IronPython
3. Phalanger compiler, http://php-compiler.net/
4. jcx software: VS.Php, http://www.jcxsoftware.com/vs.php
5. Microsoft DLR, http://www.codeplex.com/dlr
6. VisualStudio JScript IntelliSense, http://msdn.microsoft.com/en-us/library/bb385682.aspx
7. IntelliSense, http://en.wikipedia.org/wiki/IntelliSense
8. Predictive text, http://en.wikipedia.org/wiki/Predictive_text
9. T. Lindahl, K. Sagonas: *Practical type inference based on success typings.* Proceedings of PPDP, ACM, Venice 2006.
10. S. MacKenzie: *Keystrokes per character as a characteristic of text entry techniques.* Proceedings of MobileHCI 2002.
11. O'Riordan et. al.: *Investigating text input methods for mobile phones.* J. Computer Sci, I (2):189-199, 2005.
12. J. Jones: *Abstract syntax tree implementation idioms*. Pattern Languages of Programs 2003, Illinois.
13. A. Abonyi., D. Balas, M. Beno, J. Misek and F. Zavoral.: *Phalanger improvements.* Technical Report, Department of Software Engineering, Charles University in Prague, 2009.

Restartovací automaty se strukturovaným výstupem a funkční generativní popis*

Martin Plátek[1] and Markéta Lopatková[2]

[1] KTIML, MFF UK, Praha, martin.platek@mff.cuni.cz
[2] ÚFAL, MFF UK, Praha, lopatkova@ufal.mff.cuni.cz

Abstrakt *Tento příspěvek se zabývá restartovacími automaty, které tvoří formální rámec pro Funkční generativní popis češtiny. Restartovací automaty pracující současně se čtyřmi rovinami jazykového popisu jsou určeny k tomu, aby prováděly redukční analýzu českých vět a umožnily tak odvodit závislostní vztahy ve větě z možných pořadí jednotlivých redukcí. Standardní model restartovacích automatů je zde obohacen o strukturovaný výstup, který umožňuje budovat tektogramatickou závislostní strukturu odvozenou z redukční analýzy. Restartovací automaty se strukturovaným výstupem představujeme v tomto příspěvku poprvé.*

1 Úvodní poznámky

Funkční generativní popis (FGP) je systém pro popis gramatiky češtiny, který se rozvíjí od 60. let 20. století, viz zejména [10,11]. Jeho základními charakteristikami jsou stratifikační a závislostní přístup k popisu jazyka.

Stratifikační přístup rozčleňuje popis jazyka do několika rovin, kde každá z rovin poskytuje množinu úplných zápisů vět (má svůj slovník a svou syntax). Zde se soustředíme zejména na dvě krajní roviny, na nejnižší rovinu označovanou jako rovina formy (dále w-rovina) a na nejvyšší rovinu, rovinu významu (dále t-rovina).[1] Zápis věty na nejnižší rovině si můžeme zjednodušeně představit jako řetězec slovních tvarů a interpunkce. Nejvyšší rovina zachycuje jazykový význam věty pomocí tektogramatické reprezentace, jejímž základem je závislostní strom, viz [12,8]. Tato rovina specifikuje zejména tzv. hloubkové role, gramatémy a aktuální členění.

Závislostní přístupy využívají k popisu syntaxe přirozeného jazyka závislostní struktury. V rámci FGP je význam věty zachycen jako (tektogramatický) závislostní strom, ve kterém jsou plnovýznamová slova reprezentována jako uzly stromu (uzel je zde komplexní jednotka zachycující lexikální význam a další významové charakteristiky); vztahy mezi jazykovými jednotkami jsou zachyceny jako hrany mezi odpovídajícími uzly, viz zejména [6].

V článcích [2,3] jsme se věnovali formalizaci metody redukční analýzy (RA). *Redukční analýza*, metoda, která modeluje práci lingvisty při rozboru vět přirozeného jazyka, je založena na postupném zjednodušování rozebírané věty (více v oddíle 2). Naším hlavním záměrem je zařazení redukční analýzy do vhodného formálního rámce pro FGP tak, aby byly zřetelné souvislosti mezi redukční analýzou a množinou závislostních struktur. Takovým rámcem mohou být *restartovací automaty*. Ukázali jsme, že pomocí restartovacích automatů lze přirozeně splnit hlavní požadavky na formální rámec pro FGP,[2] tedy požadavky na rozlišení množiny správně utvořených vět daného přirozeného jazyka (zde ji budeme označovat L_w), dále na množiny významových reprezentací TR a na stanovení relace SH mezi L_w a TR popisující vztahy synonymie a homonymie, viz [3] (tam lze též nalézt rozsáhlou bibliografii).

Standardní model restartovacího automatu je definován jako akceptor, tj. zařízení rozhodující, zda zadaný řetěz patří do jazyka definovaného automatem, viz zejména [5]. V práci [3], kde je FGP modelován jako formální překlad, byl využit koncept restartovacího automatu jako převodníku vstupního řetězu symbolů na výstupní řetěz symbolů. Takový model ovšem neodpovídá zcela přímočaře závislostně orientovaným přístupům k popisu přirozeného jazyka, se kterými pracuje formální lingvistika, neboť ty explicitně užívají stromové struktury pro popis významu. Přirozeným požadavkem je tedy obohacení formálního modelu tak, aby přímo generoval stromové struktury. Proto zde ke standardnímu modelu restartovacího automatu přidáváme strukturovaný výstup – chápeme ho tedy jako převodník, jehož výstupem jsou složitější tektogramatické závislostní stromy odvozené z redukční analýzy (oddíl 3).

2 Principy redukční analýzy

Redukční analýza dovoluje zkoumat závislostní strukturu věty a přitom přiměřeně zachytit její slovosledné varianty. To je nezbytné zejména pro jazyky s relativně

* Tento příspěvek vznikl za podpory GA AV ČR v rámci programu Informační společnost, projekt č. 1ET100300517.

[1] Další roviny, m-rovina a a-rovina, zachycují morfologickou, resp. povrchově syntaktickou stavbu věty.

[2] Tyto obecné požadavky na formální popis přirozených jazyků přebíráme z práce [7].

volným slovosledem, jako je čeština, kde závislostní struktura a slovosled souvisí pouze volně.

Redukční analýza je založena na postupném zjednodušování (vstupního) zápisu zpracovávané věty – každý krok RA spočívá ve vypuštění vždy alespoň jednoho slova (symbolu) vstupního zápisu, které může (ale nemusí) vést k přepisu nějakého dalšího symbolu. Tento postup umožňuje určit jazykové závislosti, podrobněji viz [2].

Při každém kroku RA je požadováno splnění několika principů, které zajišťují lingvisticky adekvátní rozbor dané věty – je to především princip zachování syntaktické správnosti (redukovaná věta musí být gramaticky správná) a princip zachování významové úplnosti věty.

Zdůrazněme, že během RA se zpracovává vstupní věta, která je doplněna metajazykovými kategoriemi ze všech rovin FGP – kromě slovních forem a interpunkce tedy obsahuje také informace morfologické, (povrchově) syntaktické a tektogramatické. Tato vstupní věta je postupně zjednodušována tak dlouho, dokud se nedospěje k *základní predikační struktuře*, která je typicky tvořena hlavním predikátem věty (finitním slovesem) a jeho (valenčními) doplněními.

Doplňme zatím alespoň neformálně, že každý krok RA musí být v jistém smyslu minimální – jakýkoliv potenciální redukční krok, který by se týkal menšího počtu symbolů ve větě, by vedl k syntakticky nesprávné nebo významově neúplné větě, a porušoval by tak výše uvedené principy RA.

Podívejme se na RA konkrétní věty (1) (viz též [3], kde je podrobně popsán způsob reprezentace slovosledu a zpracování jednotlivých typů závislostí). Na této větě budeme ilustrovat postupnou redukci a z ní vyplývající budování tektogramatického závislostního stromu.

Příklad 1 Našeho Karla plánujeme poslat na příští rok do Anglie. (viz [11], p. 241, rozvinuto)

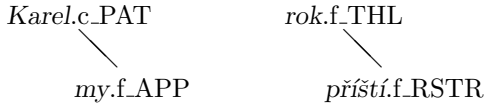
Zápis této věty (mírně zjednodušený) na všech čtyřech rovinách FGP je uveden na obrázku 1 – každý sloupec tabulky odpovídá jedné rovině jazykového popisu (po řadě *w*-, *m*-, *a*- a *t*-rovina, viz oddíl 1); řádky tabulky reprezentují jednotlivé slovní formy a interpunkční znaménka (vždy jeden nebo více řádků odpovídají jedné povrchové slovní formě či interpunkci, více viz [3]).

V prvních krocích RA můžeme v libovolném pořadí redukovat slova ‚*našeho*' a ‚*příští*' (a celé řádky, které těmto slovům odpovídají) – výsledná redukovaná věta je správně gramaticky utvořená a je významově úplná. Z tohoto pozorování vyplývá, že slova ‚*našeho*' a ‚*příští*' jsou vzájemně nezávislá a rozvíjejí některá ze slov v redukované větě, podrobněji viz [2].

w-rovina	*m*-rovina	*a*-rovina	*t*-rovina
Našeho	*můj*.PSMS4	Atr	
Karla	*Karel*.NNMS4	Obj	
			[*my*].t_ACT
plánujeme	*plánovat*.VB-P-	Pred	*plánovat*.f_PREDVF1
			Karel.c_PAT
			my.f_APP
			[*my*].t_ACT
poslat	*poslat*.Vf- - -	Obj	*poslat*.f_PAT.VF2
na	*na*.RR- - 4	AuxP	
příští	*příští*.AA4IS	Atr	
rok	*rok*.NNIS4	Adv	*rok*.f_THL
			příští.f_RSTR
do	*do*.RR- - 2	AuxP	
Anglie	*Anglie*.NNFS2	Adv	*Anglie*.f_DIR3
.	..Z: - - -	AuxK	

Obr. 1. Reprezentace věty (1) na čtyřech rovinách FGP.

Na základě povrchově syntaktických a morfematických kategorií lze zájmeno ‚*našeho*' a přídavné jméno ‚*příští*' analyzovat jako doplnění rozvíjející substantiva ‚*Karla*', resp. ‚*rok*'. Rozebíraným redukcím tedy na *t*-rovině odpovídají závislostní hrany mezi příslušnými uzly, tedy ‚*Karel*.c_PAT—*my*.f_APP' a ‚*rok*.f_THL—*příští*.f_RSTR', graficky znázorněno:



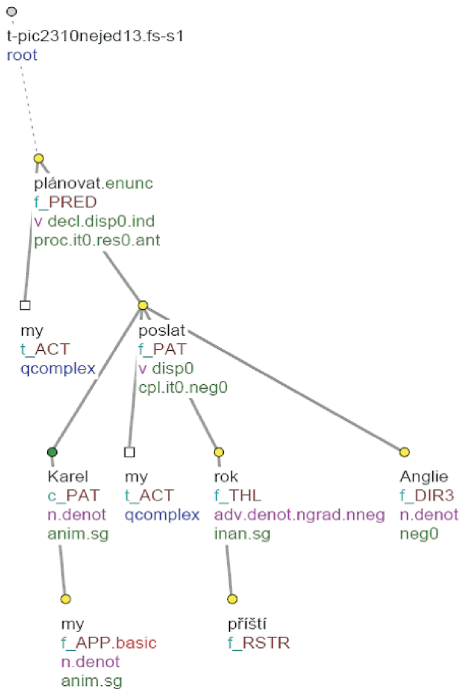
V dalším kroku lze redukovat předložkovou skupinu ‚*na rok*', která je na *t*-rovině reprezentována symboly ‚*rok*.f_THL' (jejím vypuštěním neporušíme gramatičnost ani úplnost věty). Podle principů RA tato předložková skupina rozvíjí některý z dosud neredukovaných větných členů; na základě slovosledu ji interpretujeme jako doplnění slovesa ‚*poslat*' a na *t*-rovině tak vytváříme závislostní hranu ‚*poslat*.f_PAT.VF2—*rok*.f_THL'.

Obdobným způsobem postupujeme dále a na základě jednotlivých redukcí budujeme závislostní reprezentaci věty (1) – výsledný závislostní strom je na obrázku 2.

3 Restartovací automaty a FGP

V této části postupně zavádíme formální model redukční analýzy FGP a popisujeme jeho vlastnosti.

V prvním oddíle 3.1 jsou zavedeny *jednoduché restartovací automaty (*sRL*-automaty)*, které slouží jako výchozí pojem. Tam jsou zavedeny i tzv. metainstrukce, sloužící k přehlednému zápisu restartovacích automatů.



Obr. 2. Tektogramatická reprezentace pro větu (1).

Tento standardní model restartovacího automatu obohacujeme v následujícím oddíle 3.2 tak, aby byly během vypouštění a přepisování z vypouštěných symbolů budovány závislostní struktury.

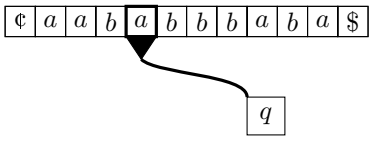
Třetí oddíl 3.3 zavádí automaty typu F_{09} jako speciální případ obohacených sRL-automatů. Jejich pomocí se postupně zavádějí následující základní pojmy pro automat M_{09} typu F_{09}:

- *charakteristický jazyk* $\mathsf{L}_C(\mathsf{M}_{09})$ odpovídající vstupním větám obohaceným o metajazykové informace;
- jazyk správně zapsaných (českých) vět L_w reprezentující nejnižší rovinu FGP;
- dTR-*jazyk* $\mathsf{dTR}(\mathsf{M}_{09})$ odpovídající množině významových stromových struktur daných FGP;
- dTR-*charakteristická relace* $\mathsf{TSH}(\mathsf{M}_{09})$ popisující synonymii a homonymii;
- TSH-*syntéza*, která představuje FGP jako generativní zařízení, tedy v jeho původní úloze;
- TSH-*analýza*, která plní úlohu syntakticko-sémantické analýzy realizované na základě FGP.

3.1 Restartovací t-sRL-automat

Nejprve neformálně popíšeme restartovací automat, který využíváme k modelování redukční analýzy RA.

Jednoduchý RL-*automat* (sRL-*automat*) M je (v obecném případě) nedeterministický stroj s konečně



Obr. 3. Restartovací automat.

stavovou řídící jednotkou (množinu stavů označujeme Q), konečným charakteristickým slovníkem Σ a pracovní hlavou. Tato hlava, která operuje na pružné pásce (seznamu) ohraničené levou zarážkou ¢ a pravou zarážkou \$, může číst a zpracovávat právě jeden symbol, viz obrázek 3.

Pro vstupní větu $w \in \Sigma^*$ je vstupním zápisem na pásce ¢w\$. Automat M začíná výpočet ve svém startovním stavu q_0 s hlavou na levém konci pásky a ve výhledu má levou zarážkou ¢. Automat M může provádět následující kroky:

- *posuny doprava* a *posuny doleva* – posouvají hlavu o jednu pozici doprava, resp. doleva a určují následný stav;
- *vypouštěcí kroky* – vypouštějí symbol pod pracovní hlavou, určují následný stav a posunují hlavu na pravého souseda vypuštěného symbolu.

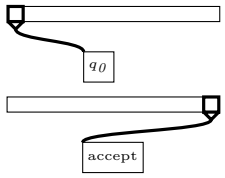
Na pravém konci pásky se M buď zastaví a ohlásí *přijmutí*, nebo se zastaví a ohlásí *odmítnutí*, nebo *restartuje*, což znamená, že přemístí hlavu na levý konec pásky a přepne se znovu do startovního stavu (obrázek 4). Požadujeme, aby M před každým restartem, resp. mezi každými dvěma restarty provedl alespoň jedno vypuštění.

Pokračujme poněkud formálněji a přesněji. Jednoduchý RL-automat je šestice $M = (Q, \Sigma, \delta, q_0, ¢, \$)$, kde:

- Q je konečná množina stavů;
- Σ je konečný slovník (charakteristický slovník);
- ¢, \$ jsou zarážky, které nepatří do Σ;
- $q_0 \in Q$ je startovací stav;
- δ je přechodová relace, tedy konečná množina instrukcí tvaru $(q, a) \rightarrow_M (p, Op)$, kde q, p jsou stavy z Q, a je symbol ze Σ a Op je operace; jednotlivé operace odpovídají jednotlivým typům kroků (posun doprava, posun doleva, vypuštění, přijímání, odmítání a restart).

Konfigurace jednoduchého restartovacího automatu M je řetěz $\alpha q \beta$, kde $q \in Q$, a buď $\alpha = \lambda$ a $\beta \in \{¢\} \cdot \Sigma^* \cdot \{\$\}$, nebo $\alpha \in \{¢\} \cdot \Sigma^*$ a $\beta \in \Sigma^* \cdot \{\$\}$; zde q reprezentuje momentální stav, $\alpha\beta$ je momentální obsah pásky a rozumí se, že hlava čte první symbol (slovo) z β. Konfigurace tvaru $q_0 ¢ w \$$ je nazývána *restartovací konfigurací*.

1. restartovací konfigurace
2. přijímací konfigurace



Obr. 4. Významné konfigurace.

Povšimněme si, že libovolný výpočet M se skládá z fází. Fáze, zvaná *cyklus*, začíná restartovací konfigurací, hlava provádí posuny a vypouští podle svých instrukcí do chvíle, kdy je proveden restart a kdy tedy nastane restartovací konfigurace. V případě, že fáze nekončí restartem, nazýváme ji *koncovkou*, neboť nutně končí zastavením. Požadujeme, aby každý cyklus automatu M provedl alespoň jedno vypuštění – tedy každá nová fáze začíná na jednodušším řetězu než ta předchozí. V koncovce se může, ale nemusí vypouštět nebo přepisovat. Používáme notaci $u \vdash^c_M v$ pro označení cyklu či (cyklové) redukce pomocí M. Cyklus začíná v restartovací konfiguraci $q_0 ¢ u \$$ a končí restartovací konfigurací $q_0 ¢ v \$$; relace $\vdash^{c^*}_M$ je reflexivním a transitivním uzávěrem $\vdash^c_M$.

Řetěz $w \in \Sigma^*$ je *přijímán* pomocí M, pokud existuje přijímací výpočet startující z počáteční konfigurace $q_0 ¢ w \$$. Jako *charakteristický jazyk* $\mathsf{L}_C(M)$ označujeme jazyk sestávající ze všech řetězů přijímaných pomocí M; říkáme, že M *rozpoznává (přijímá) jazyk* $\mathsf{L}_C(M)$. Jako $\mathsf{S}_C(M)$ označujeme *jednoduchý jazyk* sestávající z řetězů, které M přijímá pomocí výpočtů bez restartu (koncovkami). $\mathsf{S}_C(M)$ je regulárním podjazykem jazyka $\mathsf{L}_C(M)$, viz [4].

Takový sRL-automat M, který používá nejvýše t vypuštění v jednom cyklu ($t \geq 1$) a zároveň libovolný řetěz z jazyka $\mathsf{S}_C(M)$ nemá více než t symbolů, budeme nazývat t-sRL-*automatem*. Symbolem t-sRL značíme třídu všech t-sRL-automatů.

V následujícím textu budeme pracovat již jen s t-sRL-automaty.

Poznámka: Takto definované t-sRL-automaty jsou dvoucestné automaty, které dovolují v každém cyklu kontrolu celé věty před prováděním změn. To připomíná chování lingvisty, který si může celou větu přečíst ještě před volbou (korektní) redukce. Model automatu je záměrně obecně nedeterministický, aby mohl měnit pořadí redukčních cyklů. To slouží jako nástroj pro verifikaci vzájemné nezávislosti některých částí věty, viz oddíl 2 o redukční analýze.

V práci [3] popisujeme t-sRL-automaty pomocí *metainstrukcí* v následující formě:

$(¢ \cdot E_0, a_1, E_1, a_2, E_2, \ldots, E_{s-1}, a_s, E_s \cdot \$)$, kde:
- $s \in \{1, \ldots t\}$,
- $E_0, E_1, \ldots, E_s$ jsou regulární jazyky (často reprezentované regulárními výrazy); nazýváme je *regularními omezeními* těchto instrukcí, a
- symboly $a_1, a_2, \ldots, a_s \in \Sigma$ odpovídají symbolům, které jsou vypuštěny automatem M během příslušného cyklu.

Výpočet pomocí M řízený touto metainstrukcí startuje z konfigurace $q_0 ¢ w \$$; výpočet zkolabuje (tedy nebude ani přijímacím výpočtem), pokud w není možno rozložit do tvaru $w = v_0 a_1 v_1 a_2 \cdots v_{s-1} a_s v_s$ takového, že $v_i \in E_i$ pro všechna $i = 0, \ldots, s$. Pokud w do takového tvaru rozložit lze, pak je jeden z nich (nedeterministicky) vybrán a restartovací konfigurace $q_0 ¢ w \$$ je redukována do restartovací konfigurace tvaru $q_0 ¢ v_0 v_1 \cdots v_{s-1} v_s \$$. Abychom popsali přijímací koncovky, používáme tzv. přijímací metainstrukce tvaru $(¢ \cdot E \cdot \$, \mathsf{Accept})$, kde E je regulární jazyk (v našem případě dokonce konečný).

Povšimněte si následujících vlastností, které hrají důležitou roli v naší aplikaci restartovacích automatů.

Definice: (Vlastnost zachování chyby)
Říkáme, že t-sRL-automat M *zachovává chybu*, pokud $u \notin \mathsf{L}_C(M)$ a $u \vdash^{c^*}_M v$ implikuje, že $v \notin \mathsf{L}_C(M)$.

Definice: (Vlastnost zachování korektnosti)
Říkáme, že t-sRL-automat M *zachovává korektnost*, pokud $u \in \mathsf{L}_C(M)$ a $u \vdash^{c^*}_M v$ implikuje, že $v \in \mathsf{L}_C(M)$.

Je známo, že každý t-sRL-automat zachovává chybu a že všechny deterministické t-sRL-automaty zachovávají korektnost. Na druhé straně lze zkonstruovat příklady nedeterministických t-sRL-automatů, které korektnost nezachovávají, viz [4].

3.2 Obohacené t-sRL-automaty

Obohacený t-sRL-automat M_o popisujeme pomocí *obohacených* metainstrukcí, umožňujících postupně budovat stromové struktury, které formalizují tektogramatickou rovinu FGP.

Obohacené metainstrukce mají dva nové rysy, (i) obohacený M_o může nově symboly také přepisovat a (ii) M_o může navíc během cyklu pokládat oblázky na pracovní pásku a tím označovat symboly na pásce. Místa označená oblázky slouží pro výběr uzlů do výstupní struktury, při restartu jsou všechny oblázky odstraněny. Oblázky v metainstrukcích zde reprezentujeme čísly v hranatých závorkách, např. $[1], [2], \ldots [s]$.

První výslednou výstupní strukturou jsou tzv. DR-stomy (delete-rewrite trees). Uzly DR-stromu jsou tvořeny výskyty vypouštěných či přepisovaných symbolů, které byly označeny pomocí oblázků. DR-stromy mají dva typy hran:

– *šikmé (vypouštěcí)* hrany vedou od aktuálně vypouštěného symbolu pod oblázkem k nějakému automatem vybranému symbolu pod oblázkem;
– *vertikální (přepisovací) hrany* vedou od původního symbolu pod oblázkem k novému symbolu na stejném místě.

Uzly DR-stromu jsou horizontálně uspořádané. Toto uspořádání uzlů je odvozeno od uspořádání (výskytů) symbolů ve vstupním slově výpočtu M_o. Uzly spojené vertikální hranou mají v tomto uspořádání stejnou (horizontální) pozici.

Při zavádění DR-stromů bychom mohli postupovat stejně jako v práci [1]. Tam je možné najít i vhodné neformální vysvětlení užití DR-stromů v lingvistice.

Poznámka: Stromové struktury FGP lze formálně popsat jako tzv. D-stromy. D-strom vznikne z DR-stromu ‚sražením' vertikálních cest do jejich nejnižšího uzlu, který nese původní symbol na příslušné pozici. Horizontální uspořádání uzlů D-stromu se dědí z příslušného DR-stromu. Informace o tom, jak z D-stromů získat zpět DR-stromy, je v FGP implicitně obsažena v kategoriích (např. ve valenčních rámcích), které uzly nesou. Více o vztahu DR-stromů a D-stromů můžeme nalézt opět v článku [1].

Restartovací metainstrukce. Pro t-sRL-automat M_o uvažujeme obohacené metainstrukce dvojího typu: restartovací a přijímací. *Restartovací metainstrukce* mají následující formu:

$I = (¢ \cdot E_0, [1]p_1, E_1, [2]p_2, \ldots E_{s-1}, [s]p_s, E_s \cdot \$, T_I)$, kde:

– $s \in \{1, \ldots t\}$,
– $E_0, E_1, \ldots, E_s$ jsou regulární jazyky podobně jako u standartních metainstrukcí;
– $[1], \ldots [s]$ označují místa s položenými oblázky, tedy místa, která budou využita jako uzly stromu T_I, viz dále;
– $p_1, p_2, \ldots, p_s$ popisují uplatnění jednotlivých operací automatu M_o během příslušného cyklu; p_i může být v jednom z následujících tvarů:
  1. $a_i \to \lambda$, značí vypuštění symbolu a_i;
  2. $a_i \to b_i$, značí přepis symbolu a_i na symbol b_i;
  3. a_i, používá se pro vyznačení místa s výskytem symbolu a_i; toto značení se využívá jen pro formulaci výstupní struktury T_I.
– T_I je popis vypouštěcího a přepisovacího stromu daného metainstrukcí I.

Požadujeme, aby vypouštěcí a přepisovací strom pro instrukci I, tedy DR-strom T_I, tvořil vrcholový (kořenový) strom. T_I tvoří následujícím uzly a hrany:

1. *Uzly* stromu T_I mají tvar uspořádané trojice, kde první položka trojice je číslo odkazující k místu na pásce, druhá položka obsahuje symboly na pásce (značíme je zde a_i, c_j, příp. d_l) a třetí položka (spolu s první položkou) určuje hranu k rodičovskému uzlu (viz bod 2.):
   – *kořen* T_I je uzel tvaru $(i, c_i, Root)$, $0 \le i \le s$;
   – *listy* T_I mají tvar $[j, a_j, h_j]$, kde $0 \le j, h_j \le s$;
   – T_I může obsahovat maximálně jeden *vnitřní uzel*[3] tvaru (l, d_l, k_l).

   Dále požadujeme, aby symboly c_i, příp. d_l byly rovny buď a_i nebo b_i, resp. a_l nebo b_l v závislosti na tvaru T_I (viz dále).
2. *Hrany* stromu T_I jsou kódovány pomocí první a třetí položky v uzlech:
   – *vypouštěcí hrana:* uplatnění operace tvaru $p_j = a_j \to \lambda$ dovoluje vytvářet následující hrany: $([j, a_j, h_j], (h_j, a_{h_j}, h_{h_j}))$, nebo $((j, a_j, h_j), (h_j, a_{h_j}, Root))$, nebo $([j, a_j, h_j], (h_j, a_{h_j}, Root))$, kde $h_j \ne j$ (jde tedy o šikmé hrany).
   – *přepisovací hrana* uplatnění operace $p_j = a_j \to b_j$ může produkovat (vertikální) hranu tvaru $((j, a_j, j), (j, b_j, Root))$; navíc požadujeme, aby T_I měl maximálně jednu hranu tohoto typu;
   – *vypouštěcí hrana do kořene:* ‚prázdná' operace $p_j = a_j$ se využívá pro označení kořene stromu T_I.

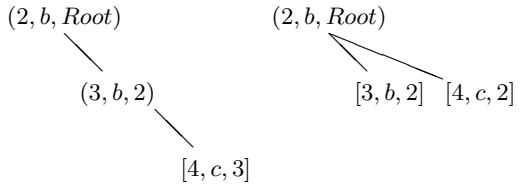
   Hrany jsou orientovány od vypouštěných a přepisovaných symbolů k těm momentálně nevypuštěným nebo právě vzniklých přepisem (tedy směrem k vrcholu (kořeni)).

Tyto požadavky kladené na DR-strom T_I říkají, že cesty v T_I mají maximálně délku 2 a do vrcholu (kořene) vede nejvýše jedna hrana; pokud je tato hrana vertikální (přepisovací), tak je to jediná vertikální hrana v T_I. Tato omezení vyplývají z požadavků na formální rámec FGP, viz oddíl 3.3.

Příklad. Uvažujme formální jazyk $L_{abc} = \{a^n b^n c^n \mid n > 0\}$. Při redukci vět tohoto jazyka budeme postupovat např. následovně: budeme vypouštět vždy poslední symbol a (označíme jej oblázkem [1]), předposlední symbol b (oblázek [2]) bude kořen stromu, dále vypouštíme poslední symbol b (oblázek [3]) a první symbol c (oblázek [4]). Zároveň budujeme výstupní strom $T_{I_{abc}}$, který blíže specifikuje závislosti mezi symboly zpracovávaného řetězce. Budeme tedy používat jedinou restartovací metainstrukci:

$I_{abc} = (¢ \cdot \{a\}^*, [1]\ a \to \lambda, \{b\}^*, [2]\ \ b, \{\lambda\}, [3]\ b \to \lambda, \{\lambda\}, [4]\ \ c \to \lambda, \{c\}^* \cdot \$, T_{I_{abc}})$, kde $T_{I_{abc}} =$

[3] Vnitřní uzel stromu je takový uzel, který není kořenem ani listem. Rozlišení listů [ ] a ostatních uzlů ( ) je technické řešení umožňující jednoduše formulovat podmínky na tvar T_I.



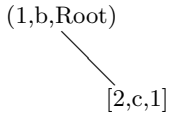
Obr. 5. Vypoušťěcí a přepisovací stromy $T_{I_{abc}}$ (vlevo) a $T'_{I_{abc}}$ (vpravo).

$\{(2, b, Root), (3, b, 2), [4, c, 3]\}$ je zobrazen na obrázku 5 vlevo. Vypouštěcí strom $T_{I_{abc}}$ zde určuje závislost vypouštěných symbolů b, c na (předposledním) b a dále skutečnost, že vypouštěný symbol a s označením [1] není do výstupní struktury vůbec zařazen. První části instrukce by např. vyhovoval též strom $T'_{I_{abc}} = \{(2, b, Root), [3, b, 2], [4, c, 2]\}$ (obrázek 5 vpravo).

Přijímací metainstrukce. Abychom popsali přijímací koncovky, používáme *přijímací metainstrukce* tvaru $[I_a, \mathsf{Accept}]$, kde I_a je instrukce podobného tvaru jako restartovací metainstrukce, podobný je i způsob jejího uplatnění. Přijímací instrukce má tedy též možnost vypouštět a přepisovat (není to však na rozdíl od ostatních instrukcí povinné). Úspěšné provedení přijímací metainstrukce znamená ukončení přijímacího výpočtu. Změny provedené v přijímací metainstrukci se nepovažují za redukci danou automatem M_o. Mají význam pouze s ohledem na výstupní strukturu neredukovatelné věty.

Příklad. Pro formální jazyk $L_{abc} = \{a^n b^n c^n |\ n \geq 0\}$ budeme potřebovat kromě jediné restartovací metainstrukce (viz výše) též jedinou přijímací metainstrukci tvaru $[Ia, \mathsf{Accept}]$, kde:

$I_a = (\mathrm{¢} \cdot \{a\},\ [1]\ b,\ \{\lambda\},\ [2]\ \ c \rightarrow \lambda,\ \$,\ T_{I_a})$, kde $T_{I_a} = \{(1, b, Root), [2, c, 1]\}$ je zobrazen na obrázku 6. Pomocí této metainstrukce se zkontroluje poslední symbol a (neprojeví se ve výsledné struktuře), poslední symbol b (oblázek [1]) bude kořenem stromu a poslední symbol c (oblázek [2]) bude redukován; strom T_{I_a} určuje, že symbol c bude záviset na symbolu b.



Obr. 6. Strom T_{I_a} pro přijímací metainstrukci I_a.

Budování DR-struktury pro danou metainstrukci. Při výpočtu obohacený t-sRL-automat M_o tedy provede současně s provedením vypouštěcí a přepisovací části instrukce též odpovídající úpravu (přírůstek) výstupní DR-struktury podle specifikace v poslední části instrukce. Metainstrukce I přiděluje jednotlivým výskytům symbolů $a_1, \ldots a_s$ v instrukci oblázky, tedy čísla v hranatých závorkách od [1] do $[s]$. Ta svazují uzly z T_I s výskyty symbolů ve zpracovávané restartovací konfiguraci během cyklu popsaného pomocí I (a tím případně i s DR-strukturou vytvořenou dřívějším použitím metainstrukcí, viz dál). Kořen (i,a_i,*Root*) reprezentuje jediný symbol z T_I, který je přítomný v následující konfiguraci. To znamená, že k vypuštěným uzlům není v žádné z následujících konfigurací přístup.

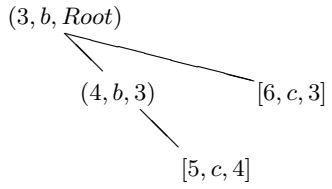
Provedením (další) metainstrukce získáme novou strukturu, která v obecném případě nemusí být souvislá. Platí však, že každá komponenta této struktury tvoří velmi speciální kořenový strom. V tomto příspěvku nás zajímají pouze t-sRL-automaty, které *přijímacím výpočtem vrací (souvislé vrcholové)* DR-*stromy*; komponenty odpovídající jednotlivým metainstrukcím jsou postupně spojovány do nové komponenty, a to (i) buď tak, že se v jednom kroku ztotožní kořeny původních komponent (využívá se pro jednoduché redukční závislosti zpracovávané restartovacími metainstrukcemi a pro přijímací metainstrukci), (ii) nebo se v jednom kroku spojí jedinou přechodovou hranou, která vede z kořene jedné komponenty do jiného uzlu než kořene nové komponenty (pro restartovací metainstrukce zpracovávající redukční komponenty).

Množinu DR-stromů, které vznikly během přijímacích výpočtů automatu M_o, budeme označovat $\mathsf{TR}(\mathsf{M}_o)$. Množinu odpovídajících D-stromů označíme $\mathsf{dTR}(\mathsf{M}_o)$ a budeme ji nazývat dTR-jazykem automatu M_o.

Příklad. Mějme formální jazyk $L_{abc} = \{a^n b^n c^n |\ n \geq 0\}$ a metainstrukce I_{abc} a I_a pro jeho zpracování definované výše. Obrázek 7 zachycuje DR-strom T daný uplatněním těchto instrukcí na řetěz tvaru *aabbcc*. Uzel tvaru $(3, b, Root)$ zobrazuje kořen stromu T, kde b je třetím symbolem řetězu *aabbcc*; dále např. uzel tvaru $[5, c, 4]$ zachycuje symbol c na pátém místě řetězu *aabbcc*, který je závislý na symbolu b na čtvrtém místě řetězu *aabbcc* (tj. z c vede hrana do b). Protože tento DR-strom neobsahuje žádné přepisovací hrany, je shodný s D-stromem.

3.3 Formální rámec redukční analýzy a jeho vlastnosti

V tomto oddíle využijeme pojmy předchozího oddílu a zavedeme formální rámec redukční analýzy pro FGP, který budeme označovat F_{09}.



Obr. 7. DR-strom T pro řetězec *aabbcc*.

Jako *automat typu* F_{09} budeme označovat takový obohacený nedeterministický t-sRL-automat M_{09}, který zachovává koreknost a jehož charakteristický slovník Σ je sestaven ze čtyř podslovníků $\Sigma_w, \Sigma_m, \Sigma_a, \Sigma_t$. Požadujeme přitom, aby v každém cyklu M_{09} byl vypuštěn alespoň jeden symbol z těchto podslovníků. Pokud M_{09} nějaký symbol přepíše, přepíše ho symbolem ze stejného podslovníku.

Automat M_{09} (typu F_{09}) je popisován pomocí obohacených metainstrukcí, které obsahují informace o tektogramatické struktuře. M_{09} přijímacím výpočtem zahrne právě všechny tektogramatické symboly (symboly z podslovníku Σ_t) charakteristického řetězu (věty) do výstupního D-stromu. Připomeňme, že množinu korektně definovaných D-stromů automatu M_{09} nazýváme dTR-jazykem automatu M_{09} a značíme ji $\mathsf{dTR}(\mathsf{M}_{09})$. Množina $\mathsf{dTR}(\mathsf{M}_{09})$ tedy reprezentuje všechny korektní tektogramatické struktury podle FGP.

Pojmy $\mathsf{L}_C(\mathsf{M}_{09})$, *charakteristického jazyka* automatu M_{09}, a $\mathsf{S}_C(\mathsf{M}_{09})$, *jednoduchého jazyka* automatu M_{09}, přebíráme z předchozího oddílu. Nechť $w \in \mathsf{L}_C(\mathsf{M}_{09})$. Potom $\mathsf{TR}(w,\mathsf{M}_{09})$ označuje množinu DR-stromů z $\mathsf{TR}(\mathsf{M}_{09})$ získaných výpočtem M_{09} nad větou w. Odpovídající množinu D-stromů značíme $\mathsf{dTR}(w,\mathsf{M}_{09})$ Požadujeme v souladu se základním záměrem FGP, aby pro každé $w \in \mathsf{L}_C(\mathsf{M}_{09})$ měla $\mathsf{dTR}(w,\mathsf{M}_{09})$ právě jeden prvek. Tento požadavek se nevztahuje na DR-stromy.

Jazykem j-roviny rozpoznávané pomocí M_{09}, kde $j \in \{w, m, s, t\}$, je množina všech vět (řetězů), které lze získat z řetězů z $\mathsf{L}_C(\mathsf{M}_{09})$ odstraněním všech symbolů nepatřících do Σ_j. Tento jazyk označíme $\mathsf{L}_j(\mathsf{M}_{09})$. $\mathsf{L}_w(\mathsf{M}_{09})$ reprezentuje množinu správně zapsaných vět definovanou pomocí M_{09}.

Nyní můžeme definovat dTR-*charakteristickou relaci* $\mathsf{TSH}(\mathsf{M}_{09})$ danou automatem M_{09}:

$\mathsf{TSH}(\mathsf{M}_{09}) = \{(u,t) \mid u \in \mathsf{L}_w(\mathsf{M}_{09})$ & $t \in \mathsf{dTR}(\mathsf{M}_{09})$ & $\exists w \in \mathsf{L}_C(\mathsf{M}_{09})$ takové, že u vzniklo z w vypuštěním symbolů nepatřících do Σ_w a $t \in \mathsf{dTR}(w,\mathsf{M}_{09})\}$.

Poznámka: dTR-charakteristická relace reprezentuje důležité vztahy v popisu přirozeného jazyka, vztahy synonymie a homonymie (víceznačnosti). Z charakteristické relace odvodíme zbývající důležité pojmy, pojmy analýzy a syntézy.

Pro libovolné $t \in \mathsf{dTR}(\mathsf{M}_{09})$ zavádíme TSH-*syntézu pomocí* M_{09} jako množinu dvojic (u,t) patřících do $\mathsf{TSH}(\mathsf{M}_{09})$, formálně

$$\mathit{synt}\text{-}\mathsf{TSH}(\mathsf{M}_{09}, t) = \{(u,t) \mid (u,t) \in \mathsf{TSH}(\mathsf{M}_{09})\}$$

TSH-syntéza tedy spojuje tektogramatickou reprezentaci, t.j. D-strom t z $\mathsf{dTR}(\mathsf{M}_{09})$, se všemi jí odpovídajícími větami u patřícími do $\mathsf{L}_w(\mathsf{M}_{09})$. Tento pojem slouží ke sledování synonymie a jejího stupně.

Na závěr zavádíme duální pojem k TSH-syntéze, pojem TSH-*analýzy řetězu u pomocí* M_{09}:

$$\mathit{anal}\text{-}\mathsf{TSH}(\mathsf{M}_{09}, u) = \{(u,t) \mid (u,t) \in \mathsf{TSH}(\mathsf{M}_{09})\}$$

TSH-analýza tedy vrací pro danou větu u všechny její tektogramatické reprezentace t z $\mathsf{dTR}(\mathsf{M}_{09})$, tedy modeluje víceznačnost jednotlivých povrchových vět. Tento pojem dává formální definici úplné syntaktickosémantické analýzy pomocí M_{09}.

Omezení na vlastnosti metainstrukcí pro FGP. Zbývá nám doplnit několik omezení na vlastnosti metainstrukcí a výpočtů M_{09}, která vyplývají z charakteru lingvistické metody FGP.

1. Připomeňme si tvar restartovací metainstrukce automatu M_{09}:

I=($\mathrm{¢}\cdot E_0, [1]p_1, E_1, [2]p_2, E_2, \ldots, E_{s-1}, [s]p_s, E_s\cdot\$, T_I$), $s \in \{1, \ldots t\}$.

Rozlišujeme dva typy restartovacích metainstrukcí: metainstrukce, které zpracovávají redukční komponenty (zejména valenční), a metainstrukce pro jednoduché redukční závislosti, viz [2]. Metainstrukce pro *redukční komponenty* musí splňovat následující omezení na T_I:

- Délka každé (úplné) cesty v T_I je právě 2, t.j. každá úplná cesta obsahuje právě dvě hrany, a T_I obsahuje právě jeden vnitřní uzel.
- Do vrcholu (kořene) T_I vede jediná hrana. Může být šikmá i vertikální. Šikmá hrana signalizuje korektnost redukce na tektogramatické rovině a významovou úplnost zredukované tektogramatické věty. Vertikální hrana signalizuje korektnost redukce na tektogramatické rovině a významovou neúplnost zredukované tektogramatické věty.
- Do vnitřního uzlu T_I vedou jen šikmé hrany. Reprezentují jednotlivá valenční doplnění.

Metainstrukce pro *jednoduché redukční závislosti* musí produkovat T_I, který obsahuje jedinou šikmou hranu.

2. Přijímací metainstrukce spolu vytváří, nebo spoluvytváří kořen výsledného DR-stromu. Nemusí na rozdíl od restartovacích instrukcí vytvářet přechodovou hranu k žádné další komponentě výsledného DR-stromu.

Z tohoto důvodu nemá T_I žádný vnitřní bod. Přijímací metainstrukce tvoří DR-stromy pouze valenčního typu. Ostatní závislosti ‚per definitione' odpovídají jednoduchým redukčním závislostem a modelují se tedy pomocí restartovacích metainstrukcí.

3. T_I v libovolné z výše popsaných metainstrukcí má ještě důležitou vlastnost, a to *vlastnost projektivity*. Projektivita znamená, že každý (úplný) podstrom T_I vymezuje nějaký souvislý úsek v horizontálním uspořádání svých uzlů, formální definici projektivity pro DR- a D-stromy lze nalézt např. v článku [1].

4 Závěrečné poznámky

Tímto příspěvkem jsme navázali na článek [3]. Obohatili jsme zde formální rámec pro Funkční generativní popis češtiny tak, že výstupem analýzy definované tímto popisem nejsou řetězy slov a symbolů, ale (tektogramatické) závislostní stromy. Tento typ stromů je vstupem pro syntézu definovanou pomocí FGP, která popisuje synonymii v češtině. Dále jsme formálně zavedli analýzu popisující víceznačnost přirozeného jazyka. Tímto krokem jsme zřetelně pokročili v popisu souvislostí mezi redukční analýzou a závislostní strukturou na tektogramatické rovině. Zformulovali jsme tak základní metodiku FGP v termínech teorie automatů, tedy v termínech obvyklých pro formulaci informačních technologií. Toto považujeme za hlavní přínos tohoto článku, neboť tato metodika byla až doposud vykládána způsobem obvyklým v tradiční lingvistice, tedy spíše vágně a především pomocí příkladů.

Poznamenejme, že náš model zatím nezohledňuje řadu hlavně povrchových jevů týkajících se zejména koordinace a tzv. segmentů. Do budoucna proto plánujeme rozšíření formálního rámce tak, abychom do něj mohli zmíněné jevy bez vážnějších konfliktů zahrnout. DR-stromy, které by v tomto ohledu měly pomoci, jsme použili jako pomocný prostředek již v této práci.

Další směr, kterým bychom se rádi vydali, je studium slovosledných jevů v souvislosti s valencí. Za tímto účelem je vhodné zavést závislostní stromy a DR-stromy také pro povrchovou rovinu. V tomto případě však budeme muset připustit i masivní neprojektivitu pro tyto stromy, viz [1].

Práce neobsahuje žádná (nová) tvrzení matematického typu. Ta je možno nalézt v bohaté literatuře o restartovacích automatech, např. v pracích [5,4]. Práce studující obohacené restartovací automaty matematickou metodou očekáváme v blízké budoucnosti.

Reference

1. T. Holan, V. Kuboň, K. Oliva, and M. Plátek: *On complexity of word order.* In Les grammaires de dépendance – Traitement automatique des langues, (S. Kahane, ed.), 41, 1, 2000, 273–300.
2. M. Lopatková, M. Plátek, and V. Kuboň: *Závislostní redukční analýza přirozených jazyků.* In Proceedings of ITAT 2004, (P. Vojtáš, ed.), 2004, 165–176.
3. M. Lopatková, and M. Plátek: *Funkční generativní popis a formální teorie překladů.* In Proceedings of ITAT 07, (P. Vojtáš, ed.), 2007, 3–14.
4. H. Messerschmidt, F. Mráz, F. Otto, and M. Plátek: *Correctness preservation and complexity of simple RL-automata.* In Lecture Notes in Computer Science, 4094, 2006, 162–172.
5. F. Otto: *Restarting automata and their relations to the Chomsky hierarchy.* In Developments in Language Theory, Proceedings of DLT'2003 (Z. Esik, Z. Fülöp. eds.) LNCS, 2710, Springer, Berlin, 2003.
6. V. Petkevič: *A new formal specification of underlying structures.* Theoretical Linguistics, 21, 1, 1995, 7–61.
7. M. Plátek: *Composition of translation with D-trees.* COLING'82, 1982, 313–318.
8. M. Plátek, and P. Sgall: *A scale of context-sensitive languages: applications to natural language.* Information and Control, 38, 1, 1978, 1–20.
9. M. Plátek, F. Mráz, F. Otto, and M. Lopatková: *O roztržitosti a volnosti slovosledu pomocí restartovacích automatů.* In Proceedings of ITAT 2005, (P. Vojtáš, ed.), 2005, 145–156.
10. P. Sgall: *Generativní popis jazyka a česká deklinace* Academia, Praha, 1967.
11. P. Sgall, E. Hajičová, and J. Panevová: *The meaning of the sentence in its semantic and pragmatic aspects.* (J. Mey, ed.) Dordrecht: Reidel and Prague: Academia, 1986.
12. P. Sgall, L. Nebeský, A. Goralčíková, and E. Hajičová: *A functional approach to syntax in generative description of language.* New York, 1969.

Acoma: Kontextový rekomendačný emailový systém

Martin Šeleng, Michal Laclavík, Emil Gatial and Ladislav Hluchý

Oddelenie paralelného a distribuovaného spracovania informácií
Ústav informatiky, Slovenská akadémia vied
Dúbravská cesta 9, 845 07 Bratislava, Slovensko

Abstrakt. *Aj v súčasnom období, keď rezonujú slovné spojenia ako Web 2.0, resp. Web 3.0 je email stále najčastejšie používanou službou na internete. Email ako taký sa potýka s rôznymi problémami ako je napr. spam alebo informačné preťaženie spojené so spracovávaním emailov, avšak prináša aj rôzne výhody pre spoločnosti, ktoré pomocou emailu komunikujú, kooperujú alebo riešia pracovné úlohy. Aj keď sa emailový klienti postupne zlepšujú, integrácia emailu (emailových klientov) s rôznymi systémami na podporu kolaborácie, resp. spracovávania úloh zostáva stále na rovnakej úrovni. V príspevku navrhujeme framework pre anotáciu emailových správ ako nový spôsob využitia predpripravených znalostí pre organizácie, ktoré využívajú emailovú komunikáciu ako súčasť svojich procesov. Ďalej v príspevku opisujeme nástroj, ktorý umožňuje poskytovanie znalostí, získaných na základe kontextu, v organizáciách (ACoMA - Automated Content-based Message Annotator) pri riešení ich pracovných postupov. Takisto predstavíme spôsob ako jednoducho rozširovať možnosti poskytovania znalostí v emailoch pomocou technológie OSGi, a tým umožniť ľubovoľné pridávanie modulov do navrhnutého systému. Súčasne navrhneme spôsob ako vyvíjať používateľské rozhrania a ako spravovať a spúšťať jednotlivé vyvinuté moduly.*

1 Úvod

Podľa najnovších prehľadov a štatistík ľudia pracujúci s informáciami posielajú a prijímajú denne 133 emailov[1] a strávia 21% pracovného času pri spracovaní emailovej komunikácie. V roku 2001 pracovníci pracujúci s informáciami prijali približne 20 a poslali 6 emailov denne[2], dnes je množstvo prijatých emailov oveľa vyššie, pričom počet odoslaných emailových správ zostal približne na rovnakej úrovni [13]. Pritom väčšina používateľov hovorí o zavalení informáciami, tzv. „information overload“[3]. Informácie vytvorené v ľubovoľnej organizácii môžu byť prínosom, ale aj záťažou - záleží na tom, ako sú využívané a manažované. Email sa v tomto ohľade nijako nelíši od iných informačných zdrojov. Môže byť vysoko efektívnym komunikačných a pracovným nástrojom a zdrojom potrebných informácií, ale iba vtedy, ak sú informácie dobre spravované a manažované.

Jedným zo základných problémov emailovej komunikácie je to, že sa používa na účely, na ktoré pôvodne nebola vytvorená. Napríklad na archivovanie informácií alebo manažment pracovných úloh [1, 2]. Ak chcú organizácie dosiahnuť stanovené ciele, základom je efektívna komunikácia, ktorá často prebieha cez emaily. Emailová komunikácia sa využíva hlavne pri spolupráci a prepojení (interoperabilite) firiem a organizácií všetkých veľkostí. Je teda vhodným médiom na zistenie kontextu používateľa a poskytovanie relevantných informácií a znalostí v tomto kontexte, ktoré používateľ potrebuje na úspešné vykonanie pracovných aktivít. Podobne ako Gmail[4] zobrazuje kontextovú reklamu a umožňuje niektoré jednoduché akcie, ako je napríklad pridanie udalosti do kalendára, systém Acoma poskytuje informácie a znalosti priamo v kontexte emailu. Tieto informácie môžu pomôcť pri práci, ktorú email reprezentuje. Problém ako pripojiť k emailu znalosti, resp. kontextové informácie bol riešený vo viacerých projektoch, ako napr. kMail [9], ktorý sa pokúšal zintegrovať emailovú komunikáciu s organizačnou pamäťou, ale nútil používateľa používať špeciálneho emailového klienta. Iný podobný nástroj je Zimbra[5]. Zimbra je webový emailový klient s funkcionalitou poskytujúcou detekciu objektov, ako sú napríklad telefónne čísla alebo adresy komunikujúcich firiem, a umožňuje aj niektoré akcie nad týmito objektmi. Podobne ako kMail aj Zimbra núti používateľa používať špeciálneho emailového klienta a emailový server a zmeniť tak existujúcu internetovú infraštruktúru v rámci organizácie.

Systém Acoma sa začal vyvíjať ako jeden z komponentov v rámci APVT projektu Raport[6] a ďalej sa rozvíja ďalej v rámci medzinárodného projektu Commius[7], ktorý rieši medzipodnikovú interoperabilitu [12] využívajúcu emailovú komunikáciu. Jednou z úloh projektu Commius je aj semi-automatická anotácia založená na vzoroch, ktorá stavia na metóde Ontea[8] vyvinutej v rámci projektu NAZOU[9].

Práve sémantická anotácia je jedným zo spôsobov riešenia problémov spojených s emailovou komunikáciou. Aby bolo možné zistiť formalizovaný kontext emailu napríklad vzhľadom na obchodný model organizácie, je potrebné mapovať text emailu na objekty v modeli organizácie.

Emaily podobne ako informácie na webe obsahujú neštruktúrovaný text, často v ešte väčšom množstve. Existujúce anotačné prístupy a riešenia, ktoré sú väčšinou zamerané na dokumenty na webe a využívajú HTML štruktúru, nie je možné použiť na anotáciu emailov. Sémantická anotácia však predstavuje možnosť, ako ďalej riešiť odvodzovanie, vyhľadávanie na základe sémantiky alebo zisťovanie kontextu emailovej komunikácie.

1 https://h30046.www3.hp.com/campaigns/2005/promo-evolution/1-1LRYR/images/Preview_Radicati.pdf

2 http://www.gallup.com/poll/4711/Almost-All-EMail-Users-Say-Internet-EMail-MadeLives-Better.aspx

3 University of Southern California 2007 Center for the Digital Future Report: http://www.digitalcenter.org/pdf/2007-Digital-FutureReport-Press-Release112906.pdf

4 http://gmail.com

5 http://www.zimbra.com/

6 http://raport.ui.sav.sk

7 http://www.commius.eu

8 http://ontea.sourceforge.net

9 http://nazou.fiit.stuba.sk

Príspevok je rozdelený do 5 častí: Úvod, Prístup a architektúra, Aplikácia a prispôsobenie, Existujúce riešenia, Záver. V časti Prístup a architektúra uvedieme prístup, ktorý sme zvolili aby sme vyriešili niektoré z problémov uvedených na začiatku tejto časti a takisto uvedieme architektúru nástroja Acoma a použité technológie. V časti Aplikácia a prispôsobenie uvedieme príklady použitia nášho riešenia a takisto uvedieme spôsob ako prispôsobiť náš framework pre potreby ľubovoľnej organizácie. V časti Existujúce riešenia porovnáme naše riešenie s inými existujúcimi riešeniami a v poslednej časti Záver, zhrnieme a uvedieme možnosti rozširovania funkcionality nástroja Acoma.

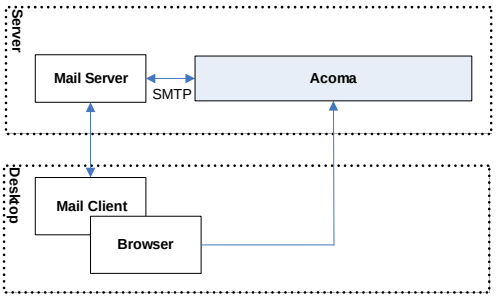
2 Prístup a architektúra

Emaily sú silne napojené na prácu v organizácii, ich obsah je však väčšinou neštruktúrovaný. Vyvinutý nástroj je priamo prepojený na pracovný kontext organizácie, takže nie je ťažké analyzovať a pochopiť kontext týkajúci sa znalostí v organizačnej pamäti. Nechceme nútiť používateľov frameworku Acoma používať nové webové rozhranie, nový plugin do existujúceho emailového klienta, resp. nového emailového klienta, ale nechať používateľa pracovať tak ako bol doteraz zvyknutý (prijímať a posielať emaily v jeho obľúbenom klientovi).

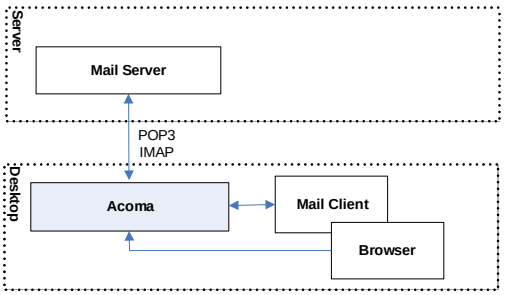
Používanie emailov umožňuje získať „aktívny“ zdieľaný znalostný kanál, pretože používateľ nemusí na získanie určitej znalosti využívať rozsiahle vyhľadávanie. Zdieľané znalosti sú priamo doručené v emailovej správe na základe aktuálneho problému alebo aktivity riešenej používateľom. Používateľ dostane email s pripojenými informáciami na konci správy (textové prílohy, resp. HTML prílohy). V emailovej správe sa tiež zobrazia informácie o ďalšom probléme alebo aktivite v danom pracovnom procese. Používanie textových (HTML) príloh sa ukazuje ako vhodné riešenie, pretože sa nemení text pôvodného emailu, len sa dopĺňa o relevantné informácie (text, hypertextové prepojenia, a pod.). Momentálne existujú dve implementácie nástroja Acoma: SMTP a POP3 (IMAP implementácia je v štádiu spracovávania) implementácia, ktorých architektúru je možné vidieť na (obr. 1) a (obr.2).

Obidve existujúce implementácie pracujú v podobných cykloch (obr. 3).

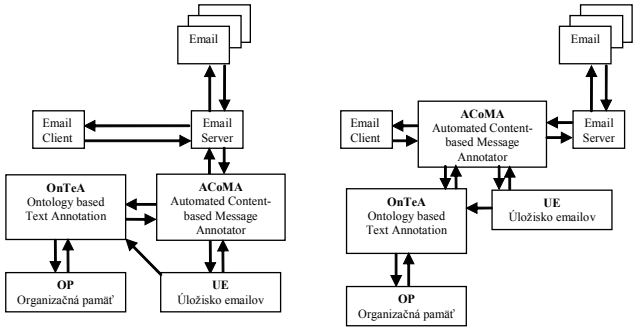
V obidvoch implementáciách funguje nástroj Acoma podobne ako proxy s tým rozdielom, že v prípade SMTP implementácie je to nepriame prepojenie (Acoma prijme od SMTP servera naraz celý email) a v prípade POP3 implementácie slúži ako priame proxy (preposiela medzi



Obr. 1. Acoma napojená na emailový server.



Obr. 2. Acoma na desktope.



Obr. 3. Cyklus práce nástroja Acoma: SMTP a POP3 implementácia.

emailovým klientom a emailovým serverom celú komunikáciu, až pokiaľ nezachytí samotný text emailu).

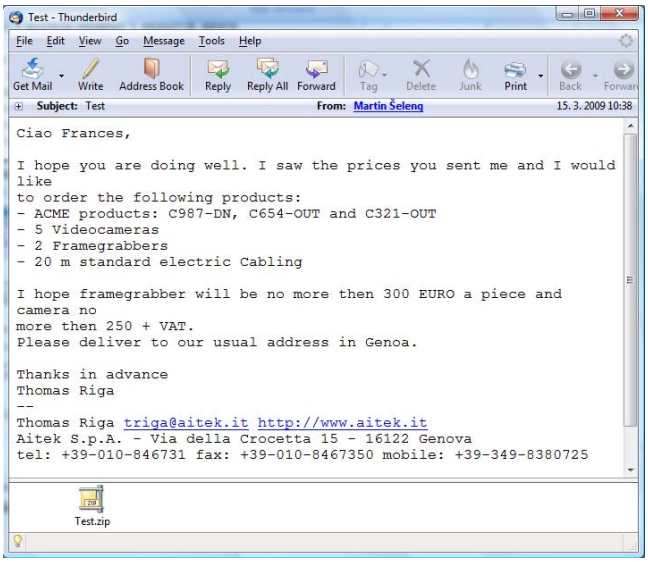
V prípade SMTP implementácie je nástroj Acoma nainštalovaný na poštovom serveri podobne ako antivírové alebo antispamové programy. Po prijatí emailu nástroj Acoma daný email rozloží na prílohy, text emailu a hlavičku a následne ich zanalyzuje pomocou sémantickej anotácie [6, 7]. Nástroj Ontea na základe získaného kontextu vyberie z organizačnej pamäte všetky relevantné informácie, ktoré následne pošle späť nástroju Acoma. Acoma tieto informácie naformátuje a pripojí k prijatému emailu a email ponechá na serveri. Používateľ následne pri preberaní pošty dostane už takto upravený email.

V prípade POP3 implementácie je Acoma spustená na klientskom počítači. Výhodou tejto implementácie je to, že v prípade viacerých používateľov jedného emailového konta (napríklad v prípade malých podnikov, keď sa pre objednávky a faktúry používané len jedno emailové konto) systém Acoma nemodifikuje email na poštovom serveri ale len u používateľa, takže ostatní používatelia, ktorí nemajú nainštalovaný nástroj Acoma, vidia pôvodný email bez zmien.

Na nasledujúcom obrázku (obr. 4) je ukážka emailovej správy z projektu Commius.

Nástroj Acoma odosielaný email (obr.4) zanalyzuje, rozloží na hlavičku a telo správy, ktoré následne uloží do úložiska emailov. Následne spracuje prílohy, rozbalí zip súbor a následne spracuje aj tieto súbory. Pre extrakciu informácií Ontea následne spracuje telo aj hlavičku emailu, pričom sa použijú regulárne výrazy (pre náš email), niektoré z nich sa nachádzajú na obr. 5.

Ontea na základe týchto regulárnych výrazov nájde nasledujúce objekty (obr. 6).



Obr. 4. Príklad emailu odoslaného používateľom (objednávka komponentov kamerového systému)

```
EURO_AMOUNT_PATTERN=\\b([1-9][0-
9]+[,.]?[0-9]*) *(EUR|EURO|euro)
EURO_AMOUNT_CLASS=amount:euro

VAT_AMOUNT_PATTERN=\\b(VAT|vat)\\b
VAT_CLASS=code:vat

AMOUNT_VAT_PATTERN=\\b([1-9][0-
9]+[,.]?[0-9]*)
*(EUR|EURO|euro|SKK|Sk|\$)? *\\+ *
(VAT|vat)
AMOUNT_VAT_CLASS=amount:number_vat

PRODUCT_TYPE_PATTERN=\\b([vV]ideocamera
s?|[Cc]ameras?|[Mm]otherboards?|CPU|[Ff
]ramegrabbers?|[Ss]ervers?|[cC]ablings?
|coaxial|RS232|network switch(es)?)\\b
PRODUCT_TYPE_CLASS=product:type

PRODUCT_CODE_PATTERN=\\b\\p{Lu}[0-
9]{3}-\\p{Lu}{2,3}\\b
PRODUCT_CODE_CLASS=product:code

BY_ZIP_PATTERN=(\\p{Lu}[a-z]+[
]*\\p{L}*)[ ]+\\p{Lu}{2}[ ]*[0-9]{5}
BY_ZIP_CLASS=Location

BY_IN_PATTERN=\\b(in|near by) +(the)?
*(\\p{Lu}\\p{Ll}+ \\p{Lu}\\p{Ll}+
\\p{Lu}\\p{Ll}+|\\p{Lu}\\p{Ll}+
\\p{Lu}\\p{Ll}+|\\p{Lu}\\p{Ll}+)
BY_IN_CLASS=Location
BY_IN_GROUP=3
```

Obr. 5. Ukážka regulárnych výrazov.

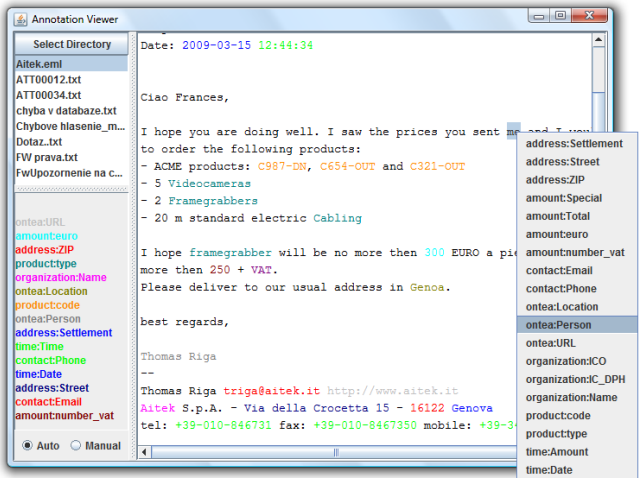
```
amount:euro 300
amount:number_vat 250
product:type Videocameras
product:type camera
product:type Framegrabbers
product:type framegrabber
product:type Cabling
product:code C987-DN
product:code C321-OUT
product:code C654-OUT
Location Genova
Location: Genoa
```

Obr. 6. Objekty nájdené nástrojom Ontea v ukážkovom emaili z obrázku 4.

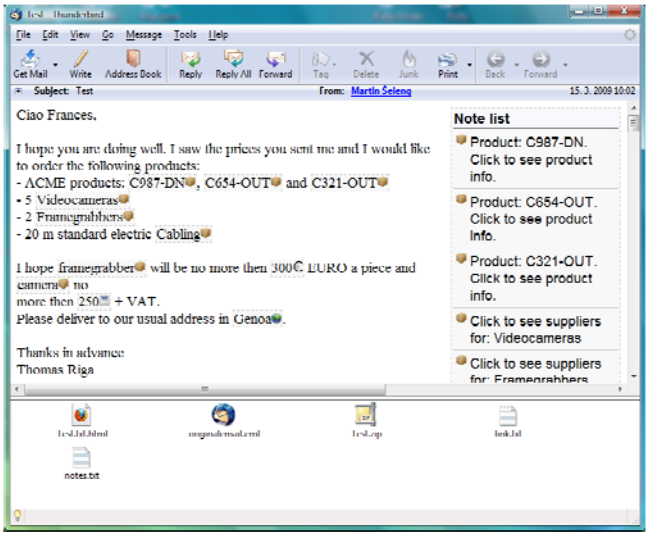
```
EURO_AMOUNT_NOTE=Amount {amount:euro}.
See in other currency
EURO_AMOUNT_URL=http://www.google.com/s
earch?q={amount:euro}EUR
EURO_AMOUNT_MATCH=amount:euro
EURO_AMOUNT_TYPE=money

AMOUNT_VAT_NOTE=Amount
{amount:number_vat} + VAT. Click to see
value
AMOUNT_VAT_URL=http://www.google.com/se
arch?q={amount:number_vat}*1.19
AMOUNT_VAT_MATCH=amount:number_vat
AMOUNT_VAT_TYPE=calc

SUPPLIER_NOTE=Click to see suppliers
for: {product:type}
SUPPLIER_URL=&module=Supplier&q_product
_type={product:type}
SUPPLIER_MATCH=product:type
SUPPLIER_TYPE=product

PRODUCT_NOTE=Product: {product:code}.
Click to see product info.
PRODUCT_URL=&module=Product&q_product_c
ode={product:code}
PRODUCT_MATCH=product:code
PRODUCT_TYPE=product

GOOGLE_MAPS_NOTE=See {Location} in
Google Maps
GOOGLE_MAPS_URL=http://maps.google.com/
maps?f=q&hl=en&geocode=&q={Location}
GOOGLE_MAPS_MATCH=Location
```

Obr.7. Objekty a im prislúchajúce textové poznámky.

Ďalej sú definované jednoduché poznámky s odkazmi na externé dátové zdroje (obr. 7), ktoré musí definovať administrátor. V budúcnosti to však bude umožnené aj bežnému používateľovi cez používateľské rozhranie (prvotná implementácia takéhoto rozhrania sa nachádza na



Obr.8. Používateľské rozhranie pre vytváranie patternov na extrakciu a anotáciu objektov v emailovej správe.



Obr. 9. Príklad emailu upraveného nástrojom Acoma.

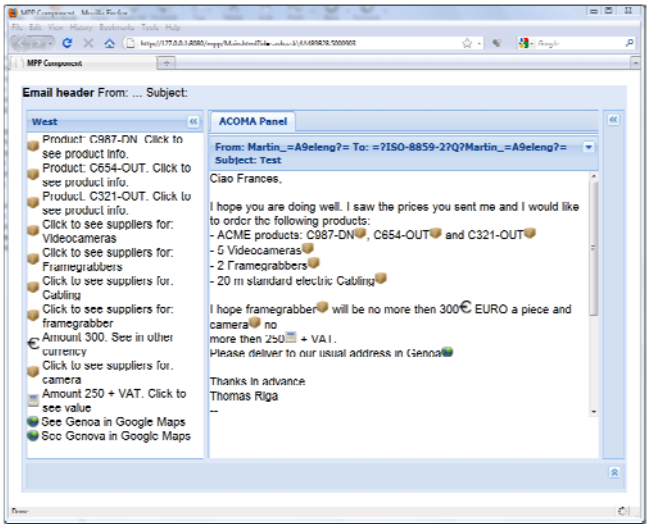
obr. 8). Vytváranie a špecifikácia jednotlivých regulárnych výrazov (obr. 6) bude neskôr možné pomocou grafického rozhrania a jednoduchých makier s preddefinovanými regulárnymi výrazmi.

Na základe nájdených objektov sa definované poznámky pridajú do emailu ako inline textová príloha, resp. naformátovaná HTML príloha (obr. 9), ak sú splnené podmienky vo vlastnosti „MATCH“.

Na nasledujúcom obrázku je zobrazený prijatý email, ktorý obsahuje prepojenia na zdroje (v tomto prípade prepojenia na webové aplikácie). Tieto zdroje súvisia s obsahom emailu a podporujú používateľa v ľahšom splnení úlohy, ktorú emailová správa reprezentuje.

Jednou z liniek pridaných do textu emailovej správy je aj linka do lokálneho webového kontajnera (vo frameworku Acoma sme použili kontajner Jetty[10]). V tomto kontajnery sa dá zobraziť spracovaná emailová správa s dodatočnými informáciami, získanými napr. pomocou systémových konektorov ako sú: prepojenie do databáz produktov, databáz dodávateľov, účtovníckych webových systémov a pod. (obr. 10).

[10] http://www.mortbay.org



Obr.10. Spracovaná emailová správa s dodatočnými informáciami o objektoch nachádzajúcich sa v správe.

V nasledujúcej časti si predstavíme jednotlivé komponenty frameworku Acoma a uvedieme zoznam použitých technológií.

Systém Acoma sa skladá zo štyroch hlavných častí:

- Acoma Core
- Acoma Server
- Acoma MultiThread
- Acoma Email
- Acoma Attachment processing
- FelixCore

Acoma Core slúži len na načítanie konfiguračného súboru so špecifickými nastaveniami pre určitú doménu.

Acoma Server je zodpovedný za počúvanie a prijímanie prichádzajúcich žiadostí o pripojenie so strany poštového klienta (v SMTP aj POP3 implementácii).

Acoma MultiThread je zodpovedný za preposielanie komunikácie medzi poštovým klientom a serverom.

Acoma Attachment processing je zodpovedný za skonvertovanie jednotlivých príloh (typu: pdf, html, do, xls, ppt, ...) do textovej formy, pre následné spracovanie nástrojmi zodpovednými za extrakciu informácií (napr. nástrojom Ontea).

Acoma Email slúži na dekompozíciu prijatého emailu, uloženie jednotlivých príloh, spúšťanie jednotlivých modulov, následné upravovanie emailovej správy a vytvorenie nového emailu s pridanými textovými/html informáciami.

FelixCore slúži na správu nainštalovaných modulov (v špecifikácii OSGi hovoríme o tzv. bundles) – pridanie, odobratie a spustenie modulu.

Framework Acoma používa na prácu s emailovými správami JavaMail API[11], pre vytváranie HTML príloh a GUI jednotlivých modulov používa GWT[12], ktorý

[11] http://java.sun.com/products/javamail

[12] http://code.google.com/webtoolkit

používa technológiu AJAX[13] a pre správu a vývoj modulov je použitá technológia OSGi[14], konkrétne implementácia Felix[15].

3 Aplikácia a prispôsobenie

Veríme, že zvolený prístup môže byť použitý v aplikáciách kde je potrebné:

- spravovanie znalostí,
- vyhľadávanie sociálnych sietí,
- manažovanie informácií v organizáciách,
- a podporu medzipodnikovej spolupráce.

Momentálne sa najviac zameriavame podporu medzipodnikovej spolupráce. V súčasnosti je množstvo emailov vytváraných automaticky pomocou Web 2.0 aplikácií[16]. Tieto informácie sú štruktúrované avšak stale relatívne jednoducho spracovateľné aj človekom. Vďaka tomu sa dajú tieto jednoducho extrahovať aj pomocou počítača a následne vkladať do iných aplikácií ako sú: databázové systémy, pracovné hárky, webových formulárov, a pod.

Prispôsobenie framework Acoma pre ľubovoľnú organizáciu je podmienené iba implementovaním modulov založených na technológii OSGi s vopred definovaným rozhraním (obr. 11).

```
public abstract class Module {
  public abstract String getName();
  public abstract String
  getDescription();
  public abstract Set<ModuleResult>
  execute(String path, String acomaid,
  String[] args);
  public abstract boolean check(String
  rawEmailText);
  public abstract void
  configuration(String[] conf);
}
```

Obr. 11. Rozhranie modulov spúšťaných frameworkom Acoma.

Rozhranie sa skladá z piatich metód:

- getName() vráti meno modulu
- getDescription() vráti popis modulu
- execute(String path, String acomaid, String[] args) spustí zvolený modul
- check(String rawEmailText) overí, či daný modul vie spracovať email
- configuration(String[] conf) nakonfiguruje daný modul

[13] http://en.wikipedia.org/wiki/Ajax_(programming)

[14] http://www.osgi.org

[15] http://felix.apache.org

[16] http://www.returnpath.net/blog/2008/07/case-study-web-20-runs-on-emai.php

4 Existujúce riešenia

Ako už bolo spomínané v úvode, boli snahy integrovať kontextové informácie do emailových správ (kMail, Zimbra a Gmail). Ďalšími snahami bolo vytvorenie pluginov do existujúcich emailových klientov ako je napr. Xobni, (plugin pre MS Outlook), ktorý poskytuje rôzne informácie vzťahujúce sa k odosielateľovi emailovej správy ako sú kontaktné informácie odosielateľa. Ďalej boli vyvinuté nasledujúce výskumné a vývojové riešenia (prototypy), ktoré sa zameriavajú na spravovanie úloh vyplývajúcich z emailovej komunikácie, archiváciu emailov, manažment informácií vyextrahovaných z emailov a na kolaboračné aspekty emailov: Telenotes, ContactMap, TaskMaster, Snarf, Remail, Priorities alebo Semanta.

Najväčší rozdiel medzi frameworkom Acoma a existujúcimi riešeniami je priame napojenie Acomy na existujúcu emailovú infraštruktúru, pričom používateľ nie je nútený meniť svoju každodennú emailovú prácu.

Podrobnejšie informácie o existujúcich riešeniach a ich porovnanie je možné nájsť tu [4, 14].

5 Záver a budúca práca

Článok opisuje možnosť využitia znalostí v organizácii tak, aby implementácia ich využitia nezasahovala do zabehnutého pracovného procesu. Pri väčšine projektov manažmentu znalostí sa v organizáciách inštalujú nové systémy, s ktorými sa používateľ musí naučiť pracovať. V prípade nástroja Acoma nie je potrebné inštalovať nové systémy - používateľ dostane relevantné informácie a znalosti priamo pri vybavovaní úloh prostredníctvom emailovej komunikácie. Dané informácie môže, ale nemusí využiť, pričom ho neobťažujú v zabehnutých pracovných postupoch. Zdá sa, že je vhodné použiť takýto systém všade tam, kde sa elektronická komunikácia používa ako primárny nástroj na manažovanie pracovného procesu.

Moduly vyvíjané ľubovoľnou spoločnosťou môžu byť jednoducho zaradené do systému Acoma, ktorý ich v prípade potreby dokáže stiahnuť z webu, doinštalovať a spustiť nad určitými typmi emailov

Aj keď čiastočné vyhodnotenie frameworku bolo uskutočnené v rámci projektu Raport, v budúcnosti sa budeme snažiť o komplexnejšie vyhodnotenie nášho riešenia.

Poďakovanie

Táto práca je vyvíjaná a podporovaná projektmi Commius FP7-213876, APVV DO7RP-0005-08, AIIA APVV-0216-07, VEGA 2/7098/27, SEMCO-WS APVV-0391-06.

Referencie

1. S. Whittaker, C. Sidner: *Email overload: exploring personal information management of email*. In Proceedings of ACM CHI'96 Conference on Human Factors in Computing Systems, 276-283.
2. D. Fisher, A.J. Brush, E. Gleave, M.A. Smith: *Revisiting Whittaker & Sidner's "email overload" ten years later*. In CSCW2006, New York ACM Press.
3. M. Laclavik: *Commius: ISU via email*. Workshop on Enterprise Interoperability Cluster: Advancing European Research in Enterprise Interoperability II at eChallenges 2007 conference.

4. M. Laclavik, M. Seleng, L. Hluchy: *Acoma: Network enterprise interoperability and collaboration using email communication*. In Proceedings of eChallenges 2007; Expanding the Knowledge Economy: Issues, Applications, Case Studies Paul Cunningham and Miriam Cunningham (Eds) IOS Press, 2007 Amsterdam ISBN 978-1-58603-801-4, 1078-1085.
5. M. Šeleng, M. Laclavík, Z. Balogh, L. Hluchý: *Automated content-based message annotator - Acoma*. In Vojtáš, Peter (editor). ITAT 2006: Information Technologies - Applications and Theory. - Department of Computer Science, Faculty of Science, P.J.Šafárik University, 2006. ISBN 80-969184-4-3, 195-198.
6. M. Laclavik, M. Seleng, E. Gatial, Z. Balogh, L. Hluchy: *Ontology based Text Annotation – OnTeA*. In: Proc. of 16-th European-Japanese Conference on Information Modelling and Knowledge Bases, EJC'2006, Y.Kiyoki et.al. eds., 2006, Dept.of Computer Science, VSB - Technical University of Ostrava, ISBN 80-248-1023-9. Trojanovice, Czech Republic, 280-284,.
7. M. Laclavik, M. Ciglan, M. Seleng, S. Krajci: *Ontea: Semi-automatic pattern based text annotation empowered with information retrieval methods*. In Hluchy, Ladislav. Tools for Acquisition, Organisation and Presenting of Information and Knowledge: Proceedings in Informatics and Information Technologies. Kosice : Vydavatelstvo STU, Bratislava, 2007. ISBN 978-80-227-2716-7, part 2, 119-129.
8. J. Habermas: *The Theory of Communicative Action*. Beacon, Boston, 1981.
9. D.G. Schwartz, D. Te'eni: *Bar-Ilan University, Tying Knowledge to Action with kMail*, MAY/JUNE 2000, IEEE Knowledge Management, 33-39.
10. D. Te'eni, D.G. Schwartz: *Contextualization in computer-mediated communication, information systems-the next generation.* L. Brooks and C. Kimble, eds., McGraw-Hill, New York, 1999, 327-338.
11. C.A. O'Reilly, L.R. Pondy: *Organisational communication, organisational behavior.* S. Kerr, ed., Grid, Columbus, Ohio, 1979, 119-150.
12. ftp://ftp.cordis.europa.eu/pub/ist/docs/directorate_d/ebuebusin/ei-roadmap-final_en.pdf
13. A. Lantz: *Does the use of email change over time*? International Journal of Human-Computer Interaction, 15, 3, June 2003, 419 – 431.
14. M. Laclavík, D. Maynard: *Motivating intelligent email in business: an investigation into current trends for email processing and communication research.* To appear in CEC2009 IEEE Proceedings, E3C Workshop, 2009, http://ikt.ui.sav.sk/archive/publications/e3c_sota_email.pdf.

Architecture of the secure agent infrastructure for management of crisis situations

Branislav Šimo, Zoltán Balogh, Ondrej Habala, Ivana Budinská and Ladislav Hluchý

Institute of Informatics, Slovak Academy of Sciences,
Dúbravská cesta 9, 845 07 Bratislava, Slovakia
{branislav.simo, zoltan.balogh, ondrej.habala, ivana.budinska, ladislav.hluchy}@savba.sk

Abstract. *This paper describes the architecture and design of the secure agent infrastructure for management of crisis situations. The purpose of this infrastructure is semi-automatic control of the crisis management process and crisis management personnel support during a crisis (natural disasters or accidents). It focuses on information provisioning from human actors and management of resources needed for crisis mitigation and resolution. One of the key features is creation of a secure agent execution environment, which enables secure information exchange among trusted parties.*[1]

1 Introduction

During the crisis it is important to effectively manage distributed material and human resources needed for crisis mitigation. Timely information delivery about the crisis situation and fast resource deployment into the crisis area are essential for minimizing the losses. The information being collected can be very sensitive, because it can include private information about citizens affected, classified military data or other information that must be kept secret.

Effective distribution of material and human resources is one the aims of the EU FP7 project SECRICOM.

1.1 Security challenges

These problems mostly relate to distributed nature of computation execution and data processing. Many security problems are already solved such as secure communication tunneling through encryption or authorization and authentication using asymmetric cryptography. Security challenges for distributed computing can be generally divided into two groups: privacy and trust. Both of these security areas can be solved either on the side of clients (initiators) or on the side of executors (servers).

Communication, security and accessibility of information are the key factors during management of crises situations. Secure communication is a technological challenge which must be solved in a complex manner. It is important to solve interconnection of multiple communication channels but also protection from misuse of information and communication flows.

In this article we present architecture of a distributed system for secure execution of mobile code implemented as mobile services in untrustworthy computing environment using secure hardware platform module for the management of crises situations.

[1] This work was supported by the following projects: SECRICOM SEC-2007-4.2-04, SEMCO-WS APVV-0391-06, VEGA No. 2/6103/6, VEGA 2/7098/27.

2 Secure agent infrastructure requirements

The role of agents in our system is primarily coordinated collection of information. Gathering of information is enacted either from legacy systems or from human end-users through mobile devices by guided dialog. In respect to requirements the overall agent infrastructure must be a secure, robust and fail resistant system. An agent as technology was selected due to the ability to fulfill such requirements through support of mobile and dynamically deployable executable code.

2.1 Infrastructure security requirements

In order to define concrete security requirements we must sketch the basic infrastructure in which agents will operate (Fig. 1). The network of Trusted Servers (TS) is the home platform for agents. According to [3] the platform from which an agent originates is referred to as the ***home platform***, and normally is the most trusted environment for an agent. This is also true for our agents – the network of TS is a managed set of systems with defined security policies and possibly managed by a central authority. From here agents are delegated to host platforms to gather data and information. TS host core services of the agent platform. Agents are mainly executed on remote sites which provide computational environment in which agents operate. We will refer to these sites as to ***host platforms*** *(or agent platform).*

In general any party which wishes to join the system and to provide information from his legacy systems or users must introduce a host platform for agents. We will refer to such parties as to Host Platform Providers (HPP). From end-user requirements the following HPPs were identified so far (Fig. 1):

- *Resource Providers* – hospitals, fire brigade, police, warehouses or any other entities which can play a role in the mitigation of crisis situation,
- *Command Centers* – mobile (nomadic) centers which coordinate locally the incident site;
- *General Command Center* and *Operators* – usually located in one place or at least tightly interconnected.

The features of the agents will encompass several carefully chosen attributes:

- *Code mobility* (without execution state) – ability to move code to different platforms and execute there, within the project we do not plan to support execution state mobility (since there is no requirement for that),

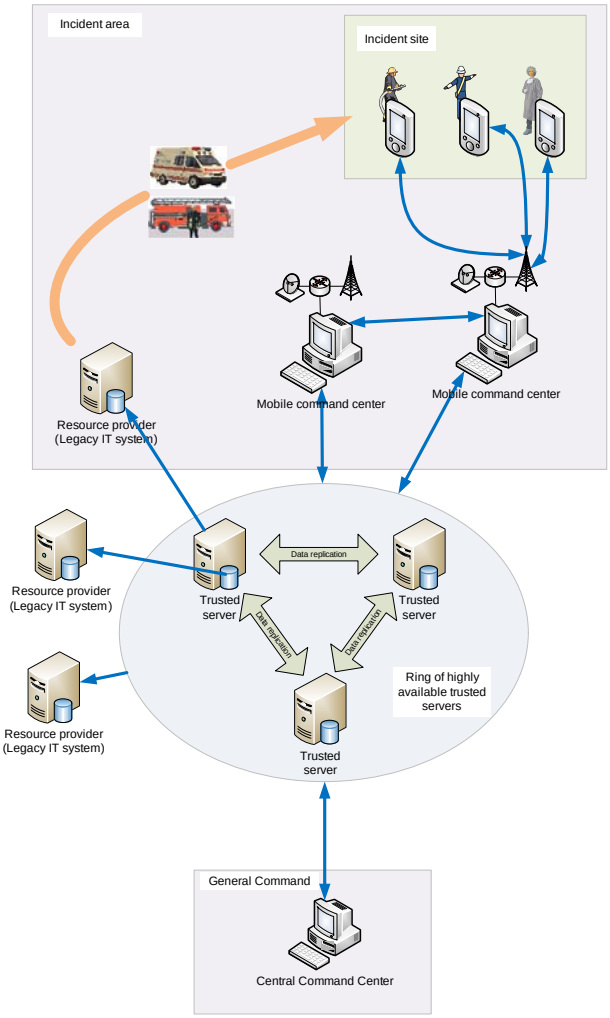


Fig. 1. Secure Agent Infrastructure deployment overview.

- *Autonomy* – ability to deliver gathered data to one or several optional destinations,
- *Reactivity* – in some cases agents will perceive the context in which they operate and react to it appropriately (e.g., agents can monitor availability of some resource and notify the requestor).

Since agents collect information which is often classified, while at the same time requirements for action or decision traceability exist, agents must be provided with secure, trusted and attested execution environment. In the following we identify main agent-related security threats. A detailed explanation of generic mobile agent security aspects is discussed in [3]. Generally four threat categories are identified:

- Agent platform attacking an agent,
- Agent attacking an agent platform,
- Agent attacking another agent on the agent platform,
- Other entities attacking the agent system.

The last category covers the cases of an agent attacking an agent on another agent platform, and of an agent platform attacking another platform, since these attacks are primarily focused on the communications capability of the platform to exploit potential vulnerabilities. The last category also includes more conventional attacks against the underlying operating system of the agent platform.

2.1.1 The host platform attacking the agent

The main threat for agents in foreign execution environment of host platforms is the "malicious host problem". This is one of main problems in the class of "an agent platform attacking an agent". Simple explanation of "malicious host problem" is provided in [4]: "Once an agent has arrived at a host, little can be done to stop the host from treating the agent as it likes". Therefore the main requirements from the agent-side are lied out in respect to the "malicious host problem". The concrete security requirements of agents in respect to the host platform are therefore the following:

- Isolated execution environment for agent execution – not only virtual isolated execution environment but dedicated isolated hardware preferred;
- Means to attest the platform required in order to detect if the host platform is in trusted state;
- Protected storage for credential data (such as PKI's secret key).

2.1.2 The agent attacking the host platform

There are also threats stemming from an agent attacking an agent host platform. Therefore reversely a host platform has also requirements in respect to agents. These requirements are more evident when provided in context of HPPs security requirements:

1. HPPs do not want to install and execute any external application (including SECRICOM system) on their systems in line with their strategic legacy applications.
2. HPPs prefer to have a dedicated and isolated system for SECRICOM which would connect to their legacy system in a secure predefined way.
3. HPPs want to be able to control what (data), when and by who (traceability) is provided to the SECRICOM system.
4. HPPs want to be able to constrain the set of applications executable on their site. Agents must be therefore audited and verified, thus mediating trust to executable agent code.

The agent platform has the following security requirements in respect to agents:

- Isolated execution environment for agent execution - agents must be executed in isolated environment (isolated hardware preferred), so an agent can not harm legacy systems;
- Means to monitor and trace agents activity;
- Means to configure the set of agents executable on the host platform;

In order to track agents, any agent in the platform must be cryptographically signed. Only agents signed with trusted authority and assigned to selected category will be trusted by a host system.

Agents need to send signed messages to Trusted Servers.

2.1.3 The agent attacking another agent

It is required that any agent which will be used in SECRICOM will need to be audited and certified by a central authority. In turn every host platform will be configured to execute only agents which are certified. These two security policies should ensure that malicious agents will not be deployed into the infrastructure. Only breach of the set security policies might lead to potential agent-to-agent security risk.

Moreover each agent will be executed in a relatively isolated virtual environment with limited access to data of other parallel executed agents on the same host platform.

2.1.4 Other entities attacking the agent system

Agents will also connect to legacy systems (third party software). Therefore a risk of attacking agent by a legacy system but also vice versa – risk of attacking legacy system by an agent exist.

The host platforms will need to provide some kind of connection to legacy systems. We explicitly presume that this will be a network connection. On any network connection there is an eavesdropping risk. Therefore another requirement which arises from agents to the host platform is:

- Secure protected connection to legacy systems.

Physical security of network connection can be achieved either by direct cable connection of the host platform with legacy system or by managed network security (managed switch with well defined security policies). The data transport security will be achieved primarily through encryption.

2.2 Agent life cycle and related security requirements

The life cycle of an agent in the secure agent infrastructure (SAI) is the primary source of security requirements of the SAI. The creation of an agent encompasses development of its code, audit and certification. After successful certification of its code it is equipped with a private key, which is (for all of its existence) available only to the agent itself. A corresponding public key is stored and accessible in a public key registry. After its certification and "priming" with a private key, the agent is stored in an agent registry (AR). From this registry, it is downloaded to a trusted docking station (TDS or just DS) which needs to use the agent's capabilities. The downloaded copy has to be at all times secure – during transfer from the AR to the requesting TDS, and also during its deployment and execution inside the TDS. The copy inside TDS is destroyed when it finishes executing and delivers its results. The life cycle of an agent ends when its certificate expires, and after this it may be deleted from the AR, since it cannot be deployed anywhere in the SAI anymore. See Fig. 2 for a graphical representation of this process.

The life cycle of an agent poses following requirements on the security infrastructure:

- The agent must contain its private key; this key must not be known to any other entity during the whole lifetime of the agent.

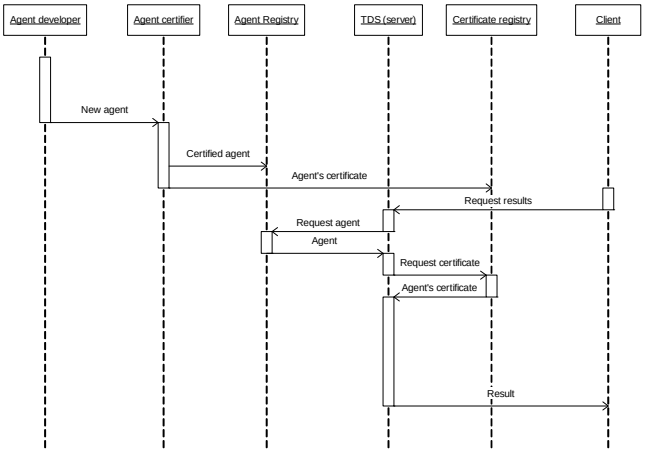


Fig. 2. Agent life cycle.

- The agent must be audited before it can be used; the audit must ensure, that the agent does only what its creator states it should do, and that it does not contain any malicious code, which may jeopardize the integrity of the execution environment.
- The agent must be protected at all times from revealing its private key; it must always be either stored in a trusted device, or encrypted when it is outside of such device.
- Each audited agent must be issued a certificate, signed by its auditor, which states the capabilities of the agent as specified by its creator and verified by the auditor.
- The execution infrastructure must contain a service for storing and accessing certificates of entities inside the infrastructure; one class of these entities are also the software agents.
- All results produced by a software agent must be protected from being revealed to any entity different than their intended recipient – the client. Also, it must be asserted that their authenticity can be verified by the client upon their reception.
- The results provided by an agent must be signed by both the agent and the device running agent's code in order to ensure the trust of the results by Process Management System. Each secured device is supposed to be connected to secure docking module (SDM) providing the encryption keys authenticating the device and its user, respectively.

3 Architecture

This section presents the architecture for systems which could profit from the combination of mobile code execution on an isolated trusted hardware connected to legacy computing resource. The architecture is designed for mobile services with agent-like features (mobility, pro-activity) which would execute on secure devices.

Such architecture in general consists of interconnected trusted (TS) and un-trusted servers (US). TS carry out the following tasks: registry of services, users and modules, public encryption keys, the agent base (base of mobile code) or generic security politics. Each agent has features and "abilities", which are used for the enactment of certain

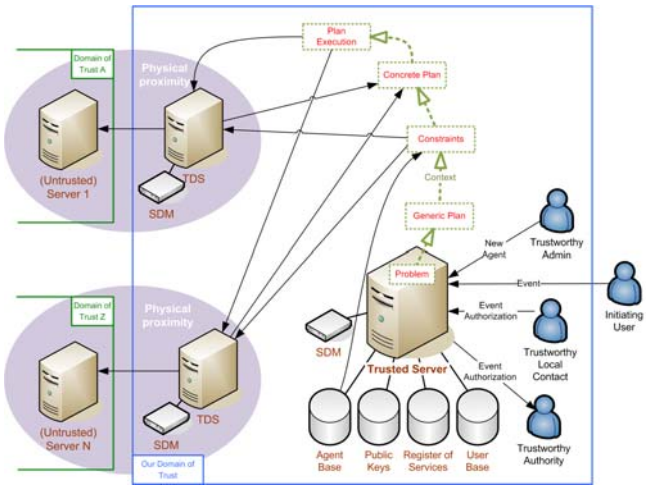


Fig. 3. The architecture for a distributed system for secure execution of agent-based mobile code based in an untrustworthy computing environment.

processes. The enactment of processes is inspired by the domain of management of crises situations in which collection of information from multiple legacy environments is required. The whole process starts with the specification of a problem in the form of dialog. Further certain agent (service) will try to specify the most serious problem which was rendered by the crises situation. Based on the type of crises situation and the region where the crises has arose appropriate actions are initiated for each crises situation type.

The system will semi-automatically generate plausible generic plans of possible solutions of rendered problems. In the next step the specification of context will be enacted in order to be able to generate the constraints of the crises situation. Relevant servers will be identified in the central database based on generated constraints. Agents which are able to query selected servers will be selected from the agent base. Information about available capacities will be retrieved from identified servers and sent back to central trusted server base. The system will then generate a concrete plan of crises situation resolution based on the retrieved disposable resource (human, material, etc.) capacities. The last step is execution of prepared plan for the concrete crises situation.

3.1 Agent Registry

Agent Registry (AR) is a service, which stores all the existing software agents in our infrastructure. The registry itself resides inside a TDS. The registry must ensure that any agent stored inside it is secure, and will be handled in a manner which will not reveal the secrets it contains to none but the authorized parties. AR has the following requirements on the security infrastructure of SECRICOM:

- Any request for a software agent to be downloaded from the registry and deployed inside a device must clearly state the recipient of the agent.
- All devices in the SECRICOM infrastructure (TDS and others) must be issued a certificate stating which agents it may receive; AR will reveal to a device only such agents, and will deny the deployment of agents for which the device is not certified.

The agent intended for deployment in a device must be protected during transport from being revealed to third parties; it must be encrypted in a manner which allows only the specific pair of a device and an agent, to which it is addressed, to be able to decrypt it and execute it.

4 Conclusion

In this paper we have described the architecture and design of the secure agent infrastructure for management of crisis situations. One of the key features is creation of a secure agent execution environment, which enables secure information exchange among trusted parties. We have also presented an architecture which is designed for execution of agent-like mobile code that executes on a secure trusted platform connected to legacy computational environment. The presented result is work in progress.

References

1. Trusted Computing Group, URL: https://www.trustedcomputinggroup.org/home
2. Trusted Platform Module, URL: https://www.trustedcomputinggroup.org/groups/tpm/Trusted_Platform_Module_Summary_04292008.pdf
3. W. Jansen, T. Karygiannis: *Mobile Agent Security – NIST* Special Publication 800-19. National Institute of Standards and Technology, Computer Security Division, Gaithersburg, MD 20899.Niklas.
4. Borselius: *Mobile agent security*. Electronics & Communication Engineering Journal, October 2002, 14, 5, IEE, London, UK, 211-218.
5. A. Ferreira, R. Cruz-Correia, L. Antunes, and D. Chadvick: *Access control: How can it improve patients' healthcare?*, In: Studies in Health Technology and Informatics, 127, June 2007, http://www.cs.kent.ac.uk/pubs/2007/2625/content.pdf

Preferencie ako usporiadanie objektov: formálny model a implementácia*

Veronika Vaneková

Pavol Jozef Šafárik University in Košice, Slovakia,
veronika.vanekova@upjs.sk

Abstrakt *V tomto článku skúmame rôzne teoretické modely používateľských preferencií. Navrhujeme model preferencií reprezentovaný ako usporiadanie objektov podľa rôznych kritérií. Usporiadanie je dané prostredníctvom členstva objektu vo fuzzy množine. Tento model definujeme v rámci jednostranne fuzzifikovanej deskripčnej logiky. Ako druhú alternatívu navrhujeme logiku, v ktorej poradie objektov nie je dané fuzzy hodnotou, ale priamo ako binárna relácia usporiadania. Popisujeme systém, ktorý implementuje tento model preferencií a na jeho základe vyhľadáva najlepšie objekty. Hlavným prínosom je model používateľa, vhodný na automatické odvodzovanie v deskripčných logikách, ako aj na efektívne dopytovanie pomocou top-k algoritmu.*

1 Úvod

Jedným z cieľov Webu 2.0 je väčšia orientácia a dôraz na používateľa. S tým súvisí vývoj sociálnych sietí, recommenderov a vyhľadávačov. Mnohé systémy menia svoju koncepciu a snažia sa v maximálnej miere prispôsobiť používateľovi. Pritom je dôležité reprezentovať, spracovať a uložiť čo najviac informácií o používateľoch a ich preferenciách pri vyhľadávaní. Vo všeobecnosti čím viac informácií o používateľovi má systém k dispozícii, tým relevantnejšie výsledky systém podľa nich vypočíta. Zadávanie informácií z pohľadu používateľa však nesmie byť príliš zdĺhavé či komplikované.

Rôzne prístupy k riešeniu tohto problému porovnávame v kapitole 2. Analyzujeme výhody a nevýhody rôznych modelov preferencií a navrhujeme vlastný model založený na usporiadaní objektov prostredníctvom fuzzy množín, ktorý podporuje preferenčné vyhľadávanie najlepších objektov. Kapitola 3 opisuje systém preferenčného vyhľadávania založený na fuzzy modeli. Systém je použiteľný pre rôzne domény a rozšíriteľný o ďalšie metódy získavania preferencií. V kapitole 4 sa venujeme fuzzy deskripčným logikám, ktoré predstavujú vhodný teoretický základ pre uvedený model. Ďalej skúmame možnosť abstrahovať od fuzzy hodnôt a interpretovať preferencie priamo ako usporiadania objektov. V kapitole 5 uvádzame niektoré zaujímavé problémy z oblasti používateľských preferencií a ďalší plánovaný výskum v tejto oblasti.

* Táto práca je čiastočne podporovaná projektami VEGA 1/0131/09 a VVGS/UPJŠ/45/09-10.

2 Modely preferencií

Preferenciu definujeme ako reláciu usporiadania na danej doméne. Každá doména obsahuje objekty hľadané používateľmi, napríklad notebooky, hotely, pracovné ponuky, letenky a podobne. Objekty viac preferované používateľom sú v preferenčnej relácii pred menej preferovanými objektmi. Podrobnejší úvod do problematiky preferencií sa nachádza v príručke [9]. Klasická preferenčná logika je definovaná v [15], iný prístup je založený na modálnej logike [16].

Preferencie sa využívajú v rôznych aplikáciách a od presného účelu aplikácie závisí aj ich reprezentácia. Prvou oblasťou, kde sa preferencie často využívajú, sú internetové obchody. Systémy nazývané *recommendery* [1] odporúčajú zákazníkom potenciálne zaujímavé výrobky a tým uľahčujú orientáciu a vyhľadávanie na stránke. Preferencie sú tu reprezentované ako vektor $(p_1, \ldots, p_n)$, kde p_i je ohodnotenie i-teho výrobku. Ohodnotenie zadá používateľ priamo na stránke (ako hlasovanie o kvalite výrobku) alebo ho systém odvodí z toho, že si používateľ daný výrobok objednal. Výrobky, ktoré používateľ nikdy nevidel, majú neznáme ohodnotenie. Úlohou recommendera je odhadnúť ohodnotenie výrobku, ktorý používateľ ešte nevidel a odporúčať mu podľa toho najlepšie výrobky.

Na určenie tohto ohodnotenia sa používa *kolaboratívne filtrovanie*, *obsahové filtrovanie* a ich kombinácie [1]. Pri kolaboratívnom filtrovaní systém určí pre daného používateľa skupinu podobných používateľov na základe podobnosti ich doterajších vektorov preferencií. Preferencie objektov, ktoré daný používateľ ešte neohodnotil, sa potom určia ako priemer v rámci skupiny. Pri obsahovom filtrovaní systém najskôr nájde skupiny podobných objektov, spočíta priemerné hodnotenie používateľa pre každú skupinu, vyberie najlepšie hodnotenú skupinu a následne odporúča všetky objekty z danej skupiny, ktoré používateľ ešte nevidel.

Preferencie hrajú dôležitú úlohu aj v *multikriteriálnom rozhodovaní* – výbere najlepšej alternatívy podľa viacerých kritérií [8]. Používateľ má k dispozícii konečnú množinu objektov a množinu vlastností objek-

tov $J = \{j_1, \ldots, j_n\}$. Objekty je možné rôzne usporiadať podľa každej vlastnosti j_i. Agregáciou získame ich celkové usporiadanie.

V oblasti multikriteriálneho rozhodovania sa veľmi často uplatňujú princípy fuzzy logiky [7]. V článku [6] sú opísané tri fuzzifikované modely preferencií. Prvým je usporiadanie objektov (binárna relácia) podľa určitého kritéria, druhým je *fuzzy preferenčná relácia* $p^k = [p^k_{ij}]$, $p^k_{ij} \in [0,1]$, ktorej prvky p^k_{ij} vyjadrujú, do akej miery je objekt x_i lepší než objekt x_j podľa kritéria k. Tretím modelom je funkcia f^k, kde $f^k(x_i)$ vyjadruje mieru preferencie objektu x_i podľa kritéria k. Ide vlastne o členskú funkciu fuzzy množiny, ktorej hodnota je zo spojitého intervalu $[0,1]$ a vyjadruje preferenciu používateľa k jednej konkrétnej vlastnosti objektu. Pri určení výsledného poradia objektov sa najčastejšie používa OWA operátor [5], teda usporiadaný vážený priemer.

Podobná metóda *rank-aware querying* [3] používa agregáciu na výpočet ohodnotenia objektov. Je implementovaná v relačnej databáze pomocou efektívnych rank-join operátorov.

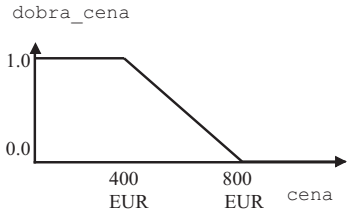
Ďalšia metóda rozhodovania podľa viacerých kritérií je založená na *skyline operátore* [4]. Každá vlastnosť objektu tvorí jeden rozmer priestoru a objekty tvoria body v priestore. Hľadáme tie objekty, pre ktoré neexistuje objekt vo všetkých vlastnostiach rovnako dobrý alebo lepší, a zároveň lepší aspoň v jednej vlastnosti. Nevýhodou je, že objekty zo skylinu nie sú usporiadané a nezohľadňujú, že niektoré vlastnosti objektov sú pri rozhodovaní dôležitejšie než iné.

Rozhodovaniu podľa neúplných kritérií sa venuje teória tzv. hrubých množín (*rough sets* [2]). Používa dva druhy atribútov, podmienkové (vlastnosti objektov) a rozhodovacie (celkové hodnotenia objektov). Objekty s rovnakými hodnotami vlastností sa nazývajú *nerozlíšiteľné* a tvoria triedu ekvivalencie. Ak množina obsahuje jeden objekt z triedy ekvivalencie a neobsahuje iný objekt, potom výber objektov do tejto množiny je na základe nejakého ďalšieho, neznámeho kritéria. Takáto množina sa nedá presne definovať na základe známych atribútov a nazýva sa hrubá množina. Cieľom je nájsť pravidlá, ktoré podľa vlastností objektov určia výsledné hodnotenie.

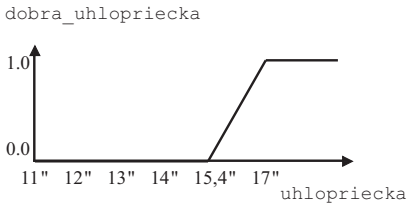
2.1 Fuzzy model používateľa

Fuzzy model používateľa [11] je určený pre vyhľadávanie najlepších objektov podľa viacerých vlastností, v ľubovoľnej doméne. Rozlišujeme *lokálne preferencie* používateľa (vzťahujú sa vždy len k jednej vlastnosti objektov a sú reprezentované ako fuzzy množiny hodnôt danej vlastnosti) a *globálne preferencie* (sú určené na výpočet výslednej hodnoty preferencie, reprezentované ako agregačné funkcie).

Ako príklad uvedieme doménu notebookov. Používatelia môžu vyberať najlepšie notebooky podľa celého radu vlastností od kapacity disku, RAM, rýchlosti procesora, ceny, uhlopriečky, počtu USB portov, hmotnosti, až po výrobcu či farbu. Každý používateľ môže mať špecifické preferencie ku každému atribútu, ale tiež mu môže záležať iba na niektorých. Ak napríklad hľadá lacný a širokouhlý notebook, jeho lokálne preferencie k atribútom `cena` a `uhlopriecka` môžu vyzerať tak, ako ukazujú obrázky 1, 2. Formálne ich definujeme ako fuzzy množiny $f^A_U : D_A \longrightarrow [0,1]$, kde D_A je množina hodnôt vlastnosti A.



Obr. 1. Fuzzy množina `dobra_cena` s nerastúcou charakteristickou funkciou.



Obr. 2. Fuzzy množina `dobra_uhlopriecka` s neklesajúcou charakteristickou funkciou.

Preferencie môžu byť vo vzájomnom konflikte, napríklad ak chceme lacný aj rýchly notebook. V tomto prípade fuzzy množiny `dobra_cena` a `dobra_rychlost` vytvoria rôzne usporiadania objektov. Globálne preferencie nám umožnia aj z rôznych usporiadaní vytvoriť jedno výsledné usporiadanie objektov. Formálne definujeme globálnu preferenciu ako monotónnu agregačnú funkciu $@_U : [0,1]^n \longrightarrow [0,1]$, ktorú aplikujeme na hodnoty lokálnych preferencií. Typickou agregačnou funkciou je vážený priemer, kde váhy vyjadrujú dôležitosť zodpovedajúcich vlastností. Ďalšou možnosťou sú *klasifikačné pravidlá* [10], napríklad:

`dobry_notebook(x)` ≥ 0.8 IF `dobra_cena(x)` ≥ 0.8 AND `dobra_rychlost(x)` ≥ 0.3

`dobry_notebook(x)` ≥ 0.4 IF `dobra_cena(x)` ≥ 0.5 AND `dobre_rozlisenie(x)` ≥ 0.6

Klasifikačné pravidlo pozostáva z hlavičky (napr. `dobry_notebook(x)` ≥ 0.8) a z tela, ktoré je konjunkciou podmienkových klauzúl (napr. `dobra_cena(x)` ≥ 0.8 a `dobra_rychlost(x)` ≥ 0.3). Ak objekt spĺňa tieto podmienky, potom výsledná hodnota preferencie bude minimálne taká ako hodnota v hlavičke pravidla. Zo všetkých splnených pravidiel vyberieme tú najvyššiu hodnotu v hlavičke, takže všetky nerovnosti budú splnené. Takéto pravidlá je možné získať indukciou z objektov ohodnotených používateľom [10].

3 Implementácia modelu používateľa pre preferenčné vyhľadávanie

Uvedený fuzzy model je implementovaný v systéme UPreA [13] na získavanie a správu používateľských preferencií v ľubovoľnej doméne. UPreA obsahuje grafické rozhranie na priame zadávanie preferencií, zabezpečuje ukladanie dát do ontológie a slúži ako rozhranie pre ostatné súčasti systému, hlavne induktívne učenie preferencií a top-k vyhľadávanie.

Používateľ po registrácii dostane zoznam atribútov z danej domény a má možnosť vybrať tie atribúty, ktoré považuje za dôležité pri rozhodovaní. UPreA podľa toho nastaví váhy v agregačnej funkcii. Následne používateľ nastaví lokálne preferencie pomocou grafického rozhrania. Zadané preferencie použijeme pri vyhľadávaní najlepších objektov pomocou top-k algoritmu. Detaily sa nachádzajú článkoch [11,13]. Výsledky môže používateľ ohodnotiť v škále 1 - 5, čím vlastne definuje nové (čiastočné) usporiadanie objektov. Na základe týchto ohodnotení získame indukciou nové globálne preferencie vo forme klasifikačných pravidiel [10].

V práci [13] opisujeme experiment so systémom UPreA – porovnávali sme usporiadanie výsledkov dané top-k algoritmom a usporiadanie dané hodnotením používateľa. Pomocou Kendallovho τ korelačného koeficientu sme vypočítali koreláciu dvoch uvedených usporiadaní. Výsledky experimentu ukázali, že pri použití klasifikačných pravidiel nastane vyššia korelácia, a teda model s pravidlami lepšie vystihuje skutočné preferencie používateľa.

UPreA ukladá preferencie do OWL ontológie (detaily sa nachádzajú v článku [11]). Teoretickými ekvivalentmi ontológií sú deskripčné logiky [14]. V nasledujúcej kapitole uvádzame dve deskripčné logiky prispôsobené preferenčnému vyhľadávaniu a porovnávame ich.

4 Deskripčné logiky s usporiadaním inštancií

Deskripčné logiky (DL) popisujú doménové znalosti pomocou *konceptov* a *rolí*. Koncepty môžeme považovať za unárne predikáty (interpretujeme ich ako množiny prvkov domény) a roly za binárne predikáty, ktoré vyjadrujú vlastnosti a vzťahy medzi konceptmi. Pomocou konceptov a rolí môžeme tvoriť výroky o jedincoch (prvkoch domény).

Každá deskripčná logika je charakterizovaná množinou konštruktorov prípustných pri vytváraní zložených konceptov a množinou tvrdení, ktoré sa môžu vyskytovať v znalostnej báze. Čím viac konštruktorov DL pripúšťa, tým zložitejšiu syntax získame. S tým úzko súvisia odvodzovacie problémy. V každej deskripčnej logike sú navrhnuté algoritmy na odvodenie implicitných znalostí z explicitne uvedených tvrdení (reasoning). Zložitejšia syntax sa teda odrazí aj na zložitosti odvodzovania nových znalostí.

4.1 $s - \mathcal{EL}(\mathcal{D})$

Okrem klasických deskripčných logík existuje značný počet fuzzifikovaných verzií. Na reprezentáciu nášho modelu preferencií používame špeciálnu DL s klasickými (crisp) rolami a fuzzy konceptmi. Označujeme ju $s-\mathcal{EL}(\mathcal{D})$ [12]. Fuzzy koncepty postačujú na reprezentovanie preferencií ako `dobry_notebook`, pričom všetky ostatné časti modelu sa dajú popísať aj bez fuzzy hodnôt. DL $s - \mathcal{EL}(\mathcal{D})$ obsahuje nasledujúce konceptové konštruktory: $\top$, konjunkcia $C \sqcap D$, existenčný kvantifikátor $\exists R.C$, konkrétne predikáty P, agregácia $@_U$ a top-k konštruktor. Základné konceptové konštruktory interpretujeme nasledovne:

$$A^{\mathcal{I}} : \Delta^{\mathcal{I}} \longrightarrow [0,1]$$

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \{1\}$$

$$(C \sqcap D)^{\mathcal{I}}(a) = \min\{C^{\mathcal{I}}(a), D^{\mathcal{I}}(a)\}$$

$$\exists R.C^{\mathcal{I}}(a) = \sup_{b \in \Delta^{\mathcal{I}}} \{C^{\mathcal{I}}(b) |\ (a,b) \in R^{\mathcal{I}}\}$$

$$\exists (u_1 \circ \cdots \circ u_k).P^{\mathcal{I}}(a) =$$

$$= \sup_{b \in \Delta^{\mathcal{D}}} \{P(b) |\ (a,b) \in (u_1^{\mathcal{I}} \circ \cdots \circ u_k^{\mathcal{I}})\}$$

$$top - k(C)^{\mathcal{I}}(a) =$$

$$= \begin{cases} C^{\mathcal{I}}(a), & \text{ak } \left\| b \in \Delta^{\mathcal{I}} |\ C^{\mathcal{I}}(a) < C^{\mathcal{I}}(b) \right\| < k \\ 0, & \text{inak} \end{cases}$$

$$@_U^{\mathcal{I}} : (\Delta^{\mathcal{D}})^n \longrightarrow \Delta^{\mathcal{D}}$$

Konkrétna doména $\mathcal{D} = (\Delta^{\mathcal{D}}, Pred(\mathcal{D}))$ sa skladá z domény $\Delta^{\mathcal{D}} = \mathbf{R}$ a z množiny predikátov $Pred(\mathcal{D}) = \{$ `nerastuca` $_{a,\ b}$, `neklesajuca` $_{a,\ b}$, `lichobeznikova`$_{a,b,c,d}\}$, ktorá obsahuje štandardné fuzzy predikáty s nerastúcou, neklesajúcou alebo lichobežníkovou členskou funkciou [11]. Ich interpretácia je pevná na rozdiel od konceptov. Napríklad množinu z obrázku 1 by sme zapísali ako `nerastuca`$_{400,800}(x)$.

4.2 $o - \mathcal{EL}(\mathcal{D})$

Fuzzy množiny majú v deskripčnej logike $s - \mathcal{EL}(\mathcal{D})$, opísanej v predchádzajúcej kapitole, špeciálne využitie: reprezentujú používateľské preferencie, ktoré sú prirodzene nepresné, vágne. Súčasne fuzzy hodnoty vytvárajú usporiadanie objektov podľa daných preferencií. Usporiadanie však môžeme reprezentovať iným spôsobom, ako binárnu reláciu. V tejto kapitole uvádzame DL so zhodnou množinou konštruktorov, ale s odlišnou interpretáciou konceptov. Na zdôraznenie tejto skutočnosti označujeme koncepty ako $C_{\leq}$ a interpretáciu ako $\mathcal{J}$.

$$A_{\leq}^{\mathcal{J}} \subseteq \Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$$

$$\top_{\leq}^{\mathcal{J}} = \Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$$

$$\text{Ak } (a_1, a_2) \in \ C_{\leq}^{\mathcal{J}} \wedge (a_1, a_2) \in D_{\leq}^{\mathcal{J}}$$

$$\text{potom } (a_1, a_2) \in (C_{\leq} \sqcap D_{\leq})^{\mathcal{J}}$$

$$(\exists R.C_{\leq})^{\mathcal{J}} \supseteq \{(a_1, a_2) |\ \forall c_1\ (a_1, c_1) \in R^{\mathcal{J}}$$

$$\exists c_2\ (a_2, c_2) \in R^{\mathcal{J}} : (c_1, c_2) \in C_{\leq}^{\mathcal{J}}\}$$

$$(\exists (u_1 \circ \cdots \circ u_k).P)^{\mathcal{J}} \supseteq \{(a_1, a_2) |\ \forall c_1\ (a_1, c_1) \in$$

$$(u_1^{\mathcal{J}} \circ \cdots \circ u_k^{\mathcal{J}})\ \exists c_2\ (a_2, c_2) \in (u_1^{\mathcal{J}} \circ \cdots \circ u_k^{\mathcal{J}}) :$$

$$P(c_1) \leq P(c_2)\}$$

$$@_U^{\mathcal{J}}(C_{\leq_1}, \ldots, C_{\leq_m}) \subseteq \Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$$

Z formálneho hľadiska interpretujeme koncept $C_{\leq}$ ako *predusporiadanie* domény, teda reflexívnu a tranzitívnu reláciu $C_{\leq}^{\mathcal{J}} \subseteq \Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$. Ak $(a, b) \in C_{\leq}^{\mathcal{J}}$, potom hovoríme, že a je preferované menej (alebo rovnako) ako b. Na rozdiel od relácie usporiadania tu nemusí platiť podmienka antisymetrie – môžu existovať dva rôzne objekty preferované rovnako, teda $(x, y) \in C_{\leq}^{\mathcal{J}}$ a $(y, x) \in C_{\leq}^{\mathcal{J}}$. Takéto objekty nazývame *nerozlíšiteľné podľa* C. Predusporiadanie je *úplné*, ak pre každú dvojicu objektov $\forall a, b \in \Delta^{\mathcal{J}} :\ (a, b) \in C_{\leq}^{\mathcal{J}} \vee (b, a) \in C_{\leq}^{\mathcal{J}}$ (platí jedna alebo obidve nerovnosti).

Top koncept $\top_{\leq}$ je interpretovaný ako úplná relácia $\Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$, kde sú všetky objekty preferované rovnako. Konštruktor $C_{\leq} \sqcap D_{\leq}$ má za výsledok čiastočné predusporiadanie, ktoré je možné rozšíriť na úplné predusporiadanie, ale rozšírenie nemusí byť jednoznačné. Každá interpretácia je daná jedným takýmto rozšírením. Okrem vyššie uvedených konštruktorov definujeme navyše $top - k$ a agregáciu:

Nech $C_a = \{c \in \Delta^{\mathcal{J}} |\ (a, c) \in C_{\leq}^{\mathcal{J}} \wedge (c, a) \notin C_{\leq}^{\mathcal{J}}\}$ je množina objektov ostro väčších než a podľa $C_{\leq}$. Potom $(a, b) \in top - k(C_{\leq})^{\mathcal{J}}$, akk:

t1) $(a, b) \in C_{\leq}^{\mathcal{J}}$ a $\|C_a\| < k$
t2) $\|C_a\| \geq k$ a $\|C_b\| < k$
t3) $\|C_b\| \geq k$

Agregáciu definujeme $@_U^{\mathcal{J}}\ (C_{\leq_1}, \ldots, C_{\leq_m}) \subseteq \Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$ pre každú m-ticu konceptov $C_{\leq_1}, \ldots, C_{\leq_m}$. Podobne ako v prípade konjunkcie, jej výsledkom je čiastočné predusporiadanie. Agregáciu definujeme podobne ako výsledkové tabuľky Formuly 1 s tým rozdielom, že pripúšťame aj remízy. Najskôr definujeme *úroveň* objektu a v $C_{\leq}$:

$level(a, C, \mathcal{J}) = \max_{l \in \mathbf{N}}\{l |\ \exists b_1, \ldots, b_l \in \Delta^{\mathcal{J}}\ \forall i \in \{1, \ldots, l\}(b_i, b_{i+1}) \in C^{\mathcal{J}} \wedge (b_{i+1}, b_i) \notin C^{\mathcal{J}} \wedge b_1 = a\}$

Ďalej definujeme *tabuľku ohodnotení* ako ostro klesajúcu postupnosť $score_{@_U}(score_1, \ldots, score_m)$, kde rozdiely medzi susednými prvkami tiež klesajú, teda napríklad (10, 8, 6, 5, 4, 3, 2, 1). Potom $(a, b) \in @_U^{\mathcal{J}}(C_{\leq_1}, \ldots, C_{\leq_m})$ vtedy, ak $\sum_{j=1}^{m} score_{level(C_{\leq_j}, a, \mathcal{J})} \geq \sum_{j=1}^{m} score_{level(C_{\leq_j}, a, \mathcal{J})}$.

4.3 Porovnanie fuzzy konceptov a predusporiadaní

Uvedieme príklad znalostnej bázy pre DL $s - \mathcal{EL}(\mathcal{D})$ a $o - \mathcal{EL}(\mathcal{D})$, ktorá obsahuje analogické tvrdenia. K dispozícii máme koncept `notebook` a roly `ma_cenu`, `ma_disk`. Znalostná báza v $s - \mathcal{EL}(\mathcal{D})$ obsahuje tvrdenia:
⟨ `HP_Pavillion`: `notebook`, 1⟩
(`HP_Pavillion`, 700): `ma_cenu`
(`HP_Pavillion`, 250): `ma_disk`
⟨ `Asus_EEE`: `notebook`, 1⟩
(`Asus_EEE`, 340): `ma_cenu`
(`Asus_EEE`, 160): `ma_disk`

Ďalej definujeme preferenčné koncepty používateľa U_1:
$\texttt{dobra\_cena}_{U_1} \equiv \exists\ (\texttt{ma\_cenu}).\texttt{nerastuca}_{400,800}$
$\texttt{dobry\_disk}_{U_1} \equiv \exists\ (\texttt{ma\_disk}).\texttt{neklesajuca}_{100,320}$
$\texttt{dobry\_notebook}_{U_1} \equiv \texttt{dobra\_cena}_{U_1} \sqcap \texttt{dobry\_disk}$

Potom každý model $\mathcal{I}$ spĺňa:
$\langle \texttt{HP\_Pavillion} : \texttt{dobra\_cena}_{U_1}, 0.25\rangle$
$\langle \texttt{HP\_Pavillion} : \texttt{dobry\_disk}_{U_1}, 0.68\rangle$
$\langle \texttt{HP\_Pavillion} : \texttt{dobry\_notebook}_{U_1}, 0.25\rangle$
$\langle \texttt{Asus\_EEE} : \texttt{dobra\_cena}_{U_1}, 1\rangle$
$\langle \texttt{Asus\_EEE} : \texttt{dobry\_disk}_{U_1}, 0.27\rangle$
$\langle \texttt{Asus\_EEE} : \texttt{dobry\_notebook}_{U_1}, 0.27\rangle$

Ak použijeme agregačnú funkciu $@_{U_1}(x, y) = \frac{x+2y}{3}$, môžeme nadefinovať koncept $\texttt{dobry\_notebook}_{U_1}$ ako $@_{U_1}(\texttt{dobra\_cena}_{U_1}, \texttt{dobry\_disk}_{U_1})$. Minimálny model bude odlišný:
$\langle \texttt{HP\_Pavillion} : \texttt{dobry\_notebook}, 0.54\rangle$
$\langle \texttt{Asus\_EEE} : \texttt{dobry\_notebook}, 0.51\rangle$

V DL $o - \mathcal{EL}(\mathcal{D})$ bude znalostná báza obsahovať rovnaké rolové tvrdenia a rovnaké definície troch preferenčných konceptov $\texttt{dobra\_cena}_{U_1}$, $\texttt{dobry\_disk}_{U_1}$ a $\texttt{dobry\_notebook}_{U_1}$. Konceptové tvrdenia majú nasledujúci formát:

$(\texttt{HP\_Pavillion}, \texttt{Asus\_EEE}) : \texttt{notebook}$
$(\texttt{Asus\_EEE}, \texttt{HP\_Pavillion}) : \texttt{notebook}$

Potom v každom modeli platia nasledujúce tvrdenia, avšak nevieme odvodiť žiadne znalosti o koncepte $\texttt{dobry\_notebook}_{U_1}$, pretože sa jedná len o čiastočné predusporiadanie:
$(\texttt{HP\_Pavillion}, \texttt{Asus\_EEE}) : \texttt{dobra\_cena}_{U_1}$
$(\texttt{Asus\_EEE}, \texttt{HP\_Pavillion}) : \texttt{dobry\_disk}_{U_1}$

Použijeme preto agregáciu a definujeme si tabuľku ohodnotení $@_{U_1} = (3, 2, 1)$ a koncept $\texttt{dobry\_disk}_{U_1}$ ako $@_{U_1}(\texttt{dobra\_cena}_{U_1}, \texttt{dobry\_disk}_{U_1})$. Potom objekt HP_Pavillion bude na prvom mieste v koncepte $\texttt{dobry\_disk}_{U_1}$, zatiaľ čo Asus_EEE bude na prvom mieste v koncepte $\texttt{dobra\_cena}_{U_1}$. Preto obidva objekty získajú tri body za prvé miesto a dva body za druhé miesto. V každom modeli bude platiť remíza:
$(\texttt{HP\_Pavillion}, \texttt{Asus\_EEE}) : \texttt{dobry\_notebook}_{U_1}$
$(\texttt{Asus\_EEE}, \texttt{HP\_Pavillion}) : \texttt{dobry\_notebook}_{U_1}$

5 Záver a ďalší výskum

Model založený na fuzzy množinách a agregácii je podporovaný induktívnymi metódami získavania preferencií, top-k vyhľadávaním aj odvodzovaním v deskripčných logikách. Implementácia tohto modelu v systéme UPreA je nezávislá od domény, v ktorej chceme vyhľadávať najlepšie objekty podľa preferencií.

V oblasti používateľských preferencií je niekoľko zaujímavých problémov, ktorým sa budeme venovať v ďalšom výskume. Do systému UPreA, ktorý preferencie ukladá a spravuje, plánujeme implementovať kolaboratívne filtrovanie. Táto metóda sa použije v prípade, že používateľ nezadá svoje preferencie ručne, ale máme k dispozícii iné informácie (údaje z registrácie, ohodnotenia objektov), podľa ktorých môžeme určiť skupinu podobných používateľov.

Ďalej plánujeme porovnať fuzzy model preferencií s inými podobnými modelmi. Ideálnym riešením je nájsť metódu, ktorá transformuje preferencie z jedného modelu do druhého a následne porovnať množiny výsledkov, ktoré poskytnú implementácie týchto modelov. Niektoré modely sú jednoduchšie, niektoré bohatšie, a teda nie vždy existuje jednoznačná transformácia. V takom prípade môžeme porovnať tieto modely pomocou experimentu, kedy tí istí používatelia sa budú snažiť hľadať objekty podľa tých istých (prirodzených) preferencií v obidvoch systémoch. Potom porovnáme výsledky tak ako v predchádzajúcom prípade. Chyby však môžu byť spôsobené odlišným rozhraním systémov alebo nekonzistentným správaním používateľa.

Pre navrhnuté deskripčné logiky $o - \mathcal{EL}(\mathcal{D})$ a $s - \mathcal{EL}(\mathcal{D})$ plánujeme prepracovať navrhnuté odvodzovacie algoritmy [12], analyzovať ich zložitosť a porovnať ich efektivitu s inými odvodzovacími algoritmami.

Referencie

1. J.L. Herlocker, J.A. Konstan, L.G. Terveen, J.T. Riedl: *Evaluating collaborative filtering recommender systems.* ACM Trans. Inf. Syst., 22, 1, 2004, 5–53.
2. L. Polkowski, A. Skowron: *Rough sets: A tutorial.* Rough Fuzzy Hybridization: A New Trend in Decision-Making (2000).
3. I.F. Ilyas, R. Shah, W.G. Aref, J.S. Vitter, A.K. Elmagarmid: *Rank-aware query optimization.* Proceedings of the 2004 ACM SIGMOD international conference on Management of data, 2004, 203-–214.
4. D. Kossmann, F. Ramsak, S. Rost: *Shooting stars in the sky: An online algorithm for skyline queries.* VLDB, 2002.
5. R. Fuller: *OWA operators in decision making.* Exploring the Limits of Support Systems, TUCS General Publications 3, Turku Centre for Computer Science, 1996, 85–104.
6. F. Chiclana, F. Herrera, E. Herrera-Viedma: *Integrating three representation models in fuzzy multipurpose decision making based on fuzzy preference relations.* Fuzzy Sets and Systems, 97, 1998, 33–48.
7. R.R. Yager: *Extending multicriteria decision making by mixing tnorms and OWA operators.* International Journal of Intelligent Systems, 20, 2005, 453–474.
8. B. Roy: *Multicriteria Methodology for Decision Analysis.* Kluwer Academic Publishers, 1996.
9. R.I. Brafman, C. Domshlak: *Preference handling – an introductory tutorial.*
   URL http://www.cs.bgu.ac.il/~brafman/tutorial.pdf
10. T. Horváth, P. Vojtáš: *Ordinal classification with monotonicity constraints.* Proceedings of ICDM, Springer, 2006.
11. P. Gurský, V. Vaneková, J. Pribolová:*Fuzzy user preference model for top-k search.* Proceedings of IEEE World Congress on Computational Intelligence (WCCI), Hong Kong, FS0377, 2008.
12. V. Vaneková, P. Vojtáš: *A description logic with concept instance ordering and top-k restriction.* Information Modelling and Knowledge Bases XX. Frontiers in Artificial Intelligence and Applications, 190, IOS Press Amsterdam, 2009, 139–153.
13. P. Gurský, T. Horváth, J. Jirásek, S. Krajči, R. Novotný, J. Pribolová, V. Vaneková, P. Vojtáš: *User preference web search – experiments with a system connecting web and user.* Computing and Informatics Journal, 2009 (to appear).
14. F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, eds.: *Description Logic Handbook.* Cambridge University Press (2002).
15. G.H. von Wright: *The Logic of Preference.* Edinburgh University Press, 1963.
16. C. Boutilier: *Toward a logic for qualitative decision theory.* Proceedings of the KR'94, 75–86 Morgan Kaufmann, 1992.

Plánovanie dátových prenosov v distribuovanom prostredí

Michal Zerola[1], Roman Barták[2], Jérôme Lauret[3], and Michal Šumbera[1]

[1] Ústav jaderné fyziky, Akademie věd, ČR
michal.zerola@ujf.cas.cz
[2] Matematicko-fyzikální fakulta, Univerzita Karlova, ČR
roman.bartak@mff.cuni.cz
[3] Brookhaven National Laboratory, USA
jlauret@bnl.gov

Abstrakt *Fyzikálne experimenty vysokých energií za posledné desaťročie smerovali k distribuovanému výpočtovému modelu v snahe paralelne spracovávať enormné objemy dát, ktoré rastú z roka na rok. Za účelom optimalizácie jednotlivých zdrojov, ktoré sú geograficky rozmiestnené je potrebné čeliť i otázke efektívnych dátových prenosov medzi jednotlivými centrami. Zaoberáme sa plánovaním dátových prenosov v distribuovanom prostredí pomocou programovania s obmedzujúcimi podmienkami (Constraint Programming). Predstavíme CP model uvažujúc rozdielne prenosové linky navzájom zdieľané jednotlivými prenosmi súborov smerujúcimi do spoločného cieľa. Ukážeme niekoľko vylepšení vedúcich k rýchlejšiemu výpočtu plánu (prenosových ciest) a k orezaniu prehľadávaného priestoru. Predstavíme heuristiku pre výber rozhodovacích premenných a jej porovnanie so simulovaným Peer-2-Peer modelom. Na záver načrtneme architektúru a komunikáciu medzi jednotlivými komponentami v reálnom prostredí.*

1 Úvod

Výpočtovo náročné experimenty, akými sú i tie z oblasti fyziky vysokých energií (High Energy and Nuclear Physics) vybudovali distribuovaný výpočtový model, aby čelili ich masívnym požiadavkám. Éra intenzívnych spracovaní dát zaiste otvorila možnosti i vedcom z oblasti informatiky pre riešenie praktických a zaujímavých problémov. Jedným experimentom z oblasti HENP je experiment STAR [4] na urýchľovači RHIC (Relativistic Heavy Ion Collider [3]) umiestnenom v Brookhaven National Laboratory, v USA.

Zmyslom tejto práce je navrhnúť a vyvinúť automatický plánovací systém, ktorý by efektívne využíval jednotlivé výpočtové a úložné prostriedky. Toto zadanie zahŕňa replikáciu/distribúciu centrálne získaných dát z detektorov na ostatné výpočtové uzly s dôrazom na ich následné spracovanie. V tomto článku sa budeme sústrediť na jednu časť tohoto problému, ktorá je v súčasnosti potrebná a hneď využiteľná fyzikmi a to: "ako preniesť požadované dáta do spoločného cieľového uzla v čo najkratšom čase?" Tento problém je možno adresovať i ako plánovanie mnohých prenosových ciest so zdieľanými linkami a minimalizáciou makespanu (času ukončenia poslednej úlohy - prenosu).

[4] http://www.star.bnl.gov

1.1 Súvisiace práce

Potreby dáta intenzívnych projektov vychádzajúce z rôznych oblastí ako bio-informatika (BIRN,BLAST), astronómia (SDSS) alebo HENP komunity (STAR, ALICE) sú skúmané a riešené vedcami už roky. Kedže klesajúca cena úložných priestorov a výpočtových jednotiek dovoľuje analyzovať čoraz viac získaných dát, apetít po efektivite v DataGridoch sa stáva ešte výraznejším.

Oddelenie rozvrhovania výpočtových úloh od dátových prenosov študovali už Ranganathan a Foster v [5]. Autori diskutovali kombinácie replikačných stratégií a rozvrhovacích algoritmov, ale neuvažovali charakteristiku siete. Povaha fyzikálnych experimentov vysokých energií, kde dáta sú centrálne získavané implikuje, že replikácia dát do ostatných centier je nutná k ich distribuovanému spracovaniu.

Sato a spol. v [6] a autori [4] sa zaoberali otázkou replikácie dát pomocou matematických podmienok modelujúc optimalizačný problém v prostredí Gridu. Riešiaci postup v [6] je založený na celočíselnom lineárnom programovaní, kým [4] používa Lagrangiovu relazačnú metódu [1]. Limitácia oboch modelov je práve charakterizácia dátových prenosov, ktorá nezohľadňuje možné prenosové cesty (ale len priame spojenia) možno vedúce k lepšiemu využitiu liniek.

My sa zameriavame na túto chýbajúcu zložku uvažujúc prenosy medzi geograficky vzdialenými centrami za účelom ich zefektívnenia. Pôvodná myšlienka modelu pochádza od Simonisa [7], kde podmienky definujúce toky boli rozšírené o prenosové rýchlosti liniek, rozvrhovaciu fázu (alokáciu prenosov v čase) a prehľadávacie heuristiky. Riešiaci prístup je založený na programovaní s obmedzujúcimi podmienkami a jeho hlavná výhoda je jednoduchá možnosť rozširovať model o ďalšie obmedzenia reálneho sveta.

2 Formalizácia problému

V tejto časti predstavíme formálny popis problému za použitia matematických podmienok, ktoré odpovedajú obmedzeniam z reálneho sveta. Vstup problému pozostáva z dvoch častí. Prvá má statický charakter a reprezentuje sieť a lokácie jednotlivých súborov. Prenosová sieť, formálne orientovaný ohodnotený graf, sa skladá z množiny vrcholov **N** a množiny ohodnotených orientovaných hrán **E**. Uzly zastupujú výpočtové centrá a hrany prenosové linky medzi nimi. Váha hrany charakterizuje prenosovú rýchlosť príslušnej linky. Informácia o lokáciách jednotlivých súborov je mapovanie z jednoznačného názvu súboru na množinu uzlov (centier) kde je súbor zreplikovaný.

Druhá časť vstupu je požiadavka od užívateľa, konkrétne zoznam súborov, ktoré požaduje v cieľovom uzle v čo najkratšom čase. Úlohou plánovača je potom vytvoriť:

- prenosovú cestu pre každý súbor, t.j. výber jednej lokácie ako zdroja a cestu počínajúc v tomto uzle a končiac v cieľovom (*plánovanie*),
- pre každý súbor a jeho prenosovú cestu alokáciu času na jeho jednotlivé prenosy po príslušných linkách cesty (*rozvrhovanie*), a to tak, že
- výsledný plán má minimálny makespan

Príklad: Výstupom plánovacej fáze pre súbor F_1 (ktorý sa nachádza v uzloch A a B a má byť prenesený do uzla C) je prenosová trasa $A \to D \to B$. Rozvrhovacia časť potom vyprodukuje alokácie jednotlivých prenosov v čase s ohľadom na prenosové rýchlosti liniek a precedencie medzi prenosmi: $A \to D$ v čase $1 \leftrightarrow 3$ a $D \to B$ v čase $4 \leftrightarrow 7$.

Riešiaci postup je teda zložený z dvoch fází, ktoré iterujú ako vidieť v algoritme 1. Aktuálne najlepší makespan je použitý ako medza (*rez*) v rozvrhovacej fáze (Branch-and-bound stratégia). Aktuálny makespan môžeme navyše efektívne využiť i v plánovacej časti. Myšlienkou je, že podľa počtu súčasne priradených súborov a ich štartovacích prenosových časov na danej linke, môžeme určiť spodný odhad makespanu tejto konfigurácie (preto je predávaný plánovacej funkcii ako parameter). Podrobnejší popis matematických podmienok je uvedený v časti 2.1.

Nasledujúci formalizmus je použitý k definovaniu podmienok plánovacieho podproblému. Množina $\mathbf{OUT}(n)$ obsahuje hrany vychádzajúce z uzla n, množina $\mathbf{IN}(n)$ hrany vchádzajúce do uzla n. Vstup získaný od užívateľa je množina žiadostí (demands) **D**, ktoré je potrebné preniesť do cieľového uzla *dest*. Pre každú požiadavku (súbor) $d \in D$ máme množinu jeho zdrojov $\mathbf{orig}(d)$ - centier kde daný súbor je už zreplikovaný/prístupný. Predstavíme *link-based* prístup pre modelovanie podmienok, inou možnosťou je tzv. *path-based* prístup, ktorého bližší popis je možné nájsť v [9].

Algorithm 1 Pseudocode for a search procedure.

```
makespan ← sup
plan ← Planner.getFirstPlan()
while plan != null do
  schedule ← Scheduler.getSchedule(plan, makespan)
  {Branch-and-Bound on makespan}
  if schedule.getMakespan() < makespan then
    makespan ← schedule.getMakespan() {better schedule found}
  end if
  plan ← Planner.getNextPlan(makespan) {next feasible plan with cut constraint}
end while
```

2.1 Plánovanie

Kľúčovou myšlienkou *link-based* prístupu je použitie rozhodovacej $\{0,1\}$ premennej X_{de} pre každú požiadavku a prenosovú linku siete, označujúc či daný súbor bude prenesený po linke alebo nie. Podmienky (1-3), zaisťujú, že ak rozhodovacie premenné majú pridelené hodnoty, tak výsledná konfigurácia obsahuje prenosové cesty. Tieto samotné podmienky ešte nazaisťujú elimináciu cyklov, tomu zabránia až *precedenčné* podmienky (4).

$$\forall d \in \mathbf{D}: \quad \sum_{e \in \cup \mathbf{OUT}(n|n \in \mathbf{orig}(d))} X_{de} = 1, \quad \sum_{e \in \cup \mathbf{IN}(n|n \in \mathbf{orig}(d))} X_{de} = 0 \tag{1}$$

$$\forall d \in \mathbf{D}: \quad \sum_{e \in \mathbf{OUT}(dest(d))} X_{de} = 0, \quad \sum_{e \in \mathbf{IN}(dest(d))} X_{de} = 1 \tag{2}$$

$$\forall d \in \mathbf{D}, \forall n \notin orig(d) \cup \{dest(d)\}: \quad \begin{matrix} \sum_{e \in \mathbf{OUT}(n)} X_{de} \leq 1 \\ \sum_{e \in \mathbf{IN}(n)} X_{de} \leq 1 \end{matrix} \quad \sum_{e \in \mathbf{OUT}(n)} X_{de} = \sum_{e \in \mathbf{IN}(n)} X_{de} \tag{3}$$

Precedenčné podmienky (4) používajú nerozhodovacie premené P_{de} s doménou nezáporných celých čísiel, ktoré reprezentujú možný počiatočný čas prenosu súboru d po linke e. Nech dur_{de} je konštantná doba prenosu súboru d po hrane e. Potom podmienka

$$\forall d \in \mathbf{D}\ \forall n \in \mathbf{N}: \quad \sum_{e \in \mathbf{IN}(n)} X_{de} \cdot (P_{de} + dur_{de}) \leq \sum_{e \in \mathbf{OUT}(n)} X_{de} \cdot P_{de} \tag{4}$$

zaisťuje správne poradie medzi prenosmi každého súboru, teda eliminuje cykly. Bohužiaľ, podmienky (4) nefiltrujú jednotlivé domény P_{de} pokiaľ hodnoty X_{de} nie sú známe a preto sme navrhli redundantné podmienky (5) k presnejšiemu spodnému odhadu každej premennej P_{de}. Nech *start* je počiatočný vrchol hrany e, ktorá neobsahuje súbor d ($start \notin orig(d)$):

$$\min_{f \in \mathbf{IN}(start)} (P_{df} + dur_{df}) \leq P_{de} \tag{5}$$

Premenné P_{de} neslúžia len k eliminácii cyklov ale tiež k odhadu makespanu celkového plánu. Podstatou je, že v závislosti na aktuálnom počte súborov priradených k danej hrane a ich možným počiatočným časom, môžeme vypočítať spodný odhad makespanu, ktorý bude zistený presne až v ďalšej rozvrhovacej fáze. Na základe toho, ak už máme horný odhad makespanu (typicky z niektorej predošlej iterácie), môžeme odmedziť prehľadávanie stavového priestoru, ktoré by viedlo len k horším plánom pomocou podmienky:

$$\forall e \in \mathbf{E} : \min_{d \in \mathbf{D}}(P_{de}) + \sum_{d \in \mathbf{D}} X_{de} \cdot dur_{de} + SP_e < makespan, \tag{6}$$

kde SP_e značí hodnotu najkratšej cesty z koncového vrcholu hrany e do $dest$.

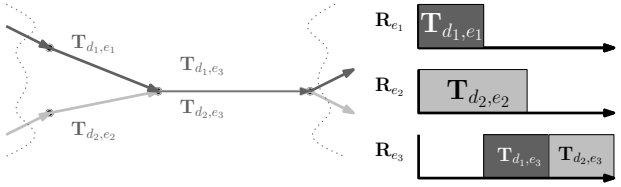
2.2 Rozvrhovanie

Cieľom rozvrhovacej časti je ohodnotenie konfigurácie ciest v zmysle minimálneho makespanu. T.j. za predpokladu, že každý súbor má určenú prenosovú trasu, určiť kedy budú linky trasy použité pre jeho prenos a to tak, aby všetky súbory dorazili do cieľa čo najskôr. V princípe, slúži ako objektívna funkcia, pretože výsledný plán nebude vykonávaný presne podľa rozvrhnutých časov aktivít, ako uvidíme v časti 4.

Budeme používať značenie z oblasti rozvrhovania, konkrétne **úlohy** a **zdroje**. Pre každú hranu grafu e, ktorá bude použitá aspoň jedným prenosom vytvoríme jednoznačný **unárny zdroj** $\mathbf{R}_e$. Podobne, pre každý súbor a jeho zvolené prenosové linky (definujúce prenosovú cestu) zavedieme množinu úloh podľa nasledovných pravidiel (znázornené na obrázku 1):

- ak súbor d má byť prenesený cez linku e (t.j. $X_{de} = 1$) tak vytvoríme úlohu $\mathbf{T}_{de}$, obsahujúcu nezápornú celočíselnú premennú $start_{de}$ (s doménou $[0, \ldots, horizon]$) a konštantu dur_{de}, ktoré popisujú počiatočný čas a dobu prenosu. Úlohu T_{de} priradíme k zdroju $\mathbf{R}_e$
- pre ľubovoľné dva súbory d_1 a d_2 priradené k zdroju R_e, podmienka $start_{d_1e} + dur_{d_1e} \leq start_{d_2e} \vee start_{d_2e} + dur_{d_2e} \leq start_{d_1e}$ musí platiť
- pre každý súbor d vybudujeme precedencie medzi jednotlivými úlohami T_{de} a to tak, že prenos súboru z uzlu ($T_{d,out}$) môže začať len po skončení predošlého prenosu do tohoto uzla ($T_{d,inc}$), t.j. $start_{d,inc} + dur_{d,inc} \leq start_{d,out}$

Dôvodom k použitiu unárnych zdrojov pred *energetickými*, ktoré sa zdajú byť výstižnejšie, je ich dostupnosť v súčasných CP frameworkoch. Vzhľadom k tomu, že prenos jedného súboru je v praxi atomická operácia, nepreemptívny rozvrhovací prístup (prenos súboru/úlohu nie je možné prerušiť) nám vyhovuje taktiež.



Obr. 1. Príklad priradenia úloh (prenosov súborov po príslušných linkách) na unárne zdroje (linky). V tomto prípade prenosové cesty súborov d_1 a d_2 zdieľajú linku e_3, a následne tiež i zdroj $\mathbf{R}_{e_3}$.

3 Prehľadávacie heuristiky

V programovaní s obmezujúcimi podmienkami heuristiky pre výber a ohodnotenia premenných určujú tvar prehľadávacieho stromu, ktorý je obvykle prechádzaný do hĺbky. Pôvodne sme v plánovacej fáze použili *dom* [2] stratégiu (uprednostňuje premenné s najmenším počtom aktuálne možných hodnôt) pre výber premenných X, čo pri boolovských premenných znamená výber v pevnom poradí. Následne sme navrhli *FastestLink* výber pre *link-based* prístup. Princípom je, že v rozhodovacej fáze z neohodnotených premenných pre súbor d, zvolené X_{de_j} odpovedá najrýchlejšej linke ($j = \arg\min_{i=1,\ldots,m} slowdown(e_i)$). V prípade, že niekoľko takých premenných existuje, prvá je zvolená z pevného poradia. Ako súčasne najlepšiu heuristiku sme navrhli *MinPath*, ktorá zohľadňuje premenné P_{de} pre lepšie odhady prenosových časov. Konkrétne, heuristika vyberá k ohodnoteniu premennú X_{de} tak, že príslušná hodnota je minimálna:

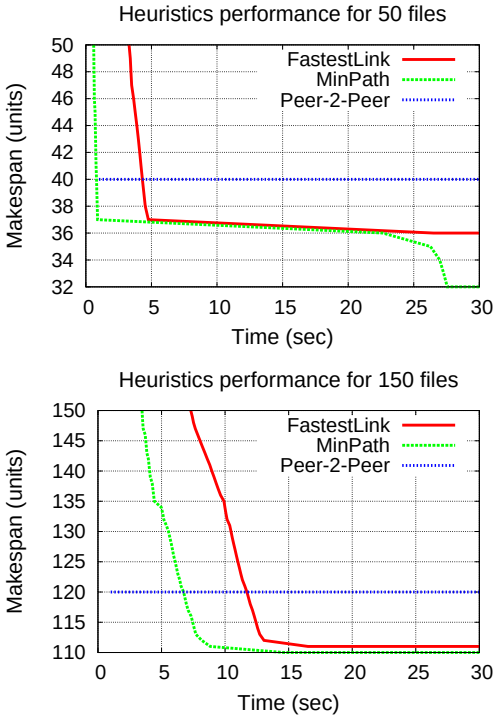
$$\inf P_{de} + dur_{de} + SP_e, \tag{7}$$

kde $\inf P_{de}$ značí najmenšiu hodnotu v aktuálnej doméne premennej P_{de}.

Porovnania jednotlivých heuristík spolu s Peer-2-Peer modelom, ktorého simulátor sme implementovali sú znázornené na grafoch 2. Princíp Peer-2-Peer modelu je založený na hladovom prístupe, kde cieľová stanica sťahuje požadované súbory priamo z niekoľkých zdrojov (obdoba *torrentu*), a najmenšou jednotkou (chunkom) je samotný súbor. Bližšie porovnanie je možné nájsť v [8].

4 Realizácia rozvrhu

Model načrtnutý v predošlých častiach predpokladá prenos jedného súboru po danej linke v čase za použitia unárnych zdrojov. V praxi sa ale väčšinou používa prenos súčasne niekoľkých súborov po linke kvôli lepšej saturácii prenosového pásma a minimalizácii réžie TCP/IP protokolu. Vytvorenie presného modelu reálnej siete a prenosu paketov by ale ťažko viedlo k optimalizácii. Nami navrhnutý mechanizmus ako vypočítaný rozvrh bude vykonávaný v reálnej sieti je nasledovný:



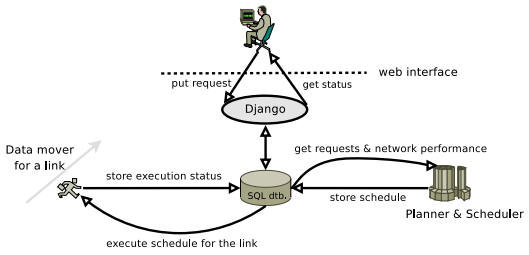
Obr. 2. Konvergencia makespanu počas prehľadávania pre *FastestLink* a *MinPath*.

– o každú linku sa stará jeden *LinkManager*, ktorý je zodpovedný za prenos súborov po linke v prípade, že linka patrí do vypočítanej prenosovej cesty
– hneď ako je súbor prístupný k prenosu (je v počiatočnom uzle danej hrany), príslušný *LinkManager* inicializuje ďalší prenos, pričom rešpektuje maximálny počet povolených paralelných instancií

Požiadavky od užívateľov sú ukladané do relačnej databáze. Jedným z vhodných prístupov je použitie webového interfacu postaveného na frameworku **Django**[5], ktorý sa stará o formuláre a templaty a zároveň poskytuje pluginy pre prístup k niekoľkým relačným databázam. Plánovač, samostatná centrálna komponenta zvolí "batch" (dávku) z čakajúcich súborov k prenosu a vypočíta plán. Výber súborov odpovedá *fair-share* objektívnej funkcii a umožňuje nám testovať niekoľko faktorov (z pohľadu užívateľov či využitia zdrojov). Plán je uložený späť do databáze udávajúc *Link Managerom*, že súbory sú pripravené na prenosy.

Takže než, aby sme vykonávali presný rozvrh, v implementácii uvažujeme len plán - vypočítané prenosové cesty, pretože nemáme limitáciu na *due-time* a jednotlivé prenosy vykonáme hladovo. Aby sme si

[5] http://www.djangoproject.com



Obr. 3. Schéma navrhnutej architektúry.

to ale mohli dovoliť, musíme si byť istí, že vypočítaný rozvrh sa nebude rádovo líšiť od vykonaného. K tomu účelu sme naimplementovali simulátor reálnej prenosovej siete a vykonali porovnania, podľa ktorých sa makespan vykonaného rozvrhu nelíši viac ako o 3%, čo je zanedbateľné. Experimenty potvrdzujú, že predstavený model poskytuje pomerne presný odhad reálneho makespanu.

Reference

1. M.L. Fisher: *The Lagrangian relaxation method for solving integer programming problems.* Management Science, 27, 1, January 1981, 1–18.
2. S.W. Golomb and L.D. Baumert: *Backtrack programming.* J. ACM, 12, 4, 1965, 516–524.
3. J. Adams, et al.: *Experimental and theoretical challenges in the search for the quark gluon plasma: The STAR collaboration's critical assessment of the evidence from RHIC collisions.* Nuclear Physics A, 757, 2005, 102–183.
4. R.M. Rahman, K. Barker, and R. Alhajj: *Study of different replica placement and maintenance strategies in data grid.* In CCGRID, IEEE Computer Society, 2007, 171–178.
5. K. Ranganathan and I. Foster: *Decoupling computation and data scheduling in distributed data-intensive applications.* IEEE Computer Society, 0, 2002, 352–258.
6. H. Sato, S. Matsuoka, T. Endo, and N. Maruyama: *Access-pattern and bandwidth aware file replication algorithm in a grid environment.* In GRID, IEEE, 2008, 250–257.
7. H. Simonis: *Constraint applications in networks.* In F. Rossi, P. van Beek, and T. Walsh, editors, Handbook of Constraint Programming, chapter 25, Elsevier, 2006, 875–903.
8. M. Zerola, R. Barták, J. Lauret, and M. Šumbera: *Planning heuristics for efficient data movement on the grid.* (To appear in proceedings of MISTA 2009), 2009.
9. M. Zerola, R. Barták, J. Lauret, and M. Šumbera: *Using constraint programming to plan efficient data movement on the grid.* (Submitted to ICTAI 2009), 2009.

ITAT'09

Information Technology – Applications and Theory

POSTERS

Improving genetic optimization by means of radial basis function networks

Lukáš Bajer[1] and Martin Holeňa[2]

[1] student of the Faculty of Mathematics and Physics, Charles University
Malostranské nám. 25, Prague 1, Czech Republic
bajeluk@matfyz.cz
[2] Institute of Computer Science, Czech Academy of Sciences
Pod Vodárenskou věží 2, Prague 8, Czech Republic
martin@cs.cas.cz

Abstract. *Evolutionary, and especially genetic algorithms have become one of the most successful methods for the optimization of empirical objective functions. However, evaluation of such fitness functions can be very expensive or time consuming. In this article, we employ radial basis function networks as a surrogate model of the original fitness function which serves as a fast approximation whenever needed. With this method, much larger populations or several generations can be simulated without waiting for expensive objective function evaluation. As a result, faster convergence in terms of the number of the original empirical fitness evaluations is expected.*

1 Surrogate modelling

Approximation of the fitness function with some regression model is a common cure for costly evaluation of empirical objective function. These models, also known as a surrogate models, simulate a behaviour of the original function while being much cheaper or much less time consuming to evaluate. They are fitted according to a limited number of selected data points with measured original fitness function values, but they predict a response in the whole search space.

1.1 RBF networks

In this article, we use radial basis function (RBF) networks as surrogate model. They compute a mapping

$$f(\mathbf{x}) = \sum_{i=1}^{g} \pi_i f_i(||\mathbf{x} - \mathbf{c}_i||) \qquad (1)$$

where $\mathbf{x}$ is the input, g the number of components, f_i are radial basis functions, $\mathbf{c}_i$ radial functions' centres, and $||.||$ is a norm. As functions f_i, we employ Gaussian functions

$$f_i(\mathbf{x}) = g_i(\mathbf{x}; \mathbf{c}_i, \beta_i) = e^{-\beta_i ||\mathbf{x}-\mathbf{c}_i||^2}. \qquad (2)$$

Detailed description of the RBF networks can be found in [2] and [3].

1.2 Evolution control

An evolution control determines when the original fitness function or the surrogate model should be used. The preferred type is an *individual-based* evolution control which evaluates all the individuals with the approximating model at first. Then, some of them are chosen for re-evaluation with the original fitness function. Some possibilities which individuals to choose are suggested by Jin in [4].

The second type is a *generation-based* evolution control. Generations are grouped into cycles of a fixed length λ. In each cycle, η of the generations are controlled by the original fitness and the rest by the model. The larger the prediction error estimate is, the larger η should be used.

2 Our strategy for using surrogate-assisted genetic optimization

Basic steps of our version of the surrogate-assisted genetic algorithm are outlined in Figure 1 and 2. After preparation of the initial population, the algorithm proceeds with a generation: the algorithm fits the model, evaluates individuals either with the original fitness or with the surrogate model, and generates a new population with genetic operators. These steps are repeated until some acceptable solution is found, or a user-specified time limit is exhausted. All the original objective function results are stored to the database and can be used for RBF network fitting later.

2.1 Model fitting

Since RBF networks are able to work only with real values, the algorithm separates nominal variables. It uses them to cluster available original fitness function data into separate groups cl_j, and creates an independent RBF network for each such a cluster using only real-valued variables. Hamming or Jaccard's distance

Surrogate GA
Input:
– a specification of the optim. task (nominal and real variables and their properties, variable-values constraints, o – the size of a popul. to be evaluated by the *orig. fitness*; optionally: p_0 – the initial population)
– evolution control parameters (*individual*, resp. *generation* based (λ and η_0 – init. value of η))
– model parameters ($s'_{\min}$ – the min. number of data for using a model, functions f_i, $s_{\min}$ – the min. number of data for fitting one RBF network, k – no. of folds in crossvalid.)
– database $\mathbf{D}$ of the original fitness func. data
Steps of the algorithm:
(1) $p \leftarrow$ initial population: take p_0 (if given on input) or randomly generate new one satisf. constraints; $\eta = \eta_0$ (if *individual*-based evol. control)
(2) **if** (not enough data in $\mathbf{D}$, i.e. $|\mathbf{D}| < s'_{\min}$)
(3) evaluate p with the *original fitness*, store the results into the $\mathbf{D}$, and go to (11)
end if
(4) *model* $\leftarrow$ FitTheModel($s_{\min}$, f_i, $\mathbf{D}$, r)
if (*individual*-based evol. control)
(5) evaluate p with surrg. *model*
(6) select o individuals for reevaluation
(7) reevaluate them with the *orig. fitness*
else (*generation*-based evol. control)
if (η of λ generations already passed)
(8) evaluate p with the surrogate *model*
else
(9) evaluate p with the *original fitness*
end if
(10) adjust η according to the *model*'s error
end if
(11) $p \leftarrow$ new population generated with elitism, selection, mutation and crossover
(12) continue with (2) **until** (acceptable solution is found **or** solving budget exhausted)
Output: all the individuals proposed by the GA

Fig. 1. Pseudo-code of the surrogate-assisted GA

is used for clustering. Among various possibilities of creating clusters, those leading to clusters with less different nominal combinations is preferred.

The network's parameters ($\mathbf{c}_i$, β_i, π_i, $i = 1, \ldots, g$, see (2)) are fitted using the least-squares error for all possible numbers of components g. Then, the best model according to Akaike's or Bayesian information criterion (AIC or BIC, [1]), or simply the model with the lowest mean squared error (MSE) is chosen.

FitTheModel($s_{\min}$, f_i, $\mathbf{D}$, r)
Arguments: $s_{\min}$ – min. size of clusters, f_i – RBF type, $\mathbf{D}$ – database, r – nr. of real-valued var.
Steps of the procedure:
(1) cluster the orig. fitness data according to the nominal variable values into clusters of size at least $s_{\min}$; $m \leftarrow$ the number of clusters; $|cl_j| \leftarrow$ size of the j-th cluster, $j = 1, \ldots, m$ $\{\mathbf{N}_j\}_{j=1}^m \leftarrow$ sets of clusters' nominal var. values
(2) $\{g_j^{\max}\}_{j=1}^m \leftarrow$ the maxim. number of components of the j-th network (for Gaussians $g_j^{\max} = \lfloor \frac{|cl_j|}{2+r} \rfloor$)
for each cluster cl_j, $j = 1, \ldots, m$
for the number of components $g = 1, \ldots, g_j^{\max}$
(3) $e_g \leftarrow$ model's MSE from k-fold cross-valid.
end for
(4) choose the best nr. of components $g^\star$ acc. to MSE or AIC/BIC criter.
(5) $model_j \leftarrow$ refitted model using all data
end for
Output: $\{model_j, e_j, \mathbf{N}_j\}_{j=1}^m$

Fig. 2. Pseudo-code of the fitting procedure

2.2 Evaluation with surrogate model

Once the model is fitted, it can be used for evaluating individuals. A cluster with the data closest to the individual's nominal values is found at first. The model returns a value computed by the cluster's RBF network where individual's real-valued components are taken as network's inputs.

3 Conclusions

Our work proposes a detailed algorithm and its implementation of the surrogate-assisted genetic algorithm employing RBF networks. It will be tested on both the benchmark and real-world problem data, the latter from catalytic materials development. Further details accompanied by a Matlab source code will be available in Lukáš Bajer's Master thesis which is expected to be released in August, 2009.

References

1. R.A. Berk: *Statistical learning from a regression perspective*. Springer, 2008.
2. C.M. Bishop: *Neural networks for pattern recognition*. Oxford University Press, 2005.
3. M.D. Buhman: *Radial basis functions: theory and implementations*. Cambridge University Press, 2003.
4. Y. Jin: *Neural networks for fitness approximation in evolutionary optimization*. Knowledge Incorp. in Evolutionary Computation, Springer, 2005, 281–306.

Testovanie implementácií prenosových protokolov pre distribuované systémy

Martin Čup, Róbert Novotný, and Peter Gurský

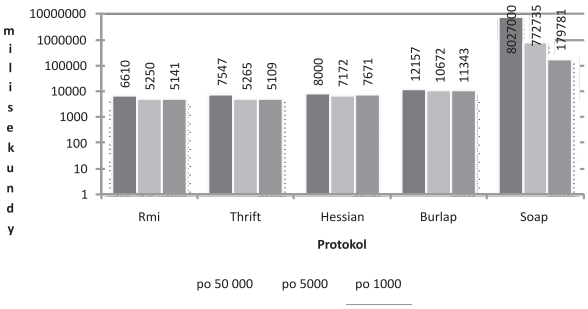
Ústav informatiky,
Univerzita P. J. Šafárika, Košice
cupomato@gmail.com, {peter.gursky,robert.novotny}@upjs.sk

Abstrakt *Skúmame a porovnávame niektoré metódy zabezpečujúce komunikáciu distribuovaných systémov. Experimentálne overujeme výkonnosť konkrétnych protokolov určených pre prenos rozsiahlych dát v distribuovaných systémoch. Navrhneme a implementujeme vlastný systém, na ktorom budeme testovať výkon týchto technológií.*

Jedným zo základných komponentov systému [1] je vyhľadávanie najlepších k objektov v distribuovanom prostredí. Výkonnosť komunikácie medzi klientom a servermi je teda kritickou požiadavkou. Budeme sa zaoberať porovnaním protokolov na prenos distribuovaných dát na platforme Java a to konkrétne technológiami RMI [2], Burlap [5], Hessian [4], Apache Thrift [3] a SOAP [6].

Protokol	Transport	Serializácia	Interop
RMI	vlastný	binárna	nie
Thrift	vlastný	binárna	áno
Hessian	HTTP	binárna	áno
Burlap	HTTP	XML	nie
Spring-WS	HTTP	XML	áno

Na riešenie problému integrácie heterogénnych služieb a protokolov existuje viacero riešení, ale pre naše účely výkonnostného testovania sme navrhli a implementovali vlastnú architektúru umožňujúcu použitie ľubovoľnej technológie (protokolu). Testovanie výkonnosti sme realizovali v rôznych prostrediach – v rámci jedného počítača, jednej lokálnej siete a v rámci siete Internet, pričom sme použili server s CPU Intel Celeron 1,73GHZ a 1024 MB RAM. Testovanie prebehlo na vzorke 1 000 000 dvojíc čísiel reprezentujúcich identifikátor objektu (int) a hodnota atribútu (double). V jednotlivých testoch boli prenášané dáta o rôznych veľkostiach. V prvom teste sme prenášali dáta v dávkach po 50 000 dvojíc, v druhom teste po 5 000 dvojíc a v tretom teste po 1000 dvojíc. Jednu dávku dát reprezentuje zoznam dvojíc. Obrázok 1 ukazuje, že celkový čas prenosu závisí od veľkosti prenášanej dávky, pričom každý protokol má najvyšší výkon pri inej veľkosti dát. Napríklad pre technológiu Hessian je najvýhodnejšie prenášať dávky veľkosti 5 000 dvojíc. Pri testovaní sme tiež brali do úvahy tzv. „zahrievacie kolo“, teda dobu trvania prenosu prvej



Obr. 1. Priemerný čas volania v jednotlivých protokoloch v milisekundách.

dávky, ktorá zaberá v priemere o 20% dlhšiu dobu než ostatné dávky.

Ukázalo sa, že najvyšší výkon majú protokoly s binárnou serializáciou, pričomo prenos SOAP správ sa ukázal ako nanajvýš nevhodný na prenos takéhoto druhu dát. Najlepšie výsledky v testoch dosiahli RMI a Thrift, ktoré dosahovali skoro totožné výsledky a vzhľadom na jednoduchú implementáciu a vysoký výkon ich odporúčame používať. Thrift navyše ponúka i čiastočnú interoperabilitu, hoci na úkor zložitejšej konfigurácie.

Práca bola čiastočne podporená z projektov VVGS PF 27/2009/I, VVGS/UPJŠ/45/09-10, 2009–2010, VEGA 1/0131/09, 2009–2011.

Referencie

1. P. Gurský, T. Horváth, J. Jirásek, S. Krajči, R. Novotný, J. Pribolová, V. Vaneková, P. Vojtáš: *User preference web search – experiments with a system connecting web and user.* Computing and Informatics Journal, 2008.
2. Java Remote Method Invocation. [online] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
3. Apache Thrift. [online] http://incubator.apache.org/thrift
4. Hessian Binary Web Service Protocol. [online] http://hessian.caucho.com/
5. Burlap XML Web Service Protocol. http://www.caucho.com/resin-3.0/protocols/burlap.xtp
6. Spring-WS. [online] http://static.springsource.org/spring-ws/sites/1.5/

Advanced data mining and integration for environmental scenarios

Ondrej Habala[1], Marek Ciglan[2], Viet Tran[2], and Ladislav Hluchý[2]

[1]Dept. of Parallel and Distributed Computing, Institute of Informatics of the Slovak Academy of Sciences
84507 Bratislava, Slovakia
[2]Institute of Informatics of the Slovak Academy of Sciences

We present our work in the project ADMIRE[3], where we use advanced data mining and data integration technologies to run an environmental application, which uses data mining instead of standard physical modeling to perform experiments and obtain environmental predictions.

The project ADMIRE (Advanced Data Mining and Integration Research for Europe [1]) is a 7th FP EU ICT project aims to deliver a consistent and easy-to-use technology for extracting information and knowledge from distributed data sources. The project is motivated by the difficulty of extracting meaningful information by mining combinations of data from multiple heterogeneous and distributed resources. It will also provide an abstract view of data mining and integration, which will give users and developers the power to cope with complexity and heterogeneity of services, data and processes. One of main goals of the project is to develop a language that serves as a canonical representation of the data integration and mining processes.

The presented environmental application currently contains three scenarios, which are in the process of being implemented and deployed in the ADMIRE testbed. These scenarios have been selected from more than a dozen of candidates provided by hydro-meteorological, water management, and pedological experts in Slovakia. The main criterion for their selection was their suitability for data mining application. The scenarios are named ORAVA, RADAR and SVP, and they are in different stages of completion, with ORAVA being the most mature one, and SVP only in the beginning stages of its design.

The scenario named ORAVA has been defined by the Hydrological Service division of the Slovak Hydrometeorological Institute, Bratislava, Slovakia. Its goal is to predict the water discharge wave and temperature propagation below the Orava reservoir, one of the largest water reservoirs in Slovakia.

The pilot area covered by the scenario (see Fig. 1) lies in the north of Slovakia, and covers a relatively small area, well suitable for the properties of testing ADMIRE technology in a scientifically interesting setting.

The data, which has been selected for data mining, and which we expect to influence the scenario's target variables – the discharge wave propagation, and temperature propagation in the outflow from the reservoir to river Orava – is depicted in Tab. 1.

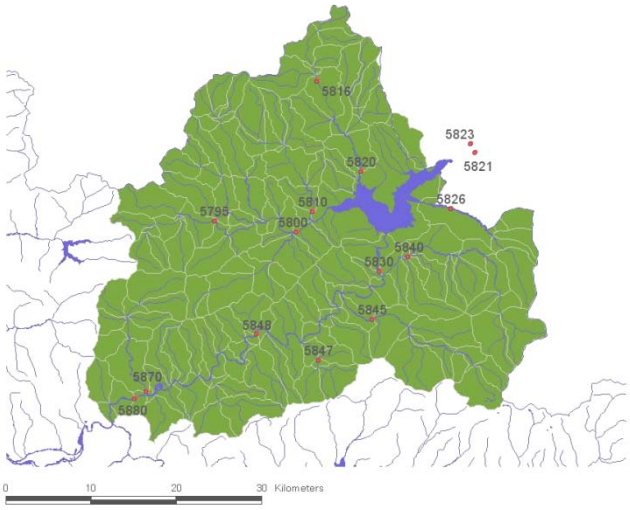


Fig. 1. The area of the pilot scenario ORAVA.

For predictors in this scenario, we have selected rainfall and air temperature, the discharge volume of the Orava reservoir and the temperature of water in the Orava reservoir. Our target variables are the water height and water temperature measured at a hydrological station below the reservoir. As can be seen in Fig. 1, the station directly below the reservoir is no.5830, followed by 5848 and 5880.

If we run the data mining process in time T, we can expect to have at hand all data from sensors up to this time (first three data lines in Tab. 1). Future rainfall and temperature can be obtained by running a standard meteorological model. Future discharge of the reservoir is given in the manipulation schedule of the reservoir.

Time	Rainfall	$Temp_{Air}$	Discharge	$Temp_{Res}$	$Height_{Station}$	$Temp_{Station}$
T-2	R_{T-2}	F_{T-2}	D_{T-2}	E_{T-2}	X_{T-2}	Y_{T-2}
T-1	R_{T-1}	F_{T-1}	D_{T-1}	E_{T-1}	X_{T-1}	Y_{T-1}
T	R	F_T	D_T	E_T	X_T	Y_T
T+1	R_{T+1}	F_{T+1}	D_{T+1}		X_{T+1}	Y_{T+1}
T+2	R_{T+2}	F_{T+2}	D_{T+2}		X_{T+2}	Y_{T+2}

Tab. 1 Depiction of the predictors and variables of the ORAVA scenario.

[3]This work is supported by projects ADMIRE FP7-215024, APVV DO7RP-0006-08, SEMCO-WS APVV-0391-06, VEGA No. 2/0211/09.

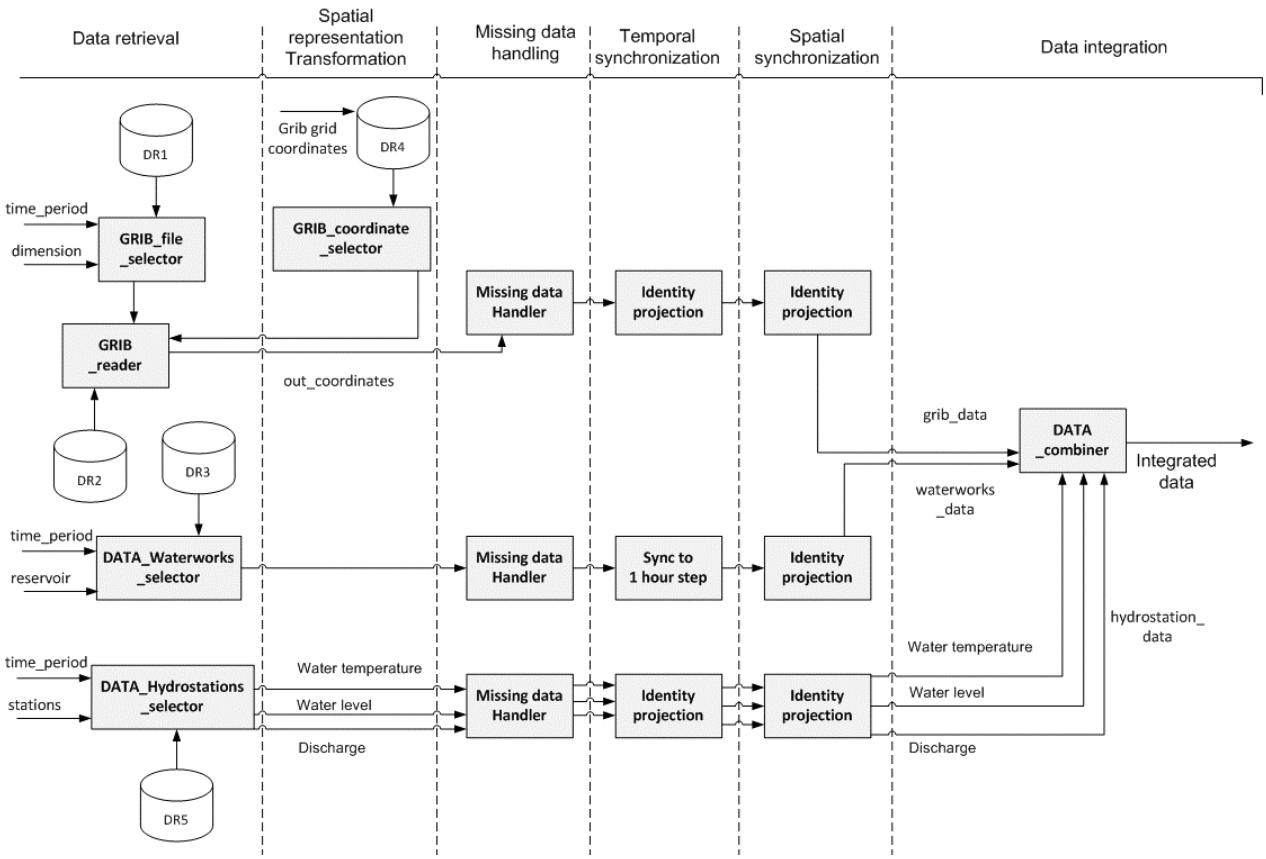


Fig. 2. Orava data management scenario.

The experimental scenario RADAR tries to predict the movement of moisture in the air from a series of radar images. Weather radar measures the reflective properties of air, which are transformed to potential precipitation before being used for data mining.

Last presented scenario, named SVP, which is still in the design phase, is the most complex of all scenarios expected to be deployed in the context of ADMIRE. It uses the statistical approach to do what the FFSC application did before ADMIRE – predict floods. The reasons why we decided to perform this experiments are mainly the complexity of simulation of floods by physical models when taking into account more of the relevant variables, and the graceful degradation of results of the data mining approach when facing incomplete data – in contrast to the physical modeling approach, which usually cannot be even tried without having all the necessary data.

For predicting floods, we have been equipped with 10 years of historical data from the Vah cascade of waterworks by the Slovak Water Enterprise, 9 years of meteorological data (precipitation, temperature, wind) computed by the ALADIN model at SHMI, hydrological data from the river Vah, again by SHMI, and additionally with measured soil capacity for water retention, courtesy of our partner Institute of Hydrology of the Slovak Academy of Sciences. We base our efforts on the theory, that the amount of precipitation, which actually reaches the river basin and contributes to the water level of the river is influenced by actual precipitation and its short-term history, water retention capacity of the soil, and to lesser extent by the evapotranspiration effect.

The described scenarios have allowed us to define and design a generic data integration engine for environmental applications – see Fig. 2. The process of environmental data integration is divided into several semi-independent stages:

- Data retrieval – operations able to retrieve the data from different, heterogeneous data sources. Data retrieval PEs are executed at data resources. This class of PEs is also responsible for transforming raw data sets to the form of tuples.
- Data transformation – operations that transform input list of tuples. These PEs can perform data transformation on per tuple basis, or can be used to aggregate tuples in the input lists. Data transformation covers spatial representation transformation, missing data handling, and temporal and spatial synchronization of data.
- Data integration – given input lists of tuples, data integration operations combine the tuples from input lists into a coherent form.

References

1. EU FP7 ICT project: Advanced Data Mining and Integration Research for Europe (ADMIRE),2008-2011. Grant agreement no.215024. http://www.admire-project.eu (accessed July 2009).

Evaluation of ordering methods for the XML-λ Framework*

Miloš Janek and Pavel Loupal

Dept. of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University
Karlovo nám. 13, 121 35 Praha 2, Czech Republic
janekm1@fel.cvut.cz, loupalp@fel.cvut.cz

1 Problem statement

In our work we deal with the problem of finding an efficient ordering method for XML. We solve this problem in context of the *XML-λ Framework* [1], an environment for evaluation of XPath/XQuery queries using a functional type system and a query language based on the simply typed lambda calculus. Inappropriate ordering is one of the most common issues that may cause poor performance results achieved by applications updating large XML documents.

Several approaches for an effective ordering already exist as a result of extensive research work up to now. We acquainted us with and implemented particularly those based on the *Dewey ordering method* [4], namely *ORDPATH*, *Extended Dewey*, and *IFDewey*.

We carried out a number of performance benchmarks of various methods and publish the results in this contribution. In the experiment, we take in account all basic operations over XML data, i.e. document parsing and serialization, and its in-memory modifications.

2 The XML-λ Framework

The initial idea of the XML-λ comes from Pokorný [2]. We adopted his approach but we *do not* use it primarily as a new language for querying XML but rather as a functional framework that serves as an environment for evaluating queries written in XPath or XQuery. Hence, we aim to describe the semantics of such languages by a functional language introduced within the framework.

The XML-λ Framework consists of three parts – a type system, a data model, and a query language. The most important part is the data model which uses sets and functions to model XML document structures. An XML document is accordingly modeled as a structure containing sets of typed values and a set of functions that aim to store and provide relationships among them (see [1]).

* This work has been partially supported by the grant No. 201/09/0990 of the Grant Agency of Czech Republic.

3 Benchmarking and results

In our experiment we use XML data created by the `XMLGEN` tool [3] that generates documents of given size according to fixed schema (specified by a DTD).

We have chosen three different scenarios for the benchmark that should give us relatively comprehensive information about qualities of the prototype. The tests cover both document parsing (bulk-loading), document serialization, and data insertion.

The results of all benchmarks confirmed the correctness of our implementation and revealed some advantages and weaknesses of particular methods.

4 Conclusion

We had successfully developed and integrated various ordering methods into the XML-λ Framework. Subsequently, we ran a number of benchmarks to prove and check our prototype. The results indicate that each variant of the ordering method excels in its intended domain. Through acquisition of these methods we have gained a solid base for further experiments.

The presented work is only a small part of our long-term effort on further development of the XML-λ Framework. The results show that it is possible to integrate our set-based solution with existing ordering methods for XML without significant loss of either functionality or performance.

References

1. P. Loupal: *XML-Lambda type system and data model revealed.* In Proceedings of DATESO '09, CEUR Workshop Proceedings, 471, 2009, 130–141.
2. J. Pokorný: *XML-λ: an extensible framework for manipulating XML data.* In Proceedings of BIS 2002, Poznan, 2002, 160–168.
3. A.R. Schmidt et al.: *The XML benchmark project.* Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April 2001.
4. I. Tatarinov et al.: *Storing and querying ordered XML using a relational database system.* In M.J. Franklin, B. Moon, and A. Ailamaki, editors, SIGMOD Conference, ACM, 2002, 204–215.

Dynamické vlastnosti pravdepodobnostných fuzzy klopných obvodov

Martin Klimo and Juraj Boroň

Katedra informačných sietí, Žilinská Univerzita, Fakulta riadenia a informatiky
Univerzitná 2, 01 026 Žilina, Slovensko

Abstrakt. *Existuje viacero spôsobov a prístupov ako rozpoznávať reč. V dnešnej dobe sa začína presadzovať prístup s využitím neurónových sietí. Avšak aj tento prístup je len určitým rámcom, v ktorom je veľké množstvo možností, ako základné myšlienky implementovať. Základným prvkom každej neurónovej siete je neurón, z ktorého je zostavená neurónová sieť o určitej štruktúre. Je však možné použiť rôzne typy neurónov, rôzne typy štruktúr a pod. V tomto článku analyzujeme základné vlastnosti nami navrhnutého neurónu a to dynamické vlastnosti pravdepodobnostného fuzzy klopného obvodu so zabúdaním. U klopného obvodu závisí výstup nielen od momentálnych hodnôt vstupov, ale aj od jedného resp. viacerých predchádzajúcich stavov obvodu. To dáva klopným obvodom možnosť zapamätať si predchádzajúci stav až po ďalšiu zmenu na vstupoch. V tomto článku ako základ vytváraných klopných obvodov nepoužijeme Booleovu algebru ale pravdepodobnostnú algebru, čo umožňuje zaviesť funkciu zabúdania. Základným klopným obvodom je klopný obvod typu RS z ktorého sa odvodzujú ďalšie typy. V tomto článku si ukážeme dynamické vlastnosti nami testovaných klopných obvodov, ktoré budú neskôr tvoriť základ pre tvorbu neurónovej siete pre rozpoznávanie reči.*

1 Úvod

Na Katedre informačných sietí Žilinskej univerzity sa výskum dlhodobo venuje oblasti prenosu a spracovania reči. Jednou z úloh v tomto rámci je aj úloha rozpoznávania difónov v reči. Ako základ pre túto úlohu sme si zvolili neurónovú sieť. Jej základným prvkom je nelineárny systém, ktorý ďalej nazývame neurónom. Ako základnú štruktúru pre takýto neurón sme zvolili pravdepodobnostný fuzzy klopný obvod so zabúdaním.

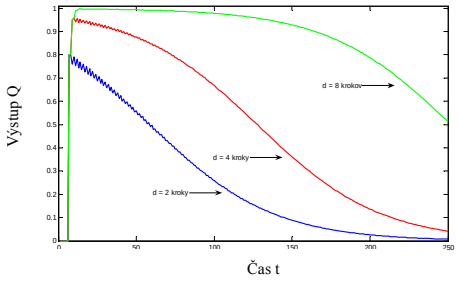
Klasické umelé neurónové siete modelujú neurón ako nelineárnu funkciu lineárnej kombinácie vstupných hodnôt, kde koeficienty lineárnej kombinácie predstavujú synaptické váhy. Úlohou fázy učenia je nájsť tieto koeficienty tak, aby sa minimalizovala chyba riešenia určitej úlohy po ukončení učenia. Naším zámerom je študovať neurónové siete bez synaptických váh, v ktorých učenie bude meniť štruktúru siete, ale nie váhu synapsií. Matica systému v takomto prípade vyjadruje len prepojenie medzi neurónmi a stavové premenné sú priamo výstupy z jednotlivých neurónov. Ako prvý krok sa nám zdá vhodné uvoľniť len podmienku dvojstavovej množiny Boolovej algebry na zavedenie spojitých vstupov, stavov a výstupov a štruktúru (t.j. väzby medzi prvkami) ponechať dvojhodnotové. Takýto prístup umožňuje siete s pravdepodobnostnou algebrou jednoducho redukovať na siete s Boolovou algebrou [1]. Preto ako definičný obor hodnôt vstupov, stavov a výstupov bol zvolený uzavretý jednotkový interval $a,b \in \langle 0,1 \rangle \subset \Re$, kde $\Re$ je množina reálnych čísel a za základnú štruktúru sme zvolili pravde-podobnostnú algebru. Čitateľ môže nájsť podrobne spracovanú problematiku fuzzy logík v [2].

2 Klopné obvody s pravdepodobnostnou algebrou

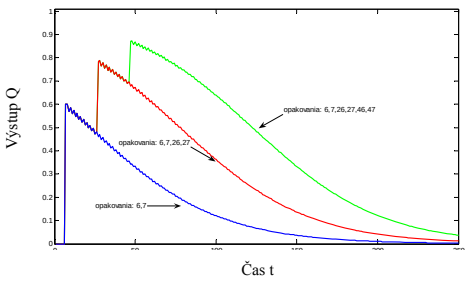
Klopné obvody majú tú vlastnosť, že si dokážu pamätať predchádzajúce stavy, t.j. ich výstup závisí nielen od momentálnych vstupov ale aj od jedného resp. viacerých predchádzajúcich stavov obvodu. Základným klopným obvodom, z ktorého sa odvodzujú ďalšie typy klopných obvodov, je binárny obvod typu RS, ktorý má dva vstupy označené symbolmi R (reset) a S (set) a výstup Q resp. Qn. Modelovanie neurónových sietí za použitia klopných obvodov a nelineárnej matematiky je taktiež popísané v [3]. Analogicky ako binárny klopný obvod typu RS, sa správa aj pravdepodobnostný klopný obvod typu RS.

Pridávaním ďalších vstupov a logických členov ku základnému klopnému obvodu typu RS, môžeme zostaviť ďalšie klopné obvody s odlišnými vlastnosťami. Hoci detailný popis štruktúry presahuje možnosti príspevku, uvedieme aspoň základné vlastnosti niektorých testovaných klopných obvodov, ktoré vychádzajú zo základného klopného obvodu typu RS a pridávaním ďalších vhodných vstupov a logických členov. Pridaním ďalšieho vstupu *zabúdanie* ku základnému obvodu typu RS a pridaním ďalších logických členov, sme dosiahli, že výstup takéhoto klopného obvodu sa preklopí ako v type RS, avšak po určitom čase sa vracia späť k pôvodnej hodnote.

Ukážme si ďalej čo sa stane ak po určitom čase znova vybudíme klopný obvod preklápajúcim vstupom.



Obr. 1. Výstup Q klopného obvodu so zabúdaním pre rôzny počet krokov budenia (2,4,8).

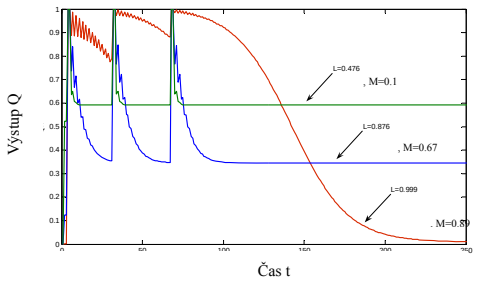


Obr. 2. Výstup Q klopného obvodu so zabúdaním pre rôzny počet opakovaní.

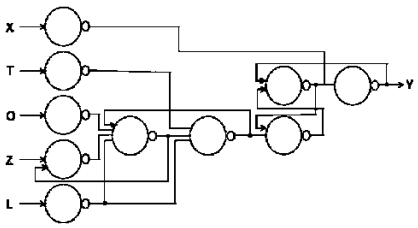
Vidíme, že opakovanie nielen zvyšuje hodnoty výstupu, na ktorú dokáže preklápací impulz vybudiť výstup, ale tým, že spätnou väzbou mení hodnotu zabúdania, mení aj charakter zabúdania. To môže mať význam v procese učenia.

Správanie pravdepodobnostného klopného obvodu ukazuje, že pri hodnotách vstupných a stavových veličín blízkych nule sa klopný obvod dostáva do pomerne stabilných stavov, ktoré nie je možné zmeniť menšími hodnotami vstupných veličín. Preto bol zavedený ďalší vstup, ktorý tieto hodnoty limituje na zadanú vstupnú hodnotu (presnejšie povedané na invertovanú hodnotu vstupu L), čím sa mení citlivosť pravdepodobnostného klopného obvodu voči vstupom.

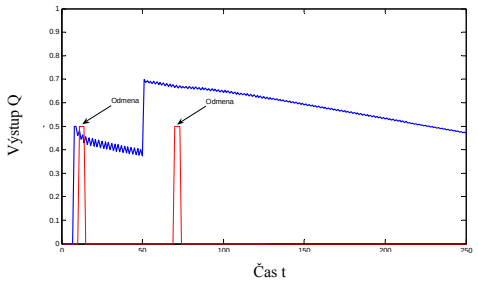
Tieto pravdepodobnostné klopné obvody plánujeme v budúcnosti využiť ako elementárne prvky sietí s časovým kódovaním. To znamená, že časti siete, ktoré sú preklopené v zhode so zámerom na ktorý je sieť určená, by mali ostať preklopené podstatne dlhší čas (mali by byť zapamätané) v porovnaní s časťami siete ktoré sú preklopené v rozpore so zámerom a ktoré by mali byť zabudnuté. Takéto posilňovanie alebo zoslabovanie však nebude určené adresne na jednotlivé klopné obvody, ale bude paušálne pre celú sieť. Preto zavádzame ďalšie vstupy T (trest), O (odmena).



Obr. 3. Výstup Q a Q_n klopného obvodu s pamäťou a s rôznymi hodnotami limitu.



Obr. 4. Klopný obvod s odmenou a trestom.



Obr. 5. Výstup Q a odmena O pri zabúdaní (0.1).

Na obrázku 5 je vidieť ako po nastavení vstupu odmena sa takýto klopný obvod správa. Výstup sa zväčší a klopný obvod je schopný pamätať si dlhšiu dobu. Analogicky pri nastavení vstupu trest si klopný obvod pamätá kratšiu dobu, zabúda rýchlejšie.

3 Záver

Článok poukazuje na to, že kombináciou klopného obvodu ako základného pamäťového prvku v digitálnych systémoch a funkcie zabúdania (dosiahnutej vďaka použitiu pravdepodobnostej algebry) je možné dosiahnuť časové kódovanie v zmysle doby pamätania si stavu elementu. Výsledky experimentov s takouto štruktúrou ukazujú, že je možné dosiahnuť niektoré želateľné správania, akými sú predlžovanie stavu vyššou hodnotou vstupu, dlhším trvaním vstupu alebo opakovaním. Je tiež možné podmieňovať trvanie stavu iným vstupom (pamäť), čo umožňuje Hebbovo učenie. Vstupy pre odmenu a trest umožňujú súčasné ovplyvňovanie stavu viacerých klopných obvodov, čo imituje nástroje učiteľa pre podporu pamätania a zabúdania. Predpokladáme a plánujeme overiť ďalším výskumom, že siete takýchto elementov by bolo možné učiť, na rozdiel od neurónových sietí, bez zmeny synaptických váh. Samotný proces učenia by v takomto prípade znamenal dovedenie nelineárneho systému (bez modifikácie „matice“ systému) do takého stavu, v ktorom systém na dané vstupy dáva požadované výstupy. Proces hľadania vhodnej matice systému by potom bolo možné oddeliť od procesu učenia a presunúť ho do sféry evolučného vývoja populácie takýchto sietí. Našou snahou bude otestovať uvedené myšlienky na úlohe rozpoznávania reči.

Referencie

1. F.P. Preparata, R.T. Yeh: *Úvod do teórie diskrétnych matematických štruktúr*, ALFA/SNTL, Bratislava, 1982, 63-563-82.
2. P. Hájek: *Metamathematics of fuzzy logic*. Trends in Logic, Kluwer, Dordrecht, 4, 1998, ISBN 0-7923-5238-6.
3. D.M. Dubois: *Hyperincursive McCulloh and Pitts neurons for designing a computing flip-flop memory*, American Institute of Physics, AIP Conference Proceedings, 1999.

Tractability of approximation by connectionistic models*

Věra Kůrková

Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague,
vera@cs.cas.cz, http://www.cs.cas.cz/∼vera

Abstrakt *The role of dimensionality in approximation by connectionistic models is investigated. Upper bounds on worst-case errors in the factorized form $\xi(d)\kappa(n)$ with an nonincreasing function $\kappa(n)$ of the number n of computational units corresponding to model complexity and a function $\xi(d)$ of the input dimension d are analyzed. Sets of functions for which the function $\xi(d)$ is a polynomial are called tractable. Methods from nonlinear approximation theory are used to describe tractable sets in approximation by connectionistic models. The results are illustrated by examples of Gaussian radial networks.*

Experimental results have shown that connectionistic models built from relatively few computational units with a simple structure (e.g., neural networks, radial-basis-function networks, kernel models, free-node splines) may obtain surprisingly good performances in high-dimensional tasks. Such models take on the form of linear combinations of all n-tuples of functions computable by computational units of various kinds (e.g., perceptrons, radial or kernel units, Hermite functions, trigonometric polynomials or splines). The integer n can be interpreted as the *model complexity* measured by the number of computational units.

Estimates of model complexity of connectionistic systems have been derived by various authors from upper bounds on rates of approximation of input-output functions with increasing number n. Such estimates have been formulated using "big o" notation, i.e., in the form $O(\kappa(n))$, with $\kappa(n) = n^{-1/2}$ (see, e.g., [1]), $\kappa(n) = n^{-1/p}$ [2] or $\kappa(n) = n^{-1/2}(\log n)^{1/2}$ [3]. So they were only focused on the dependence of approximation errors on model complexity n but they hide dependence on the dimension d of input data, which was by some authors even called "constant". However in [8,7], we have shown that in some cases dependence on input dimension d may be even exponentional.

In this paper we investigate upper bounds on rates of approximation in the form

$$\xi(d)\,\kappa(n),$$

where $\xi : \mathbb{N} \to \mathbb{R}_+$ is a function of the number d of network inputs (which is equal to the number of variables of input-output functions) and $\kappa : \mathbb{N} \to \mathbb{R}_+$ is a nonincreasing function of the model complexity n. Model complexity sufficient for approximation of functions from given class within a prescribed accuracy $\varepsilon = \xi(d)\,\kappa(n)$ can be estimated from such bounds as a function of the dimension d. For example, for $\kappa(n) = n^{-1/2}$, we get $n = \frac{\xi(d)^2}{\varepsilon^2}$.

To describe cases when model complexity grows only polynomially with the dimension d, in [5] we introduced the concept of *tractability*. We defined it for approximation by connectionistic models of the form

$$\mathrm{span}_n G_d := \left\{ \sum_{i=1}^{n} w_i g_i \,|\, w_i \in \mathbb{R}, g_i \in G_d \right\},$$

where linear combinations correspond to input-output functions of networks with n units from the set of computational units G_d with d inputs. We say that functions from sets A_d can be *tractably approximated by elements of* $\mathrm{span}_n G_d$ with approximation error measured by the norm $\|.\|_{\mathcal{X}_d}$ when the upper bound

$$\delta(A_d, \mathrm{span}_n G_d)_{\mathcal{X}_d} := \sup_{f \in A_d} \inf_{g \in \mathrm{span}_n G_d} \|f - g\|_{\mathcal{X}_d} \leq \xi(d)\kappa(n), \quad (1)$$

holds with the function $\xi(d)$ of the input dimension d *polynomial*. G_d is usually called a *dictionary*.

Description of sets of functions which can be tractably approximated by networks with various types of units can provide some understanding to the role of dimensionality in neurocomputing. We describe such tractable sets as sets of functions satisfying suitable quantitative constrains on their derivatives or smoothness. We formulate such constrains in terms of various norms. Thus we investigate upper bounds in the form (1) for sets $A_d = B_{r_d}(\|.\|_F)$, where $B_{r_d}(\|.\|_F)$ denotes the ball of radius r_d centered at zero in the norm $\|.\|_F$.

Inspection of such bounds implies conditions on growth or decrease of radii r_d which guarantee tractability of approximation of functions from $B_{r_d}(\|.\|_F)$, i.e., $\xi(d)\,\kappa(n)$ with $\xi(d)$ polynomial, which depends on r_d (often it is a product of r_d with other factors).

We first describe tractable sets in the form of balls in norms tailored to computational units from general

* This work was partially supported by the project 1ET100300517 of the program Information Society of the National Research Program of the Czech Republic and the Institutional Research Plan AV0Z10300504.

dictionaries. The norm called *G-variation* and denoted $\|.\|_{G,\mathcal{X}}$ is defined

$$\|f\|_{G,\mathcal{X}} := \inf\left\{c>0 \,|\, c^{-1}f \in \mathrm{cl}_{\mathcal{X}}\,\mathrm{conv}\,(G\cup -G)\right\}.$$

The term "variation" is motivated by the special case of the dictionary formed by functions computable by Heaviside perceptron networks for which in the one-variable case the variational norm is equal (up to a constant) to the concept of total variation from integration theory. The class of variational norms also includes as a special case the ℓ_1-norm (for G orthonormal). For dictionaries G corresponding to Heaviside perceptrons and Gaussian radial units, G-variation can be estimated using Sobolev and Bessel norms [6,4].

The following theorem gives estimates of worst-case errors of functions in balls in variational norm with respekt to general dictionsaries.

Theorem 1. *Let d be a positive integer, $(\mathcal{X}_d, \|.\|_{\mathcal{X}_d})$ be a Banach space of d-variable functions, G_d its bounded subset with $s_d = \sup_{g\in G_d}\|g\|_{\mathcal{X}_d}$, $r_d > 0$ and n be a positive integer. Then*
(i) for $(\mathcal{X}_d, \|.\|_{\mathcal{X}_d})$ a Hilbert space,
$\delta(B_{r_d}(\|.\|_{G_d}), \mathrm{span}_n\, G_d)_{\mathcal{X}_d} \le s_d\, r_d\, n^{-1/2}$;
(ii) for $(\mathcal{X}_d, \|.\|_{\mathcal{X}_d}) = (\mathcal{L}^p(\Omega_d,\rho), \|.\|_{\mathcal{L}^p})$ with $p\in(1,\infty)$ and ρ a measure on $\Omega_d \subseteq \mathbb{R}^d$,
$\delta(B_{r_d}(\|.\|_{G_d}), \mathrm{span}_n\, G_d)_{\mathcal{L}^p} \le 2^{1+1/\bar{p}}\, s_d\, r_d\, n^{-1/\bar{q}}$,
where $\frac{1}{p}+\frac{1}{q}=1$, $\bar{p}=\min(p,q)$, and $\bar{q}=\max(p,q)$;
(iii) for $(\mathcal{X}_d, \|.\|_{\mathcal{X}_d}) = (\mathcal{M}(\Omega_d), \|.\|_{\sup})$ the space of measurable functions on $\Omega_d \subseteq \mathbb{R}^d$ with the supremum norm and G_d a subset of the sets of characteristic functions on Ω_d such that the co-VC-dimension $h^*_{G_d}$ of G_d is finite, $\delta(B_{r_d}(\|.\|_{G_d}), \mathrm{span}_n G_d)_{\sup} \le$
$6\sqrt{3}\,\left(h^*_{G_d}\right)^{1/2} r_d\,(\log n)^{1/2}\; n^{-1/2}$.

In the upper bounds from Theorem 1, the functions $\xi(d)$ are of the forms $\xi(d) = c\, s_d\, r_d$ and $c\,\left(h^*_{G_d}\right)^{1/2} r_d$, where c is an absolute constant. So Theorem 1 implies tractability when $s_d\, r_d$, and $h^*_{G_d}\, r_d$, resp., grow polynomially with d increasing. Note that s_d and $h^*_{G_d}$ are determined by the choice of the dictionary G_d, but r_d can be chosen in such a way that $\xi(d)$ is a polynomial.

Dependence of tractability of balls $B_{r_d}(\|.\|_{G_d})$ on the size of their radii r_d can be illustrated by an example of Gaussian radial-basis networks. Let $\gamma_{d,b} : \mathbb{R}^d \to \mathbb{R}$ denote the *d-dimensional Gaussian function of the width $b>0$*, defined as

$$\gamma_{d,b}(x) = e^{-b\|x\|^2}.$$

We denote by $G^\gamma_d(b)$ the set of *Gaussian radial-basis d-variable functions with a fixed width b and varying centroids*, i.e., $G^\gamma_d(b) = \left\{(\gamma_{d,b}(.-y)\,|\, y\in\mathbb{R}^d\right\}$. A simple calculation shows that for any $b>0$, $\|\gamma_{d,b}\|_{\mathcal{L}^2(\mathbb{R}^d)} = (\pi/2b)^{d/4}$. Thus the sets $G^\gamma_d(b)$ of Gaussians with fixed widths are bounded in the spaces $(\mathcal{L}^2(\mathbb{R}^d), \|.\|_{\mathcal{L}^2})$ with $\sup_{g\in G^\gamma_d(b)}\|g\|_{\mathcal{L}^2} = (\pi/2b)^{d/4}$. Applying Theorem 1(i) to approximation by Gaussians with fixed widths we get the upper bound

$$\delta(B_{r_d}(\|.\|_{G^\gamma_d}(b)), \mathrm{span}_n\, G^\gamma_d(b))_{\mathcal{L}^2} \le r_d\,\left(\frac{\pi}{2b}\right)^{d/4}\; n^{-1/2}. \tag{2}$$

In this upper bound, $\xi(d) = (\pi/2b)^{d/4}\, r_d$. Thus for $b=\pi/2$, (2) implies tractability for r_d growing with d polynomially, for $b>\pi/2$, it implies tractability even when r_d is increasing exponentially fast, while for $b<\pi/2$, it merely implies tractability for r_d decreasing exponentially fast. Hence, the width b of Gaussians has a strong impact on the size of radii r_d of balls in $G^\gamma_d(b)$-variation for which $\xi(d)$ is a polynomial. The narrower the Gaussians, the larger the balls for which (2) implies tractability.

For perceptron and Gaussian radial functions, variational norms can be estimated using more norms defined in terms of norms defined using various properties of derivatives and smoothness (such as Sobolev and Bessel norms) [6,4]. Thus Theorem 1 also implies tractability results for functions which can be described in terms of conditions on their derivatives.

Reference

1. A.R. Barron: *Universal approximation bounds for superpositions of a sigmoidal function.* IEEE Transactions on Information Theory, 39, 1993, 930–945.
2. C. Darken, M. Donahue, L. Gurvits, and E. Sontag: *Rate of approximation results motivated by robust neural network learning.* In Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, The Association for Computing Machinery, New York, 1993, 303–309.
3. L. Gurvits and P. Koiran: *Approximation and learning of convex superpositions.* J. of Computer and System Sciences, 55, 1997, 161–170.
4. P.C. Kainen, V. Kůrková, and M. Sanguineti: *Complexity of Gaussian radial basis networks approximating smooth functions.* J. of Complexity, 25, 2009, 63–74.
5. P.C. Kainen, V. Kůrková, and M. Sanguineti: *On tractability of neural-network approximation.* In M. Kolehmainen et al., editor, ICANNGA 2009, LNCS 5495, Berlin, Heidelberg, Springer-Verlag, 2009, 11–21.
6. P.C. Kainen, V. Kůrková, and A. Vogt: *A Sobolev-type upper bound for rates of approximation by linear combinations of Heaviside plane waves.* J. of Approximation Theory, 147, 2007, 1–10.
7. V. Kůrková: *Minimization of error functionals over perceptron networks.* Neural Computation, 20, 2008, 252–270.
8. V. Kůrková, P. Savický, and K. Hlaváčková: *Representations and rates of approximation of real–valued Boolean functions by neural networks.* Neural Networks, 11, 1998, 651–659.

Modelovanie QoS mechanizmu Weighted Round Robin s využitím teórie hromadnej obsluhy

Dušan Nemček

Fakulta riadenia a informatiky, Žilinská univerzita v Žiline
Univerzitná 8215/1, 010 26 Žilina, Slovensko

1 Cieľ

Kvalita služieb (*QoS – Quality of Service*) je termín, ktorý predstavuje komplexné opatrenia, snažiace sa zaručiť koncovému používateľovi doručenie dát v požadovanej kvalite nezávisle od ich typu, od zariadení po trase alebo prenosovej technológii a médiu 1. Základ kvality služieb spočíva v obmedzovaní a prideľovaní šírky pásma potrebným tokom. V súčasných sieťach veľmi často nasadzovaný a konfigurovaných QoS mechanizmus, ktorý sa priamo podieľa na delení celkovej šírky pásma medzi jednotlivé triedy služieb, je tzv. *Weighted Round Robin* (WRR) 2. Cieľom práce je na jednoduchom príklade ukázať ako je s využitím teórie hromadnej obsluhy modelovať tento mechanizmus so zameraním na výpočet jedného z kľúčových parametrov, ktorým je pravdepodobnosť odmietnutia paketov z frontov tzn. pravdepodobnosť straty paketov.

2 Východiská

WRR patrí do skupiny QoS mechanizmov *Queue Scheduling Disciplines* (QSD) 3. QsD je množina QoS mechanizmov, ktoré riadia rozdelenie celkovej šírky pásma výstupnej linky medzi viaceré triedy sieťových služieb (servisné triedy). Tento proces sa deje v smerovači pri výbere paketu z frontov (vyrovnávacích pamätí), ktoré sú pridelené príslušným servisným triedam a jeho poslaní na výstupnú linku. WRR podporuje toky servisných tried s významne odlišnými požiadavkami na šírku pásma. Každému frontu môže byť pridelený rôzny podiel zo šírky pásma výstupného portu resp. linky. WRR definuje tzv. servisné kolo, v ktorom obslúži každý front niekoľkokrát. Pri obsluhe frontu sa z frontu odoberie práve jeden paket. Ak je front prázdny, tak sa neobsluhuje. Hrubé rozdelenie výstupnej prenosovej kapacity linky je dosahované tým, že pre každý front je určené, koľko krát sa z neho odošle paket v jednom servisnom kole. Z každého neprázdneho frontu je v jednom kole odoslaný aspoň jeden paket, čím sa prechádza blokovaniu neprioritných paketov.

Uvažovaný systém modelujúci činnosť WRR má jeden vysielač paketov, ktorý zdieľajú dva nezávislé toky paketov radené do konečných frontov s maximálnymi dĺžkami L_1 resp. L_2. Maximálne dĺžky frontov sú dané kapacitou vyrovnávacích pamätí, v ktorých sa fronty aj fyzicky nachádzajú. Do frontov prichádzajú pakety s intenzitou λ_1 resp. λ_2. Vstupný tok je pri tom *Poissonov*. Pakety pochádzajúce z prvého toku sú pakety prvého typu, pakety pochádzajúce z druhého toku sú pakety druhého typu. Pakety sú vo frontoch organizované podľa metódy FIFO. Vysielač vyberá z frontov pakety a následne ich vysiela bit po bite na výstupnú linku. Vysielaný môže byť v jednom momente vždy len jeden paket a to buď paket prvého alebo druhého typu. Po dokončení vysielania nasleduje okamžite výber ďalšieho paketu z frontov. Výber paketov z frontov však nie je deterministický ako v reálnom algoritme WRR, ale sa modeluje zavedením pravdepodobností výberov paketov z jednotlivých frontov, tzn., že zo súčasne neprázdnych frontov sa pakety vyberajú s pevne stanovenou pravdepodobnosťou p_1 resp. $p_2 = 1 - p_1$. Hodnota p_1 resp. p_2 tak určuje váhu pridelenú príslušnému frontu. Ak je jeden z frontov prázdny, z neprázdneho sa vyberajú s pravdepodobnosťou $p_1 = 1$ (resp. $p_2 = 1$) a systém sa tak v tomto špeciálnom prípade správa ako jednofrontový systém, čo plne zodpovedá koncepcii algoritmu WRR. Doby vysielania paketov majú exponenciálne rozdelenie so strednou dobou τ_1 resp. τ_2. Ich prevrátené hodnoty μ_1 resp. μ_2 sú stredné intenzity vysielania paketov, avšak len v prípade, že susedný front je prázdny. Keďže výstupný tok je rovnako ako vstupný tok bez pamäte, pričom v jednom okamihu môže nastať len jedna elementárna udalosť (vstup paketu do frontu, vybranie paketu z frontu), jedná sa o *Markovov systém*. Pre modelovanie uvedeného systému tak možno využiť teóriu *Markovových reťazcov* 4, ktorá je základnou súčasťou teórie hromadnej obsluhy.

3 Metódy

Pre strednú intenzitu vysielania všetkých paketov na výstupnú linku v uvažovanom systéme sa dá za predpokladu, že ide o Markovov systém, odvodiť nasledujúci vzťah:

$$\mu = \frac{\mu_1 \mu_2}{\mu_1 p_2 + \mu_2 p_1}$$

Táto intenzita je pri tom rovná intenzite, s akou sú pakety z frontov vyberané do vysielača.

Pod pojmom intenzita vysielania paketov *i*-teho frontu (μ_i') sa chápe intenzita, s akou sú pakety *i*-teho typu vysielané na výstupnú linku a to v prípade, že v oboch frontoch sú nejaké pakety, teda v prípade, že oba fronty sú neprázdne. Platí pre ňu, že $\mu_i' <= \mu_i$, kde μ_i je intenzita vysielania paketov *i*-teho frontu v prípade, že susedný front je prázdny, počítaná ako prevrátená hodnota stredného času vysielania paketu (τ_i). Pri charakteristike uvažovaného systému sa uvádza práve táto hodnota. Pre výpočet parametrov μ_1' a μ_2' sa dá odvodiť nasledovný vzťah:

$$\mu_i' = p_i \mu = p_i \frac{\mu_1 \mu_2}{\mu_1 p_2 + \mu_2 p_1}$$

Rozoberaný systém možno názorne popísať grafom prechodov pozostávajúceho z vrcholov a orientovaných hrán. Vrcholy grafu pritom predstavujú stavy do ktorých sa

systém môže v čase dostať. Nech $L_1 = m$ a $L_2 = n$, pričom $m, n \in$ N. Keďže sa jedná o dvojfrontový systém na popis je najvýhodnejšie použiť dvojrozmerný graf s rozmermi $((m+1)_x(n+1))$ vrcholov. Ak je systém v stave, ktorému zodpovedá vrchol daný dvojicou súradníc (i,j), znamená to, že je v ňom i paketov prvého typu a j paketov druhého typu. Orientované hrany predstavujú možné prechody systému z jedného stavu do bezprostredne ďalšieho a ich ohodnotenia sú rovné intenzitám, s akými k týmto prechodom dochádza.

Vývoj uvažovaného systému v čase možno popísať systémom *Kolmogorových diferenciálnych rovníc* 4:

$$p'(t) = p(t)Q$$

$p(t)$ je vektor pravdepodobností všetkých stavov systému v čase t a Q je matica intenzít prechodov medzi týmito stavmi. Na základe uvedeného vzťahu možno odvodiť napríklad pre neokrajové stavy prechodového grafu $(0 < i < m,\ 0 < j < n)$ nasledovný systém rekurentných rovníc:

$$\begin{aligned} p_{i,j}(t+\Delta t) = \Delta t(& \lambda_1 p_{i-1,j}(t) + \mu_2' p_{i,j+1}(t) + \\ & + \mu_1' p_{i+1,j}(t) + \lambda_2 p_{i,j-1}(t)) + \\ & + (1 - \Delta t(\lambda_2 + \mu_1' + \mu_2' + \lambda_1)) p_{i,j}(t) \end{aligned}$$

Ten sa využíva na numerický výpočet pravdepodobností stavov systému v určitom čase. V každom kroku, ktorý predstavuje uplynutie časového okamihu Δ, sa pre každý stav (vrchol prechodového grafu) systému volá práve raz príslušná rovnica tak, že jej výsledok $(p_{ij}(t+\Delta t))$ z prechádzajúceho kroku je vstupom $(p_{ij}(t))$ pre krok aktuálne vykonávaný a jeho výsledok sa stáva následne vstupom pre krok nasledujúci. Spomínaný postup vyplýva z rekurentnej povahy uvedených rovníc. Na to, aby sa mohli pravdepodobnosti jednotlivých stavov systému prehlásiť za ustálené (nezávislé na čase), je nutné vykonať takýchto krokov 1000 až 10000 pri veľmi malom Δ. Prvý krok vychádza z ľubovoľne zvoleného rozdelenia pravdepodobnosti stavov, pre ktoré však musí platiť, že súčet pravdepodobností všetkých stavov bude rovný 1.

Uvažovaný systém predpokladá konečné dĺžky frontov. Môže tak dochádzať k situácii, že aspoň jeden z frontov bude plný, tzn., že pakety radené do takéhoto frontu budú odmietané. Nech je maximálna dĺžka prvého frontu $L_1 = m$ a druhého frontu $L_1 = n$, pričom $m, n \in N$. Nech $\pi_{i,j}$ je pravdepodobnosť, že po ustálení bude systém v náhodne zvolenom okamihu v stave (i,j). Potom pre pravdepodobnosti odmietnutia paketov z jednotlivých frontov Ps_1 resp. Ps_2 platí:

$$Ps_1 = \sum_{j=0}^{n} \pi_{m,j} \qquad Ps_2 = \sum_{i=0}^{m} \pi_{i,n}$$

Inými slovami pravdepodobnosť s akou dochádza k situácii, že paket smerujúci do príslušného frontu bude odmietnutý, je rovná súčtu pravdepodobností všetkých stavov ustáleného systému, kedy je tento front plný.

Pre celkovú pravdepodobnosť odmietnutia paketov Ps, kde $\lambda = \lambda_1 + \lambda_2$, platí:

$$Ps = \frac{\lambda_1}{\lambda} Ps_1 + \frac{\lambda_2}{\lambda} Ps_2$$

4 Výsledky

Pre získanie konkrétnych výsledkov bolo nutné zostaviť aplikáciu, ktorá prevádza časovo náročný rekurentný výpočet pravdepodobností všetkých stavov systému a z nich pre zadané parametre systému následne spomínaným spôsobom získa jednotlivé hodnoty Ps, Ps_1, Ps_2. Napríklad pre $p_1 = 0.4$, $p_2 = 0.6$, $\lambda_1 = 5$, $\mu_1 = 6$, $\lambda_2 = 4$, $\mu_2 = 7$, $L_1 = 3$, $L_2 = 5$ boli získané hodnoty $Ps_1 = 0.441$, $Ps_2 = 0.15$, $Ps = 0.312$. Aplikácia umožňovala graficky vyhodnotiť aj závislosti Ps, Ps_1, Ps_2 od zmeny ľubovoľných dvoch parametrov, ktoré boli zvolené ako parametre premenné na danom intervale.

5 Záver

Na základe vykonaných experimentov uskutočnených prostredníctvom testovacej aplikácie je možné dôjsť k nasledovným záverom.

Prvý faktor, ktorý vplýva na vzájomne ovplyvňovanie frontov je samotné delenie jednotkovej pravdepodobnosti medzi ne. Ak sa o určitú hodnotu zvýši pravdepodobnosť výberu z jedného frontu, presne o tú hodnotu sa pravdepodobnosť výberu z druhého frontu zníži. To má za následok zníženie alebo zvýšenie pravdepodobnosti odmietnutia paketov príslušného frontu.

Druhý faktor, ktorý sa výrazne vplýva jeden front na druhý vychádza zo zdieľania času jedného vysielača medzi pakety dvoch frontov. Kým je vysielaný prvý paket z jedného frontu, prvý paket z druhého frontu musí čakať, keďže nebol vybraný. Čím dlhšie je paket z prvého frontu vysielaný tým dlhšie paket v druhom fronte musí čakať na vysielanie, čo znižuje aj skutočnú intenzitu vysielania paketov z druhého frontu oproti intenzite, keď by prvý front neexistoval tzn. oproti intenzite μ_2.

Tretím faktorom, ktorým jeden front ovplyvňuje výsledky druhého je intenzita stavu, kedy vo fronte nie je žiadny paket. Ak dôjde k takejto situáciu, z druhého frontu sú vyberané pakety s pravdepodobnosťou rovnou jednej bez ohľadu na jemu priradenú pravdepodobnosť výberu. Platí teda, že čím častejšie je jeden front prázdny, tým častejšie sú z druhého frontu vyberané pakety s pravdepodobnosťou rovnou 1, tzn. vysielané s intenzitou μ_i a tým menšie sú jeho straty.

Tento rozšírený abstrakt vo svojej podstate stručne popísal numerický model mechanizmu WRR v systéme s dvoma frontami a na základe experimentov s ním vykonaných naznačil niektoré závery. Rozšírený článok sa v budúcnosti môže stať dobrým základom pre modelovanie ostatných QoS mechanizmov ako aj mechanizmu WRR v zložitejších systémoch.

Literatúra

1. M. Klimo, a kol.: *Modelovanie IP prevádzky*, 2008, dostupné na ŽU FRI KIS.
2. CISCO: *Planning for Quality of Service*. Documentation of CISCO, 2004.
3. Ch. Semeria: *Supporting differentiated service classes: Queue schdeuling principles*, dostupné na: http://cn.juniper.net/solutions/literature/white_papers/200020.pdf , 26.4.2009.
4. Š.Peško, J. Smieško: *Stochastické modely operačnej analýzy*, Žilinská univerzita v Žiline, Žilina, 1999.

Kolaboratívna anotácia webových stránok

Róbert Novotný and Peter Kál
Ústav informatiky, Univerzita P. J. Šafárika, Košice

Abstrakt. *Prezentujeme návrh a architektúru systému na kolaboratívnu, používateľmi podporovanú anotáciu webových stránok, ktorá slúži ako podklad pre extrakciu objektových dát z nich.*

1 Úvod a motivácia

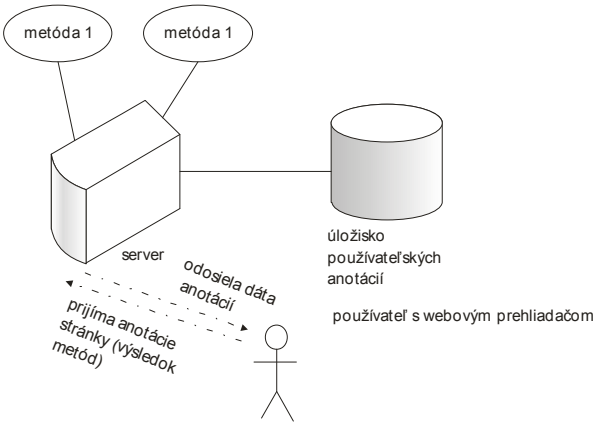
Predajcovia a poskytovatelia služieb často prezentujú svoje produkty či riešenia v podobe štruktúrovaných webových stránok. Ukazuje sa, že štandardne sú prezentované v dvoch formách: buď v podobe prehľadových stránok zahrňujúcich dáta o viacerých produktoch (typicky v tabuľkovej forme) alebo v detailných stránkach, ktoré zobrazujú dáta o jedinom produkte.

Extrakcia dát z takýchto stránok je realizovaná viacerými existujúcimi prístupmi, napr. strojovým učením, ktoré generuje extrakčné automaty[3]. Inými príkladom sú nástroje ako Lixto [2], ktoré ponúkajú používateľsky prítulný spôsob grafického vyznačovania relevantných prvkov. Takto anotovaná stránka môže slúžiť ako predloha pre extrakciu podobných stránok. Iným, plnoautomatickým príkladom je systém spomínaný v [1], ktorý je založený na vyhľadávaní dátových regiónov a dátových záznamov v prehľadových stránkach, doplňaný extrakciou dát z detailných stránok na základe porovnávania štruktúry HTML kódu.

Ukazuje sa však, že množstvo nástrojov si vyžaduje buď značné úsilie používateľa, alebo istú technickú zdatnosť, prípadne znalosť princípov metodológie (napr. výber vzorky pre strojové učenie). V našom prístupe prezentujeme filozofiu *kolaboratívnej anotácie*, kde viacero používateľov graficky vyznačuje relevantné elementy webových stránok pomocou webového prehliadača a odosiela ich na server. Odoslané dáta slúžia ako východisko pre rôzne štatistické odvodzovanie relevantných stránok, prípadne ako nástroj pre korekciu plnoautomatických metód extrakcie dát.

2 Architektúra systému

Návrh systému vychádza z architektúry klient-server (pozri obr. 1). Klientská strana je implementovaná v podobe rozšírenia prehliadača Mozilla Firefox, ktorá kladie na používateľa minimálne nároky na inštaláciu softvéru. Klientská strana predovšetkým prezentuje webové stránky (s využitím zobrazovacích schopností prehliadača). V rámci klientskej strany dokáže používateľ označovať relevantné časti stránky a voliteľne k nim dodať metadáta (napr. názov atribútu, ktorý obsahuje označený text). Zároveň si vie zobrazovať anotácie, ktoré predstavujú výsledok aplikácie metód na strane servera, resp. agregované anotácie odvodené z označení ostatných používateľov. Existujúcu sadu anotácii je tiež možné použiť na inú stránku s rovnakým obsahom. (Množstvo produktových stránok z jedného portálu zachováva relatívne rovnakú štruktúru, kde je možné anotácie ľahko migrovať).



Obr. 1. Architektúra systému.

Serverovská strana predovšetkým prijíma používateľské anotácie (typicky v tvare usporiadanej štvorice [ID používateľa, URL stránky, XPath výraz identifikujúci anotovaný element a popis anotovaného atribútu.] Tieto údaje ukladá do úložiska používateľských anotácií. Na strane servera sa tiež spúšťajú konkrétne metódy pre extrakciu dát, ktorých výsledky je možné agregovať a spárovať s používateľskými anotáciami danej stránky.

3 Konkrétne použitie na príklade služby

Prototyp existujúcej architektúry sme experimentálne overili na extrakcii detailových stránok [1] reprezentujúcich dáta o filmoch. Zmienená metóda vyhľadáva rozdiely v HTML štruktúre dvojice stránok. Jej primárnym nedostatkom je neschopnosť identifikovať atribúty objektov, ktoré majú rovnakú hodnotu v oboch porovnávaných stránkach. Pomocou kolaboratívnej anotácie môže používateľ označiť nedetegované atribúty a tým nielen vylepšiť presnosť extrakcie danej dvojice stránok, ale zavedie tým do systému anotácie, ktoré je možné použiť aj pre ďalšie porovnávania dvojíc stránok.

4 Budúci výskum

V ďalšej práci je priestor na porovnanie a agregáciu viacerých metód na strane servera, ktoré navrhnú používateľovi predpokladanú sadu anotácií. Používateľ potom nebude nútený anotovať neznámu stránku nanovo, ale bude môcť jednoduchým spôsobom upraviť predanotované elementy.

Bibliografia

1. D. Maruščák, R. Novotný, P. Vojtáš: *Unsupervised structured web data and attribute value extraction*. In: Proceedings of Znalosti 2009
2. R. Baumgartner, S. Flesca, G. Gottlob: *Visual web information extraction*. VLDB Conference, 2001.
3. N. Kushmerick: *Wrapper induction: efficiency and expressiveness*. Artificial Intelligence, 118:15-68, 2000.

Winston: asistent dolovania dát*

Štefan Pero and Tomáš Horváth

Institute of Computer Science, Faculty of Science
Pavol Jozef Šafárik University in Košice
stepero@gmail.com, Tomas.Horvath@upjs.sk,

Dátová analýza resp. dolovanie dát patria do skupiny predmetov, ktoré môžu byť veľmi užitočné nielen pre študentov informatiky ale aj pre "neinformatikov" (biológovia, chemici, fyzici, ...), ktorí by tieto vedomosti vedeli dobre zužitkovať pri vyhodnocovaní svojich experimentov. Študenti týchto predmetov by mali mať možnosť "pohrať sa" s dátami, odskúšať rôzne metódy a techniky, prípadne naprogramovať niektoré známe (alebo aj vlastné) algoritmy.

Existuje niekoľko systémov dolovania dát, ktoré majú vizuálne rozhranie. Najznámejšími sú systémy Weka [5], Ferda [4], SumatraTT [1], Orange [3] a KDD package [2]. Tieto systémy sú však pre neinformatikov príliš zložité a teda môžu byť pre nich demotivujúce. Na druhej strane, k vytvoreniu vlastných metód je potrebné si naštudovať zložitejšie dátové štruktúry resp. skriptovacie jazyky týchto systémov. Toto však kladie vyššie nároky aj na študentov informatikov a odpútava ich pozornosť od samotných algoritmov.

Našim hlavným cieľom bolo vytvoriť systém na podporu výučby, ktorý je jednoduchý nielen z používateľského ale aj z vývojového hľadiska a teda študenti sa môžu lepšie sústrediť na samotné metódy a techniky dátovej analýzy a dolovania dát.

Aplikácia

Systém sa skladá z modulov, ktoré môžu byť zadelené do troch balíkov. Balík `DataSources` obsahuje moduly pre načítanie dát z externých zdrojov (Text, Excel, Access, MySQL). Balík `Preprocessing` obsahuje metódy predspracovania dát (napr. selekcia riadkov a stlpcov, spájanie stlpcov, ...). Algoritmy dolovania dát sú zahrnuté do balíka `DataMining`.

Vo vizuálnom prostredí si používateľ vyberá a spája rôzne moduly a nastavuje ich atribúty, ktoré sú definované v XML konfiguračných súboroch (každý modul má svoj vlastný konfiguračný subor). Moduly v balíku `DataSources` majú na vstupe surové dáta, na výstupe tabuľku. Ostatné moduly majú na vstupe aj na výstupe tabuľku, čo umožňuje dané moduly prepájať.

Jazykové nastavenia sú konfigurovateľné v XML súboroch, čo povoľuje pridávanie ďalších jazykov bez zásahu do zdrojového kódu aplikácie.

* Systém Winston je podporovaný projektmi VEGA 1/0131/09 a VVGS/UPJŠ/45/09-10.

Kvôli jednoduchosti sme aplikáciu navrhli tak, aby používateľ videl na obrazovke "všetko" a teda sa nemusel preklikávať cez ďalšie okná (obrázok 1).

V hlavnom menu sú tri položky: `Projects` obsahuje položky pre vytvorenie, uloženie a načítanie projektov. V položke `Settings` môžeme nastaviť jazyk aplikácie a spustiť sprievodcu pre pridávanie vlastných modulov. Pre pomoc slúži položka `Help`.

Na ľavej strane je stromová štruktúra balíkov a modulov, ktorých atribúty môžeme vidieť a nastavovať v časti Nastavenia, na pravej strane okna (kde sa dajú takisto nastaviť typy stĺpcov v dátach.

Pracovná plocha obsahuje dve záložky: Na záložke `Schema` si používateľ vytvára svoj projekt umiestnením modulov a ich prepájaním. V každom kroku používateľ môže nahliadnúť do dát a teda vidieť výstupy jednotlivých modulov v záložke `View`.

Správy z aplikácie (chybové hlásenia, stručné návody, atď.) sa vypisujú na konzolu.

Vývoj modulov

To, že vstupy aj výstupy sa reprezentujú v takej istej dátovej štruktúre (v tabuľke) značne zjednoduší prácu pri implementovaní vlastného modulu.

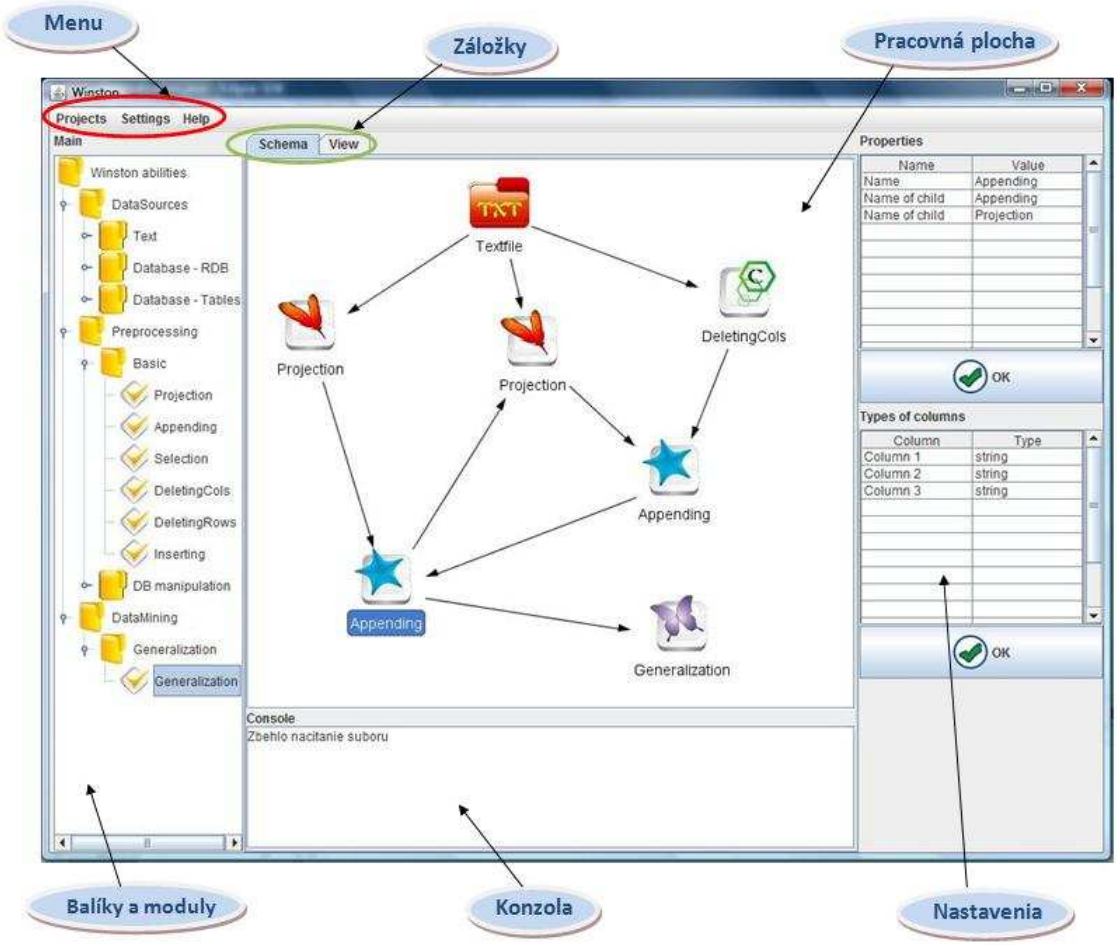
Práca každého modulu spočíva v načítaní vstupnej tabuľky, v načítaní atribútov modulu, v samotnom výpočte a v uložení výsledku do výstupnej tabuľky. K vytvoreniu vlastného modulu potrebujeme použiť nasledovnú štruktúru:

```
WinstonSerializableObject selectedObject;
// načítanie vstupných dát modulu
List<List> tabulka =
   selectedObject.getChildTab().getTable();
// načítanie atribútov modulu
Property[]props = selectedObject.getProps();
... algoritmus pre výpočet ...
// uloženie výstupu
selectedObject.setTable(List<List> newTab);
```

Vstupné parametre (atribúty) vytvoreného vlastného modulu potrebujeme ešte špecifikovať v XML konfiguračnom súbore.

Vlastný modul môžeme vložiť do systému aj pomocou zabudovaného sprievodcu, v ktorom je možnosť aj vytvorenia konfiguračného XML súboru.

Systém je implementovaný v jazyku Java.



Obrázok 1. Vzhľad aplikácie Winston.

Záver

Úspešné využitie Winstona vo výučbe a v menších data-mining projektoch očakávame hlavne kvôli jednoduchosti ovládania, ako aj možnosti programovania vlastných modulov v jazyku Java, ktorý sa vyučuje na väčšine vysokých škôl.

Systém Winston plánujeme nasadiť do výučby v nasledujúcom školskom roku. Systém sa neustále bude rozširovať novými modulmi.

Aplikáciu je možné stiahnuť na adrese `http://inka.ics.upjs.sk/winston/`

Referencie

1. P. Aubrecht, F. Železný, P. Mikšovský, O. Štepánková: *SumatraTT: towards a universal data preprocessor.* In: Cybernetics and Systems 2002. Vienna: Austrian Society for Cybernetics Studies. 2002 - ISBN 3 85206 160 1, 818–823.
2. P. Bednar, J. Paralic: *KDD package.* In: Proceedeings of the Znalosti 2003 Conference, Ostrava, February 2003, Czech Republic, ISBN 80-248-0229-5, 113–122.
3. J. Demsar, B. Zupan, G. Leban: *Orange: from experimental machine learning to interactive data mining.* White Paper (www.ailab.si/orange), Faculty of Computer and Information Science, University of Ljubljana, 2004.
4. M. Kováč, T. Kuchař, A. Kuzmin, M. Ralbovský: *Ferda, nové vizuální prostředí pro dobývání znalostí.* In: Znalosti 2006. Ostrava: TU VŠB, 2006 - ISBN 80-248-1001-8, 118-–129.
5. I.H. Witten, E. Frank: *Data mining: practical machine learning tools and techniques.* 2nd Edition. Morgan Kaufmann, San Francisco, 2005.