Preface

The 7th workshop **ITAT'07 – Information Technology – Applications and Theory** (`http://ics.upjs.sk/itat`) was held in Horský hotel Poľana, Slovak Ore Mountains (`http://www.polana.sk/`) located 1260 meter above the sea level in district Detva, Slovakia, in end of September 2007.

ITAT workshop is a place of meeting of people working in informatics from former Czechoslovakia (official languages for oral presentations are Czech, Slovak and Polish; proceedings papers can be also in English).

Emphasis is on exchange of information between participants, rather than make it highly selective. Workshop offers a first possibility for a student to make a public presentation and to discuss with the "elders". A big space is devoted to informal discussions (the place is traditionaly chosen at least 1000 meter above the sea level in a location not directly accessible by public transport).

Thematically workshop ranges from foundations of informatics, security, through data and semantic web to software engineering.
These proceedings consists of

- one invited lecture paper and
- 16 original scientific papers

All original scientific papers were refereed by at least two independent referees. There were 45 abstracts submitted.

The workshop was organized by Institute of Informatics of University of P.J. Šafárik in Košice; Institute of Computer Science of Academy of Sciences of the Czech Republic, Prague; Faculty of Mathematics and Physics, School of Computer Science, Charles University, Prague; Faculty of Informatics and Information Technology of the Slovak Technical University, Bratislava and Slovak Society for Artificial Intelligence.

Partial support has to be acknowledged from projects of the Program Information Society of the Thematic Program II of the National Research Program of the Czech Republic 1ET100300419 "Intelligent models, algorithms, methods and tools for the semantic web realization" and 1ET100300517 "Methods for intelligent systems and their application in data mining and natural language processing".

Peter Vojtáš

Program Committee

Peter Vojtáš, (chair), *Charles University, Prague, CZ*
Gabriela Andrejková, *University of P.J. Šafárik, Košice, SK*
Mária Bieliková, *Slovak University of Technology in Bratislava, SK*
Viliam Geffert, *University of P.J. Šafárik, Košice, SK*
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*
Věra Kůrková, *Institute of Computer Science, AS CR, Prague, CZ*
Pavol Návrat, *Slovak University of Technology in Bratislava, SK*
Ján Paralič, *Technical University in Košice, SK*
Dana Pardubská, *Comenius University, Bratislava, SK*
Martin Plátek, *Charles University, Prague, CZ*
Jaroslav Pokorný, *Charles University, Prague, CZ*
Karel Richta, *Czech Technical University, Prague, CZ*
Gabriel Semanišin, *University of P.J. Šafárik, Košice, SK*
Václav Snášel, *Technical University VŠB in Ostrava, CZ*
Vojtěch Svátek, *University of Economics in Prague, CZ*
Jirka Šíma, *Institute of Computer Science, AS CR, Prague, CZ*
Július Štuller, *Institute of Computer Science, AS CR, 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*
Katarína Mičková, *University of P.J. Šafárik, Košice, SK*
Róbert Novotný *University of P.J. Šafárik, Košice, SK*

Organization

ITAT 2007 Information Technologies – Applications and Theory was organized by
University of P.J. Šafárik, Košice, SK
Institute of Computer Science, AS CR, Prague, CZ
Faculty of Mathematics and Physics, Charles University, Prague, CZ
Faculty of Informatics and Information Technology of the Slovak Technical University, Bratislava, SK
Slovak Society for Artificial Intelligence, SK

Table of Contents

ITAT'07

Information Technology – Applications and Theory

INVITED PAPER

Funkční generativní popis a formální teorie překladů*

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 *V tomto příspěvku se zabýváme Funkčním generativním popisem češtiny (FGP) na pozadí formální teorie překladů a formální teorie jazyků. Klademe důraz na propojení formálních modelů s jejich jazykovou (lingvistickou) náplní. FGP byl rozvíjen od šedesátých let. Jednou z jeho formálních podob byla kompozice překladů pomocí pěti překladových gramatik. Zavedeme zde rozlišovací sílu takových systémů. Dále představíme nový formální model založený na překladových restartovacích automatech.*
Naším hlavním záměrem je představit metodu redukční analýzy pro FGP a její souvislost s formálním překladem. Z vlastností formalizované redukční analýzy lze vyčíst i požadavky na vlastnosti formálních rámců pro FGP. Proto v poslední části diskutujeme přímé uplatnění překladových restartovacích automatů v roli formálního rámce pro FGP.

1 Úvodní poznámky

Tento text by byl před pár lety zařazen do oblasti, které se říkalo algebraická lingvistika. Obsahuje definice a tvrzení matematicko-informatického charakteru, lingvistické příklady, lingvistické diskuze a jisté informaticko-lingvistické hypotézy. Navazuje na práce [8], [17], [11] a [13]. Při psaní jsme se snažili vyhýbat konfliktům mezi informatickými a lingvistickými zvyklostmi.

Funkční generativní popis je závislostní systém pro popis gramatiky češtiny, který se rozvíjí od 60. let 20. století, viz zejména [20], [22], [21]. Jeho základní charakteristikou je stratifikační přístup – popis jazyka je rozčleněn do několika rovin, každá z rovin je množinou zápisů vět (má svůj slovník a svou syntax). Zápis věty na nejnižší rovině si můžeme zjednodušeně představit jako řetězec slovních tvarů a interpunkce. Nejvyšší rovina zachycuje jazykový význam věty pomocí tektogramatické reprezentace (dále TR), jejímž základem je lineární zápis závislostního stromu, realizovaný vhodným uzávorkováním, viz [23], [18].

Formální rámec pro FGP byl původně navržen jako generativní zařízení pomocí sériově řazených zásobníkových automatů (odtud slůvko generativní v názvu), viz [23], [18]. Ten byl také počítačově implementován a postupně vyplňován generativními a překladovými instrukcemi, které nesly zakódované lingvistické poznatky, viz např. [15].

* Autoři jsou podporováni programem 'Informační společnost', projekt č. 1ET100300517.

Zde navazujeme na práci [17] z osmdesátých let, kde byl navržen formální rámec pro FGP pomocí (syntaxí řízených jednoduchých) překladových schémat, která byla interpretována dvojím způsobem – jednak jako generativní, jednak jako analytický systém, a která obsahovala časové a prostorové odhady složitosti jejich práce.

Z práce [17] přejímáme obecné požadavky na formální popisy přirozených jazyků L – každý takový systém by měl explicitně rozlišovat:

- množinu správně utvořených vět daného přirozeného jazyka L, kterou dále budeme označovat LC;
- formální jazyk LM reprezentující všechny možné tektogramatické (významové) reprezentace TR jazyka L, tedy množinu (korektních) tektogramatických stromových struktur daného jazyka;
- relaci SH mezi LC a LM popisující vztahy synonymie a homonymie;

Na relaci SH lze pohlížet jako na speciální případ formálního překladu. Nese informaci jak o svém vstupním jazyce (LM), tak o svém výstupním jazyce (LC).

Abychom porozuměli vývoji formální stránky FGP, budeme se nejprve věnovat základům formální teorie překladu (oddíl 2), kde připomeneme pojmy formálního překladu a překladových gramatik (oddíl 2.1) a skládání překladů (oddíl 2.2), na jejichž základech byl zaveden formální rámec FGP ze [17] – zde jej budeme značit F_{82} (oddíl 2.3).

V následující části se věnujeme metodě redukční analýzy. Redukční analýza, metoda, která modeluje práci lingvisty při rozboru vět přirozeného jazyka, je založena na postupném zjednodušování rozebírané věty – představíme ji v oddíle 3, kde vysvětlíme její základní principy (oddíl 3.1) a ukážme je na konrétních jazykových jevech (oddíl 3.2).

Naším hlavním záměrem je zařazení redukční analýzy do vhodného formálního rámce pro FGP – věnujeme mu část 4. Takovým rámcem mohou být restartovací automaty, viz např. [14], [11], zde oddíl 4.1. Ukážeme, že pomocí restartovacích automatů lze přirozeně splnit hlavní požadavky na formální rámce FGP, tedy požadavky na rozlišení množin LC a LM a relace SH i na zřetelné rozlišení jednotlivých

rovin FGP. Tento nový formální rámec FGP, který označujeme symbolem F_{07}, přibližujeme v oddílech 4.2 a 4.3.

Navíc nastíníme představu, jak pomocí restartovacích automatů a jejich vlastností vystavět bohatou taxonomii formálního překladu. Navážeme tak na bohatou literaturu, která restartovací automaty prezentuje jako flexibilní prostředek pro rozpoznávání formálních jazyků, viz např. [14]. Od automatů rozpoznávajících jazyky k automatům definující překlady přejdeme podobným obratem, jakým se přechází od generativních gramatik k překladovým gramatikám v části 2.

Poznámka: Restartovací automaty nechápeme jako model procedury rozpoznávání vět v našem vědomí či jako model těžko pozorovatelné procedury lidského porozumění větě přirozeného jazyka. Naší ambicí je pomocí restartovacích automatů přímočaře modelovat chování lingvisty, který provádí (na tabuli nebo na papíře) specializovanou činnost, redukční analýzu, která je důležitým přípravným i kontrolním krokem pro rozbor (české) věty.

Obraťme se nyní k věcné náplni FGP. Počet jazykových rovin, se kterými FGP pracuje, se v jednotlivých fázích jeho vývoje měnil. Zde pracujeme se čtyřmi rovinami, stejně jako např. Pražský závislostní korpus (PZK, viz zejména [1], je aplikací metodiky FGP na velký, elektronicky přístupný soubor českých vět). Jsou to následující roviny:

- rovina slovních tvarů a interpunkce (v PZK tzv. w-rovina), která zde odpovídá jazyku LC;
- morfematická rovina (v PZK m-rovina), která pro každý slovní tvar či interpunkční znaménko udává jeho lemma a morfologické kategorie;
- rovina povrchové syntaxe[3] (v PZK a-rovina) zachycující vztahy jako 'býti_subjektem', 'býti_objektem', 'býti_přívlastkem' ...;
- rovina jazykového významu, označovaná jako tektogramatická rovina (t-rovina v PZK), která specifikuje zejména tzv. hloubkové role, gramatémy a aktuální členění; zde tato rovina odpovídá formálnímu jazyku LM, viz výše.

2 Formální překlad a FGP

V úvodu jsme zmínili pojem SH-relace jako relace mezi jazykem korektních českých vět (LC) a jazykem jejich významů (LM), která pro jednotlivé věty rozlišuje různé významy a naopak pro jednotlivé významové zápisy rozlišuje jejich korektní vyjádření v LC. Na SH-relaci, které budeme též říkat *rozlišovací relace* a jejíž postupné vybudování je hlavním cílem FGP, pohlížíme jako na důležitý příklad (víceznačného) formálního překladu. Proto v této části nejprve vyložíme základy teorie (víceznačných) formálních překladů. Využijeme přitom pojmu překladové gramatiky.

Předpokládáme u čtenáře základní znalosti o formálních generativních gramatikách. Na nich budujeme pojmy pro překladové gramatiky a formální překlad.

Zavedeme Chomského hierarchii překladů a potom její podstatné zjemnění založené na skládání překladů. Podle této hierarchie budeme klasifikovat rozlišovací sílu formálních rámců pro FGP.

2.1 Překlad a překladové gramatiky

Nejprve zavedeme pojmy formální překlad a překladová gramatika. Překladovou gramatiku poté využijeme jako prostředek pro zadávání formálních překladů.

Terminologická poznámka: Abychom předcházeli nedorozuměním v lingvistické interpretaci navrhovaných pojmů, používáme často původní Chomského terminologii. Tedy např. *slovník* na místě abecedy, *slovo* nebo *symbol* jako prvek slovníku či *věta* nebo *řetěz* na místě slova ze současné terminologie obvyklé v teorii formálních jazyků.

(Formálním) překladem budememe nazývat množinu dvojic tvaru (u, v), kde u, v jsou řetězy symbolů nad daným slovníkem.

Pětici $G = (V_N, I, O, P, S)$ nazveme *překladovou gramatikou*, pokud

- V_N a I a také V_N a O jsou konečné disjunktní množiny symbolů (I a O disjuktní být nemusí),
- P je konečná množina pravidel tvaru $\alpha \to \beta$, kde $\alpha \in (V_N \cup I \cup O)^+$ obsahuje alespoň jeden symbol z V_N a $\beta \in (V_N \cup I \cup O)^*$,
- $S \in V_N$.

Množinu I nazveme *vstupním slovníkem*, množinu O *výstupním slovníkem* gramatiky G, dále množinu $IO = I \cup O$ nazveme *překladovým slovníkem* gramatiky G a jazyk $L_{IO}(G) = \{w \in IO^* \mid S \Rightarrow^* w\}$ *překladovým jazykem* gramatiky G.

Dále definujme *překlad podle gramatiky* G, $T(G)$, jako množinu dvojic vět vytvořených následujícím způsobem:

$T(G) = \{(u, v) \mid \exists\ w \in L_{IO}(G) : u$ vzniklo z w odstraněním všech symbolů nepatřících do I&v vzniklo z w odstraněním všech symbolů nepatřících do $O\}$.

[3] Poznamenejme, že v současné specifikaci FGP Petr Sgall, autor FGP, považuje tuto rovinu za redundantní.

Označme dále $L_I(G)$ množinu všech vět (řetězů), které lze získat z řetězů z $L_{IO}(G)$ odstraněním všech symbolů nepatřících do I. Tento jazyk nazýváme *vstupním jazykem překladu* podle gramatiky G.

Analogicky $L_O(G)$ označuje množinu všech vět, které lze získat z řetězů z $L_{IO}(G)$ odstraněním všech symbolů nepatřících do O. Tento jazyk nazýváme *výstupním jazykem překladu* podle gramatiky G.

Předchozí definice (formálního) překladu, založená na pojmu překladových gramatik, umožňuje mluvit o Chomského hierarchii (formálních) překladů a překladových jazyků. Máme tedy třídu překladů bez omezení, tedy typu 0, třídu kontextových překladů, tedy typu 1, třídu bezkontextových překladů, tedy typu 2, a třídu regulárních překladů, tedy typu 3. Podobně je to s překladovými jazyky.

Důležitá fakta: Není těžké nahlédnout, že třídy Chomského hierarchie překladů lze oddělit pomocí podobných příkladů jako třídy Chomského hierarchie jazyků. Stačí uvažovat překladové gramatiky, které tyto separující jazyky kopírují. Poznamenejme dále, že třídy regulárních překladů lze charakterizovat pomocí jednocestných konečných automatů s výstupem a třídy bezkontextových překladů pomocí zásobníkových automatů s výstupem. Povšimněme si ještě, že fakt, že nějaký překladový jazyk (překlad) je kontextový (typu 1), nezaručuje kontextovost příslušného vstupního jazyka, ani kontextovost příslušného výstupního jazyka. Jinak je tomu u zbylých tříd Chomského hierarchie jazyků, díky uzavřenosti těchto tříd na operaci homomorfismu (viz např. [24]).

2.2 Skládání překladů a Chomského hierarchie

Abychom mohli charakterizovat klasické modely FGP, budeme se zabývat speciálními typy složených překladů a ukážeme, že tvoří podstatné zjemnění Chomského hierarchie překladů. Toto zjemnění se bude týkat kontextových překladů. Uvnitř Chomského třídy kontextových překladů tak získáme nekonečnou hierarchii překladů v závislosti na počtu užitých překladových gramatik v kompozici.

Nechť T_1, T_2 je (uspořádaná) dvojice překladů. Symbolem $Comp(T_1, T_2)$ budeme označovat *složení překladů* T_1, T_2, definované vztahem $Comp(T_1, T_2) = \{(u_1, v_2) \mid \exists\ u_2 : (u_1, u_2) \in T_1, (u_2, v_2) \in T_2\}$. Je zřejmé, že $Comp(T_1, T_2)$ je také překlad. Jako $Comp(T_1, T_2, ..., T_i)$ označíme překlad, který vznikl postupným složením překladů $T_1, T_2, ..., T_i$ zleva doprava.

Dále budeme skládat pouze překlady dané bezkontextovými gramatikami. Bez újmy na obecnosti se omezíme na korektní gramatiky, t.j. na gramatiky, které generují bez chybných kroků. To znamená, že z každé větné formy takové gramatiky (odvoditelné ze startovacího symbolu), lze odvodit větu jejího překladového jazyka. Kdybychom uvažovali takovéto překlady bez dalsího omezení, byli bychom schopni již se třemi překlady simulovat libovolný Turingův stroj. Proto si povšimněte následujících omezujících podmínek.

Nechť $G_1, G_2, ..., G_k$ jsou korektní bezkontextové překladové gramatiky. Budeme říkat, že $P = Comp(T(G_1), T(G_2), ..., T(G_k))$ je *lineárně korektním překladem (LK-překladem) stupně* k, pokud splňuje následující podmínky:

- Existuje kladná konstanta j taková, že pro každé $1 \leq i \leq k$ platí, že $|u| \leq j|v|$, kde $(u, v) \in Comp(T(G_1), ..., T(G_i))$.
  To zhruba řečeno znamená, že překlady definované jednotlivými gramatikami k-stupňového překladu P na výstupu zachovávají délku vstupu (až na multiplikativní konstantu).
- $P = Comp(T(G_1), T(G_2), ..., T(G_k))$ je zleva korektní, což znamení, že výstupní jazyk $T(G_i)$ je podmnožinou vstupního jazyka $T(G_{i+1})$ pro $1 \leq i < k$.
  To znamená, že vše co, G_1 vygeneruje, P opravdu postupně přeloží. Na základě předchozí podmínky přitom při překladu zkracuje pouze nepodstatně.

Nechť k je celé kladné číslo. Jako LK_k označíme třídu LK-překladů stupně k, tedy složení k překladů splňující předchozí podmínky. Nepříliš složitou transformací výsledků a postupů z [18], [17] dostáváme následující fakta.

Důležitá fakta: LK_1 je třídou bezkontextových překladů a třídy LK_k tvoří nekonečnou hierarchii separovaných tříd překladů v rámci Chomského třídy 1. To znamená, že pro libovolné přirozené k je LK_k vlastní podmnožinou LK_{k+1}.

Této škály využijeme v následujícím oddíle pro charakterizaci překladové, tedy rozlišovací síly FGP.

V této části jsme zavedli Chomského hierarchii překladů a její výrazné a důležité zjemnění. Pod pojmem *rozlišovací síla překladu* P (či zařízení, které ho definuje) rozumíme zařazení P do některé z tříd této (nebo jiné) hierarchie. V následujících oddílech se budeme zabývat rozlišovací silou překladů, definovaných některými modely FGP. Přirozeně nám jde o to, aby formální rámec pro FGP měl pokud možno optimální rozlišovací sílu vzhledem k FGP odladitelnosti a lingvistickou adekvátnosti i vzhledem k efektivitě algoritmů SH-analýzy a SH-syntézy (viz oddíl 4.2). Tento požadavek nás vede ke studiu

rozlišovací síly a dalších relevantních vlastností potenciálních formálních rámců pro FGP.

2.3 Formální rámec klasického FGP a síla jeho rozlišení

Pomocí symbolu F_{82} značíme formální rámec Funkčního generativního popisu ze 70. let, viz [17]. Takový model lze charakterizovat jako LK-překlad stupně 5, tedy jako lineárně korektní překlad daný pěti překladovými bezkontextovými gramatikami, které označujeme jako G_1, G_2, G_3, G_4 a G_5. Množina větných významů LM_{82} (tektogramatická rovina) popsaná pomocí F_{82} odpovídá vstupnímu jazyku gramatiky G_1. Množina korektních českých vět LC_{82} popsaná pomocí F_{82} odpovídá výstupnímu jazyku gramatiky G_5. Rozlišovací relace ve F_{82} je tedy překladem mezi řetězy z LC_{82} a LM_{82}.

Prvky z LM_{82} jsou dobře uzávorkovanými výrazy, které reprezentují tzv. projektivní závislostní stromy, viz [17].

Gramatika G_1 simuluje generativní gramatiku, její vstupní jazyk se rovná jejímu výstupnímu jazyku. Gramatiky G_2, G_3, G_4 postupně překládají vstupní (tektogramatické) struktury na struktury odpovídající dalším rovinám FGP. Všechny tyto překlady transformují projektivní závislostní stromy opět na projektivní závislostní stromy. Teprve překlad pomocí gramatiky G_5 odstraňuje závorky, aby převedl projektivní závislostní strom do obvyklé typografické formy české věty.

Z dnešního pohledu a v dnešní terminologii lze výše zmíněné reprezentace projektivních závislostních stromů chápat jako XML-struktury. Gramatiky G_2, G_3, G_4 pak podle této terminologie realizují jisté XML-transformace, které vzhledem k zařazení do třídy překladů LK_5, nemůžeme považovat za příliš složité.

Ve zbytku tohoto oddílu zmíníme důvody, proč budeme uvažovat nové formální rámce pro FGP.

Přirozeně chceme, aby nově navrhovaný formální rámec pro FGP umožňoval modelovat a kontrolovat nejen všechny podstatné jevy, které modeloval F_{82}, ale i významné jevy, které původní model nezachycoval.

Jedním z takových jevů je například český (povrchový) větný slovosled, který připouští neprojektivity vysokého stupně, viz [5]. Netriviální a důležité je zachycení jeho vztahu k valenci, k hloubkovému slovosledu a aktuálnímu členění.

Metodu přímočarého zachycení vztahu valence a slovosledu pomocí redukcí nazýváme *redukční analýzou (pro* FGD*)*. Cílem následující části je přiblížit čtenáři práci lingvisty provádějícího redukční analýzu českých vět, a tuto činnost formálně modelovat. K modelování využijeme konceptu (překladových) restartovacích automatů. Následující část vychází z přímé spolupráce autorů s Petrem Sgallem, autorem FGD, jejímž výsledkem byl článek [8].

3 Redukční analýza pro FGP

Tento oddíl je věnován redukční analýze (RA) přirozeného jazyka – napřed vysvětlíme základní principy RA (oddíl 3.1), pak ukážeme použití RA při zpracování dvou konkrétních jazykových jevů: zachycení slovosledu a určování závislostí ve větě (oddíl 3.2).

3.1 Principy redukční analýzy

Redukční analýza modeluje metodu práce lingvisty při rozboru vět přirozeného jazyka. Dovoluje zkoumat závislostní strukturu věty a přitom přiměřeně zachytit její slovosledné varianty. To je nezbytné zejména pro jazyky s relativně volným slovosledem, jako je čeština, kde závislostní struktura a slovosled souvisí pouze volně.

Redukční analýza je založena na postupném zjednodušování (vstupního) zápisu zpracovávané věty – krok RA spočívá ve vypuštění vždy alespoň jednoho slova (symbolu) vstupního zápisu.[4] Tento postup umožňuje určit jazykové závislosti, viz [7].

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 tvořena:

- řídícím slovesem (predikátem) nezávislé slovesné klauze a jeho valenčními doplněními, nebo
- řídícím substantivem nezávislé nominativní klauze a jeho valenčními doplněními, např. *Názory čtenářů*, nebo
- řídícím substantivem nezávislé vokativní klauze, např. *Jano!*, nebo
- řídícím slovem nezávislé citoslovečné klauze, např. *Pozor!*

Při každém kroku RA je požadováno splnění následujících principů:

- zachování syntaktické správnosti zpracovávané věty (tedy ze syntakticky správné věty vznikne po každém redukčním kroku opět syntakticky správná věta);

[4] Zde pro zjednodušení pracujeme pouze s operací *vypouštění (delete)*, v [7] se předpokládá též operace *přepisování (rewrite)*.

– zachování lemmatu a množiny morfologických kategorií u nevypuštěných slov;
– zachování významu jednotlivých slov, reprezentovaného např. jako položka ve (valenčním) slovníku;
– zachování tzv. významové úplnosti věty; v tomto textu požadujeme zachování úplného valenčního rámce (t.j. aktantů a obligatorních volných doplnění specifikovaných u daného slova ve valenčním slovníku) všech lexikálních položek, které nesou valenční informaci.

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.

Tyto požadavky ilustrujeme rozborem příkladu v následujícím oddíle.

3.2 Redukční analýza v příkladech – slovosled a závislostní struktura

Čeština je jazyk s poměrně vysokou mírou volnosti slovosledu. To přirozeně neznamená, že by věty lišící se slovosledem byly zcela synonymní, neboť slovosled v češtině odráží aktuální členění (a některé věty lišící se aktuálním členěním nelze považovat za synonymní, neboť mohou mít různé pravdivostní podmínky). Změna slovosledu však s sebou nemusí nést změnu závislostní struktury (a nemusí být narušena gramatičnost věty). Slovosled a závislostní struktura spolu souvisejí jen volně, závislostní relace (binární vztah mezi řídící lexikální jednotkou a jednotkou závislou / rozvíjející) není možné určit pouze ze slovosledných pozic těchto jednotek. Metoda RA umožňuje odvodit závislostní vztahy z možných pořadí redukce jednotlivých slov ve ve větě, viz [7].

Slovosled. Vyjasnění role slovosledu v jazycích s volným slovosledem je věnováno velká pozornost, viz např. [2], [5], [4], [3], abychom citovali alespoň nejnovější publikace pro češtinu.

Zopakujme dva základní principy pro tektogramatickou reprezentaci věty, která ve FGP zachycuje význam, viz [22], [2]:

– Slovosled na TR (hloubkový slovosled) spolu se speciálními kategoriemi odráží aktuální členění věty – vyjadřuje škálu výpovědní dynamičnosti věty (a tedy se může lišit od povrchového slovosledu).
– Princip projektivity pro TR umožňuje adekvátně zachytit škálu výpovědní dynamičnosti věty povrchové věty (projektivní i neprojektivní)

Než se podíváme, jakým způsobem se tyto principy promítají do zpracování konkrétní věty, zavedeme konvenci, která nám později umožní zachytit několikarovinnou reprezentaci věty jako vstup restartovacího automatu.

Konvence: Zápis věty na všech rovinách reprezentujeme (např. na vstupní pásce automatu) jako řetězec čtveřic, kde každá čtveřice odpovídá nějakému vstupnímu slovu. Na obrázcích zachycujících příkladové věty jsou tyto čtveřice pro přehlednost na samostaných řádcích. První položka čtveřice vždy zachycuje slovní tvar (příp. interpunkční znaménko), druhá položka zachycuje (morfologické) lemma a morfologické kategorie daného slova, třetí položka jeho (povrchovou) syntaktickou funkci a konečně čtvrtá položka udává pro plnovýznamová slova[5] příslušné tektogramatické informace – tektogramatické lemma, funktor, identifikátor valenčního rámce, gramatémy a informaci o aktuálním členění. Navíc uvažujeme čtveřice, které reprezentují aktuální elipsy – ty mají neprázdné pouze tektogramatické položky, viz Poznámku za příkladem (1).

Každé vstupní slovo je reprezentováno jednou nebo dvěma čtveřicemi: (i) jednou čtveřicí, pokud povrchový slovosled souhlasí se slovosledem hloubkovým nebo pokud dané slovo nemá samostatnou položku v TR (tedy nejde o plnovýznamové slovo), příp. má neprázdnou pouze tektogramatickou položku (jde o elipsu); (ii) dvěma čtveřicemi, pokud se povrchový slovosled liší od slovosledu hloubkového (jedna čtveřice popisuje slovní tvar, lemma, morfologické kategorie a syntaktickou funkci; druhá čtveřice zachycuje tektogramatické informace).

Uspořádání čtveřic odráží slovosled na jednotlivých rovinách popisu – sledujeme-li první až třetí položky čtveřic, dostaneme povrchový slovosled, poslední položky čtveřic udávají hloubkový slovosled.

Nyní se tedy podívejme, jakým způsobem se slovosledné principy stanovené FGD promítají do zpracování konkrétní věty (řídíme se zde pravidly i notací, která jsou použita v PZK, viz [1] a zejména [12]).

Příklad:

(1) *Našeho Karla plánujeme poslat na příští rok do Anglie.* (viz [22], p. 241, rozvinuto)
≈ Co se týká našeho Karla, tak toho plánujeme poslat na příští rok do Anglie.

Vlastní jméno *Karel* tvoří kontrastivní topik věty (tfa = 'c', tedy jde o známou informaci, kontextově

[5] Význam pomocných a funkčních slov, jako jsou např. předložky, pomocná slovesa nebo podřadící spojky, je zachycen pomocí funktorů a gramatémů u jejich řídících slov.

Našeho	*můj*.PSMS4	Atr	
Karla	*Karel*.NNMS4	Obj	
			[my].ACT_t
plánujeme	*plánovat*.VB-P-	Pred	*plánovat*.PRED.Fr1_f
			Karel.PAT_c
			my.APP_f
			[my].ACT_t
poslat	*poslat*.Vf- - -	Obj	*poslat*.PAT.Fr2_f
na	*na*.RR- - 4	AuxP	
příští	*příští*.AA4IS	Atr	
rok	*rok*.NNIS4	Adv	*rok*.THL_f
			příští.RSTR_f
do	*do*.RR- - 2	AuxP	
Anglie	*Anglie*.NNFS2	Adv	*Anglie*.DIR3.basic_f
.	..Z: - - -	AuxK	

Obrázek 1. Vstupní reprezentace věty (1).

zapojenou, která je v textu strukturována jako kontrastivní, s přízvukem v mluvené podobě výpovědi). Podle pravidel aktuálního členění je kontrastivní topik v povrchové realizaci věty umístěn na začátek věty (i s členy na něm závislými, zde přívlastek *náš*), mezi něj a jeho řídící větný člen (sloveso *poslat*, viz druhá část příkladu) se dostávají další členy věty, což způsobuje neprojektivitu povrchové struktury.

Teoretický předpoklad projektivity TR si vynucuje změny v hloubkovém slovosledu – příslušná položka *Karel*.PAT_c je v TR přemístěna na pozici před řídící slovo *t-poslat*.PRED.Fr1_f (podle přijatých pravidel před nekontrastivní topik, zde nevyjádřeným aktorem [my].ACT_t, viz [12]). Zároveň se přesouvají i všechny členy na ní závislé.

Další pravidlo určuje, že přívlastky mají vyšší výpovědní dynamiku než lexikální jednotka, kterou rozvíjejí, jejich pozice v hloubkovém slovosledu je tedy za řídícím substantivem.

Poznámka: Uvědomme si ještě, že TR je z definice významově úplná, jsou v ní doplněny aktuální elipsy (obligatorní položky, které nejsou v povrchové realizaci věty vyjádřeny, posluchač/čtenář si je doplňuje na základě kontextu). Takové doplněné elipsy jsou reprezentovány čtveřicí, která obsahuje pouze zjednodušenou tektogramatickou informaci, sestávající pouze z funktoru,[6] viz i bod (2b) dále (ve větě (1) např. dvakrát aktor [my], který odpovídá nevyjádřenému podmětu).

Redukční analýza věty (1) má tedy na vstupu řetězec z obrázku 1.

Určování závislostní struktury. Nyní se podívejme na redukční fázi analýzy, která umožňuje určit závislosti mezi jednotlivými slovy ve větě (podrobněji viz [7]).

Redukcí/vypuštěním zde rozumíme odstranění všech symbolů, které reprezentují jedno vstupní slovo, viz Konvenci výše. Jde tedy v prototypickém případě, kdy povrchový slovosled souhlasí se slovosledem hloubkovým, o odstranění jedné čtveřice symbolů; obdobně v případě, kdy jde o funkční či pomocné slovo (a tedy má prázdnou tektogramatickou položku), nebo kdy jde o elipsu (a naopak tektogramatická položka je jediná neprázdná). Pokud se povrchový a hloubkový slovosled liší věty liší, jsou při redukčním kroku odstraněny obě čtveřice pro vstupní slovo.

Závislosti se zpracovávají dvěma způsoby, podle funkce daného slova či skupiny slov ve větě:

1. Volná doplnění, která nezaplňují valenční doplnění žádné lexikální jednotky ve větě, jsou postupně, v libovolném pořadí, vypouštěna.
2. Redukční komponenty (tvořené slovy, která musí být redukována v jednom kroku RA, aby se vyloučily negramatické, tedy neúplné tektogramatické reprezentace, viz první Poznámku níže) jsou zpracovávány v závislosti na své funkci ve větě:
   (a) Pokud 'hlava' redukční komponenty nenaplňuje valenční požadavky (např. přívlastek nebo přívlastková věta), je celá redukční komponenta vypuštěna.
   (b) Pokud 'hlava' redukční komponenty naplňuje valenční požadavky, potom:
   (i) vše až na 'hlavu' redukční komponenty je vypuštěno;
   (ii) položka reprezentující 'hlavu' redukční komponenty je zjednodušena – všechny symboly příslušné čtveřice až na funktor jsou vypuštěny; výsledkem takového zjednodušení je položka, kterou lze interpretovat jako reprezentaci nulové lexikální realizace obligatorní položky, viz též Poznámku výše.

Poznámka: Redukční komponenty jsou typicky tvořeny buď jedním větným členem (např. předložkové skupiny, jako *do Anglie*), nebo lexikální položkou vyžadující valenční doplnění a těmito doplněními, jako *[my]*.ACT *pošleme Karla*.PAT *domů*.DIR3, viz [7].

Poznámka: Uvedený postup redukcí odráží skutečnost, že určité slovo je závislé na jiném slově, pokud je vždy vypuštěno dříve; slova jsou navzájem nezávislá, pokud je lze vypouštět v libovolném pořadí, viz [7].

Vraťme se tedy k větě (1) – podíváme se na postupné určování její závislostní struktury.

[6] Funktor specifikuje syntakticko-sémantický vztah mezi příslušnou položkou a její řídící lexikální jednotkou.

→	(3 kroky)		
Karla	*Karel*.NNMS4	Obj	[my].ACT_t
plánujeme	*plánovat*.VB-P-	Pred	*plánovat*.PRED.Fr1_f
			Karel.PAT_c
			[my].ACT_t
poslat	*poslat*.Vf- - -	Obj	*poslat*.PAT.Fr2_f
do	*do*.RR- - 2	AuxP	
Anglie	*Anglie*.NNFS2	Adv	*Anglie*.DIR3.basic_f
.	..Z: - - -	AuxK	

Obrázek 2. Reprezentace věty (1) po třech krocích RA.

Příklad:

(1) *Našeho Karla plánujeme poslat na příští rok do Anglie.*

V prvních krocích můžeme v libovolném pořadí redukovat slova *našeho* a *příští* – výsledná reprezentace je gramatická, tato slova nezaplňují žádné valenční požadavky a jsou vzájemně nezávislá. V dalším kroku lze redukovat předložkovou skupinu *na rok* – jde opět o volné doplnění, jeho vypuštěním neporušíme úplnost věty.

Je zřejmé, že stejně dobře jsme mohli redukovat slovo *našeho* až po vypuštění předložkové skupiny *na rok* (jde o vzájemně nezávislá slova); naproti tomu předložkovou skupinu *na rok* nelze redukovat dříve než slovo *příští*, je tedy jeho řídícím členem.

**Našeho Karla plánujeme poslat na příští do Anglie.*

Po třech krocích RA tedy získáme zjednodušenou větu (2), jejíž reprezentace je na obrázku 2.

(2) *Karla plánujeme poslat do Anglie.*

Tuto větu již nelze zjednodušit prostým vypouštěním, aniž bychom porušili principy RA (konkrétně zachování úplnosti věty):

**Karla plánujeme poslat.*
**Karla plánujeme do Anglie.*
**Karla poslat do Anglie.*
**Plánujeme poslat do Anglie.*

Podíváme-li se na lexikální obsazení věty (2), zjistíme, že sloveso *poslat* má (v tomto významu) tři valenční doplnění, 'kdo', aktor (ACT), posílá 'koho', patient (PAT) 'kam', směr (DIR3); sloveso *plánovat* má dvě valenční doplnění, 'kdo, aktor (ACT), plánuje 'co/inf', patient (PAT). Sloveso v infinitivním tvaru *poslat* tedy tvoří se svými doplněními [my].ACT, *Karla*.PAT a *do Anglie*.DIR3 redukční komponentu,

→			
			[my].ACT_t
plánujeme	*plánovat*.VB-P-	Pred	*plánovat*.PRED.Fr1_f
			[ ].PAT_f
.	..Z: - - -	AuxK	

Obrázek 3. Reprezentace věty (1) po redukci komponenty [my] *Karla poslat do Anglie.*

jejíž je 'hlavou'; tato 'hlava' pak zaplňuje valenční požadavek slovesa *plánovat*.

Postupujeme tedy podle bodu 2b: (i) vypustíme valenční doplnění slovesa *poslat*, tedy čtveřice reprezentující vstupní slova [my].ACT, *Karla*.PAT a *do Anglie*.DIR3; (ii) čtveřici reprezentující sloveso *poslat*.PAT zjednodušíme – vypustíme všechny symboly až na funktor PAT. Výsledná reprezentace takto zjednodušené věty je na obrázku 3.

Tato reprezentace tvoří základní predikační strukturu (jde o řídící sloveso nezávislé slovesné klauze a jeho valenční doplnění), redukční analýza tedy končí. Věta (1) je postupně zredukována tak, že každou redukcí získáme správně utvořenou věta jazyka, přičemž ta poslední nelze dále redukovat.

Redukční analýzu, kterou jsme zde ukázali na příkladu konkrétní věty, lze formálně modelovat pomocí restartovacích automatů, kterým je věnován následující oddíl.

4 Restartovací automaty a FGP

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

V prvním oddíle jsou zavedeny *jednoduché restartovací automaty (*sRL*-automaty)*, které slouží jako výchozí pojem. Tam jsou zavedena i tzv. metainstrukce, sloužící k přehlednému zápisu restartovacích automatů a jejich postupnému budování po nezávislých krocích. Tento model restartovacího automatu jsme vybrali hlavně proto, že lze upravit tak, aby byl schopen stavět z vypouštěných symbolů závislostní struktury.

Druhý oddíl zavádí automaty typu F_{07} jako speciální případ sRL-automatů. Jejich pomocí se postupně zavádějí následující základní pojmy pro automat M_{07} typu F_{07}:

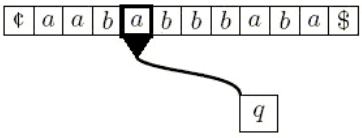
- *charakteristický jazyk* $L_C(M_{07})$;
- čtyři formální jazyky reprezentující jednotlivé roviny FGP, přičemž jazyky reprezentující první a poslední rovinu odpovídají po řadě jazyku správně utvořených vět LC a jazyku tektogramatických reprezentací LM z kapitoly 1;
- *překladový jazyk* $L_{03}(M_{07})$;

– *charakteristickou relaci* SH(M_{07});
– SH-*syntézu*, která představuje FGP jako generativní zařízení, tedy v jeho původní úloze;
– SH-*analýzu*, která plní úlohu syntakticko-sémantické analýzy realizované na základě FGP.

4.1 t-sRL-automat

Nejprve popíšeme neformálně restartovací automat, který využíváme k modelování RA. Tento oddíl je adaptací kapitoly definic z [11].

Jednoduchý RL-*automat* (sRL-*automat*) M je (v obecném případě) nedeterministický stroj s konečně stavovou řídící jednotkou (množinu stavů označujeme Q), konečným charakteristickým slovníkem Σ a pracovní hlavou se schopností číst a pracovat právě s jedním symbolem, která operuje na pružné pásce (seznamu) ohraničené levou zarážkou ¢ a pravou zarážkou \$, viz obrázek 4.



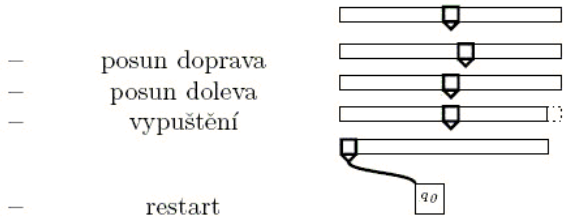
Obrázek 4. Restartovací automat.

Pro vstupní větu $w \in \Sigma^*$ je vstupním zápisem na pásce ¢w\$. M začíná výpočet ve svém startovním stavu q_0 s hlavou na levém konci pásky, ve výhledu má levou zarážkou ¢. Automat M může provádět následující kroky (viz též obrázek 5):

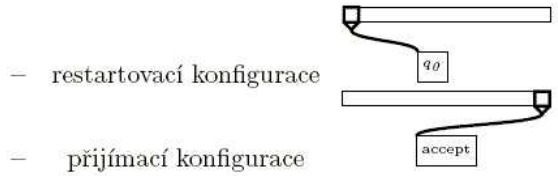
– *posuny doprava* a *posuny doleva* – posouvají hlavu o jednu pozici doprava, resp. doleva a určují následný stav;
– *vypouštěcí kroky* – vypouštějí symbol pod pracovní hlavou, určují následný stav a posunují hlavu na pravého souseda vypuštěného symbolu.

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 6). Požadujeme, aby M před každým restartem a 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:



Obrázek 5. Operace.



Obrázek 6. Významné konfigurace.

– Q je konečná množina stavů;
– Σ je konečný slovník (charakteristický slovník);
– ¢, \$ jsou zarážky, {¢, \$} nepatří do Σ;
– q_0 z Q je startovacím stavem;
– δ je přechodovou relací ≈ konečnou množinou instrukcí tvaru: $(q, a) \rightarrow {}_M(p, Op)$, kde q, p jsou stavy z Q, a je symbol ze Σ, a Op je operace, kde 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, token) z β. Konfigurace tvaru q_0¢w\$ je nazývána *restartovací konfigurací*.

Povšimněme si, že libovolný výpočet M se skládá z fází. Fáze, zvaná *cyklus*, začíná restartovací konfigurací, hlava provádí posuny a vypouští podle 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. Používáme notaci $u \vdash {}^c{}_M\ v$ pro označení cyklu M, který začíná v restartovací konfiguraci q_0¢u\$ a končí restartovací konfigurací q_0¢v\$; relace $\vdash {}^{c^*}{}_M$ je reflexivním a transitivním uzávěrem $\vdash {}^c{}_M$. Pár RS(M) := $((\Sigma)^*, \vdash {}^c{}_M)$ nazýváme *redukčním systémem* indukovaným automatem M.

Řetěz $w \in \Sigma^*$ je *přijímán* pomocí M, pokud existuje přijímací výpočet startující z počáteční konfigurace $q_0 ¢ w \$$. Jako *charakteristický jazyk* $L_C(M)$ označujeme jazyk sestávající ze všech řetězů přijímaných pomocí M; říkáme, že M *rozpoznává (přijímá) jazyk* $L_C(M)$. Jako $S_C(M)$ označujeme *jednoduchý jazyk* přijímaný pomocí M, který sestává z řetězů, které M přijímá pomocí výpočtů bez restartu (koncovkami). Snadno lze nahlédnout, že $S_C(M)$ je regulárním podjazykem jazyka $L_C(M)$.

Takový sRL-automat M, který používá nejvýše t vypuštění v jednom cyklu ($t \geq 1$) a zároveň libovolný řetěz z $S_C(M)$ 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 (vypouštění). 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 část 3 o redukční analýze. Dalším podstatným rysem je skutečnost, že se omezujeme na práci s modelem, pro který existuje přirozené číslo t dominující počtu vypuštění v cyklu a velikosti přijímaného neredukovatelného řetězu. Velikost čísla t dává (hrubý) horní odhad stupně valence příslušného jazyka.

Podobně jako v [11] můžeme t-sRL-automaty popisovat pomocí *metainstrukcí* ve formě

$$(¢ \cdot E_0, a_1, E_1, a_2, E_2, \ldots, E_{s-1}, a_s, E_s \cdot \$) ,$$

$1 \leq s \leq t$, kde

- $E_0, E_1, \ldots, E_s$ 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 $i = 0, \ldots, s$. Na druhou stranu, pokud w lze rozložit do takového tvaru, pak jeden z nich je vybrán nedeterministicky a restartovací konfigurace $q_0 ¢ w \$$ je redukována do 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).

Příklad: Nechť $t \geq 1$ a nechť $L_{R_t} = \{ c_0 w c_1 w c_2 \cdots c_{t-1} w \mid w \in \{a, b\}^* \}$. Pro tento jazyk sestrojíme t-RL-automat M_t se slovníkem $\Sigma_t = \{c_0, c_1, \ldots, c_{t-1}\} \cup \Sigma_0$, kde $\Sigma_0 = \{a, b\}$, pomocí následující sady metainstrukcí:

$(¢ c_0, a, {\Sigma_0}^* \cdot c_1, a, {\Sigma_0}^* \cdot c_2, \ldots, {\Sigma_0}^* \cdot c_{t-1}, a, {\Sigma_0}^* \cdot \$)$,
$(¢ c_0, b, {\Sigma_0}^* \cdot c_1, b, {\Sigma_0}^* \cdot c_2, \ldots, {\Sigma_0}^* \cdot c_{t-1}, b, {\Sigma_0}^* \cdot \$)$,
$(¢ c_0 c_1 \ldots c_{t-1} \$, \mathsf{Accept})$.

Snadno nahlédneme, že platí $L(M_t) = L_{R_t}$.

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

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

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

Je celkem zřejmé, že každý t-sRL-automat zachovává chybu a že všechny deterministické t-sRL-automaty zachovávají korektnost. Na druhé straně lze snadno zkonstruovat příklady nedeterministických t-sRL-automatů nezachovávajících korektnost.

4.2 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_{07}.

Jako *automat typu* F_{07} budeme označovat takový t-sRL-automat M_{07}, který zachovává korektnost a jehož charakteristický slovník Σ je sestaven ze čtyř podslovníků $\Sigma_0, \ldots, \Sigma_3$. Požadujeme přitom, aby v každém cyklu M_{07} byl vypuštěn alespoň jeden symbol z každého jeho podslovníku $\Sigma_0, \ldots, \Sigma_3$.

Automat M_{07} je obvykle popisován pomocí metainstrukcí.

Poznámky: Každý z podslovníků $\Sigma_0, \ldots, \Sigma_3$ reprezentuje jednu rovinu FGP, viz oddíl 1.

Zachovávání korektnosti u M_{07} má zajišťovat věrnou simulaci lingvisty provádějícího redukční analýzu. Podobně jako lingvista by automat M_{07} neměl během redukční analýzy dělat chyby. Pokud je tomu jinak, je něco špatně, např. je špatně navržen charakteristický jazyk. Tuto situaci je možné vylepšit například přidáním nových metakategorií (tedy symbolů z jiné roviny než nulté) nebo nových metain-

strukcí. Vlastnosti, které zaručují zachování korektnosti, lze přitom formulovat tak, aby je bylo možno automaticky testovat (a tedy s výhodou využívat při ladění FGP).

Důležitý požadavek na vypuštění nejméně jednoho symbolu z každého podslovníku v každém cyklu (což odpovídá redukci vždy celé čtveřice, příp. obou čtveřic reprezentujících (povrchové) slovo v jediném kroku RA, viz oddíl 3.2) reprezentuje požadavek na zobecněnou lexikalizaci FGP.

Pojmy $L_C(M_{07})$, *charakteristického jazyka* automatu M_{07}, a $S_C(M_{07})$, *jednoduchého jazyka* automatu M_{07}, přebíráme z předchozího oddílu. Odvodíme z nich několik dalších důležitých pojmů.

Jako první zavádíme pojem *redukčního systému* daného automatem M_{07}, budeme značit $\mathsf{RS}(M_{07})$, jako následující pětici :

$$\mathsf{RS}(M_{07}) := (\Sigma^*, \vdash^{c}_{M_{07}}, S_C(M_{07}), \Sigma_0, \ldots, \Sigma_3)$$

Redukční systém (automatu M_{07}) formalizuje pojem redukční analýzy FGP algebraickým, neprocedurálním způsobem, tedy ve stylu tzv. analytických modelů, viz [9]. Povšimněme si, že pro každý řetězec symbolů $w \in \Sigma^*$ platí, že $w \in L_C(M_{07})$, právě když pro nějaký řetěz $v \in S_C(M_{07})$ platí $w \vdash^{c^*}_{M_{07}} v$ (tedy věta w patří do charakteristického jazyka automatu M_{07}, právě když ji lze redukovat na větu v z jednoduchého jazyka automatu M_{07}).

Poznámka: Redukční systémy jsou z lingvistického hlediska důležité. Dovolují formulovat vlastnosti společné pro konečné překlady a jazyky (typicky nám jde o konečná pozorování o přirozených jazycích) a nekonečné formální překlady a jazyky. Pomocí takových vlastností se snažíme matematickým způsobem budovat užitečné taxonomie, převážně ve formě hierarchií. Zajímavé jsou zde zejména společné vlastnosti konečných a nekonečných překladů, které by charakterizovaly jistá zjemnění Chomského hierarchie a případně vypovídaly něco zajímavého z hlediska časové či prostorové složitosti překladů. Většina vlastností restartovacích automatů, které studujeme, jsou omezení vypozorovaná z vlastností konečných redukčních systémů vytvořených na základě redukční analýzy českých vět. To nám připadá v době velkého rozvoje jazykových korpusů zajímavé a užitečné.

Jazykem j-té roviny rozpoznávané pomocí M_{07}, kde $0 \leq j \leq 3$, je množina všech vět (řetězů), které lze získat z řetězů z $L_C(M_{07})$ odstraněním všech symbolů nepatřících do Σ_j. Tento jazyk označíme $L_j(M_{07})$. Je zřejmé, že $L_0(M_{07})$ reprezentuje množinu správně utvořených vět LC definovanou pomocí M_{07}; podobně je $L_3(M_{07})$ je jazykem tektogrammatických reprezentací LM definovaným pomocí M_{07} (viz část 1, jde o obdobu definic z oddílu 2.1).

Překladovým jazykem rozpoznávaným pomocí M_{07} je množina všech řetězů, které lze získat z řetězů z $L_C(M_{07})$ odstraněním všech symbolů nepatřících do Σ_0 nebo do Σ_3. Tento jazyk označíme $L_{03}(M_{07})$. (Jde tedy o paralelu k překladovému jazyku $L_{IO}(G)$ z oddílu 2.1.)

Nyní můžeme definovat *charakteristickou relaci* $\mathsf{SH}(M_{07})$ danou automatem M_{07}:

$\mathsf{SH}(M_{07}) = \{(u, y) \mid u \in L_0(M_{07})$ & $y \in L_3(M_{07})$ & $\exists w \in L_C(M_{07})$ takové, že a u vzniklo z w vypuštěním symbolů nepatřících do Σ_0 a y vzniklo z w vypuštěním symbolů nepatřících do $\Sigma_3\}$.

Vidíme, že charakteristická relace je formálním překladem. V připomeňme si, že v každém cyklu se při redukční analýze vypouští z každé roviny zhruba stejně symbolů (až na malou multiplikativní konstantu). Odtud lze vydedukovat, že jazyky $L_{03}(M_{07})$, $L_0(M_{07})$ a $L_3(M_{07})$ patří do třídy 1 (kontextových) Chomského hierarchie jazyků. Tedy $\mathsf{SH}(M_{07})$ je kontextovým překladem z kontextového jazyka do kontextového jazyka.

Poznámka: Charakteristická relace reprezentuje důležité vztahy v popisu přirozeného jazyka, vztahy synonymie a homonymie (víceznačnosti) přirozeného jazyka L. Jinak řečeno, reprezentuje překlad z jazyka správně utvořených vět LC do tektogramatického jazyka LM a naopak. Z charakteristické relace odvodíme zbývající důležité pojmy, pojmy analýzy a syntézy.

Pro libovolné $y \in L_3(M_{07})$ (tedy tektogramatickou reprezentaci z LM) zavádíme SH-*syntézu pomocí* M_{07} jako množinu dvojic (u, y) patřících do $\mathsf{SH}(M_{07})$. Formálně:

$$\mathit{synt}\text{-}\mathsf{SH}(M_{07}, y) = \{(u, y) \mid (u, y) \in \mathsf{SH}(M_{07})\}$$

SH-syntéza spojuje tektogramatickou reprezentaci (řetěz y z LM) se všemi jí odpovídajícími větami u patřícími do LC. Tento pojem dovoluje kontrolovat synonymii a její stupeň, tak jak ji zachycuje automat M_{07}. Lingvistickým záměrem je postupně snižovat stupeň synonymie zachycené automatem M_{07} pomocí postupného zjemňování M_{07}.

Na závěr zavádíme duální pojem k SH-syntéze, pojem SH-*analýzy řetězu u pomocí* M_{07} :

$$\mathit{anal}\text{-}\mathsf{SH}(M_{07}, u) = \{(u, y) \mid (u, y) \in \mathsf{SH}(M)\}$$

SH-analýza vrací pro danou větu u všechny její tektogramatické reprezentace, tedy dovoluje kontrolovat víceznačnost jednotlivých povrchových vět. Tento pojem dává formální definici úkolu úplné syntakticko-sémantické analýzy zadané pomocí M_{07}. Lingvistickým úkolem je postupně zjemňovat M_{07} se zvláštním ohledem na adekvátní zachycení víceznačnosti vět.

V tomto oddíle jsme představili F_{07} jako vhodný formální model pro redukční analýzu se čtyřmi rovinami symbolů. Ukázali jsme také, že F_{07} splňuje základní požadavky na formální rámec pro FGP. V následujím oddíle naznačíme možnosti pro studium jeho síly. Ukážeme, že formální rámec F_{07} dává řadu možností jak formulovat omezení (či naopak rozvolnění) jejich rozlišovací síly.

4.3 Poznámky o rozlišovací síle a složitosti automatů typu F_{07}

Automat M_{07} typu F_{07} je možno budovat postupně podle složitosti lingvistických jevů. M_{07} je stavbou, která roste postupným přidáváním metainstrukcí. Připomeňme ještě, že M_{07} má dodržovat korektnost. Zhruba naznačíme, jak je možné postupovat. Nejprve se vkládají přijímací metainstrukce. Ty automaticky zachovávají korektnost. Pak se postupně přidávají redukční instrukce, které lze použít při redukční analýze dříve než instrukce již vložené. Pokud nějaké použití instrukce není korektní, buď opravíme uvažovanou instrukci nebo přidáme jednu nebo více instrukcí, které upraví její nekorektní kroky na kroky korektní (nekorektními kroky jsou míněny kroky vedoucí k zastavení automatu v nepřijímací konfiguraci v případě, kdy jiná větev výpočtu nad stejnou větou vede k jejímu přijetí). Lze předpokládat, že takovýto model je možné ladit snadněji než model ze sedmdesátých let. Budeme se snažit tento předpoklad podpořit exaktním (složitostním) výzkumem.

Velikost či momentální velikost M_{07} můžeme měřit pomocí počtu metainstrukcí. Domníváme se, že nebude těžké ukázat na základě práce [11], že rozlišovací síla automatů typu F_{07} roste s povoleným počtem metainstrukcí.

Očekává se, že M_{07} se bude postupně budovat od metainstrukcí pro jednoduché jazykové jevy a že postupně se bude přecházet k metainstrukcím pro složitější jevy. Zmiňme zde možnosti, jak lze charakterizovat sílu M_{07} v jednotlivých fázích jeho naplnění a jak formulovat požadavky na jeho očekávanou sílu, s ohledem na plánované zpracování jazykových jevů.

Prvním typem charakteristiky je omezení na počet vypouštění pomocí jednotlivých metainstrukcí. Z práce [11] vyplývá, že rozlišovací síla automatů typu F_{07} s tímto omezením roste. Podobné je to s dalšími omezeními pro restartovací automaty, jako je např. vícenásobná monotonie, které byly studovány pro formální jazyky. Stačí se v tomto smyslu ohlédnout na práce uveřejněné již dříve na této konferenci (ITAT), jako např. [19].

Tímto způsobem získáváme řadu pojmů, které nám umožní formulovat, které třídy jazykových jevů jsou již do automatu reprezentujícího FGP zahrnuty a na kterých je třeba ještě pracovat. Momentálně je např. zjevné, že je ještě dosti práce na složitějších slovosledných a interpunkčních jevech, a potom zejména na koordinačních a apozičních jazykových konstrukcích, které jsme v tomto textu nechávali zcela stranou.

5 Výhledy a závěrečné poznámky

V blízkém budoucnu plánujeme modelovat pomocí restartovacích automatů i SH-syntézu a SH-analýzu. Snadno nahlédneme, že tyto automaty budou muset používat nejen operaci vypouštění, ale i operace přepisování a připisování (i když jen omezeným způsobem). Budeme se snažit popisovat postupy, jak přecházet od automatů typu F_{07} popisujících redukční analýzu k automatům modelující SH-syntézu a hlavně SH-analýzu. Domníváme se, že formální rámec pro FGP zachycující redukční analýzu usnadní i automatizovaný přechod od FGP k dalším závažným aplikacím, jako je například vyhledávání a lokalizace gramatických chyb v českých textech.

Čtenář zběhlý v počítačové lingvistice si jistě již dříve povšiml, že použití formálního překladu jako aparátu pro vyjádření rozlišovací relace je typické pro FGP. Domníváme se, že tato volba Petra Sgalla a jeho spolupracovníků ze šedesátých let byla šťastná. Umožňuje průhledným způsobem přenášet lingvistické problémy do teorie automatů a formálních jazyků a exaktním způsobem formulovat jejich řešení. Naší snahou je i nadále této techniky využívat. Nebudeme přitom zanedbávat ani techniky zachycující rozlišovací relace složitěji strukturovanými prostředky.

Reference

1. Hajič J., Complex Corpus Annotation: The Prague Dependency Treebank. Insight into Slovak and Czech Corpus Linguistic, M. Šimková, ed., Veda Bratislava, Slovakia, 2005, 54–73
2. Hajičová E., Partee B., and Sgall P., Topic-Focus Articulation, Tripartite Structures, and Semantic Content. Kluwer, Dordrecht (1998)
3. Hajičová E., K některým otázkám závislostní gramatiky. Slovo a slovesnost, 67, 2006, 3–26
4. Havelka J., Projectivity in Totally Ordered Rooted Trees. The Prague Bulletin of Mathematical Linguistics, 84, 2005, 13–30
5. Holan T., Kuboň V., Oliva K., and Plátek M., On Complexity of Word Order. In Les grammaires de dépendance – Traitement automatique des langues, S. Kahane, ed., 41, 1, 2000, 273–300
6. Lopatková M., Plátek M., and Kuboň V., Závislostní redukční analýza přirozených jazyků. In Proceedings of ITAT 2004, P. Vojtáš, ed., 2004, 165–176

7. Lopatková M., Plátek, M., and Kuboň V., Modeling Syntax of Free Word-Order Languages: Dependency Analysis by Reduction. In TSD 2005, Proceedings, V. Matoušek, P. Mautner, T. Pavelka, eds., LNCS, 3658, 2005, 140–147
8. Lopatková, M., Plátek M., and Sgall P., Towards a Formal Model for Functional Generative Description, Analysis by Reduction and Restarting Automata. The Prague Bulletin of Mathematical Linguistics, 87, 2007, 7–26
9. Marcus S., Algebraické modely jazyka. Academia, Praha, 1969
10. Mel'čuk I.A., Dependency Syntax: Theory and Practice. State University of New York Press, Albany, 1998
11. Messerschmidt H., Mráz F., Otto F., and Plátek M., Correctness Preservation and Complexity of Simple RL-Automata. In Lecture Notes in Computer Science, 4094, 2006, 162–172
12. Mikulová M., Bémová A., Hajič J., Hajičová E., Havelka J., Kolářová V., Lopatková M., Pajas P., Panevová J., Razímová M., Sgall P., Štěpánek J., Urešová Z., Veselá K., Žabokrtský Z., and Kučová L., Anotace na tektogramatické rovině Pražského závislostního korpusu. Anotátorská příručka. Technical report, Prague, MFF UK, 2005
13. Mráz F., Otto F., and Plátek M., On the Gap-Complexity of Simple RL-Automata. In Developments in Language Theory, Proceedings of DLT 2006, O. Ibarra and Z. Dang, eds., LNCS, 4036, Springer, 2006, 83–94
14. Otto F., 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, 2003
15. Panevová J., From Tectogrammatics to Morphemics. Transducing Components of Functional Generative Description 1. In Explizite Beschreibung der Sprache und automatische Textbearbeitung, IV., Prague, MFF UK, 1979
16. Petkevič V., A New Formal Specification of Underlying Structure. Theoretical Linguistics, 21, 1, 1995
17. Plátek M., Composition of Translation with D-trees. COLING'82, 1982, 313–318
18. Plátek M. and Sgall P., A Scale of Context-Sensitive Languages: Applications to Natural Language. Information and Control, 38, 1, 1978, 1–20
19. Plátek M., Mráz F., Otto F., and Lopatková M.: O roztržitosti a volnosti slovosledu pomocí restartovacích automatů. In Proceedings of ITAT 2005, P. Vojtáš, ed., 2005, 145–156
20. Sgall P.: Generativní popis jazyka a česká deklinace, Academia, Praha, 1967
21. Sgall P., Underlying Structure of Sentences and its Relations to Semantics In Festschrift für Viktor Jul'evič Rozencvejg zum 80. Geburtstag, Wiener Slawistischer Almanach, T. Reuther, ed., 33, 1992, 273–282
22. Sgall P., Hajičová E., and Panevová J., The Meaning of the Sentence in Its Semantic and Pragmatic Aspects, J. Mey, ed., Dordrecht: Reidel and Prague: Academia, 1986
23. Sgall P., Nebeský L., Goralčíková A., and Hajičová E., A Functional Approach to Syntax in Generative Description of Language. New York, 1969
24. Sgall P. a kolektiv: Úvod do syntaxe a sémantiky, Některé nové směry v teoretické lingvistice. Academia, Praha, 1986

ITAT'07

Information Technology – Applications and Theory

ORIGINAL SCIENTIFIC PAPERS

Optimizing XQuery/XSLT programs using backward analysis*

David Bednárek

Department of Software Engineering
Faculty of Mathematics and Physics, Charles University Prague
david.bednarek@mff.cuni.cz

Abstract. *The formal semantics of XQuery 1.0, XPath 2.0, and XSLT 2.0 relies on sequences as the only compound data type assigned to all variables and expressions throughout the execution. Since concatenation of sequences is not commutative, it is difficult to reorder the operations during static optimization or to parallelize their execution. In this paper, we suggest enriching the execution model with several types that carry less information than sequences and whose operations are easier to optimize and to evaluate. Various XQuery operators and XSLT instructions access their operands in different ways; backward propagation of the usage information allows using weaker types for variables and expressions. Thus, the execution cost of an XQuery/XSLT program may be reduced in both time and space and more sophisticated methods of static optimization or parallelization become available.*

1 Introduction

With the introduction of XPath 2.0 [11] and its companions XQuery 1.0 [12] and XSLT 2.0 [13], the universe in which their expressions are evaluated was enlarged. Instead of separated universes of input node sets, output tree fragments, and atomic values, universal sequences were introduced that may contain arbitrary mixture of atomic values and input, temporal, or output nodes. Although this change unified the semantics and improved the expressing power of the languages, it brought difficulties to the implementation. Moreover, real-life queries and transformations usually do not intensively mix values of different kinds; therefore, the cost of the implementation is concentrated in the handling of corner cases that are quite rare.

The unified sequence-based universe does not fit into the mathematical models used to reason about XML traversals, queries, and transformations. The most often used models [5] of XPath expressions, including monadic second order (MSO) logic [4], automata [2] or attribute grammars [1], are well-suited for queries that select sets of nodes or transform unordered trees.

Of course, newer methods dealing with XML Query are aware of mixed sequences. For instance, in [7], an extensive theoretical framework is developed. Nevertheless, there is a gap between the older, sophisticated methods focused on a fragment of the language [9, 8], and the unlimited universal frameworks. Therefore, older optimized evaluation methods are not directly applicable to the current languages. This problem is not only academical – 18 months after the first Call for Implementations by W3C [10], only a few of production-quality XPath or XSLT engines was succesfully upgraded to the version 2.0 of these languages.

In this paper, we suggest to return back to the node-set based evaluation whenever possible. We define a hierarchy of *modes* of evaluation, ranging from the canonical sequence-based evaluation through node sets to atomic values. In most parts of real-life programs, the required evaluation mode may be reduced to node-set mode or atomic-value-sequence mode. Thus, the evaluation cost is diminished and some of the algorithms devised for XPath 1.0 may be applied to portions of XPath 2.0 programs.

Special care is dedicated to two difficult areas of XQuery and XSLT: Constructors and effective boolean value. Constructors are expressions that create new nodes with new identity; thus, violating the functional character of the language. Effective boolean value is a concept of implicit conversion, distorted by its backward compatibility.

Since the XSLT and XQuery are related languages and the translation from XSLT to XQuery is known (see [3]), the approach applies also to XSLT. Of course, it may be also applied to stand-alone XPath 2.0 environments. In this paper, we describe only the core of the languages and rely on the conversion of other features to the core, as shown for instance in [6].

The rest of the paper is organized as follows: First, the evaluation modes are defined and the behavior of important operators with respect to the modes is shown. As a justification of the approach, the section 4 defines the semantics of selected XQuery operators, based on the set-theoretic building blocks defined in section 3. This alternative definition provides a background for the evaluation modes and also for further optimizations not described in this paper.

* Project of the program "Information society" of the Thematic program II of the National research program of the Czech Republic, No. 1ET100300419

mode	sequential	set	navigation	children	atomic	boolean
seq-NA	•	•	•	•	•	•
seq-TA	•	•		•	•	•
seq-A	•	•			•	•
set-A		•			•	
seq-N	•	•	•	•		•
set-N		•	•	•		•
XEBV						•

Table 1. Evaluation modes and their capabilities.

2 Evaluation modes

Table 1 shows the list of evaluation modes defined by our method together with their capabilities. XQuery operators access various facets of their operands:

Sequential enumeration of the items in the stored order
Set membership test
Navigation to ancestors, descendants or siblings; comparisons based on node identity and document order
Children – enumeration of children and their descendants
Atomic value
Boolean – extraction of effective boolean value

The seven defined modes are partially ordered by the inclusion of their capabilities; the resulting hierarchy is shown in Fig. 1. The behavior of some operators make use of *suprema* wrt. the partial order.

XEBV mode (*extended effective boolean value*) is a five-valued logic representing the five conditions distinguished in the definition of *effective boolean value* [11]:

empty – empty sequence
false – singleton atomic value that converts to false
true – singleton atomic value that converts to true
node – non-empty sequence beginning with node
error – sequence of two or more items, beginning with atomic value

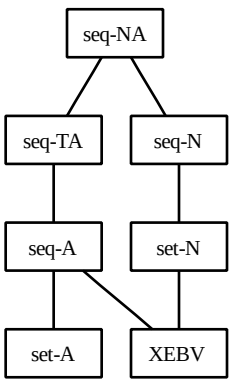


Fig. 1. Mode hierarchy.

The conversion of a sequence to boolean value is now performed in two steps: Conversion of the sequence to the XEBV mode and final conversion to boolean (when *empty* is converted to *false* and *node* is converted to *true*). The five values are selected so that the conversion to the XEBV mode may be distributed inside the sequence (comma) operator, as shown in Table 2. After the distribution, the evaluation of comma operators is trivial; moreover, their operands may be evaluated in reduced mode since they are immediately converted to the XEBV mode.

a, b	empty	false	true	node	error
empty	empty	false	true	node	error
false	false	error	error	error	error
true	true	error	error	error	error
node	node	node	node	node	node
error	error	error	error	error	error

Table 2. Sequence operator in XEBV mode.

Table 3 shows the mode behavior for important operators: When an operator is evaluated in mode m, the table defines the modes in which the operands shall be evaluated. The `for` and filter expressions form scopes in which an explicit or implicit (. or *current*) variable is defined. Therefore, their mode behavior depends on the supreme of modes (M_{var} or $M_{current}$, respectively) in which the variable is evaluated. The behavior is described in tables 4 and 5.

operator in mode m	operand	operand mode
`element` constructor	content	seq-TA
`for`-expression	`return`-expression	m
`for`-expression	`in`-expression	see Table 4
filter-expression	condition	XEBV
filter-expression	filtered expression	see Table 5
axis step	context	set-N
atomization		seq-A
sequence	both	m

Table 3. Mode behavior of important operators.

3 Value models

The XML Path language defines *primitive atomic* types, such as `xs:integer` or `xs:string`. Our method is not limited to a particular collection of types, although some of these types play special role in the method. We will denote the sets (*carriers*) of their values $W_{integer}$, W_{string}, etc. We assume total orders $<_{integer}$, $<_{string}$, etc. on the carriers.

m	$sup(M_{var})$	m_{in}
seq-*any*	seq-NA	seq-NA
seq-*any*	seq-TA, XEBV	seq-TA
seq-*any*	seq-A, set-A	seq-A
seq-*any*	seq-N, set-N	seq-N
seq-*any*	$\emptyset$	seq-TA
set-*any*	seq-NA	seq-NA
set-*any*	seq-TA, XEBV	seq-TA
set-*any*	seq-A, set-A	set-A
set-*any*	seq-N, set-N	set-N
set-*any*	$\emptyset$	seq-TA
XEBV	seq-NA	seq-NA
XEBV	seq-TA, XEBV	seq-TA
XEBV	seq-A, set-A	seq-A
XEBV	seq-N, set-N	set-N
XEBV	$\emptyset$	seq-TA

`for` X_{var} `in` X_{in} `return` X_{ret}

Table 4. Determination of `in`-expression mode.

m	$sup(M_{current})$	m_{expr}
seq-NA	*any*	seq-NA
seq-TA	seq-NA, seq-N, set-N	seq-NA
seq-TA	seq-TA, seq-A, set-A, XEBV	seq-TA
seq-TA	$\emptyset$	seq-TA
seq-A	seq-NA, seq-N, set-N	seq-NA
seq-A	seq-TA, XEBV	seq-TA
seq-A	seq-A, set-A	seq-A
seq-A	$\emptyset$	seq-A
set-A	seq-NA, seq-N, set-N	seq-NA
set-A	seq-TA, seq-A, set-A, XEBV	set-A
set-A	$\emptyset$	set-A
seq-N	seq-NA	seq-NA
seq-N	seq-N, set-N	seq-N
seq-N	seq-TA, seq-A, set-A, XEBV	seq-TA
seq-N	$\emptyset$	seq-N
set-N	seq-NA	seq-NA
set-N	seq-N, set-N	set-N
set-N	seq-TA, seq-A, set-A, XEBV	seq-TA
set-N	$\emptyset$	set-N
XEBV	seq-NA	seq-NA
XEBV	seq-N, set-N	set-N
XEBV	seq-TA, seq-A, set-A, XEBV	seq-TA
XEBV	$\emptyset$	XEBV

X_{expr} `[` X_{filter} `]`

Table 5. Determination of filtered expression mode.

For simplicity, we assume that the carriers are pairwise disjoint. Then, we can define *universal atomic carrier* as

$$W_{atomic} = W_{integer} \cup W_{string} \cup \ldots$$

The union of total orders on particular types form a partial order $<_{atomic}$ on W_{atomic}.

To keep track of the execution of an XML Query or XSLT program (or the evaluation of an XML Path expression), an alphabet of *markers* W_{marker} is used. Markers are attached to positions in the program (or the expression), i.e. to functions, templates, XSLT instructions and some XML Path subexpressions. For a particular program, the set of used markers is finite, proportional to the size of the program. To cover all programs, the alphabet W_{marker} must be infinite.

The same alphabet of markers is also used to label edges of input documents (see below).

In the most general case, an XML Path expression produces a sequence of items from some universe W. A sequence s may be modeled by a mapping $s : S_s \rightarrow W$, where S_s is a finite totally ordered set. In our approach, S_s is a finite set of words over an alphabet Σ, ordered by the lexicographic order induced by a total order on Σ.

This approach is further extended to encode unranked ordered trees: Assume a prefixless language $L_{edge} \subset \Sigma^*$, i.e. $a, ab \in L_{edge} \Rightarrow b = \lambda$. To encode a tree, each edge is labeled with a word in L_{edge} so that sibling edges have different labels and that the order of edges in the tree corresponds to the lexicographic order of their labels. Then, each node in the tree is uniquely identified by the concatenation of edge labels along the path from the root to the node; we will call the resulting word the *branch* of the node. Moreover, the set of branches of all nodes is sufficient to encode the tree.

The *branch alphabet* Σ contains atomic types, execution markers, and two special symbols – *brackets*:

$$\Sigma = W_{atomic} \cup W_{marker} \cup \{[,]\}$$

Of course, we assume that the three sets are pairwise disjoint. The brackets are used to define the prefixless language of edge labels: L_{edge} is the smallest subset of Σ^* such that

$$[(W_{atomic} \cup W_{marker} \cup L_{edge})^*] \subset L_{edge}$$

Thus, each edge label is a sequence of atomic values, markers, or other edge labels, enclosed in a pair of brackets.

Consequently, *branch* is a word in

$$L_{branch} = L_{edge}^*$$

A finite set $B \subset L_{branch}$ encodes a tree, if and only if, for each a, b,

$$ab \in B \wedge b \in L_{edge} \Rightarrow a \in B$$

When encoding sequences as mappings, the following domain is used:

$$L_{seq} = (W_{atomic} \cup W_{marker} \cup L_{edge})^*$$

To distinguish trees created during the execution of a program, the same domain is used under new name $L_{tree} = L_{seq}$.

The elements and attributes of output documents and temporary trees have names from the *name universe* W_{name}. In most cases, these names are explicitly stated in the program, as a part of XML Query *constructors* or corresponding XSLT instructions. In general, the names may be arbitrary strings (or QName's) generated during the execution.

The *property universe* corresponds to the properties that may be attached to a node:

$$W_{prop} = W_{atomic} \cup W_{name} \cup (W_{name} \times W_{atomic})$$

For simplicity, we assume that the three unioned sets are pairwise disjoint. The first set is used to store the values of text nodes or stand-alone atomic values, the second to store the names of element nodes, and the third to pair attributes with their values.

XML Query and XSLT languages allow construction of new nodes using constructor expressions or instructions. The presence of constructors violates functional character of the language: Each invocation of a constructor generates a node with new identity; thus, two invocations return different values even if invoked with the same arguments. Thus, constructors and functions containing them are not pure functions. Fortunately, this impurity may be removed by the following method:

As defined in the language specification, each invocation of a constructor creates a new tree by creating a new root node and copying the nodes of the content. This tree will never be altered or joined with other nodes during its lifetime. Therefore, to handle the identity of nodes, it is sufficient to handle the identity of trees – whenever a constructor is invoked, the whole tree is assigned a unique string from L_{tree} – the *tree identifier*. Tree identifiers are explicitly generated and distributed through the execution of the program using additional parameters so that each constructor receives a tree identifier corresponding to the dynamic invocation path that lead from the start of the program to the invocation of the constructor.

Furthermore, the node identity may play its role only when the constructed tree is navigated by path expressions, compared, or manipulated by set operators. In the most typical case, constructed trees are used only in the contents of other constructors (or as the final output of the transformation); in this case, their identity is not relevant and their tree identifiers are not needed. Our backward analysis method is able to determine this situation and remove the request for tree identifiers. After the analysis, there are two kinds of constructors, *plain* and *unique*. Only the latter receive the tree identifiers and assign them to the trees.

4 Canonic sequence model

The formal semantics of the XPath states that each expression evaluates to a *sequence* containing zero or more *items*. Each item is either an *atomic value* or a *node*. While an atomic value consist only of a single value, a node carries more information:

- Several properties attached to the node (kind, name, typed value, string value etc. as defined by the XML standard)
- Node identity
- The whole tree which contains this node

In a sequence, several nodes may belong to the same tree and it will not be wise to replicate the tree with the nodes; therefore, our mathematical model consists of two parts: An *environment*, containing the trees, and a *value*, containing the sequence of nodes (as references to the trees) and atomic values. The environment will be modeled by a *TBV-set* while the sequence will be represented by an *SATB-set*, defined below. Note that all the sets are actually partial mappings; nevertheless, we are rather using the word *set* since they will be often manipulated by set operations.

TBV-set is a partial mapping

$$X_{TBV} : L_{tree} \times L_{branch} \to W_{prop}$$

TBV-set is called *complete* if, for each t, the set

$$B(t, X_{TBV}) = \{b | \langle t, b, v \rangle \in X_{TBV}\}$$

encodes a tree.

SATB-set is a partial mapping

$$X_{SA} : L_{seq} \to (L_{tree} \times L_{branch} \cup W_{atomic})$$

assuming that $L_{tree} \times L_{branch}$ and W_{atomic} are disjoint.

In this section, the semantics of XPath and XQuery operators and XSLT instructions will be expressed in the SATB- and TBV-set model of sequences. First, we define the model of the dynamic context as required by the XPath language; then, we will show the semantics of important operators.

The dynamic context consists of the following components:

- The *context item* is represented by the pair $\langle cur, env \rangle$, where

  $$cur \in W_{atomic} \cup (L_{tree} \times L_{branch})$$

  is the atomic value of the context item or the context node (identified by its tree identifier and branch) and *env* is a TBV-set (the environment) containing (at least) the tree to which the context node belongs.

– The *context position* and *context size* are represented by the pair $\langle pos, size\rangle$ of atomic values.
– Each *variable value* is represented by a pair $\langle val, env\rangle$, where val is a SATB-set storing the sequence and env is a TBV-set containing the environment of trees for this sequence. For variables declared by the `for`, `some`, and `every` expressions, val is always a singleton.

Note that XML Path language defines more components of the dynamic context like available documents or current date. We do not explicitly model these components because they are constant during the execution of the program.

Based on the presented representation, an XPath function `f($`X_1`,...,$`X_n`)` will be modeled by the pair of functions

$$val_f(t, C, P, X_1, \ldots, X_n)$$

$$env_f(t, C, P, X_1, \ldots, X_n)$$

where $t \in L_{tree}$ is the tree identifier and

$$C = \langle cur, env\rangle, P = \langle pos, size\rangle, X_i = \langle val_i, env_i\rangle$$

The val_f and env_f functions return the value sequence and the tree environment, respectively.

The semantics of *concatenation operator*

`f($`X_1`,...,$`X_n`) =`
`  g($`X_1`,...,$`X_n`), h($`X_1`,...,$`X_n`)`

is expressed by the following equations:

$$\begin{aligned}
&val_f(t, C, P, X_1, \ldots, X_n) = \\
&\quad = \{\langle m_1 s_g, v_g\rangle | \\
&\qquad \langle s_g, v_g\rangle \in val_g(t \cdot m_1, C, P, X_1, \ldots, X_n)\} \cup \\
&\quad \cup \{\langle m_2 s_h, v_h\rangle | \\
&\qquad \langle s_h, v_h\rangle \in val_h(t \cdot m_2, C, P, X_1, \ldots, X_n)\} \\
&env_f(t, C, P, X_1, \ldots, X_n) = \\
&\quad = env_g(t \cdot m_1, C, P, X_1, \ldots, X_n) \cup \\
&\quad \cup env_h(t \cdot m_2, C, P, X_1, \ldots, X_n)
\end{aligned}$$

where m_1, m_2 are markers such that $m_1 < m_2$.

The semantics of a *for expression* like

`f($`X_1`,...,$`X_n`) =`
`  for $`X_{n+1}` in g($`X_1`,...,$`X_n`)`
`    return h($`X_1`,...,$`X_n`,$`X_{n+1}`)`

is expressed by the following equations:

$$\begin{aligned}
&val_f(t, C, P, X_1, \ldots, X_n) = \\
&\quad = \{\langle [s_g][s_h], v_h\rangle | \\
&\qquad \langle s_g, v_g\rangle \in val_g(t \cdot m, C, P, X_1, \ldots, X_n) \wedge \\
&\qquad \wedge \langle s_h, v_h\rangle \in val_h(t[s_g], C, P, X_1, \ldots, X_n, \\
&\qquad\qquad \langle\{\langle \lambda, v_g\rangle\}, env_g(t \cdot m, C, P, X_1, \ldots, X_n)\rangle)\} \\
&env_f(t, C, P, X_1, \ldots, X_n) = \\
&\quad = \bigcup\{env_h(t[s_g], C, P, X_1, \ldots, X_n, \\
&\qquad\qquad \langle\{\langle \lambda, v_g\rangle\}, env_g(t \cdot m, C, P, X_1, \ldots, X_n)\rangle) | \\
&\qquad\qquad \langle s_g, v_g\rangle \in val_g(t \cdot m, C, P, X_1, \ldots, X_n)\}
\end{aligned}$$

where $m \in W_{marker}$.

The semantics of a *computed element constructor*

`f($`X_1`,...,$`X_n`) =`
`  element` $QName$ `{ g($`X_1`,...,$`X_n`) }`

is expressed by the following equations:

$$\begin{aligned}
&val_f(t, C, P, X_1, \ldots, X_n) = \\
&\quad = \{\langle \lambda, \langle t, \lambda\rangle\rangle\} \\
&env_f(t, C, P, X_1, \ldots, X_n) = \\
&\quad = \{\langle t, \lambda, QName\rangle\} \cup \\
&\quad \cup \{\langle t, [s_g]b_2, v_g\rangle | \\
&\qquad \langle s_g, \langle t_g, b_1\rangle\rangle \in val_g(t, C, P, X_1, \ldots, X_n) \wedge \\
&\qquad \wedge \langle t_g, b_1 b_2, v_g\rangle \in env_g(t, C, P, X_1, \ldots, X_n)\}
\end{aligned}$$

5 Conclusion

We have presented a method of backward analysis and transformation that replaces the original XML Path operators and XSLT instructions by operators of reduced strength. This replacement is justified by backward analysis of the data flow.

For XQuery/XSLT programs containing no recursive functions (as well as stand-alone XPath queries), the application of the method is straightforward. However, under presence of recursive functions or templates, a more sophisticated algorithm is required. Currently, a precise exponential algorithm and a polynomial conservative approximation are known. Our future work will be focused on finding an effective precise algorithm for the recursive case.

There are two levels of benefits brought by the strength reduction. First, when the transformed program is interpreted directly in the canonical order, the reduction diminishes both the time and space consumption demands. Second, some of the reduced operators are commutative; thus, transformations based on reordering of their operands become available.

We have defined an alternative way of expressing the values, based on sets of string tuples. In this representation, sequence, for, and other major operators are replaced by string manipulations and set unions. Such a representation opens new options in analysis and optimization, including the transformation to

joins and parallel execution of the query, which is subject of our future work.

References

1. Bloem R. and Engelfriet J., A Comparison of Tree Transductions Defined by Monadic Second Order Logic and by Attribute Grammars. J. Comput. Syst. Sci. 61, 1, Aug. 2000, 1–50
2. Boneva I., Talbot J.-M., Automata and Logics for Unranked and Unordered Trees. In: 20th International Conference on Rewriting Techniques and Applications, LNCS, SV, 2005
3. Fokoue A., Rose K., Simon J., and Villard L., 2005. Compiling XSLT 2.0 into XQuery 1.0. In Proceedings of the 14th International Conference on World Wide Web (Chiba, Japan, May 10 - 14, 2005), ACM Press, New York, NY, 682–691
4. Gottlob G., Koch C., and Pichler R., Efficient Algorithms for Processing XPath Queries. In Proc. of the 28th International Conference on Very Large Data Bases (VLDB 2002)
5. Gottlob G., Koch C., and Pichler R. 2003. XPath Processing in a Nutshell. SIGMOD Rec. 32, 2, Jun. 2003, 21–27
6. Hloušek P.: XPath, XQuery, XSLT: Formal Approach. Doctoral thesis, 2005. `http://kocour.ms.mff.cuni.cz/~hlousek/`
7. Hidders J., Michiels P., Paredaens J., and Vercammen R., LiXQuery: a Formal Foundation for XQuery Research. SIGMOD Rec. 34, 4, Dec. 2005, 21–26
8. Neven F. and Schwentick T., XPath Containment in the Presence of Disjunction, DTDs, and Variables. International Conference on Database Theory, 2003. `http://citeseer.ist.psu.edu/neven03xpath.html`
9. P. T. Wood. Containment for XPath fragments under DTD constraints. ICDT 2003. `http://citeseer.ist.psu.edu/wood03containment.html`
10. XML Path Language (XPath) 2.0. W3C Candidate Recommendation 3 November 2005. `http://www.w3.org/TR/2005/CR-xpath20-20051103/`
11. XML Path Language (XPath) 2.0. W3C Recommendation 23 January 2007. `http://www.w3.org/TR/xpath20/`
12. XQuery 1.0: An XML Query Language. W3C Recommendation 23 January 2007. `http://www.w3.org/TR/xquery/`
13. XSL Transformations (XSLT) Version 2.0. W3C Recommendation 23 January 2007. `http://www.w3.org/TR/xslt20/`

Vizualizácia RDF dát pomocou techniky zlučovania vrcholov*

Jiří Dokulil[1] and Jana Katreniaková[2]

[1] Katedra softwarového inženýrství, MFF UK, Praha
jiri.dokulil@mff.cuni.cz
[2] Katedra informatiky, FMFI UK, Bratislava
katreniakova@dcs.fmph.uniba.sk

Abstrakt *Rovnako ako veľká časť XML dokumentov neobsahuje definíciu použitej schémy, je možné očakávať, že budú existovať aj RDF dokumenty bez RDF schémy alebo ontológie. Pri prezentácií týchto dát používateľovi s nimi môžeme pracovať čisto ako so všeobecným ohodnoteným orientovaným grafom a zdá se prirodzené ho prezentovať vhodným nakreslením tohoto grafu. Keďže RDF dáta môžu byť potenciálne veľmi veľké, zobrazovanie celého grafu je prakticky nemožné. Graf preto vykresľujeme postupne počnúc vhodným počiatočným vrcholom. Počas prehľadávania dát pritom poskytujeme používateľovi možnosti ďalších smerov navigácie vďaka technike 'zlučovania vrcholov', ktorá zároveň šetrí priestorové nároky na vizualizáciu grafu. Kombináciou vhodného nakreslenia a spôsobu navigácie získame nástroj, vďaka ktorému môže používateľ nadobudnúť predstavu o obsahu a štruktúre dát.*

1 Úvod

Skúsenosti z XML sveta ukazujú, že napriek tomu, že existujú pokročilé prostriedky na definovanie prípustnej štruktúry pre XML dokumenty (DTD, XML schémy, ...), reálne XML dokumenty často buď túto definíciu nevyužívajú alebo jej nezodpovedajú (nie sú validné) [7]. Je možné, že obdobná situácia nastane aj pri RDF dokumentoch. Preto má zmysel uvažovať nad nástrojom, ktorým by bolo možné preskúmávať obsah RDF dokumentu bez akejkoľvek znalosti o jeho štruktúre. Keďže RDF dáta sú chápané ako orientovaný ohodnotený graf, je prirodzené ponúknuť používateľovi ich grafické znázornenie. Pri umožnení vhodnej navigácie v tomto grafe bude používateľ schopný získať predstavu o skutočnom obsahu dát.

Práca s RDF dátami prináša niekoľko problémov.

1. Dáta môzu by potenicálne veľmi rozsiahle. V našom modelovom prípade máme k dispozícií graf s miliónmi uzlov a desiatkami miliónov hrán [3]. Zrejme preto neprichádza do úvahy ich súčasné zobrazenie, ale dokonca ani možnosť mať dáta pri zobrazovaní celé uložené v pamäti.
2. V dátach sa budú pravdepodobne vyskytovať uzly s veľmi veľkým stupňom. Modelové dáta obsahujú viacero uzlov so stupňom rádovo desať tisíc a dokonca jeden uzol, do ktorého vedie niekoľko miliónov hrán. Pre takýto uzol si nemožeme dovoliť graficky zobrazovať všetkých jeho susedov.
3. Pri práci s jedným uzlom je možné jednoducho pracovať s jeho bezprostredným okolím. V bežných prípadoch je možné zvládnuť aj prehľadávanie vo vzdialenosti 2. Väčšie okolie už môže predstavovať potenciálne príliš veľkú časť celého grafu.
4. Často nastáva situácia, kedy z uzlu, ktorý reprezentuje nejaký objekt, vychádzajú hrany reprezentujúce vlastnosti daného objektu. Takéto hrany ďalej nepokračujú. Vytvárajú teda hviezdu, kde stredom hviezdy môže byť napríklad nejaká osoba a vychádzajúcimi hranami jej vlastnosti (meno, dátum narodenia, ...).

Pri návrhu algoritmu vizualizácie a navigácie sme vychádzali zo skúsenosti s tvorbou dátoveho stohu [1]. Implementovaná vizualizácia dát stohu (známa pod názvom Tykadlo) bola napriek odlišnosti formátu dát výraznou inšpiráciou pri návrhu metódy zlučovania uzlov. Z dátového stohu sme tiež získali rozsiahle RDF dáta, ktoré vznikli zo skutočného života, takže sme si mohli vytvoriť predstavu o problémoch, s ktorými sa pri vizualizácii budeme stretávať.

Nástroj pre vizualizáciu RDF dát je vytváraný nad infraštruktúrou pre sémantický web, ktorá vzniká na MFF UK Praha [10].

2 Vizualizácia dát

Vizualizácia dát je pre používateľov dôležitá najmä z psychologického hľadiska, keďže zrakový vnem je pre človeka veľmi podstatný. Ponúka sa však nespočetné množstvo možností, ako sa dáta dajú vizualizovať. V oblasti kreslenia grafov sa za 'dobré' nakreslenie považuje také, ktoré zachováva dobre merateľné kritériá (plocha grafu, počet krížení hrán, počet ohybov hrán, atď.). Keďže tieto kritériá nevychádzajú z reálnych experimentálnych dát [6], nie je z pohľadu vizualizácie dát situácia taká jednoznačná. Pojem 'pekná vizualizácia dát' je navyše veľmi subjektívny. Najmä v prípade operácií pri navigácií vo vizualizovaných dátach

* Táto práca je čiastočne podporovaná Grantom Univerzity Komenského UK/359/2007 a Narodním programem výzkumu (projekt Informační společnost 1ET100300419).

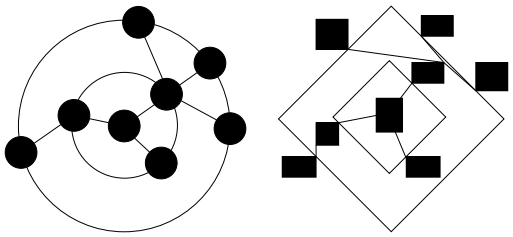
sa však v poslednej dobe považuje za kľúčové zachovávanie mentálnej mapy, t.j. snaha o minimálne zmeny v pozícií vrcholov, ktorých sa operácia netýka. Zachovávanie mentálnej mapy však prudko kontrastuje s merateľnými kritériami pre 'pekné' nakreslenie a je preto dôležité nájsť vhodný kompromis.

Pri vizualizácii RDF dát bez znalosti RDF schémy alebo ontológie získávame ohodnotený orientovaný graf. Veľkosť vzniknutého grafu však môže spôsobovať niekoľko zásadných problémov. V prvom rade môže vzniknúť problém už s ukladaním veľkého množstva dát, ktoré potrebujeme na vykreslenie kompletného grafu. Ďalším problémom môže byť príliš dlhý čas na výpočet pozícií vrcholov grafu. Tento problém je podstatný najmä pokiaľ chceme v grafe robiť aj nejaké modifikácie alebo navigáciu. V neposlednom rade môžeme v počte zobrazovaných objektov dospieť až k hodnote, ktorú už nemáme ako zobraziť alebo vieme zobraziť iba veľmi neprehľadne (napr. môže byť vo výslednom zobrazení problém rozlíšiť vrcholy a hrany, nehovoriac o čitateľnosti obsahu vrcholov).

Medzi obvyklé spôsoby riešenia problémov vizualizácie veľkého grafu patria podľa [5]:

- **Geometricky zoom** zabezpečuje podrobnejší náhľad zväčšením vybranej časti grafu. Najväčším problémom býva strata kontextu v okolí tejto vybranej časti. Jedným z riešení, ktoré sa snaží riešiť zoom bez straty kontextu je tzv. fisheye pohľad, ktorý však tiež funguje iba obmedzene.
- **Sémantický zoom** poskytuje možnosť meniť mieru zobrazenia podrobností v grafe. Je však potrebné predspracovanie, kde na základe semantiky dát určíme sémanticky podobné prvky a vytvoríme z nich tzv. clustre.
- **Postupné preskúmavanie** vytvára zobrazovaný graf postupne. Na začiatku zobrazujeme iba malú časť grafu (v extrémnych prípadoch iba počiatočný vrchol) a zvyšné časti grafu sa zobrazujú iba v prípade potreby.

Keďže nemáme žiadne znalosti RDF schémy ani ontológie je približovanie na základe semantiky dát nemysliteľné. Zvolili sme preto techniku postupného preskúmavania a vykresľovania grafu. Ten zároveň nevyžaduje prácu s celým grafom, ale iba s vykreslenou časťou a je teda menej náročný na pamäť a čas. Pre aktuálne vykreslený podgraf poskytneme používateľovi možnosť preskúmavať okolie vykreslenej časti, t.j. rozšíriť podgraf o jeden (alebo viac) priamych susedov niektorého vykresleného vrcholu. Takýmto spôsobom vytvárame *prehľadávací strom* pre dáta. V každom momente chceme mať vizualizovaný tento prehľadávací strom spolu s ostatnými (nestromovými) hranami medzi zobrazenými vrcholmi.



(a) Radiálne rozloženie (b) Štvorcové rozloženie

Obrázok 1. Používané rozloženia pri kreslení veľkých stromov.

2.1 Algoritmus vykresľovania

Aktuálne vykresľovaný graf máme určený vykresľovanými vrcholmi a prehľadávacím stromom. Ako podstatnú štruktúru považujeme práve strom, ktorý nám pre každý vrchol určuje cestu k počiatočnému vrcholu.

Jedným z obľúbených spôsobov vizualizácie veľkých grafov a špeciálne stromov je radiálne rozloženie, kde vrcholy umiestňujeme na sústredné kružnice so stredom v niektorom významnom vrchole (pozri Obr. 1(a)). Keďže v našom prípade sú vrcholy obdĺžniky bez možnosti rotácie (chceme umožniť pozerať obsah vrcholu), potrebujeme také rozloženie vrcholov, kde počet výškových bodov (t.j. rozdiel y-ových súradníc) oblasti zodpovedajúcej nejakému uhlu α bude nezávislý od umiestnenia tejto oblasti. Preto sme sa rozhodli pre modifikáciu radiálneho rozloženia tzv. *štvorcové rozloženie* (pozri Obr. 1(b)). Polomerom štvorca budeme v ďalšom texte označovať polomer kružnice opísanej tomuto štvorcu.

V prvom rade sa zameriame na vykreslenie stromu. Vrcholy v stromu budeme reprezentovať obdĺžnikmi (označme $\Gamma(v)$), kde výška bude nanajvýš rovná šírke obdĺžnika. Táto podmienka je prijateľná, pretože mená hrán, ktoré vo vrcholoch uchovávame sú URL, ktoré sú obvykle dlhé. Naproti tomu, výšku vrcholu môžeme ľahko ovplyvniť tým, že nevykreslíme všetky jeho riadky a umožníme si riadky prezerať. Vrcholy umiestňujeme zvonku na strany sústredných štvorcov (v prvom kvadrante je na štvorci umiestnený ľavý dolný roh, v ostatných kvadrantoch analogicky). Roh obdĺžnika $\Gamma(v)$, ktorý leží na jednom zo sústredných štvorcov budeme označovať $\gamma_0(v)$ a protiľahlý vrchol $\gamma_1(v)$. Vzdialenosť od koreňa stromu určuje jednoznačne štvorec, na ktorom sa vrchol zobrazuje. Hrany stromu budú priame čiary spájajúce vrcholy stromu. Hrana (u, v) kde u je predchodcom v je reprezentovaná úsečkou spájajúcou $\gamma_1(u)$ s $\gamma_0(v)$. Pritom spôsob ukladania jednotlivých bodov nám zaručuje, že na tejto úsečke neleží žiaden iný vrchol a nepretína ju iná stromová hrana.

Majme teda strom $T = (V, E)$ s koreňom vo vrchole r_T. Vrcholy, ležiace v hĺbke h stromu označme $L(h)$ (zrejme $L(0) = \{r_T\}$). Pre každý vrchol v priradíme jednoznačnú sféru vplyvu t.j. uhol $(\alpha_1(v), \alpha_2(v))$, do ktorého sa umiestni podstrom s koreňom vo v. Pokiaľ obmedzíme $\alpha_1(v), \alpha_2(v)$, iba na prvý kvadrant, potom pokiaľ r je polomer štvorca a $(\alpha_1(v), \alpha_2(v))$ je uhol, do ktorého ideme vykresľovať, tak počet vykreslených výškových bodov pre zónu vplyvu vrcholu v bude $d = r.\left(\frac{sin\ \alpha_1(v)}{sin\ \alpha_1(v)+cos\ \alpha_1(v)} - \frac{sin\ \alpha_2(v)}{sin\ \alpha_2(v)+cos\ \alpha_2(v)}\right)$. Ostatné kvadranty sa vypočítajú analogicky. V prípade, že $(\alpha_1(v), \alpha_2(v))$ presahuje viacero kvadrantov, je počet výškových bodov súčtom výškových bodov v týchto kvadrantoch. Sumu týchto výškových bodov budem v ďalšom označovať D. Do týchto bodov umiestňujeme potomkov vrcholu v. Bez ujmy na všeobecnosti nech sú to vrcholy $v_1 \dots v_k$, kde v_i má rozmery $H(v_i) \times W(v_i)$. Ak požadujeme, aby medzi jednotlivými vrcholmi bola minimálna vzdialenosť δ, potom je pre synov vrchola v potrebných $\Sigma_{i=1}^{k}(H(v_i) + \delta)$ výškových bodov. Z uvedeného musí teda platiť nerovnosť $D > \Sigma_{i=1}^{k}(H(v_i)+\delta)$.

Samotný algoritmus vykresľovania najprv vykreslí počiatočný vrchol so stredom v počiatku súradnicovej sústavy. Následne bude postupne prechádzať strom a pre hĺbku h ($h > 0$) bude pracovať nasledovne (pozri tiež Obr. 2 a 3). Nech sa doteraz vykreslené vrcholy stromu dajú vpísať do štvorca s polomerom r_{obs}.
Pre každý vrchol v z predchádzajúcej úrovne $(h-1)$ mám určený uhol, do ktorého patria jeho potomkovia $v_1 \dots v_k$ s výškami $H(v_1) \dots H(v_k)$. Na základe ich veľkostí a nerovnosti $D > \Sigma_{i=1}^{k}(H(v_i) + \delta)$ viem určiť minimálny potrebný polomer r_{min} pre jeho priamych potomkov.
Maximum z týchto polomerov a polomeru doteraz obsadeného štovrca r_{obs} je polomer, na ktorý ideme vykresľovať úroveň h.
Nový polomer určuje pre každý podstrom zodpovedajúci počet výškových bodov, do ktorých budeme rozmiestňovať vrcholy. Preto aj vzdialenosť medzi nimi je nutné znovu prepočítať (opäť použitím nerovnosti $D > \Sigma_{i=1}^{k}(H(v_i) + \delta)$).
Pre každý vrchol z úrovne $h-1$ vypočítame súradnice a sféry vplyvu pre každého jeho potomkov. Využívam pri tom vypočítaný polomer r a minimálnu vzdialenosť $\delta(v)$ zodpovedajúcu tomuto podstromu. Nech teda vrchol v má sféru vplyvu $(\alpha_1(v), \alpha_2(v))$ a potomkov s výškami $H(v_1) \dots H(v_k)$. Budeme predpokladať, že celá sféra vplyvu je iba v prvom kvadrante. Vypočítame si sféru vplyvu $(\alpha_1(v_i), \alpha_2(v_i))$ a y-ovú súradnicu γ_0 (ozn. $y(v_i)$). Z $y(v_i)$ a polomeru r vieme jednoznačne určiť aj druhú súradnice γ_0.
V prípade, že sa sféra vplyvu nachádza celá v inom kvadrante, vypočítajú sa nové hodnoty analogicky. Ak sa sféra vplyvu zasahuje do viacerých kvadrantov, rozdelíme jednotlivých synov medzi tieto kvadranty a v každom postupujeme samostatne.

Umiestnime počiatočný vrchol t_r so stredom $[0, 0]$.
Pre jednotlivé hĺbky stromu $h = 1, 2 \dots$
  Pre všetky vrcholy $v \in L(h-1)$ vypočítam $r_{min}(v)$
  $r \leftarrow max\{max\{r_{min}(v) \mid v \in L(h-1)\}, r_{obs}\}$
  Pre všetky vrcholy $v \in L(h-1)$ vypočítam $\delta(v)$
  Pre všetky vrcholy $v \in L(h-1)$.
    Nech v má sféru vplyvu $(\alpha_1(v), \alpha_2(v))$ a potomkov $v_1 \dots v_k$ s výškami $H(v_1) \dots H(v_k)$.
    $\alpha_1(v_0) \leftarrow \alpha_2(v)$
    Pre $i = 1 \dots k$
      $\alpha_2(v_i) \leftarrow \alpha_1(v_{i-1})$
      $y_{pom} \leftarrow \frac{r.sin\ \alpha_2(v_i)}{sin\ \alpha_2(i)+cos\ \alpha_2(i)}$
      $y(v_i) \leftarrow y_{pom} - H(v_i) - \frac{\delta(u)}{2}$
      $\alpha_1(v_i) \leftarrow arctg\left(\frac{y(v_i)-\frac{\delta(u)}{2}}{r-y(v_i)-\frac{\delta(u)}{2}}\right)$
      $x(v_i) \leftarrow r - y(v_i)$

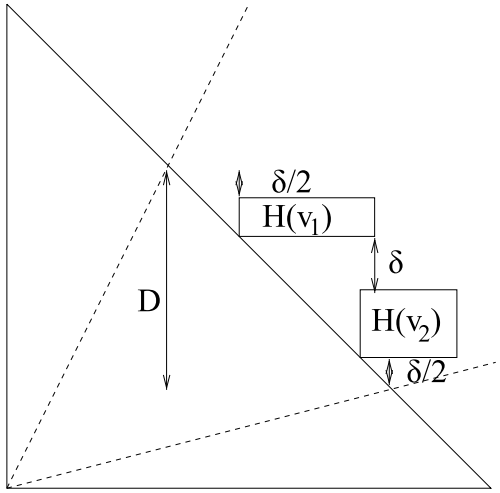
Obrázok 2. Algoritmus vykresľovania stromu (sféry vplyvu obmedzené na 1. kvadrant).

Zvyšné nestromové hrany grafu môžeme vykresľovať tiež ako spojnice medzi vrcholmi. Nemáme však zaručené, že táto úsečka nebude pretínať iné hrany alebo vrcholy. Nepredpokladáme však príliš veľké množstvo nestromových hrán. Navyše v rámci navigácie umožníme meniť pohľad v prípade, že sa graf stane neprehľadným.

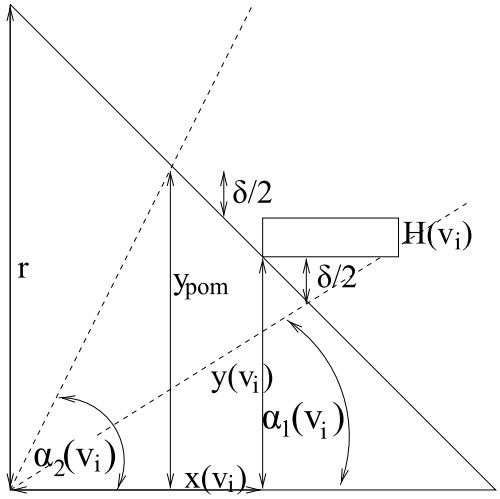
2.2 Navigácia

Pre používateľa je potrebné okrem prehľadného nakreslenia grafu zabezpečiť aj možnosť si pohľad na graf prispôsobiť svojim požiadavkám. Ako vhodný spôsob preskúmavania grafu sme zvolili rozširovanie pohľadu o niektorého suseda už vykresleného vrchola.

Zlučovanie vrcholov. Počas preskúmávania grafu využívame techniku zlučovania vrcholov. Vykreslený vrchol neobsahuje iba svoj popis ale aj zoznam vychádzajúcich a vchádzajúcich hrán. Týmto umožníme prehliadať susedov vrchola bez toho, aby to vyžadovalo zväčšené nároky na vykresľovací priestor. Významnou výhodou je, že používateľ si sám vyberie, ktorí susedia ho zaujímajú a rozšíri pohľad iba o tieto vrcholy. Tým odbúrame problém so špecifickými vrcholmi, ktoré môžu mať desaťtísíce susedov, ktoré však nie sú zaujímavé. Pri priamom vykresľovaní by sme však často narazili na problém, ako tieto vrcholy všetky zrazu vykresliť alebo ktoré z týchto vrcholov vykresliť. V prípade zlúčeného vrcholu môžeme používateľovi v prípade uzlov s menším množstvom susedov poskytnúť možnosť ich prelistovať a v prípade špecifických uzlov umožniť vyhľadávanie. Zlučovanie vrcholov tiež



(a) Počet výškových bodov



(b) Vysvetlenie premenných

Obrázok 3. Vysvetlenie výpočtu algoritmu.

umožní lepšiu prehľadnosť u ďalšieho typu vrcholov. V RDF dátach sa vyskytujú aj uzly reprezentujúce nejaký objekt, kde vychádzajúce hrany reprezentujú vlastnosti tohoto objektu. Ako príklad môže byť osoba, kde vychádzajúce hrany môžu byť meno, dátum narodenia, atď. Zlúčený uzol bude obsahovať tieto hrany prehľadne a nebude mať ďalšie priestorové nároky na susedov, z ktorých je ďalšia navigácia nemožná. Pritom sa jedná o potenciálne veľké množstvo susedov (desiatky hodnôt), z ktorých však nevychádzajú ďalšie hrany.

Zvolenie počiatočného vrcholu. Na začiatku prehľadávania si zvolíme počiatočný vrchol, z ktorého sa začne preskúmávanie grafu. Pokiaľ je graf nesúvislý, je možné, že pri niektorých voľbách počiatočného vrchola sa používateľ nedostane k pre neho zaujímavým dátam. Našťastie situácie, kedy je graf nesúvislý sú dosť zriedkavé.

Na výber počiatočného vrcholu môžeme uvážiť niekoľko možností.

- **Voľba náhodného vrcholu.** Jednoduchá metóda vychádzajúca z predpokladu, že pokiaľ zvolíme akýkoľvek počiatočný uzol, tak na niekoľko krokov sa používateľ sám jednoducho dostane k zajímavým dátam.
- **Voľba vrcholu s najvyšším stupňom.** Táto možnosť síce umožní používateľovi najrýchlejší prístup k čo najväčšiemu počtu ďalších vrcholov, ale vrchol s veľkým počtom susedov môže byť často iba vrchol pridaný z technických dôvodov (napríklad vrchol reprezentujúci nezadanú hodnotu) a môže byť k ďalšej navigácií absolútne nevhodný.
- **Analýza štruktúry grafu.** Teoreticky je možné skúmať vlastnosti celého RDF grafu a z nich odvodit vhodný počiatočný vrchol, napríklad ako stred grafu. Avšak s ohľadom na veľkosť dát je táto metóda v praxi len ťažko použiteľná.
- **Využitie ontológie.** Používaným postupom [2] je nechať používateľa vybrať vhodný počiatočný bod z ontologie nebo dátovej schémy. Keďže však chceme riešenie, ktoré funguje na ľubovoľných RDF dátach, nie je v našom prípade táto možnosť použiteľná.
- **Dotaz.** Dalšia reálne používaná možnosť je dovoliť používateľovi položiť dotaz v niektorom dotazovacom jazyku (napríklad SPARQL [8]) a výsledok dotazu použiť jako počiatočný bod. Aj toto riešenie pre nás nie je použiteľné, pretože predpokladáme, že používateľ nemá na začiatku žiadnu predstavu o tom, ako dáta vyzerajú, takže ani nebude schopný položiť vhodný dotaz.
- **Analýza používateľov.** Je možné sledovať, akým spôsobom predchádzajúci používatelia prechádzali graf a na základe toho zvoliť významný uzol, ako napríklad najčastejšie navštevovaný.

Ako najvhodnejšia sa nám v súčasnosti javí prvá možnosť, pričom v budúcnosti očakávame jej kombinovanie s poslednou možnosťou. Takto zvolený vrchol zobrazíme do počiatku súradnicovej sústavy ako *zlúčený vrchol*. Aktuálny vykresľovaný graf ako aj prehľadávací strom bude pozostávať iba z tohoto vrchola.

Rozšírenie pohľadu. Majme aktuálny zobrazený graf $G = (V, E)$ a prehľadávací strom $T = (V, E')$, kde $E' \subseteq E$. Nech je vybraný vrchol $v \in V$, s množinou susedov $N(v)$. V prípade, že $N(v) \subseteq V$, potom vrchol má už vykreslených všetkých svojích susedov a nemôžeme v ňom pokračovať. Nech teda $N'(v) = N(v) \setminus V$ je množina doteraz nezobrazených susedov vrchola v a $u \in N'(v)$. Potom môžeme rozšíriť aktuálne zobrazený graf a strom nasledovne:

- Nová množina vrcholov bude rozšírená o vrchol u, t.j. $V \leftarrow V \cup \{u\}$
- Množina hrán bude rozšírená o všetky hrany z RDF grafu susediace s vrcholom u a s niektorým už vykresleným vrcholom.
  $(E \leftarrow E \cup \{(u,w) \mid w \in V \ \wedge \ \exists p : (u,p,w) \in RDF\} \cup \{(w,u) \mid w \in V \wedge \exists p : (w,p,u) \in RDF\})$
- Do stromu $T = (V, E')$ pridáme hranu z vrcholu v do vrcholu u. Potom $E' \leftarrow E' \cup \{(v,u)\}$.

Vykreslenie nového pohľadu je možné vykonať viacerými spôsobmi. Prvou triviálnou možnosťou je kompletné prekreslenie celého grafu. Z hľadiska optimalizácie priestoru po pridaní vrcholu sa jedná o veľmi dobrú možnosť. Jej zásadnou nevýhodou je, že nezachováva mentálnu mapu používateľa, keďže vrcholy sa môžu pri kompletnom prekreslení výrazne posunúť. Pri rozšírení pohľadu však nepotrebujeme ani kompletne prekresliť celý aktuálny graf. Vo väčšine prípadov bude potrebné prekresliť iba podstrom zakorenený vo predchodcovi pridaného vrcholu. Môžeme teda prekresliť iba oblasť sféry vplyvu vrcholu, ktorému sme pridali nového potomka. Iba v prípade, že sa vrchol nezmestí do sféry vplyvu svojho predchodcu, bude potrebné zväčšiť polomer štvorca, na ktorý vrchol u potrebujeme umiestniť. Pri tejto zmene však pre už vykreslené vrcholy nemusíme prerátávať nové súradnice pomocou vykresľovacieho algoritmu. Nové súradnice vzniknú jednoduchou projekciou na nový štvorec (posunutím zobrazenia vrchola v v smere vektora $\gamma_0(v)$ po úroveň štvorca, na ktorý sa má vykresliť). Táto zmena sa môže týkať všetkých štvorcov s polomerom väčším ako štvorec, na ktorý sa mal vrchol u umiestniť. Jedná sa však o jednoduchú transformáciu, ktorá sa navyše nevykonáva často. Samozrejme, táto metóda nie je optimálna z hľadiska minimalizácie vykresľovacieho priestoru, avšak kompletne zachováva mentálnu mapu používateľa a preto sme sa rozhodli ju v navigácií uplatniť. V budúcnosti uvažujeme aj nad možnosťou jej vylepšenia, keď v prípade nevyužívania sfér vplyvu niektorých vrcholov budeme môcť rozšíriť sféru vplyvu používaného vrcholu.

Zúženie pohľadu. Je možné, že počas navigácie v grafe používateľ bude považovať niektoré podstromy prehľadávacieho stromu za nepodstatné. Preto umožníme aj zúženie pohľadu o vrchol resp. celý podstrom prehľadávacieho stromu. Pritom si vrcholy, ktoré v pohľade zostali zachovajú súčasné súradnice (v budúcnosti uvažujeme aj nad vhodnou úpravou vykresleného stromu avšak so zachovaním mentálnej mapy, t.j. so zachovaním smerov hrán medzi vrcholmi, ktoré v grafe ostali). Napriek tomu, že graf neprekresľujeme, získame týmto krokom miesto pre ďalšie rozširovanie pohľadu o vrcholy, ktoré sú pre používateľa relevantné. Rovnako ako pri rozšírení pohľadu existuje samozrejme možnosť prekreslenia celého grafu, alebo vhodného kompromisu medzi týmito možnosťami.

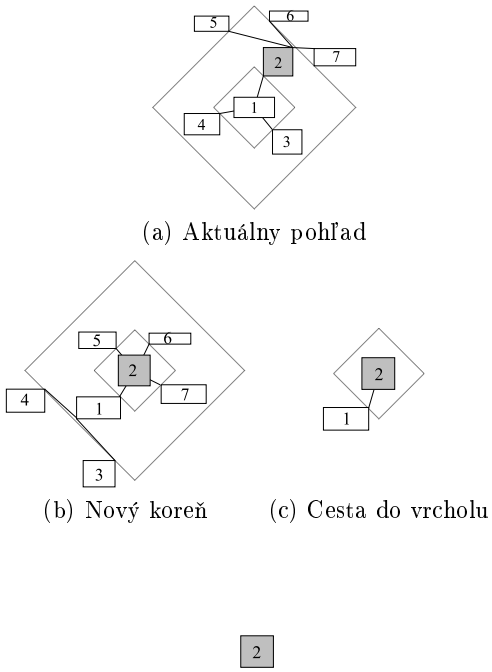
Zvolenie nového pohľadu. Používateľ môže počas preskúmávania grafu dospieť k záveru, že aktuálne vykreslený graf pre neho stratil význam, ale napriek tomu chce pokračovať v prehliadaní dát. V takýchto prípadoch umožníme nový pohľad na preskúmané dáta niektorým s nasledovných spôsobov:

- **Zmena počiatočného vrcholu** (pozri Obr. 4(b)) ponechá všetky zobrazené vrcholy. Zmení iba zobrazovaný strom tým, že za koreň stromu zvolí vybraný vrchol. Tento spôsob je vhodný najmä v prípade, že používateľ plánuje pokračovať v preskúmávaní z vybraného vrcholu, ale zaujíma sa o prepojenie so zvyškom už preskúmaného grafu.
- **Zachovanie cesty** do vybraného vrcholu (pozri Obr. 4(c)) zruší celý zobrazovaný graf okrem cesty od koreňa stromu do vybraného vrcholu, pričom za nový koreň označíme vybraný vrchol. Umožníme tým zachovanie postupnosti krokov, ktoré viedli k objaveniu vrcholu. Pritom tento spôsob nemá veľké nároky na vykresľovací priestor.
- **Zachovanie vrcholu** (pozri Obr. 4(d)) je najextrémnejšou zmenou pohľadu na preskúmávané dáta. Z existujúceho grafu je zachovaný iba vybraný vrchol (samozrejme v tvare zlúčeného vrcholu) a tento je vyhlásený za nový počiatočný vrchol, z ktorého môže používateľ preskúmávať graf.

3 Existujúce prístupy k vizualizácii RDF dát

Keďže už v súčasnosti existujú rôzne RDF dáta, pokúšajú sa ich prehliadanie pomocou vizualizácie riešiť mnohé aplikácie. Väčšina však vizualizuje celý graf, čo v prípade väčších dát naráža na už spomenuté problémy s ukladaním grafu a neprehľadnosťou vizualizácie. Z množstva existujúcich riešení sme vybrali niekoľko príkladov, ktoré zastupujú hlavné trendy v oblasti vizualizácie RDF dát:

- **RDF Gravity [4]** pracuje primárne s celým grafom a iba ako možnosť naviac umožňuje skryť niektoré uzly. Preto se hodí najmä na proskúmávanie ontológií alebo malých dát.
- **Node-centric RDF Graph Visualization [9]** sa ako jedno z mála riešení nepokúša zobraziť presne graf s dátami, ale vždy zobrazuje strom predchodcov a nasledovníkov zvoleného uzlu. V prípade, že je k nejakému uzlu možné prísť viacerými cestami (pri presnom vykresľovani grafu by vznikla nestromová hrana), potom je takýto uzol zobrazený na všetkých týchto cestách a tým je zachovaná



(a) Aktuálny pohľad

(b) Nový koreň (c) Cesta do vrcholu

(d) Vrchol

Obrázok 4. Nové pohľady na graf.

čisto stromová štruktúra. Nevýhodou tohoto riešenia je, že zobrazuje iba uzly do vzdialenosti 2 od počiatočného uzlu a nerieši problém uzlov s veľkým stupňom.

– **Paged Graph Visualization (PGV) [2]** Toto riešenie je podobné nášmu v tom, že sa nepokúša zobraziť naraz celý RDF graf. Na zobrazovanie uzlov s veľkým stupňom používa metódu *Ferris-Wheel*, ktorá zobrazí susedov vrcholu do hviezdice okolo vrcholu. Je však použiteľná iba pre uzly so stupňom rádovo niekoľko stoviek. Navyše pokračovanie prehľadávania v niektorých vrcholoch vedie k neprehľadnosti zobrazovania alebo úplnej strate mentálnej mapy v okolí vybraného vrcholu.

Napriek rozmanitosti riešení existuje veľmi málo aplikácií, ktoré je možné použiť aj na rozsiahle RDF dáta. Žiadna z týchto aplikáciií však nevyužíva techniku podobnú nami navrhovanému zlučovaniu uzlov a navyše všetky majú problémy s vrcholmi veľkých stupňov, ktoré sa v reálnych veľkých dátach často vyskytujú.

4 Záver

V článku sme popísali algoritmus na vizualizáciu RDF dát a navigáciu v nich. Tento algoritmus pracuje na všeobecných RDF dátach bez znalosti podrobnejšej štruktúry a preto sa dá využiť aj pri preskúmávaní dát, ktoré neobsahujú RDF schému alebo ontológiu. Navyše je vhodný aj pre veľké dáta a na rozdiel od existujúcich aplikácií zvláda aj prácu s dátami, ktoré obsahujú uzly z veľkých stupňom, vďaka technike zlučovania vrcholov. Pri vytváraní algoritmu sme skúmali vlastnosti veľkých reálnych dát, ktoré sme mali k dispozícií. Samotný algoritmus je budovaný nad infraštrukúrou pre Semantický web, ktorá sa vyvýja na Matematicko-Fyzikálnej Fakulte Univerzity Karlovej.

V budúcnosti by sme chceli dopracovať systém na výber počiatočného vrcholu navigácie. Ďalšou dôležitou oblasťou, ktorej sa plánujeme venovať je vylepšenie vizualizácie, kde plánujeme zlepšiť vykresľovanie nestromových hrán a modifikovať výpočet veľkosť sféry vplyvu podľa veľkosti podstromu. Tento krok si vyžaduje aj úpravy v oblasti navigácie a preskúmanie vplyvu na prehľadnosť a zachovanie mentálnej mapy.

Referencie

1. D. Bednárek, D. Obdržálek, J. Yaghob, and F. Zavoral, Data Integration Using DataPile Structure. In ADBIS Research Communications, 2005
2. L. Deligiannidis, K. J. Kochut, and A. P. Sheth, User-Centered Incremental RDF Data Exploration and Visualization. Submited to ESWC 2007, 2006
3. J. Dokulil, Transforming Data from DataPile Structure into RDF. In Proceedings of the Dateso 2006 Workshop, 2006, 54–62
4. S. Goyal and R. Westenthaler, RDF Gravity (RDF Graph Visualization Tool)
5. I. Herman, G. Melançon, and M. S. Marshall, Graph Visualization and Navigation in Information Visualization: A survey. IEEE Trans. Vis. Comput. Graph., 6(1), 2000, 24–43
6. W. Huang and P. Eades, How People Read Graphs. In Proceedings of Asia-Pacific Symposium on Information Visualisation, 2005, 51–58
7. I. Mlynková, K. Toman, and J. Pokorný, Statistical Analysis of Real XML Data Collections. In COMAD'06: Proceedings of the 13th Int. Conf. on Management of Data, 2006, 20–31
8. E. Prud'hommeaux and A. Seaborne, SPARQL Query Language for RDF. W3C Working Draft, 2005
9. C. Sayers, Node-Centric RDF Graph Visualization. Technical Report HPL-2004-60, HP Laboratories Palo Alto, April 2004
10. J. Yaghob and F. Zavoral, Semantic Web Infrastructure Using Datapile. In Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Itelligent Agent Technology, 2006, 630–633

SVD and HOSVD decomposition of SOM neural network

Jiří Dvorský and Jana Kočíbová

VŠB – Technical University of Ostrava,
{jiri.dvorsky,jana.kocibova}@vsb.cz,
http://www.cs.vsb.cz

Abstract. *Self-Organizing Maps performs a non-linear mapping from a high-dimensional data space to a low-dimensional space, typically two dimensional, aiming to preserve the topological relations of the data. SOM networks are based on unsupervised learning. The quality of the learning process is usually measured by using mean square error. Alternative quality judgment methods based on Singular Value Decomposition resp. High Order SVD and BMU movements monitoring are proposed. Some experimental results are provided.*

1 Introduction

The *Self-Organizing Map* (SOM), also called Kohonen network, is an unsupervised neural network algorithm developed by Teuvo Kohonen [3] that provides us with two useful operations in exploratory data analysis. These are clustering, reducing the amount of data into representative categories, and projection (non-linear), aiding in the exploration of proximity relations in the patterns. The SOM algorithm has some resemblance with vector quantization algorithms [5]. The great distinction from other vector quantization techniques is that the neurons are organized on a regular grid and along with the selected neuron also its neighbors are updated.

Self-organizing maps were born to emulate the human brain characteristic of topological and geometrical organization of information. The training algorithm aims at finding analogies between similar incoming data in a nonsupervised process. The algorithm places the weight vectors such that geometrically close vectors (in weight space) are also topologically close in the grid represented by the network. In other words, for similar incoming vectors (in input space), the neurons responding more vigorously should also be similar (in terms of their weight vectors) and located in nearby positions in the network grid.

A comprehensive description of the SOM algorithm can be seen in [3]. Here we will just make a brief presentation. The SOM consists of an array of elements called neurons or units $\mathbf{m_i}$, usually arranged in a low dimensionality grid (1D or 2D), the map, for ease of visualization. The grid may have several forms like rectangular, hexagonal. For each input vector $\mathbf{x}(t)$, the unit (neuron) that best matches input vector is selected. This neuron is called *Best Matching Unit* (BMU).

Then the weights of the BMU and its neighborhood will be adapted as follows:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + \eta(t)h(t)(\mathbf{x}(t) - \mathbf{m}_i(t)), \quad \forall \mathbf{m}_i$$

where η, $0 < \eta < 1$ is the learning factor, which determines the speed of weight adaptation, and ($h(t)$ is neighborhood size determining function.

The paper is organized as follows. Section 2 briefly review Singular Value Decomposition of given matrix, and High Order SVD of given tensor. Section 3 describes the proposed SVD resp. HOSVD decomposition of SOM network. The last section 4 presents experimental results.

2 SVD and HOSVD

In this paper, tensors are denoted by calligraphic upper-case letters ($\mathcal{A}, \mathcal{B}, \ldots$), matrices by uppercase letters ($A, B, \ldots$), scalars by lower case letters ($a, b, \ldots$), vectors by bold lower case letters ($\mathbf{a}, \mathbf{b}, \ldots$).

2.1 Matrix SVD

The SVD of a matrix is visualized in Figure 1. For a $I_1 \times I_2$ matrix F, it can be written as the product:

$$F = U^{(1)} \cdot S \cdot U^{(2)}$$

where $U^{(1)} = (\mathbf{u}_1^{(1)}\mathbf{u}_2^{(1)} \ldots \mathbf{u}_{I_1}^{(1)})$ and $U^{(2)} = (\mathbf{u}_1^{(2)}\mathbf{u}_2^{(2)} \ldots \mathbf{u}_{I_2}^{(2)})$ are the matrices of the left and right singular vectors. The column vectors $\mathbf{u}_i^{(1)}$, $1 \le i \le I_1$ and $\mathbf{u}_j^{(2)}$, $1 \le j \le I_2$ are orthogonal. $S = diag(\sigma_1, \sigma_2, \ldots, \sigma_{min(I1,I2)})$ is the diagonal matrix of singular values which satisfy $\sigma_1 \ge \sigma_2 \ge \ldots \ge \sigma_{min(I1,I2)})$ By setting the smallest $(minI_1, I_2 - k)$ singular values in S to zero, the matrix F is approximated with a rank-k matrix and this approximation is the best approximation measured in terms of reconstruction error. Theoretical details on matrix SVD can be found in [2].

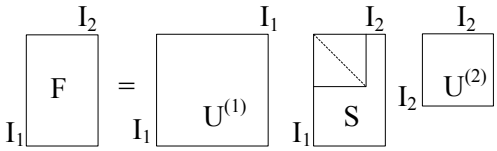


Fig. 1. Visualization of matrix SVD.

2.2 Tensor and HOSVD

A tensor is a higher order generalization of a vector (first order tensor) and a matrix (second order tensor). Higher order tensors are also called multidimensional matrices or multi-way arrays. The order of a tensor $\mathcal{A} \in R^{I_1 \times I_2 \times \cdots \times I_N}$ is N. Elements of $\mathcal{A}$ are denoted as $a_{i_1 \cdots i_n \cdots i_N}$ where $1 \leq i_n \leq I_n$. In tensor terminology, matrix column vectors are referred to as mode-1 vectors and row vectors as mode-2 vectors. The mode-n vectors of an N-th order tensor $\mathcal{A}$ are the I_n-dimensional vectors obtained from $\mathcal{A}$ by varying the index i_n and keeping the other indices fixed, that is the column vectors of n-mode matrix unfolding $\mathcal{A}_{(n)} \in R^{I_n \times (I_1 I_2 \cdots I_{n-1} I_{n+1} \cdots I_N)}$ of tensor $\mathcal{A}$. See [1] for details on matrix unfoldings of a tensor.

The n-mode product of a tensor $\mathcal{A} \in R^{I_1 \times I_2 \times \cdots \times I_N}$ by a matrix $M \in R^{J_n \times I_n}$ is an $I_1 \times I_2 \times \cdots \times I_{n-1} \times J_n \times I_{n+1} \times \cdots \times I_N$-tensor of which the entries are given by

$$(\mathcal{A} \times_n M)_{i_1 \cdots i_{n-1} j_n i_{n+1} \cdots i_N} = \\ = \sum_{i_n} a_{i_1 \cdots i_{n-1} i_n i_{n+1} \cdots i_N} m_{j_n i_n}$$

Note that the n-mode product of a tensor and a matrix is a generalization of the product of two matrices. It can be expressed in terms of matrix unfolding:

$$B_{(n)} = M A_{(n)}$$

where $B_{(n)}$ is the n-mode unfolding of tensor $\mathcal{B} = \mathcal{A} \times_n M$.

In terms of n-mode products, the matrix SVD can be rewritten as $F = S \times_1 V^{(1)} \times_2 V^{(2)}$. By extension, HOSVD is a generalization of matrix SVD: every $I_1 \times I_2 \times \cdots \times I_N$ tensor $\mathcal{A}$ can be written as the n-mode product [1]:

$$\mathcal{A} = S \times_1 V^{(1)} \times_2 V^{(2)} \cdots \times_N V^{(N)}$$

as illustrated in Figure 2 for $N = 3$. V_n contains the orthonormal vectors (called n-mode singular vectors) spanning the column space of the matrix $A_{(n)}$ (n-mode matrix unfolding of tensor $\mathcal{A}$). $\mathcal{S}$ is called core tensor. Instead of being pseudodiagonal (nonzero elements only occur when the indices satisfy $i_1 = i_2 = \cdots = i_N$), $\mathcal{S}$ has the property of all-orthogonality. That is, two subtensors $\mathcal{S}_{i_n} = \alpha$ and $\mathcal{S}_{i_n} = \beta$ are orthogonal for all possible values of n , α and β subject to $\alpha \neq \beta$. At the same time, the Frobenius-norms $\sigma_i^n = ||\mathcal{S}_{i_n=i}||$ are n-mode singular values of $\mathcal{A}$ and are in decreasing order: $\sigma_1^n \geq \sigma_2^n \geq \cdots \geq \sigma_{I_n}^n \geq 0$ $\mathcal{S}$ is in general a full tensor and governs the interactions among V_n.

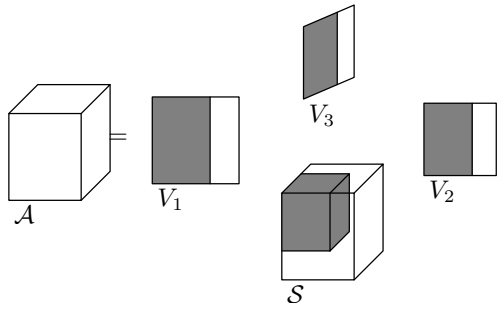


Fig. 2. Visualization of a 3-order Singular Value Decomposition.

3 SVD and HOSVD approximation of SOM

One problem that still remains in SOM research and/or in their application is problem of quality of network training. Usually mean square error is used to measure a quality of learning process. This error is just a number without any dimension or scale, and may be hard to understand. Alternative approach for measurement of quality of learning process is the goal of our research.

BMU is found for each training vector during learning process. As SOM network learns the structure of training set, the BMU of given training vector usually changes its position within the network. Movement of the BMU at the initial phase of learning process will be probably very rapid, and as the network converges to stable configuration the movement of the BMU will be very tight. The learning process could be stopped, when user specific maximal number of moved BMUs is reached.

In this way, the number of moved BMUs can be taken as alternative learning process quality measurement. The number of changes of BMUs' positions between successive iterations was considered as measure in our initial work. But this approach is not very helpful. Some movements of the BMU still remain.

3.1 SVD approach

The second approach to measurement of BMU movement uses SVD decomposition of SOM network. It is supposed, that movement of BMUs among original SOM and its rank-k approximations will be very low, when stable configuration of the SOM is reached.

To verify this hypothesis following experiment was performed:

1. The SOM network S is transformed to $rc \times m$ matrix A, where r is the number of rows, c is the number of columns of SOM S, and m is the dimension of input.

$$A = \begin{pmatrix} \mathbf{m}_{1,1}(t) \\ \mathbf{m}_{1,2}(t) \\ \vdots \\ \mathbf{m}_{1,c}(t) \\ \mathbf{m}_{2,1}(t) \\ \vdots \\ \mathbf{m}_{r,c}(t) \end{pmatrix}$$

   Note: Each $\mathbf{m}_{i,j}(t)$ is m dimensional vector.
2. Rank k approximation A_k of matrix A is computed, $1 \leq k \leq m$.
3. Network $S(k)$ is created from matrix A_k.
4. The BMU for each training vector is computed with original SOM S.
5. The BMU(k) is computed with rank k approximation $S(k)$ for each training vector.
6. If BMU is different from BMU(k) the movement is encountered.

3.2 HOSVD approach

As it is demonstrated in Figure 2 SOM network can be understood as 3-order tensor, and High Order SVD can be applied onto SOM network. There is no need to form matrix by transformation of SOM network. 3-order SVD can directly decompose SOM network. It is expected that High order SVD preserves better relationships among neurons and structure of SOM network.

Experiment similar as in SVD approach was performed:

1. The SOM network S is transformed to tensor $\mathcal{A} \in R^{r \times c \times m}$, where r is the number of rows, c is the number of columns of SOM S, and m is the dimension of input.
2. $\mathcal{A}_{(k_1,k_2,k_3)}$ approximation of tensor $\mathcal{A}$ was computed. This approximation generalizes rank-k approximation of matrix in SVD.
3. Network $S(k_1, k_2, k_3)$ is created from tensor $\mathcal{A}_{(k_1,k_2,k_3)}$.
4. The BMU for each training vector is computed with original SOM S.
5. The BMU$((k_1, k_2, k_3)$ is computed with approximated network $S(k_1, k_2, k_3)$ for each training vector.
6. If BMU is different from BMU(k_1, k_2, k_3) the movement is encountered.

4 Experimental results

A number of experiments were carried out to prove our hypothesis. One of them is provided in this paper. Parameters of used SOM network S are given in table 1. The input data comes from experiments done by Kudelka et al. [4].

# of rows	50
# of columns	50
SOM shape	toroid
input dimension	10
# of input vectors	approx 25,000
# of iterations	5000

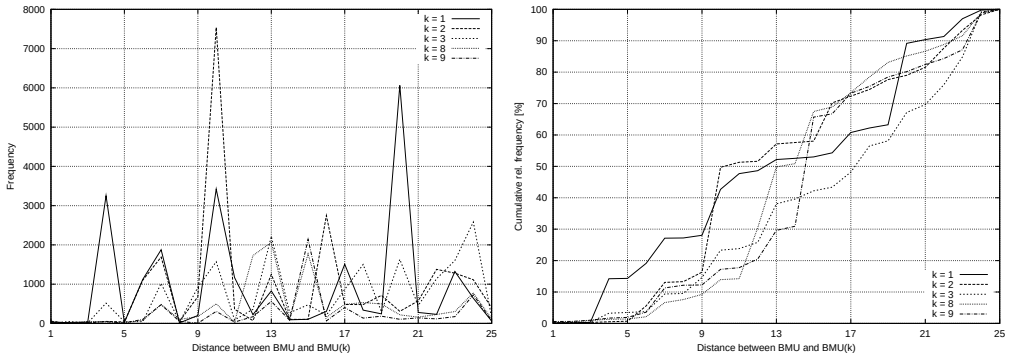
Table 1. Experimental SOM network S parameters.

The experimental results are summarizing in following tables and graphs. Table 2 provides insight to learning process. Each rank-k, $1 \leq k \leq 10$, approximation $S(k)$ of SOM network S is computed and number of moved BMUs are stored, moreover a distance of the movement.

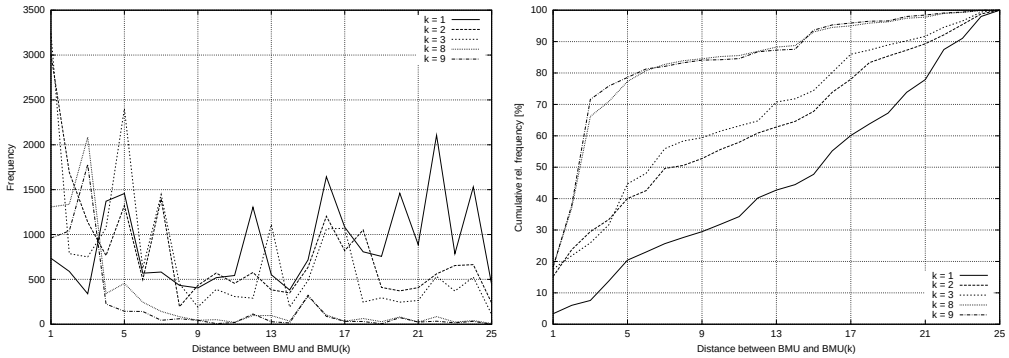
In initial phase of learning process, there is no significant difference among $S(k)$ approximations. The number of moved BMUs varies from 99% to 26% in iteration 0. Average distance of movement is also near constant, from 13.6 to 15.4.

There are significant difference among each $S(k)$ approximations after 4000 and 5000 iterations. $S(1)$ approximation has 76% of moved input vectors, but $S(8)$ resp. $S(9)$ has only 9% resp. 2.6% of moved input vectors after 5000 iterations. The average distance is also very small, only 2.4 resp. 3.

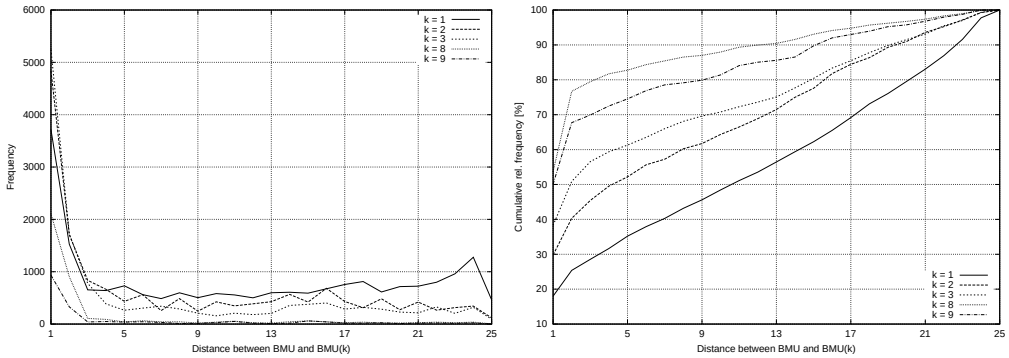
Although maximal distance of the movement remains very high in all phases of learning process, the average distance of BMUs movement decreases. Histograms of these distances can provide very useful information about learning. Figure 3(a) demonstrates initial state of SOM network S. The distribution of distances is random; there are plenty of short movements and also a lot of long movement of BMUs. The charts 3(d) have different shape. Histogram of distances has strong hyperbolic shape, i.e. very short movements are predominate. Cumulative histogram



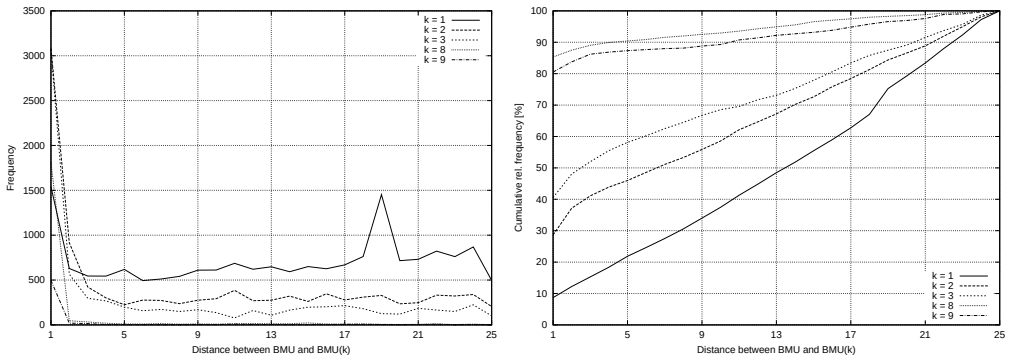
(a) Initial SOM with random weights



(b) SOM after 500 iterations



(c) SOM after 4000 iterations

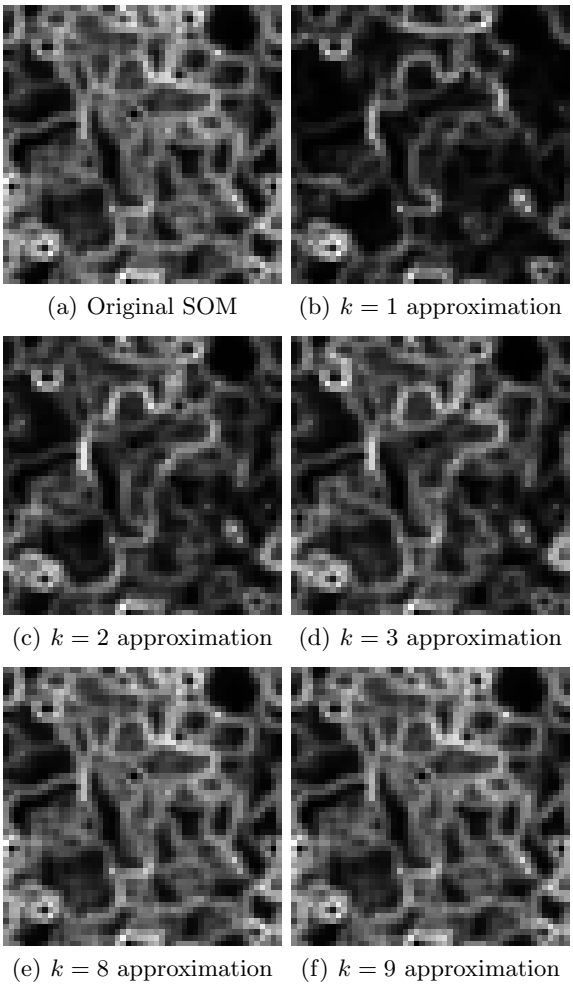


(d) SOM after 5000 iterations (final state)

Fig. 3. Frequency and cumulative frequency histograms of distances between BMU and BMU(k).

shows, that movements of length 1 (less than 20%) comprise slightly more than 80% of all movements for $S(8)$ resp. $S(9)$ approximations. These results coincide with the widely known rule called "Paretos 80/20 law". In other words, networks $S(8)$ resp. $S(9)$ are very close to original network S, consequently network S after 5000 iterations completed contains very small amount of noise (compare $S(9)$ approximation after 500 iterations completed). We can conclude, that well trained SOM network is resistant to SVD decomposition.

U-matrices [6] of experimental SOM network S and its approximations $S(k)$ are given in figure 4. These figures show, that even $S(1)$ approximation is used, the main features, main structure, of original SOM network is preserved. $S(2)$ resp. $S(3)$ approximation add more details to basic structure of $S(1)$ approximation. $S(8)$ resp. $S(9)$ approximations are not easily distinguishable from original SOM network.



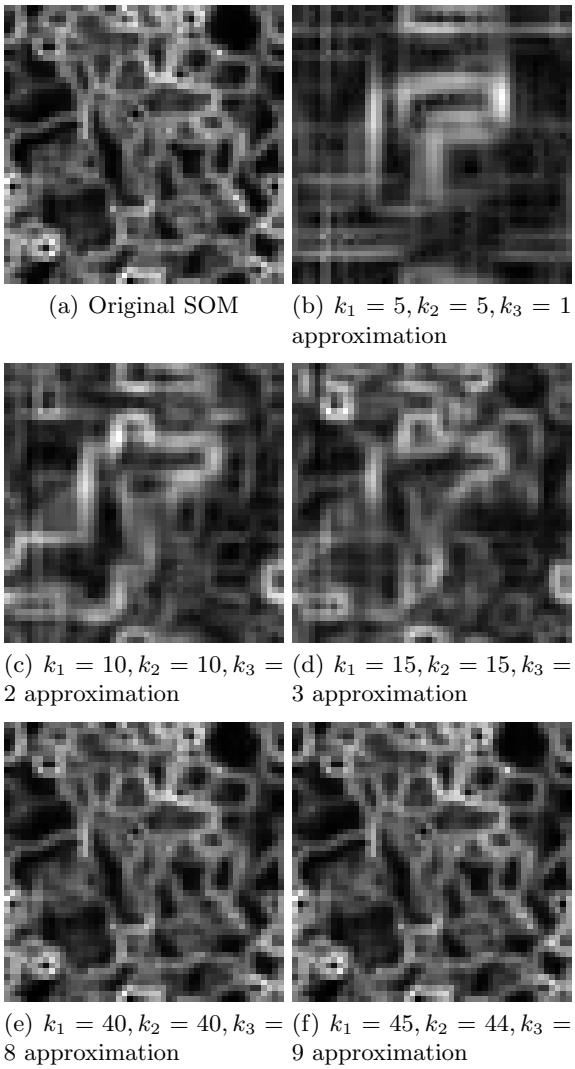
(a) Original SOM (b) $k = 1$ approximation

(c) $k = 2$ approximation (d) $k = 3$ approximation

(e) $k = 8$ approximation (f) $k = 9$ approximation

Fig. 4. SVD rank-k approximations of final SOM S (after 5000 iterations).

On the other hand performance of HOSVD approxiamtion of SOM S was very disappointing. U-matrices of approximated SOM are very similar to SVD ones, see Figure 6. Also cumulative histograms 5 of BMUs' movements show very similar characteristics. But when absolute values are compared, there are veryu high diffrences, see table 4 e.g. only 616 patterns were moved after 5,000 iterations in rank-9 SVD approximation, which is only 2.63 % of all input patterns. But in similar HOSVD approximation more than 9,000 patterns were moved. It is topic of our future work explain this discrepancy.

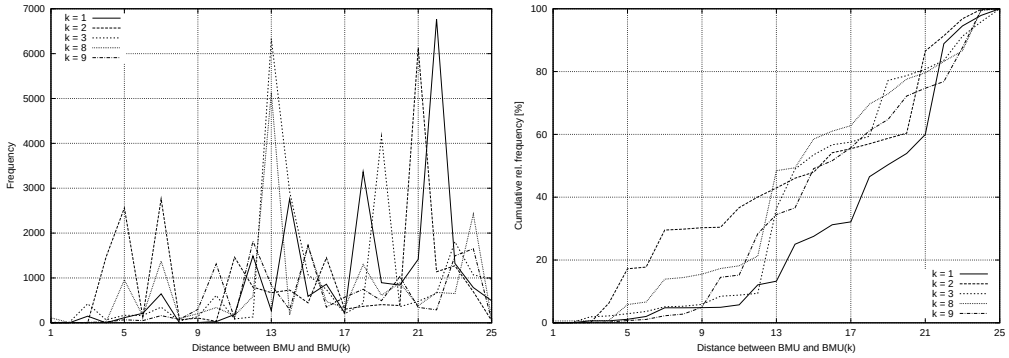


(a) Original SOM (b) $k_1 = 5, k_2 = 5, k_3 = 1$ approximation

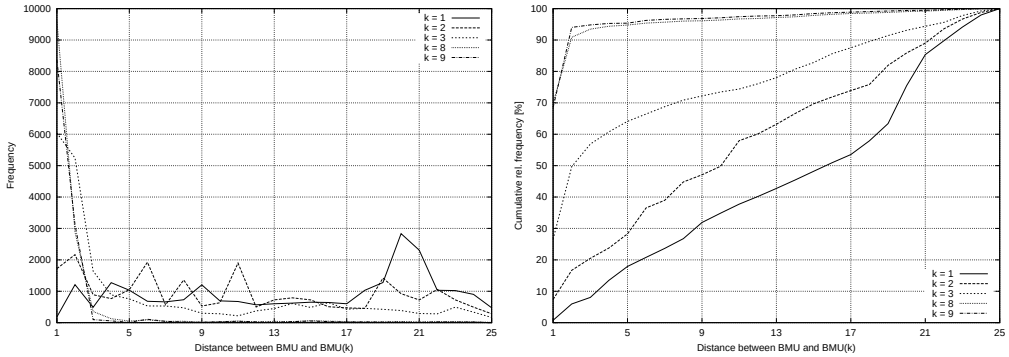
(c) $k_1 = 10, k_2 = 10, k_3 = 2$ approximation (d) $k_1 = 15, k_2 = 15, k_3 = 3$ approximation

(e) $k_1 = 40, k_2 = 40, k_3 = 8$ approximation (f) $k_1 = 45, k_2 = 44, k_3 = 9$ approximation

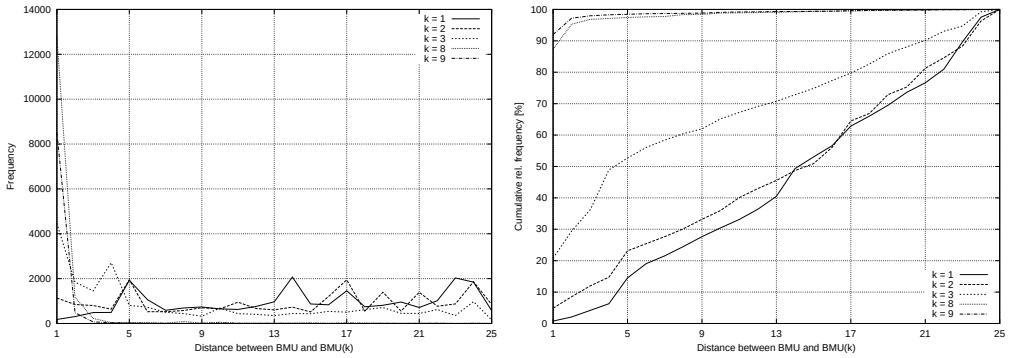
Fig. 6. $S(k_1, k_2, k_3)$ HOSVD approximations of final SOM S (after 5000 iterations).



(a) Initial SOM with random weights



(b) SOM after 4000 iterations



(c) SOM after 5000 iterations (final state)

Fig. 5. Frequency and cumulative frequency histograms of distances between BMU and BMU((k_1, k_2, k_3).

	Number of singular values k									
	1	2	3	4	5	6	7	8	9	10
	0 completed iterations									
MSE(k)	0.784	0.74	0.712	0.64	0.614	0.57	0.536	0.537	0.522	0.517
# of moved patterns	23,381	22,570	18,053	15,897	14,963	12,326	11,031	10,654	6,162	0
Moved patterns [%]	99.825	96.362	77.077	67.872	63.884	52.626	47.097	45.487	26.309	0
Max. distance	25	25	25	25	25	25	25	25	25	
Average distance	13.6	13.9	16.6	17.7	17.2	16.3	17	14.9	15.4	
	500 completed iterations									
MSE(k)	0.603	0.456	0.366	0.303	0.244	0.209	0.172	0.132	0.106	0.07
# of moved patterns	22,000	19,805	18,468	16,496	13,823	11,681	9,788	7,147	5,283	4
Moved patterns [%]	93.929	84.557	78.849	70.43	59.017	49.872	41.79	30.514	22.556	0.017
Max. distance	25	25	25	25	25	25	25	25	24	1
Average distance	14.3	10	9.2	6.6	6.4	5.9	5.5	5	4.8	1
	4000 completed iterations									
MSE(k)	0.595	0.452	0.356	0.284	0.236	0.184	0.143	0.092	0.04	0.015
# of moved patterns	20,643	16,252	13,698	11,964	9,353	8,185	4,907	3,925	1,865	0
Moved patterns [%]	88.135	69.388	58.483	51.08	39.933	34.946	20.95	16.758	7.963	0
Max. distance	25	25	25	25	25	25	25	25	24	
Average distance	11.4	8	6.9	5.8	4.8	3.8	4.2	3.7	4.7	
	5000 completed iterations									
MSE(k)	0.595	0.452	0.355	0.284	0.229	0.18	0.119	0.083	0.038	0.005
# of moved patterns	1,7748	10,760	7,555	6,868	5,058	3,781	2,564	2,125	616	1
Moved patterns [%]	75.775	45.94	32.256	29.323	21.595	16.143	10.947	9.073	2.63	0.004
Max. distance	25	25	25	25	25	25	25	25	25	1
Average distance	13.2	9.1	7.5	5.5	5	4	3.7	2.4	3	1

Table 2. Parameters of SVD rank-k approximations of given SOM.

	Approximation parameters (k_1, k_2, k_3)								
	(5, 5, 1)	(10, 10, 2)	(15, 15, 3)	(20, 20, 4)	(25, 25, 5)	(30, 30, 6)	(35, 35, 7)	(40, 40, 8)	(45, 45, 9)
	0 completed iterations								
MSE(k)	1.485	1.218	1.136	0.979	0.887	0.754	0.696	0.665	0.602
# of moved patterns	23,422	23,421	23,407	23,104	23,299	23,109	21,829	18,876	13,799
Moved patterns [%]	100	99.996	99.936	98.642	99.475	98.664	93.199	80.591	58.915
Max. distance	25	25	25	25	25	25	25	25	25
Average distance	18.4	14.6	16.3	16.7	15	15.9	15.9	15.4	16.7
	4000 completed iterations								
MSE(k)	0.595	0.452	0.356	0.285	0.237	0.184	0.143	0.092	0.04
# of moved patterns	23,404	23,304	22,722	21,671	20,141	18,890	17,265	13,761	12,218
Moved patterns [%]	99.923	99.496	97.011	92.524	85.992	80.651	73.713	58.752	52.165
Max. distance	25	25	25	25	25	25	25	25	25
Average distance	14.3	11	6.6	6.5	5	2.5	2.3	2	1.8
	5000 completed iterations								
MSE(k)	0.595	0.453	0.356	0.285	0.23	0.182	0.12	0.084	0.039
# of moved patterns	23,398	23,220	21,419	18,217	19,305	18,132	15,638	14,639	9,252
Moved patterns [%]	99.898	99.138	91.448	77.777	82.423	77.414	66.766	62.501	39.501
Max. distance	25	25	25	25	25	25	25	25	25
Average distance	14.6	13.7	8.6	6	3.4	2.2	1.8	1.4	1.3

Table 3. Parameters of $S(k_1, k_2, k_3)$ HOSVD approximations of given SOM S.

5 Conclusion

Alternative approach of quality measurement of SOM network learning process was presented. This approach uses SVD decomposition of the SOM network, and number of moved BMUs are counted among each consecutive rank-k approximations of original SOM network. Our experiments show some properties of SVD decomposition of SOM network. On the other hand High Order SVD cause some problems, because number of moved patterns is very high. We should check if this behavior is feature or bug of this method or if it is a matter of coincidence.

References

1. De Lathauwer L., Moor B.D., and Vandewalle J., A Multilinear Singular Value Decomposition. SIAM Journal on Matrix Analysis and Applications, 21, 4, 2000, 1253–1278
2. Golub G.H. and Loan C.F.V., Matrix Computations. Johns Hopkins University Press, 1996
3. Kohonen T., Self-Organizing Maps. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001
4. Kudelka M., Snasel V., Lehecka O., and El-Qawasmeh E., Semantic Analysis of Web Pages Using Web Patterns. 2006 IEEE/WIC/ACM International Conference on Web Intelligence, 0, 2006, 329–333
5. Linde Y., Buzo A., and Gray R.M., An Algorithm for Vector Quantiozation Desing. IEEE Transactions on Communications, COM-28, 1980, 84–95
6. Ultsch A., Self-Organizing Neural Networks for Visualization and Classification. In O. Opitz, B. Lausen, and R. Klar, eds, Information and Classification, London, UK, Springer, 1993, 307–313

Zložitosť vyhodnocovania zostupných XPath výrazov v kontexte sekvenčných XSLT transformácií *

Jana Dvořáková

Katedra informatiky, Fakulta matematiky, fyziky a informatiky
Univerzita Komenského, Bratislava
dvorakova@dcs.fmph.uniba.sk

Abstrakt *Pri XSLT transformáciach sa na lokalizáciu elementov vstupného XML dokumentu používajú XPath výrazy. Počas jedného kroku transformácie je potrebné vyhodnotiť a spracovať šablónu pozostávajúci z dvoch položiek - mena vstupného kontextového uzlu a postupnosti XPath výrazov. V tomto článku analyzujeme zložitosť vyhodnocovanie šablón v prípade, ktorý sa často vyskytuje v praxi, keď vstupný XML dokument je možné čítať iba sekvenčne a podobne výstupný dokument je možné generovať iba sekvenčne. Potom postupnosť XPath výrazov v šablóne určuje poradie spracovania elementov vstupného dokumentu, na ktoré odkazujú. Toto poradie je vo všeobecnosti permutáciou poradia, v akom sa elementy sprístupňujú sekvenčne. Pre šablónu navrhujeme prirodzené sekvenčné miery zložitosti (počet transformačných hláv, počet prechodov vstupného dokumentu), ktoré matematicky definujeme ako minimálne pokrytie takejto permutácie podpostupnosťami. Našim hlavným výsledkom je metóda pre výpočet rozsahu hodnôt sekvenčnej zložitosti pre šablóny obsahujúce zostupné XPath výrazy odkazujúce menom. Dôležitiou črtou výpočtu je nezávislosť na vstupnom dokumente, využíva sa iba informácia obsiahnutá vo vstupnej schéme.*

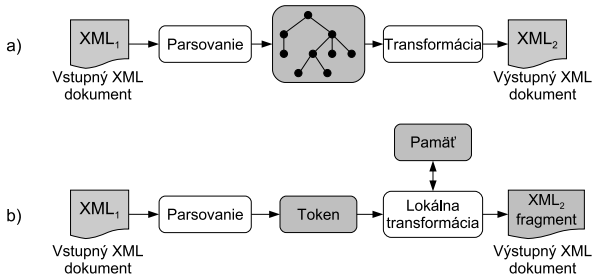
1 Úvod

XSLT [13] je populárny jazyk pre transformácie XML dokumentov, pre ktorý existuje množstvo implementácií. XSLT program sa skladá z podprogramov - šablón, z ktorých každá definuje jeden transformačný krok. Šablóny sa môžu v programe navzájom nepriamo volať. Jednoduchý príklad šablóny je uvedený nižšie.

```
<xsl:template match="a">
    <output>
    <xsl:apply-templates select="child::b">
    <xsl:apply-templates select="child::c">
    </output>
</xsl:template>
```

Atribút `match="a"` v hlavičke šablóny špecifikuje meno elementu, pri návšteve ktorého sa má šablóna aktivovať. Telo šablóny obsahuje dva konštantné reťazce `<output>` a `</output>` tvoriace časť výstupu a dve inštrukcie `apply-templates`, ktoré predstavujú volania pre spracovanie ďalších elementov vo vstupnom

* Táto práca bola podporená grantom GUK 358/2007.



Obrázok 1. (a) Stromové spracovanie a (b) sekvenčné spracovanie XSLT programu.

XML dokumente. Elementy sú adresované výrazmi jazyka XPath [12] uvedenými v atribúte `select`. Oba výrazy adresujú deti aktuálneho elementu - prvý deti s menom *b*, druhý deti s menom *c*. XSLT program sa vždy začína v koreňovom elemente vstupného dokumentu.

Stromové spracovanie. Klasické XSLT procesory spracovávajú XSLT programy *stromovo*, t.j. najskôr načítajú celý vstupný dokument do pamäte ako internú hierarchickú štruktúru, typicky DOM strom [11], a potom na tejto štruktúre aplikujú jednotlivé transformačné kroky (Obr. 1 (a)). V súčasnosti sa v praxi stále častejšie stretávame s XML dokumentami, pre ktoré tento prístup nie je vhodný. Jedná sa rozsiahle XML dokumenty (napr. výstupy z databáz), ktoré sa nemusia celé zmestiť do operačnej pamäte a ich stromové spracovanie vedie k nárastu využívania virtuálnej pamäte a spomaleniu procesu transformácie. Ako ukazujú výsledky experimentov [7], veľmi veľké dokumenty klasické procesory nedokážu spracovať vôbec. Pri XML dátových tokoch (napr. dáta z monitorovacích zariadení) takisto nie je vhodné použiť klasický prístup, nakoľko sa typicky vyžaduje spracovanie v reálnom čase.

Sekvenčné spracovanie. Tieto dôvody viedli k skúmaniu *sekvenčného spracovania* XML transformácií. Pri sekvenčnom prístupe sa vstupný dokument číta po častiach, ktoré sa hneď spracovávajú a transformujú na časti výstupu (Obr. 1 (b)). Kľúčovým problémom je stanovenie využitia prostriedkov počítača (najmä

operačnej pamäte) potrebných pre spracovanie rôznych tried XSLT programov.

V tomto článku najskôr definujeme abstraktné miery zložitosti sekvenčného spracovania a naším hlavným prínosom je uvedenie metódy pre výpočet hodnôt definovaných mier pre jednotlivé šablóny XSLT programu. Výpočet je založený na analýze obsahu šablóny a vstupnej XML schémy, popisujúcej štruktúru vstupných XML dokumentov.

Súvisiaci výskum. Väčšina predchádzajúcej práce bola zameraná na výskum sekvenčného vyhodnotenia samotných XPath výrazov [3,6,9,10]. Bolo implementových niekoľko procesorov pre sekvenčné spracovanie XML transformácií. Procesory pre jazyk XQuery [2,5,8] pokrývajú síce veľké množiny transformácií, ale autori neanalyzujú spotrebu operačnej pamäte pri spracovaní rôznych tried transformácií. Pokiaľ je nám známe, doteraz bol vyvinutý iba jeden sekvenčný procesor pre jazyk XSLT [7]. Procesor vôbec nevyužíva pamäť pre dočasné ukladanie dát, a preto je schopný spracovať iba veľmi jednoduché transformácie. Naše výsledky teda predstavujú krok ďalej, nakoľko analyzujeme komplexnejšie transformácie a ich nároky na využitie prostriedkov počítača.

2 Pojmy a označenia

Najskôr definujeme potrebné matematické pojmy a abstraktné modely. Abstrakcia XML dokumentu vychádza z modelu predstaveného v [1].

XML strom. XML dokument reprezentujeme prirodzene ako strom, ktorý vznikne mapovaním elementov na jednotlivé uzly. Množinu *XML stromov* nad Σ označujeme $\mathcal{T}_\Sigma$[1]. Nech $t \in \mathcal{T}_\Sigma$. Množinu všetkých *uzlov* t označujeme $Nodes(t) \subseteq (\mathbb{N}\{.\})^*$. Na identifikáciu uzlov používame dynamické hierarchické číslovanie, kde každý uzol má priradený reťazec čísel oddelených ".". Jednotlivé čísla označujú pozíciu uzla v danej úrovni stromu a v danej vetve, t.j., koreň je identifikovaný reťazcom 1, jeho deti reťazcami 1.1, 1.2, 1.3, ..., atď. (Obr. 2).

Postupnosti uzlov. Ďalej potrebujeme zaviesť niekoľko pojmov týkajúcich sa postupností uzlov. Poradie elementov v XML dokumente zodpovedá v stromovej reprezentácií postupnosti uzlov v preorderi. Reláciu usporiadania v preorderi na postupnosti uzlov označíme symbolom $\prec$. Pokrytie postupnosti uzlov s je množina podpostupností $\{s_1, \ldots, s_n\}$ taká, že každý uzol z s patrí práve do jednej z nich. Minimálne pokrytie je pokrytie minimálne na počet prvkov. Minimálne pokrytie postupnosti uzlov s rastúcimi podpostupnosťami označíme $MinCover(s, inc)$ a rastúcimi spojitými podpostupnosťami (behmi) $MinCover(s, run)$.

XSLT šablóna. XSLT šablóna je definovaná ako $(n+1)$-tica $tmp = (\sigma, exp_1, \ldots, exp_n)$, kde $\sigma \in \Sigma$ je meno kontextového uzlu a $exp_1, \ldots, exp_n$ je postupnosť XPath výrazov nad Σ. V tomto článku sa obmedzíme na zostupné XPath výrazy odkazujúce menom, pričom používame nasledujúcu zjednodušenú syntax: $exp = child[\alpha_1]/\ldots/child[\alpha_k]$, ktorá v XPath syntaxi zodpovedá výrazu

$$child::\alpha_1/child::\alpha_2 \ldots /child::\alpha_k.$$

Ako kontextový uzol uvažujeme koreň XML stromu, čo je pre naše účely postačujúce[2]. Nech t je XML strom nad Σ. Vyhodnocovacia funkcia pre XPath výraz exp v kontexte t je označená $eval(exp, t)$ a vracia postupnosť uzlov t v preorderi, ktoré zodpovedajú uzlom odkazovaným výrazom exp v strome t. Napr. ak $exp = child[\alpha]$ a $t = \sigma(\alpha, \beta, \alpha)$, potom $eval(exp, t) =$ 1.1, 1.3[3].

XML schéma. XML schéma nad Σ je z abstraktného pohľadu množina pomenovaných regulárnych výrazov nad Σ. V regulárnych výrazoch uvažujeme päť konštruktorov so štandardným významom: "|", ",", "*", "+", "?". Priorita konštruktorov je vyjadrená uzátvorkovaním. Inštanciami schémy sú XML stromy, ktoré spĺňajú štrukturálne podmienky špecifikované regulárnymi výrazmi. *Množinu inštancií* schémy xs označujeme $\mathcal{T}_{xs}$.

Pokiaľ nie je uvedené inak, v ďalšom texte uvažujeme XML stromy, XSLT šablóny a XML schémy nad ľubovoľnou, ale pevne danou abecedou Σ.

3 Sekvenčné spracovanie XSLT šablóny

Nech $tmp = (\sigma, exp_1, \ldots, exp_n)$ je XSLT šablóna. Šablóna sa počas transformácie aplikuje pri návšteve každého vstupného uzla pomenovaného σ. XPath výrazy $exp_1, \ldots, exp_n$ sa potom vyhodnocujú relatívne vzhľadom na tieto uzly. My sa v tomto článku zameriavame na zložitosť sekvenčného spracovania jednej aplikácie danej XSLT šablóny. Najskôr definujeme základný pojem potrebný pre jej určenie - permutáciu vyvolanú šablónou. Potom prestavíme abstraktný model pre sekvenčné spracovanie XSLT šablón a nakoniec ukážeme, že zložitosť sekvenčného modelu pre spracovanie danej šablóny sa dá definovať pomocou vyvolanej permutácie.

[1] Neuvažujeme ostatné prvky XML dokumentu, ako atribúty, dátové hodnoty, menné priestory.

[2] Stačí uvažovať taký podstrom vstupného stromu transformácie, na ktorom sa daná XSLT šablóna aplikuje.

[3] Kvôli prehľadnosti jednotlivé uzly v postupnosti oddeľujeme čiarkou.

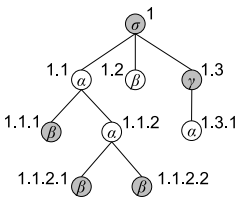
Permutácia vyvolaná šablónou. Keďže pri sekvenčnom spracovaní potrebujeme výstup transformácie generovať najskôr, ako je to možné (t.j. minimálnym využitím pamäte pre dočasné uloženie výstupu), uzly adresované XPath výrazmi v XSLT šablóne musia byť navštívene a potom spracované v poradí, v akom sa jeden po druhom vyhodnotia. Z hľadiska sekvenčného spracovania sú kritické práve situácie, keď postupnosť vyhodnotených uzlov nezodpovedá preorderovému poradiu, v akom sa vyskytujú vo vstupnom strome. V tomto prípade je potrebné využiť aj pamäť na dočasné uloženie vstupu resp. výstupu. *Permutáciu* vyvolanú šablónou tmp v kontexte t definujeme ako postupnosť uzlov vrátenú postupným vyhodnotením XPath výrazov $exp_1, \ldots, exp_n$.

$$Permutation(tmp, t) = \\ eval(exp_1, t)\, eval(exp_2, t)\ \ldots\ eval(exp_n, t).$$

Permutácia vyvolaná šablónou je teda explicitným vyjadrením poradia, v akom musia byť adresované uzly navštívené pri spracovaní vstupného stromu.

Príklad 1. Majme šablónu $tmp = (\sigma, exp_1, exp_2, exp_3, exp_4)$, kde
$exp_1 = child[\alpha]/child[\beta]$,
$exp_2 = child[\gamma]$,
$exp_3 = child[\alpha]/child[\alpha]/child[\beta]$,
$exp_4 = /$,
a XML strom $t = \sigma(\alpha(\beta, \alpha(\beta, \beta)), \beta, \gamma(\alpha))$ (Obr. 2). Potom
$eval(exp_1, t) = 1.1.1$
$eval(exp_2, t) = 1.3$
$eval(exp_3, t) = 1.1.2.1,\ 1.1.2.2$
$eval(exp_4, t) = 1$
Vybrané uzly je potrebné spracovať v poradí
$Permutation(tmp, t) = 1.1.1,\ 1.3,\ 1.1.2.1,\ 1.1.2.2,\ 1$

Sekvenčný model. Sekvenčný model predstavuje jednoduchú abstrakciu sekvenčného procesora, v rozšírenej podobe sme ho prestavili v [4]. Je definovaný ako šestica $M = (Q, \Sigma, \Delta, R, k, l)$, kde Q je množina stavov, Σ je vstupná abeceda, Δ je výstupná abeceda, R je množina pravidiel, k je počet hláv a l je počet prechodov ponad vstupný dokument. Pravidlá sekvenčného modelu sú tvaru



Obrázok 2. XML strom t z Príkladu 1. Uzly adresované XPath výrazmi šablóny tmp sú vyznačené sivou farbou.

$$(q, (\sigma_1, tag_1, la_1), \ldots, (\sigma_k, tag_k, la_k)) \rightarrow \\ str(q', m_1, \ldots, m_k)$$

kde q, q' sú aktuálny a nový stav, $\sigma_i \in \Sigma$ je meno vstupného elementu, $tag_i, la_i \in \{start, end\}$ je druh aktuálneho tagu a nasledujúceho tagu (lookahead tag), str je konštantný reťazec, ktorý tvorí časť výstupu a $m_i \in \{stay, advance, skip, restart\}$ sú akcie hláv s nasledovným významom:

- *stay* - hlava sa nehýbe
- *advance* - hlava sa posunie o jeden krok v preorderi,
- *skip* - hlava prejde celý podstrom pod aktuálnym uzlom (zodpovedá prechodu na koncový tag aktuálneho elementu),
- *restart* - hlava sa presunie späť na koreň (začiatok nového prechodu)

Konfigurácia sekvenčného modelu v kontexte t_{in} je $(k+1)$-tica

$$(q, (u_1, tag_1), \ldots, (u_k, tag_k), t_{out}),$$

kde q je aktuálny stav, dvojia (u_i, tag_i) vyjadruje pozíciu hlavy i a t_{out} je XML strom prestavujúci doteraz vygenerovaný výstup. Počiatočná konfigurácia v kontexte t_{in} je $(q_0, (1, start) \ldots, (1, start), \epsilon)$, koncová konfigurácia je $(q_0, (1, end) \ldots, (1, end), t_{out})$. Transformácia indukovaná M je množina dvojíc (t_{in}, t_{out}), kde t_{out} je výstup vygenerovaný prechodom z počiatočnej konfigurácie do koncovej konfigurácie v kontexte t_{in} a s využitím maximálne l reštartovacích pravidiel. Miery zložitosti sekvenčného modelu M sú

$$Heads(M) = k,\ Passes(M) = l.$$

Spracovanie šablóny sekvenčným modelom. Sekvenčný model budeme používať ako formalizmus pre sekvenčné spracovanie XSLT šablóny. Keďže uvažujeme šablóny bez generovania výstupu, na pravej strane pravidiel bude vždy iba stav a množinu inštrukcií pre pohyby hláv, výstupný reťazec bude prázdny. Aby sme vedeli identifikovať poradie nájdenia adresovaných uzlov, rozlišujeme pre každú hlavu i špeciálne stavy zhody $Q_{(match,i)} \subseteq Q$. Pokiaľ sa sekvenčný model nachádza v takomto stave, znamená to, že hlava i je nastavená vo vstupnom strome na niektorý z adresovaných uzlov. Sekvenčný model pre spracovanie XSLT šablóny tmp v kontexte t je potom taký model, ktorý počas výpočtu prejde postupne stavmi zhody $q_1, \ldots, q_m$, príslušné hlavy v stavoch zhody sú nastavené na uzly $u_1, \ldots, u_m$ a platí, že táto postupnosť uzlov je práve permutácia vyvolaná šablónou tmp na t. Množinu takýchto modelov označíme $\mathcal{M}_{(temp,t)}$ a konštrukciu demonštrujeme na príklade.

Príklad 2. Majme šablónu $tmp = (\sigma, exp_1, exp_2)$

a XML strom t, kde exp_1, exp_2, t sú definované ako v Príklade 1. Jednoprechodový jednohlavový sekvenčný model pre spracovanie tmp skonštruujeme nasledovne. $M = (Q, \Sigma, \Delta, q_0, R, 1, 1)$, kde
$Q = \{q_0, q_{(0,0)}, q_{(1,0)}, q_{(match,0)}, q_{(0,match)}\}$,
$Q_{(match,1)} = \{q_{(match,0)}, q_{(0,match)}\}$,
$\Sigma = \{\sigma, \alpha, \beta, \gamma\}$, $\Delta = \emptyset$, a R pozostáva z pravidiel:

Inicializácia:
$(q_0, \sigma, start, start) \to (q_{(0,0)}, advance)$,
$(q_0, \sigma, start, end) \to (q_0, advance)$

Vyhodnocovanie prvého kroku exp_1:
$(q_{(0,0)}, \alpha, start, la) \to (q_{(1,0)}, advance)$,
$(q_{(0,0)}, x, start, la) \to (q_{(0,0)}, skip), x \neq \alpha$,
$(q_{(0,0)}, x, end, start) \to (q_{(0,0)}, advance)$
$(q_{(0,0)}, x, end, end) \to (q_0, advance)$

Vyhodnocovanie druhého kroku exp_1:
$(q_{(1,0)}, \beta, start, la) \to (q_{(match,0)}, skip)$,
$(q_{(1,0)}, x, start, la) \to (q_{(1,0)}, skip), x \neq \beta$
$(q_{(1,0)}, x, end, start) \to (q_{(1,0)}, advance)$,
$(q_{(1,0)}, x, end, end) \to (q_{(0,0)}, advance)$,

Aktualizácia stavu po nájdení uzla adresovaného exp_1:
$(q_{(match,0)}, \beta, end, start) \to (q_{(1,0)}, advance)$,
$(q_{(match,0)}, \beta, end, end) \to (q_{(0,0)}, advance)$.

Vyhodnocovanie prvého kroku exp_2:
$(q_{(0,0)}, \beta, start, la) \to (q_{(0,match)}, skip)$,

Aktualizácia stavu po nájdení uzla adresovaného exp_l:
$(q_{(0,match)}, \beta, end, start) \to (q_{(0,0)}, skip)$,
$(q_{(0,match)}, \beta, end, end) \to (q_0, skip)$.

Sekvenčný model M vyhodnocuje obidva XPath výraz. Pri konštrukcii sme využili informáciu, že uzly adresované exp_1 sa v t nachádzajú pred uzlami adresovanými exp_2 (vzhľadom na preorder), inak by poradie navštívenia adresovaných uzlov nemuselo zodpovedať permutácií vyvolanej tmp na t. Samotné pravidlá však nie sú závislé na vstupe.

Pre šablónu $tmp = (\sigma, exp_1, \ldots, exp_n)$ má sekvenčný model počiatočný stav q_0 a ďalej stavy tvaru $q_{(s_1,\ldots,s_n)}$, kde s_i môže mať hodnoty $\{0, \ldots, steps_i - 1, match\}$, $steps_i$ je počet lokalizačných krokov v XPath výraze exp_i. Hodnota s_i indikuje, aká časť výrazu exp_i už bola vyhodnotená.

Sekvenčná zložitosť šablóny. Vo všeobecnosti XSLT šablónu nie je možné spracovať pomocou jednoduchého jednohlavového a jednoprechodového sekvenčného modelu ako v Príklade 2. V nasledovnom texte sa zameriame na hľadanie metódy pre výpočet sekvenčnej zložitosti šablóny, t.j., výpočet minimálneho počtu hláv a počtu prechodov, ktoré musí mať sekvenčný model pre jej spracovanie.

$$Heads(tmp, t) = min\{Heads(M) \mid M \in \mathcal{M}_{(tmp,t)}, Passes(M) = 1\}$$

= minimálny počet hláv potrebný na sekvenčné spracovanie tmp v kontexte t v jedinom prechode,

$$Passes(tmp, t) = min\{Passes(M) \mid M \in \mathcal{M}_{(tmp,t)}, Heads(M) = 1\}$$

= minimálny počet prechodov potrebný na sekvenčné spracovanie tmp v kontexte t pomocou jedinej hlavy.

Ukážeme, že sekvenčnú zložitosti vieme jednoducho vypočítať analýzou XSLT šablóny a vstupného dokumentu, pričom využijeme permutáciu vyvolanú šablónou. Pozrime sa bližšie na spôsob práce sekvenčného modelu.

Viachlavový jednoprechodový model. Pri konštrukcii viachlavového modelu potrebujeme poznať priradenie XPath výrazov jednotlivým hlavám, t.j. pre každú hlavu množinu XPath výrazov, ktoré ňou budú vyhodnotené. Na začiatku transformácie sú hlavy nastavené na koreň a počas aplikácie transformačných krokov sa každá môže hýbať iba v preorderi alebo nehýbať vôbec. Zrejme pokiaľ permutácia vyvolaná šablónou tmp je rastúca postupnosť, vieme ju spracovať jednohlavovým sekvečným modelom. Akonáhle však permutácia obsahuje nejakú inverziu, potrebujeme viac hláv, pretože jedna hlava nie je schopná navštíviť dva uzly v opačnom poradí, ako je ich poradie vo vstupnom dokumente. Každej hlave teda môže byť priradená na vyhodnotenie množina XPath výrazov, ktorá vracia rastúcu postupnosť uzlov vzhľadom na ich preorderové usporiadanie vo vstupnom strome.

Viacprechodový jednohlavový model. Situácia je podobná ako v prípade hláv - v jednom prechode vieme jednou hlavou spracovať podpostupnosť uzlov, ktorá musí byť rastúca. Rozdielom je, že podpostupnosť musí byť aj spojitá, pretože v prípade viachlavého modelu mohli hlavy navzájom komunikovať, avšak v prípade viacprechodového modelu možnosť komunikácie medzi jednotlivými prechodmi neexistuje.

Dostávame teda, že obe definované miery zložitosti môžeme vypočítať ako minimálne pokrytia permutácie vyvolanej šablónou, pričom v prípade počtu hláv je to pokrytie rastúcimi podpostupnosťami a v prípade počtu prechodov pokrytie rastúcimi a spojitými podpostupnoťami, teda behmi.

$$Heads(tmp, t) = MinCover(p, inc),$$
$$Passes(tmp, t) = MinCover(p, run),$$

kde $p = Permutation(tmp, t)$. Permutácia p sa dá určiť priamo vyhodnotením XPath výrazov z tmp v kontexte t. Potom je už iba potrebné vypočítať minimálne pokrytie p rastúcimi podpostupnosťami, resp. behmi a dostaneme výslednú sekvenčnú zložitosť.

Príklad 3. V Príklade 1 permutáciu vyvolanú šablónou tmp v kontexte t tvorila postupnosť uzlov

1.1.1, 1.3, 1.1.2.1, 1.1.2.2, 1

Túto postupnosť vieme minimálne pokryť rastúcimi podpostupnosťami {(1.1.1, 1.3), (1.1.2.1, 1.1.2.2, 1)}, ktoré sú spojité. Sekvečná zložitosť spracovania šablóny je teda $Heads(tmp, t) = Passes(tmp, t) = 2$.

Samostatný XPath výraz sa vždy dá spracovať jednou čítacou hlavou v jednom prechode. Preto hornou hranicou pre $Heads(tmp, t)$ aj $Passes(tmp, t)$ je počet XPath výrazov v dotaze *tmp*. Beh je vždy rastúca postupnosť, ale rastúca postupnosť nemusí byť nutne behom. Preto počet hláv potrebný pre spracovanie dotazu *tmp* tvorí dolnú hranicu počtu prechodov potrebného pre spracovanie *tmp*.

$$Heads(tmp, t) \leq Passes(tmp, t) \leq n,$$

kde n je počet XPath výrazov v *tmp*.

4 Výpočet sekvenčnej zložitosti vzhľadom na vstupnú schému

Uvedený postup výpočtu sekvenčnej zložitosti pre šablóny ma jednu podstatnú nevýhodu. Keďže v praxi sekvenčne typicky spracovávame veľké dokumenty alebo dátové toky, je žiadúce, aby sme vedeli vypočítať sekvenčnú zložitosť šablón nezávisle na vstupnom dokumente. Pokiaľ by sme uvažovali XPath výrazy odkazujúce indexom, situácia by bola jednoduchá - permutácia vyvolaná dotazom by bola v každom kontexte rovná postupnosti indexov zoradených v poradí výskytu v jednotlivých výrazoch. My sa však zaoberáme XPath výrazmi, ktoré odkazujú na vstupné uzly menom. Ako vidno z nasledovného príkladu, v tomto prípade môžeme pre rôzne kontexty dostať rôzne permutácie a hodnoty sekvenčnej zložitosti.

Príklad 4. Majme šablónu $tmp = (\sigma, child[\beta], child[\alpha])$ a XML stromy $t_1 = \sigma(\beta, \alpha)$ a $t_2 = \sigma(\alpha, \beta)$. Potom

$$Permutation(tmp, t_1) = 1.1,\ 1.2 \neq$$
$$Permutation(tmp, t_2) = 1.2,\ 1.1$$

Prirodzeným riešením, ktoré navrhujeme, je využitie informácie o štruktúre vstupného dokumentu obsiahnutú vo vstupnej schéme. XML schéma je typicky malá v porovnaní s veľkosťou vstupných XML dokumentov a jej analýzou vieme pre danú šablónu určiť množinu možných permutácií a presný rozsah hodnôt pre sekvenčnú zložitosť. Vo všeobecnom XPath existujú konštrukcie, pri ktorých sa množina permutácií nedá určiť ani za pomoci vstupnej schémy, vo väčšine prípadov sa im však dá vyhnúť.

V nasledovnom texte uvažujeme ľubovoľnú, ale pevne danú šablónu $tmp = \{\sigma, exp_1, \ldots, exp_n\}$, XML strom t a XML schému xs. Z dôvodu obmedzeného priestoru sa zameriavame na výpočet rozsahu hodnôt pre $Heads(tmp, t)$.

Sekvenčná zložitosť vzhľadom na schému. Minimálny a maximálny počet hláv potrebných na spracovanie *tmp* nad xs definujeme

$$MinHeads(tmp, xs) =$$
$$\quad min\{Heads(tmp, t) \mid t \in \mathcal{T}_{xs}, root(t) = \sigma\},$$
$$MaxHeads(tmp, xs) =$$
$$\quad max\{Heads(tmp, t) \mid t \in \mathcal{T}_{xs}, root(t) = \sigma\}.$$

Naším cieľom je vypočítať obe hodnoty. Výpočet priamo podľa definície nie je možný, pretože schéma xs môže mať vo všeobecnosti nekonečne veľa inštancií. Konečný počet inštancií majú iba schémy bez konštruktorov "*ä "+". Jadrom našej metódy výpočtu je modifikácia danej schémy xs na *ohraničenú schému* xs_r s konečným počtom inštancií, pričom platí, že sekvenčná zložitosť *tmp* nad xs_r je rovnaká ako sekvečná zložitosť *tmp* nad xs.

Konštrukcia ohraničenej schémy. Ohraničenú schému xs_r získame z pôvodnej schémy substitúciou podvýrazov:

1. podvýraz $(r)^+$ nahradíme podvýrazom $(r)^n$,
2. podvýraz $(r)^*$ nahradíme podvýrazom $(r)^n$,

kde n je počet XPath výrazov v šablóne xs. Stačí nám ukázať, že pre takto zostrojenú schému platí

$$MinHeads(tmp, xs_r) = MinHeads(tmp, xs)$$
$$MaxHeads(tmp, xs_r) = MaxHeads(tmp, xs)$$

Prvá časť tvrdenia je zrejmá. Iterácie nám iba pridávajú ďalšie podstromy do XML inštancií a teda vkladajú podpostupnosti do permutácií vyvolaných šablónou *tmp*. Permutácia vyvolaná na XML strome z $\tau_{xs} - \tau_{xs_r}$ je teda vždy nadpostupnosťou permutácie vyvolanej na niektorom z XML stromov patriacich do τ_{xs_r}. Minimálne pokrytie nadpostupnosti má vždy väčší alebo rovnaký počet prvkov ako minimálne pokrytie samotnej postupnosti z čoho jasne vyplýva platnosť rovnosti. Náročnejšiou úlohou je ukázať platnosť druhej rovnosti. Nebudeme uvádzať formálny dôkaz, ktorý je značne technický, ale iba pomocou príkladu načrtneme jeho základnú ideu.

Príklad 5. Majme šablónu $tmp = (\sigma, exp_1, exp_2, exp_3)$, kde

$$exp_1 = child[\alpha], exp_2 = child[\beta], exp_3 = child[\gamma].$$

Ďalej majme XML schému xs obsahujúcu jediné pravidlo $\sigma \rightarrow (\alpha, \beta, \gamma)^*$. XPath výrazy z *tmp* vyhodnotíme v kontexte tých XML inštancií schémy xs, ktoré vznikli jednou, dvomi, resp. tromi iteráciami podvýrazu α, β, γ v pravidle schémy.

$t_1 = \sigma(\alpha\beta\gamma):$ $eval(exp_1, t_1) = 1.1,$
$eval(exp_2, t_2) = 1.2,$
$eval(exp_3, t_3) = 1.3,$

$t_2 = \sigma(\alpha\beta\gamma\alpha\beta\gamma),:$ $eval(exp_1, t_1) = 1.1, 1.4,$
$eval(exp_2, t_2) = 1.2, 1.5,$
$eval(exp_3, t_3) = 1.3, 1.6,$

$t_3 = \sigma(\alpha\beta\gamma\alpha\beta\gamma\alpha\beta\gamma),:$ $eval(exp_1, t_1) = 1.1, 1.4, 1.7,$
$eval(exp_2, t_2) = 1.2, 1.5, 1.8,$
$eval(exp_3, t_3) = 1.3, 1.6, 1.9.$

Určíme minimálne pokrytia permutácií vyvolaných tmp v kontexte stromov t_1, t_2, t_3 a počet hláv potrebný pre sekvenčné spracovanie.

$MinCover(tmp, t_1) = \{(1.1, 1.2, 1.3)\},$
$MinCover(tmp, t_2) = \{(1.1, 1.2, 1.3), (1.4, 1.5, 1.6)\},$
$MinCover(tmp, t_3) =$
$\{(1.1, 1.2, 1.3), (1.4, 1.5, 1.6), (1.7, 1.8, 1.9)\} \cup$
$\{(1.1, 1.4, 1.7), (1.2, 1.5, 1.8), (1.3, 1.6, 1.9)\},$

$Heads(tmp, t_1) = 1,$
$Heads(tmp, t_2) = 2,$
$Heads(tmp, t_3) = 3.$

Pre minimálne pokrytie permutácie vyvolanej tmp v kontexte t_3 máme dve možnosti. Ľahko vidno, že druhá možnosť pokrytia podpostupnosťami sa dá analogicky použiť pre permutácie v kontexte XML inštancií s počtom iterácií väčším ako 3. Ďalšie adresované uzly sa budú jednoducho pridávať na konce pospotupností v minimálnom pokrytí pre t_3. Preto sa počet hláv potrebný na spracovanie tmp pri ďalších iteráciach už nezvýši.

Myšlienku načrtnutú v príklade je možné zovšeobecniť a tvorí jadro dôkazu platnosti tvrdenia pre výpočet maximálneho počtu hláv potrebných na spracovanie šablóny nad XML schémou.

5 Záver

V článku sme analyzovali zložitosť vyhodnocovania zostupných XSLT šablón počas sekvenčných XSLT transformácií. Vytvorili sme vhodný matematický základ - sekvenčné miery zložitosti spracovania XSLT šablóny sme definovali prirodzeným spôsobom ako minimálne pokrytia permutácie elementov vstupného XML dokumentu vyvolané dotazom. Uviedli sme metódu pre výpočet sekvenčnej zložitosti, ktorá je nezávislá na vstupnom dokumente, pri výpočte využíva iba informáciu o štruktúre vstupného dokumentu zo vstupnej schémy. Metódu sme implementovali a začlenili do prototypu sekvenčného XSLT transformátora.

V článku sme uvažovali viaceré obmedzenia kladené jednak na XML dokument (uvažovali sme iba elementy), XPath výrazov (uvažovali sme podmnožinu zostupných výrazov), ako aj XSLT šablóny. Použité abstraktné modely však zachytávajú základné problémy vyhodnocovania XSLT šablón v kontexte sekvenčných XSLT transformácií a ako nám ukázala praktická skúsenosť, mnohé z obmedzení môžu byť preklenuté jednoduchými modifikáciami algoritmu.

V ďalšej práci sa plánujeme zaoberať sekvenčným vyhodnocovaním XSLT šablón obsahujúcich spätné osi, pričom plánujeme využiť známe metódy pre prepisovanie spätných osí na dopredné [6,9], a takisto postupným vyhodnocovanim skupiny XSLT šablón, t.j. kompletného XSLT programu.

Referencie

1. G. J. Bex, S. Maneth and F. Neven, A Formal Model for an Expressive Fragment of XSLT. In Information Systems, 27, 1, 2002, 21–39
2. F. Bry, F. Coskun, S. Durmaz, T. Furche, D. Olteanu and M. Spannagel, The XML Stream Query Processor SPEX. In Proceedings of ICDE'05, 2005, 1120–1121
3. Y. Chen, S. B. Davidson, and Y. Zheng, An Efficient XPath Query Processor for XML Streams. In Proceedings ICDE'06, 79, 2006
4. J. Dvořáková and B. Rovan, A Transducer-Based Framework for Streaming XML Transformations. SOFSEM 2007, 2007
5. D. Florescu, C. Hillery, D. Kossmann, P. Lucas, F. Riccardi, T. Westmann, M. J. Carey, A. Sundararajan and G. Agrawal: The BEA/XQRL Streaming XQuery Processor. In VLDB 2003, 2003, 997–1008
6. P. Genevès and K. Rose, Compiling XPath for Streaming Access Policy. In DOCENG'05, 2005, 52–54
7. Z. Guo, M. Li, X. Wang and A. Zhou, Scalable XSLT Evaluation. In Proceedings of APWeb 2004, LNCS 3007/2004, Springer, 2004, 190–200
8. B. Ludäscher, P. Mukhopadhyay and Y. Papakonstantinou, A Transducer-Based XML Query Processor. Proceedings of VLDB 2002, 2002, 227–238
9. D. Oltenau, H. Meuss, T. Furche and F. Bry, XPath: Looking Forward. In XMLDM 2002, 2002, 109–127
10. F. Peng and S. S. Chawathe, XPath Queries on Streaming Data. In Proceedings of SIGMOD 2003, 2003
11. W3C. Document Object Model (DOM), `http://www.w3.org/DOM"`
12. W3C. XML Path Language (XPath), version 1.0, W3C Recommendation, 1999. `http://www.w3.org/TR/xpath`
13. W3C. XSL Transformations (XSLT) Version 1.0, W3C Recommendation, 1999. `http://www.w3.org /TR/xslt/`

Geometry in the data space*

Zdeněk Fabián

Ústav informatiky AVČR, Praha
zdenek@cs.cas.cz

Abstract. *We introduced in [1] new characteristics of probability distributions. The central tendency of a distribution is characterized by the Johnson mean. We discuss its estimation and construction of confidence bounds.*

1 Inference function

The usual parametric model of observed data $x_1, ...x_n$ is a parametric family $\{F_\theta, \theta \in \Theta\}$ supported by $\mathcal{X} \subseteq \mathbb{R}$ with parameter $\theta = (\theta_1, ..., \theta_m), \Theta \subseteq \mathbb{R}^m$. Data are considered to be a random sample from F_{θ_0}, i.e., a realization of independent random variables $X_1, ...X_n$ identically distributed according F_{θ_0} with unknown vector parameter θ_0.

Two questions arise:

i/ How to estimate θ_0 (to know, approximately, the underlying distribution F_{θ_0})?

ii/ How to characterize the data by a few numbers (to use them, perhaps, in further processing)?

Let us discuss the solutions consisting of choosing a suitable inference function Q and a study of averages of the type $\frac{1}{n}\sum_{i=1}^n Q(x_i;\theta)$.

If Q is the identity function, the solution of both problems is easy. Supposing that $\theta_0 = (\mu, \sigma)$ where μ is the mean and σ^2 the variance of the underlying distribution, a 'center' of the data is the average $\bar{x}$: $\frac{1}{n}\sum_{i=1}^n (x - \bar{x}) = 0$ and a measure of dispersion of the values around it is the sample variance $\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^n (x - \bar{x})^2$. However, both the mean and variance are not good characteristics of skewed distributions and, moreover, they may not exist for some 'heavy-tailed' distributions with densities slowly decaying to zero for which the integrals defining moments may not converge. In such cases, $\bar{x}$ and $\hat{\sigma}^2$ are not appropriate characteristics of the data.

The inference function of the classical statistics is a vector-function $\mathbf{U} = (U_1(x;\theta), ..., U_m(x;\theta))$, where, for j=1,...,m,

$$U_j(x) = \frac{\partial}{\partial \theta_j} \log f(x;\theta)$$

are the partial scores for the component θ_j. The system of so called maximum likelihood equations

$$\frac{1}{n}\sum_{i=1}^n U_j(x_i;\theta) = 0, \qquad j = 1, ..., m \tag{1}$$

gives estimates $\hat{\theta}_n = (\hat{\theta}_1, ..., \hat{\theta}_m)$ of θ_0 with in a certain sense best properties. However, with exception of one-parameter models, the pseudometric introduced in the sample space by vector function $\mathbf{U}$ is too complicated to offer simple characteristics of a 'center' and variability of the data.

Inference functions of the robust statistics are called 'psi-functions'; they are to a certain extent arbitrary bounded functions, making possible to introduce location and scale parameters, not taking into account the actual parameters of the underlying distribution. Thus, for instance, the M-estimate $\hat{\mu}$ of location μ is the solution of equation

$$\frac{1}{n}\sum_{i=1}^n \psi_j(x_i - \mu) = 0.$$

Such $\hat{\mu}$ is shown to be under reasonable regularity conditions consistent ($E\hat{\mu} = \mu$) and its distribution asymptotically (that is, for large n) normal, $N(\mu, v^2/n)$, where

$$v^2 = \frac{E\psi^2}{\left[E\left(\frac{\partial \psi(x-\mu)}{\partial \mu}\right)\right]^2} \tag{2}$$

(see e.g. ([5])). The boundedness of inference function ψ guarantees the suppression of the influence of possible outlier observations. However, the approach does not take into account possible prior knowledge of the underlying distribution.

It was shown in [1] that any continuous probability distribution with arbitrary interval support $\mathcal{X} \in \mathbb{R}$ can be characterized, besides the distribution function $F(x)$ and density $f(x)$, by its Johnson score $S(x)$ ([1], Definition 1). Instead of the usual moments, we can use the Johnson score moments

$$ES^k = \int_{\mathcal{X}} S^k(x) f(x)\, dx, \qquad k = 1, 2, ... \tag{3}$$

It was shown that $ES = 0$ and it can be shown that $ES^k < \infty$ if $ES^2 < \infty$ (the last condition is equivalent

* The work was supported by GA ASCR under grant No.1ET 400300513.

to the usual regularity requirements). Thus, Johnson score is a scalar function characterizing the distribution and giving a possibility to find simple characteristics of distributions.

2 Johnson data characteristics

As a 'center' of the distribution we find ([1]) the Johnson mean $x^* : S(x) = 0$ and as a measure of variability of values around x^* the Johnson variance, we introduce an analogy of (2),

$$\omega^2 = \frac{ES^2}{(ES')^2}. \tag{4}$$

For distributions with Johnson parameter, (4) is equal to the suggestion given in [1], for other distributions, (4) is better.

Densities, Johnson scores, Johnson means and Johnson variances of distributions presented below as examples are given for reference in Table 1. Apart from the normal distribution, the support of all other distributions in Table 1 is $\mathcal{X} = (0, \infty)$.

Distribution	$f(x)$	$S(x)$	x^*	ω^2
normal	$\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$	$\frac{x-\mu}{\sigma^2}$	μ	σ^2
lognormal	$\frac{\beta}{\sqrt{2\pi}x} e^{-\frac{1}{2}\log 2(\frac{x}{t})^\beta}$	$\frac{\beta}{t}\log(x/t)^\beta$	t	t^2/β^2
Weibull	$\frac{\beta}{x}\left(\frac{x}{t}\right)^\beta e^{-(\frac{x}{t})^\beta}$	$\frac{\beta}{t}[(x/t)^\beta - 1]$	t	t^2/β^2
gamma	$\frac{\gamma^\alpha}{x\Gamma(\alpha)} x^\alpha e^{-\gamma x}$	$\gamma(\frac{x}{\alpha/\gamma} - 1)$	α/γ	α/γ^2
inv. gamma	$\frac{\gamma^\alpha}{x\Gamma(\alpha)} x^{-\alpha} e^{-\gamma/x}$	$\alpha(1 - \frac{\gamma/\alpha}{x})$	γ/α	γ^2/α^3
beta-prime	$\frac{1}{xB(p,q)} \frac{x^p}{(x+1)^{p+q}}$	$\frac{q}{p}\frac{qx-p}{x+1}$	p/q	$\frac{p(p+q)^2}{q^3(p+q+1)}$

Table 1. Some distributions and their characteristics.

Fig.1 shows densities and Johnson scores of three inverse gamma distributions with $\gamma = \alpha = 0.6(1), 1$ and $1.5(3)$. The densities are heavy-tailed so that extremely large values can be observed. Johnson scores are bounded in infinity, so that averages of $S(x_i)$ are not sensitive to large x_i (on the other hand, it is apparent from the figure that the averages of $S(x_i)$ can be heavily influenced by x_i near zero). The mean of distribution 1 and 2 do not exist, the mean of distribution 3 is denoted by the star. All three distributions have the same Johnson mean $x^* = 1$, which seems to give a reasonable description of the position of a distribution on the x-axis.

Unlike the usual moments, the sample versions of the Johnson score moments cannot be determined without an assumption about the underlying distribution family. On the other hand, by substituting the empirical distribution function into (3), the resulting system of equations,

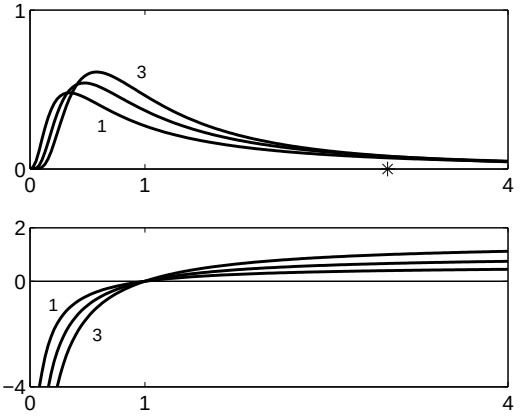


Fig. 1. Densities and Johnson scores of inverse gamma distributions.

$$\frac{1}{n}\sum_{i=1}^{n} S^k(x_i;\theta) = ES^k(\theta), \qquad k = 1, ..., m \tag{5}$$

appears to be an alternative to system (1). The estimates $\hat{\theta}_n$ from (5) are M-estimates so that they are for large n approximately normally distributed with mean θ_0 and variance given by expression similar to (2). The variances are slightly larger than that of the maximum likelihood estimates, but the estimates are robust in cases of heavy tailed distributions.

Generally, Johnson mean $x^* = x^*(\theta_1, ..., \theta_m)$. If $\theta_j = const.$ for $j = 2, ..., m$ so that $x^* = x^*(\theta_1)$ or if $x^* = \varphi(\theta_2/\theta_1)$ and $\theta_j = const.$ for $j = 3, ..., m$, the first equation of system (5) can be written in the form

$$\sum_{i=1}^{n} S(x_i; x^*) = 0, \tag{6}$$

where x^* plays the role of a parameter. For example, in case of the beta-prime distribution,

$$S(x; p, q) = \frac{q}{p}\frac{qx - p}{x_i + 1} = \frac{q^2}{p}\frac{x - x^*}{x + 1} \tag{7}$$

so that (6) has a form

$$\sum_{i=1}^{n} \frac{x_i - x^*}{x_i + 1} = 0. \tag{8}$$

Let us call the estimate $\hat{x}_n^*$ of x^* a *sample Johnson mean*. Since $\hat{x}_n^*$ is an M-estimate, we have immediately the following result:

Theorem 1. *The sample Johnson mean $\hat{x}_n^*$ determined from (6) is approximately distributed by $N(x^*, \omega^2/n)$ where ω^2 is given by (4).*

3 Johnson distance

Definition 1 *Let S be Johnson score of distribution F with support $\mathcal{X}\subseteq\mathbb{R}$. A Johnson difference of $x_1, x_2\in\mathcal{X}$ from distribution F is*

$$d(x_1, x_2) = S(x_2) - S(x_1). \tag{9}$$

If the Johnson score is written as $S(x; x^*)$, $d(x_1, x_2) = S(x_2; x^*) - S(x_1; x^*)$. In what follows, $S'(x^*; x^*)$ means $(dS(x; x^*)/dx)|_{x=x^*}$. We approach to the main result of this paper.

Theorem 2. *Let $\hat{x}^*$ be the estimate of the Johnson mean x^* of distribution F from (6). Then $d(\hat{x}_n^*, x^*)$ is approximately $N(0, [S'(x^*; x^*)]^2\omega^2/n)$.*

Proof follows from Theorem 1 and so called invariance principle saying that if $\sqrt{n}(\hat{\theta}_n^*-\theta)\to N(0,\sigma^2)$ as $n\to\infty$ in distribution and ϕ is continuous, then

$$\sqrt{n}(\phi(\hat{\theta}_n^*) - \phi(\theta^*)) \to N(0, [\phi'(\theta^*)]^2\sigma^2)$$

as $n\to\infty$ in distribution.

By Theorem 2, the approximate $(100-\alpha)\%$ confidence bounds for Johnson mean can be determined from condition

$$D(\hat{x}_n^*, x^*) = \frac{|d(\hat{x}_n, x^*)|}{S'(\hat{x}_n^*; \hat{x}_n^*)\hat{\omega}_n} = \frac{|S(\hat{x}_n^*, x^*)|}{S'(\hat{x}_n^*; \hat{x}_n^*)\hat{\omega}_n} \le \lambda_n \tag{10}$$

where $\lambda_n = u_{\alpha/2}/\sqrt{n}$ and $u_{\alpha/2}$ is the $(\alpha/2)$-th quantile of the normal distribution (in (10), we used relation $S(\hat{x}_n^*; \hat{x}_n^*) = 0$ and replaced ω by its estimate).

4 Examples

As an example of a procedure taking into account the geometry in the sample space of the given (assumed) distribution we determine the confidence intervals for Johnson mean of distributions from Table 1. We work with simplified formulas for Johnson scores without constant multiplicative terms, since they are cancelled in (10).

In case of the normal distribution, the first two Johnson moment equations (5) are identical with the maximum likelihood equations (1) so that $\hat{x}_n^* = \bar{x}$ and $\hat{\omega}^2 = \hat{\sigma}^2$. For large n, the approximate $(100-\alpha)\%$ confidence interval is, as usually,

$$(\bar{x} - \lambda_n\hat{\sigma}_n, \bar{x} + \lambda_n\hat{\sigma}_n)$$

(in this case we know, of course, the exact distribution of $(x-\bar{x})/\hat{\sigma}$).

Sample Johnson mean of distributions with Johnson parameter t is $\hat{x}_n^* = \hat{t}_n$ and its asymptotic variance $\omega^2 = 1/ES^2$. In case of the lognormal distribution, $S(x,t) = \log(x/t)$, by (6) $\hat{t}_n = \frac{1}{n}\sum_{i=1}^{n}\log x_i$, $D(\hat{t}_n, t) = |S(\hat{t}_n, t)|/(1/t * t/\beta)$ and the approximate $(100-\alpha)\%$ Johnson confidence interval for x^* is

$$(\hat{t}_n e^{-\lambda_n/\beta}, \hat{t}_n e^{\lambda_n/\beta}).$$

From the second eq. of (5), $\hat{\beta}_n^2 = n/\sum_{i=1}^{n}\log^2(x_i/\hat{t}_n)$. In case of the Weibull distribution, $S(x;t) = (x/t)^\beta - 1$, $\hat{t}_n = (\frac{1}{n}\sum_{i=1}^{n} x_i^\beta)^{1/\beta}$, which is the β-th mean, and, since $D(\hat{t}_n, t) = |S(\hat{t}_n, t)|/(\beta/t * t/\beta)$, the approximate $(100-\alpha)\%$ Johnson confidence interval for x^* is

$$(\hat{t}_n(1-\lambda_n)^{1/\beta}, \hat{t}_n(1+\lambda_n)^{1/\beta}).$$

Let us determine confidence intervals for distributions without Johnson parameter. In case of the gamma distribution, $S(x, x^*) = \gamma x - \alpha$ so that $\hat{x}_n^* = \alpha/\gamma = \bar{x}$. From (10) we obtain $D(\bar{x}, x^*) = \sqrt{\hat{\alpha}_n}|x^*/\bar{x} - 1|$ and the approximate $(100-\alpha)\%$ Johnson confidence intervals is

$$(\bar{x} - \lambda_n\rho_n, \bar{x} + \lambda_n\rho_n)$$

where $\rho_n^2 = 1/\hat{\alpha}_n = \frac{1}{n}\sum_{i=1}^{n} x_i^2/\bar{x}^2 - 1$ (it follows from the second eq. of (5)). In a particular case $\alpha = 1$, $\gamma = 1/t$ (the exponential distribution), Johnson distance is identical with the well-known Rao distance.

In case of the inverse gamma distribution, $S(x, x^*) = \alpha - \gamma/x$ so that the sample Johnson mean $\hat{x}_n^* = \gamma/\alpha = n/\sum_{i=1}^{n} 1/x_i = \bar{x}_H$ is the harmonic mean. Since $S'(x^*; x^*) = \alpha^2/\gamma$, $ES^2 = \alpha$, $E(-\partial S(x; x^*)/\partial x^*) = \alpha^2/\gamma$ and $D(x^*, \bar{x}_H) = \sqrt{\hat{\alpha}_n}|1 - \bar{x}_H/x^*|$, the approximate $(100-\alpha)\%$ Johnson confidence interval for x^* is

$$\left(\frac{\bar{x}_H}{1+\lambda_n\rho_n}, \frac{\bar{x}_H}{1-\lambda_n\rho_n}\right),$$

where, from the second equation of (5), $\rho_n^2 = 1/\hat{\alpha}_n = \bar{x}_H^2/\frac{1}{n}\sum_{i=1}^{n} 1/x_i^2 - 1$.

By (6), the sample Johnson mean of the beta-prime distribution is

$$\hat{x}_n^* = \frac{\sum_{i=1}^{n}\frac{x_i}{1+x_i}}{\sum_{i=1}^{n}\frac{1}{1+x_i}}. \tag{11}$$

Since $S(x; x^*) = (x - x^*)/(x+1)$, $S'(x^*; x^*) = 1/(p+q)$, $ES^2 = pq/(p+q+1)$ and $E(-\partial S(x; x^*)/\partial x^*) = q^2/(p+q)$, the approximate $(100-\alpha)\%$ Johnson confidence interval for x^* is

$$\left(\frac{\hat{x}_n - \lambda_n\rho_n}{1+\lambda_n\rho_n}, \frac{\hat{x}_n + \lambda_n\rho_n}{1-\lambda_n\rho_n}\right),$$

where $\rho_n = \hat{\omega}_n/(\hat{x}_n^* + 1)$.

In Table 2 we present confidence intervals (c_n^-, c_n^+) for Johnson mean of distributions from Table 1 with the values of parameters chosen such that all the distributions have the same (assumed) values $\hat{x}_n^* = 2$ and $\hat{\omega}_n = 1$.

distribution	c_n^-	c_n^+
Weibull	1.700	2.260
gamma	1.723	2.277
lognormal	1.741	2.297
beta-prime	1.746	2.305
inverse gamma	1.757	2.322

Table 2. Confidence intervals for $n = 50$ and $\alpha = 0.05$.

With exception of the interval for gamma distribution, all intervals are non-symmetric. Their lengths are similar, which is the consequence of equal Johnson variances of all distributions. The results of simulation experiments are in a good agreement with the predicted Johnson confidence intervals.

References

1. Fabián Z., Geometry of Probabilistic Models. Proc. ITAT 2006, 2006, 35–40
2. Fabián Z., New Measures of Central Tendency and Variability of Continuous Distributions. To appear in Commun. in Statist., Theory Methods, 2, 2008
3. Fabián Z., Induced Cores and Their Use in Robust Parametric Estimation. Commun. Statist., Theory Methods, 30, 3, 2001, 537–556
4. Johnson N.L., Kotz S., and Balakrishnan N., Continuous Univariate Distributions 1, 2. Wiley, 1994, 1995
5. Marrona A.R., Martin R.D., and Yohai V.J., Robust Statistics. Theory and Methods. Wiley, 2006

On radio communication in grid networks

František Galčík*

Institute of Computer Science, P.J. Šafárik University, Faculty of Science
Jesenná 5, 041 54 Košice, Slovak Republic
frantisek.galcik@upjs.sk

Abstract. *The paper considers deterministic communication in radio networks whose underlying reachability graphs are undirected 2-dimensional grids. Radio communication is synchronous and differs from other communication models in a way, how simultaneously incoming messages are processed. A node receives a transmitted message iff exactly one its neighbor transmits in a given time slot. In our setting, we assume that the nodes (networks stations) are spread in a region in a regular way. Particularly, they are located at grid points of a square mesh. Initial knowledge of a node is limited only to its unique identifier. The node is not aware of its position in the grid as it was assumed in other papers. We design an algorithm completing the broadcasting task in $O(ecc + \log N)$ rounds using total $O(n+\log N)$ transmissions, where ecc is the eccentricity of the source and N is an upper bound of unique integer identifiers assigned to the nodes. Moreover, we present a modification of the algorithm that solves a task of computation grid coordinates of each node in the asymptotically same time. All presented algorithms are asymptotically optimal according to both considered complexity measures.*

1 Introduction

A *radio network* is a collection of autonomous stations that are referred as *nodes*. The nodes communicate via sending messages. Each node is able to receive and transmit messages, but it can transmit messages only to nodes, which are located within its transmission range. The network can be modelled by a directed graph called *reachability graph* $G = (V, E)$. The vertex set of G consists of the nodes of the network and two vertices $u, v \in V$ are connected by an edge $e = (u, v)$ iff the transmission of the node u can reach the node v. In such a case the node u is called a *neighbor* of the node v. If the transmission power of all nodes is the same, then the reachability graph is symmetric, i.e. a symmetric radio network can be modelled by an undirected graph. In what follows, we focus on communication in a radio grid, i.e. an underlying reachability graph of the radio network is a grid graph. Our aim is to model radio communication in the real-world case when the nodes (networks stations) are spread in a region in a regular way. Particularly, they are located at all grid points of a square mesh and the transmission radius of a node is equal to 1.

Definition 1. *For $p, q > 1$, an undirected graph $G_{p,q}$ is called a grid graph iff:*

$$V(G_{p,q}) = \{(i,j) | 0 \leq i < p, 0 \leq j < q\}$$
$$E(G_{p,q}) = \{((i,j),(i',j')) | (i' = i \wedge j' = j \pm 1) \vee (i' = i \pm 1 \wedge j' = j)\}.$$

Radio communication is synchronous and differs from other communication models in a way, how simultaneously incoming messages are processed. A node receives a transmitted message iff exactly one its neighbor transmits in a given time slot. In particular, the nodes of a network work in synchronized steps (time slots) called *rounds*. In every round, a node can act either as a *receiver* or as a *transmitter*. A node u acting as transmitter sends a message, which can be potentially received by each its neighbor. In the given round, a node, acting as a receiver, receives a message only if it has exactly one transmitting neighbor. The received message is the same as the message transmitted by the transmitting neighbor. If in the given round, a node u has at least two transmitting neighbors we say that a *collision* occurs at node u. We assume that the nodes cannot distinguish an *interference noise* (at least two transmitting neighbors) and a *background noise* (no transmitting neighbor), i.e. the collision at the node u seems for the node u as the round in which no its neighbor transmits. We discuss communication complexity of tasks. Hence the time for determining an action in a round is insignificant, i.e. the complexity of an internal computation is not considered.

The goal of *broadcasting* is to distribute a message from one distinguished node, called a *source*, to all other nodes. Remote nodes of the network are informed via intermediate nodes. The time, required to complete an operation, is important and widely studied parameter of mostly every communication task. Since the nodes (radio stations) used in real-world applications are very often powered by batteries, another important complexity measure is a *energetic complexity*. It is denoted as the total number of transmissions that occur during the execution of an algorithm in the network.

* Author thanks to an anonymous reviewer for his suggestions and references concerning 2-distance coloring.

In this paper, we consider both complexity measures: time and energetic complexity. The complexity of an algorithm is described as a function of three parameters of radio network: the number of nodes (denoted as n), the unknown upper bound of identifiers assigned to the nodes (denoted as N), and the largest distance from the source to any other node of the network (denoted as ecc).

According to different features of stations forming a radio network, many models of radio networks have been developed and discussed. They differ in used communications scenarios and in an initial knowledge assumed for nodes. The overview of the models of radio networks can be found e.g. in [9]. Through this paper, we focus on the deterministic distributed algorithms in unknown radio networks with a grid topology. An initial knowledge of a node is limited only to its unique integer identifier (label). The node is not aware of its position in the grid. Moreover, neither the size of an underlying reachability graph (the number of nodes) nor an upper bound of identifiers are known to the nodes. Most of previous works deal with the assumption that $N = O(n^p)$, for a constant $p > 0$. In this paper, we consider that there is no relationship between N (an upper bound of identifiers) and n (the number of nodes), except trivial $N \geq n$. It models the case when each network station possesses a unique factory identifier (e.g. MAC address) of large scale, but the number of nodes forming a network is relatively small. Note that considered settings are not a typical real-world case since the network topology is fixed to a specific graph topology. On the other hand, all considered assumptions cover the case immediately after appropriate arrangement of nodes (with only a factory initial knowledge) in an area. In this sense, the algorithms proposed in this paper could be seen as the algorithms computing an auxiliary information that serves also for establishment of a fast communication mechanism.

1.1 Related work

The first distributed algorithms for unknown radio networks were presented by Diks et al. in [4]. The authors considered a restricted class of networks having the nodes situated in a line. Moreover, they assumed that each node could reach directly all nodes within a certain distance. Systematic study of deterministic distributed broadcasting has been initiated by Chlebus et al. in [2]. Recently, Czumaj and Rytter [3] gave currently best deterministic broadcasting algorithm that completes the task in time $O(n \log^2 D)$ where D is the diameter of the reachability graph. Note that $D \leq 2.ecc$. In the case when underlying reachability graph is symmetric, i.e. radio networks is modelled by a undirected graph, more efficient broadcasting algorithm has been constructed. In [7], Kowalski and Pelc gave a $\Omega(n \frac{\log n}{\log (n/D)})$ lower bound for symmetric radio networks and designed a $O(n \log n)$-rounds broadcasting algorithm.

Radio communication in networks with grid topology was investigated by Kranakis et al. in [8]. The authors discussed fault tolerant broadcasting in known topology radio grid networks (each node knows its coordinates and dimensions of the grid graph). Bermond et al. [1] considered modified model of communication in know topology radio grid networks that follows from a problem proposed by France Telecom. In this model, transmission of a node causes an interference in some nodes that are not within the transmission range of the node. This region is called an *interference range* of the node. In standard model, the interference range and the transmission range of a node are the same. The authors studied a *gathering task* where the goal is to gather information from all nodes of the network to a distinguished central node.

2 Radio broadcasting in grid networks

In this section, we focus on the broadcasting task in unknown radio networks whose underlying reachability graph is a grid graph $G_{p,q}$, for $p, q > 1$. Considering this setting, we design a deterministic distributed broadcasting algorithm that completes the task asymptotically optimal in $O(ecc + \log N)$ rounds. The algorithm consists of three parts. At first, the source selects one of its neighbors during the first part. The goal of the second part is to compute an initialization information which is later used in third part of the broadcasting algorithm. We shall compute the initialization information only in the neighbors of the source and in the nodes that have in their neighborhood two neighbors of the source. Finally, the third part of the algorithm disseminates a source message through the network. Simultaneously, a control information that is similar to the initialization information, is computed for newly informed nodes.

Definition 2. *A node is referred to as a 2-neighbor of the source iff it is adjacent with two neighbors of the source.*

2.1 Common subroutines and techniques

In this subsection, we present some supplementary techniques that are applied in the first and the second part of the broadcasting algorithm.

Definition 3. *Let v be a node with identifier $ID(v) > 0$ and let $(a_1, a_2, \ldots, a_k)_2$ be a binary representation of*

the number $ID(v)$. An infinite binary sequence $1, a_k, a_{k-1}, \ldots, a_2, a_1, 0, 0, \ldots$ is called a transmission sequence corresponding to the identifier $ID(v)$.

Example 1. The transmission sequence corresponding to the identifier $11 = (1011)_2$ is $1, 1, 1, 0, 1, 0, 0, 0, \ldots$.

Note that the previous definition implies that transmission sequences corresponding to different identifiers differ at least in one position.

Consider the following case. A node u has at least one and at most two active (participating) neighbors, say v and w, that become informed in some unknown rounds (possibly different). We have to deliver an information from one of participating neighbors to the node u as soon as possible. This task can be easily solved applying previously defined transmission sequences. Suppose that a participating neighbor becomes informed in the round 0. In the i-th round, it transmits its information iff the i-th element of the transmission sequence corresponding to its identifier is equal to 1. This subroutine is referred to as *TAI* (transmission according to identifier). We show that at most $O(\log Max(ID(v), ID(w)))$ rounds are enough to inform the node u by one of its participating neighbors.

Lemma 1. *The TAI-subroutine completes the task in at most $O(\log Max(ID(v), ID(w)))$ rounds. The total number of transmissions is $O(\log Max(ID(v),ID(w)))$.*

Proof. Suppose that the node v is informed in the round i and the second participating neighbor w (if exists) is not already informed in this round. Since the first element of the transmission sequence is always 1, v transmits in the round $i + 1$ due to the *TAI*-subroutine. The node w does not transmit in the round $i + 1$ and thus u becomes informed. Now suppose that v and w are informed simultaneously in the round i. According to *TAI*, they transmit in the round $i+1$, but the interference causes that u is not informed. Since $ID(v) \neq ID(w)$, the transmission sequences corresponding to $ID(v)$ and $ID(w)$ are different. Hence, there must be an index j such that j-th elements of their binary transmission sequences are different. It implies that exactly one participating neighbor of u transmits in the round corresponding to the index j (i.e. in the round $i + j$). Therefore the node u is informed in this round. □

In order to avoid interaction during a simultaneous execution of several communication tasks, we can use a time division multiplexing strategy. Particularly, the i-th task from the set of k tasks is executed only in each round j such that $j \equiv i \ (mod\, k)$. In our setting, we do not allow spontaneous transmission, i.e. a node (except the source or the initiator) cannot transmit before successful receiving of a message transmitted by other node. If we include the number of actual round modulo k in each transmitted message, newly informed nodes can synchronize. Thus all nodes participating in the algorithm can simultaneously execute the same task in a given round.

2.2 Selection of a neighbor of the source

Now, we describe an algorithm that selects one of the neighbors of the source. We shall utilize the *TAI*-subroutine. If a message which is transmitted in the *TAI*-subroutine, includes an identifier of sender, successful receiving of the message implies that the receiver can select one of the senders. *TAI*-subroutine works only in the case when there are at most two participating senders. On the other hand, the source has at most 4 neighbors in the grid graph. Hence, a direct application of the *TAI*-subroutine is not possible. With the assistance of 2-neighbors of the source, we split the selection process into several applications of *TAI*-subroutine, where at most 2 nodes participate in process of transmitting information towards a specific node.

In the first round, the source transmits an initialization message. It is received exactly by all neighbors of the source. Subsequently, we start 4 simultaneous tasks (each of them in a separate time slot modulo 4). In the first task, each neighbor of the source tries to inform adjacent 2-neighbors by its identity. Since exactly two neighbors of the source have to transmit towards a 2-neighbor, we utilize the *TAI*-subroutine. After at most $O(\log N)$ rounds, at least one 2-neighbor becomes informed. Note that the nodes are not aware of the fact whether they are 2-neighbors of the source or not. We can solve this problem by a modification of the *TAI*-subroutine in such a way that the first transmission of a node contains a special message. If a node receives this special message, it knows that it is not 2-neighbor. Indeed, all neighbors of the source transmit in the second round of the algorithm (the first round of the *TAI*-subroutine) the special message that due to interference cannot be received by 2-neighbors. An informed 2-neighbor starts its activity during the second task and ignores all messages received in the rounds assigned for the first task. Particularly, it attempts to send a message that includes the received identifier of one its neighbor. Again, we utilize the *TAI*-subroutine. All nodes, except neighbors of the source, ignore transmitted messages during execution of this task. Further, a neighbor of the source that receives its identifier in a round of the second task, starts an execution of the third task and attempts to send its identifier in a message to the source utilizing the *TAI*-subroutine. If a neigh-

bor of the source has received identifier of other node, it finishes its participation in all tasks, expect the fourth task. Moreover, each node that is active in the third task transmits a message in all rounds of the first task. Indeed, it blocks receipt of a message by its adjacent 2-neighbors. The source acknowledges the receipt of a identifier of one its neighbor by transmission of a message in the round reserved for the fourth task. This transmission stops execution of the first and the third task by all neighbors of the source because the selection task is accomplished. Besides, the neighbors ignore all received messages transmitted in rounds of the second task. The goals of tasks can be summarized as follows:

- first task - neighbors of the source send their identifiers towards 2-neighbors
- second task - 2-neighbors of the source send received identifier (that was received in the execution of the first task) towards neighbors of the source
- third task - neighbors of the source that are informed in a round of the second task, send their identifiers towards the source
- fourth task - the source acknowledges selection of a neighbor

Lemma 2. *The source selects one of its neighbors in $O(\log N)$ rounds and using total $O(\log N)$ transmissions, where N is the unknown upper bound of identifiers assigned to the nodes of the network.*

Proof. Time and energetic complexity of this part of the algorithm are obvious. The proof of correctness will appear in the full version of the paper [6]. It is based on the case analysis of all possible states (and future transmissions due to the algorithm) in the round when a 2-neighbor of the source has received a message at the first time. □

2.3 Computation of initialization information

During this part, an initialization process is performed. The goal is to mark neighbors of the source by distinct labels from the set $\{A, B, C, D\}$ in such a way that a node marked by A (B) has not a common neighbor with the node marked by C (D, respectively). Furthermore, we require that each 2-neighbor of the source knows the labels of both adjacent neighbors of the source. Thus these nodes (2-neighbors of the source) can be marked by distinct labels from the set $\{AB, BC, CD, AD\}$. Desired initial labelling of the nodes is described on figure 1.

Note that only neighbors and 2-neighbors of the source participate in this part of the algorithm. All other nodes ignore transmitted messages.

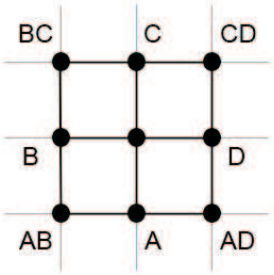


Fig. 1. The initial marking of nodes.

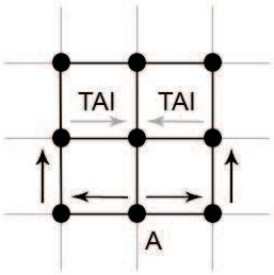


Fig. 2. Scheme of the initial marking computation.

1. The source transmits an initialization message containing the identifier of a selected neighbor.
2. Selected neighbor transmits a message and marks itself with label A.
3. 2-neighbors of the source that received the message in the previous round, retransmit the received message. A neighbor of the source that does not receive a message in this round, marks itself with the label C.
4. Each unmarked neighbor of the source executes the *TAI*-subroutine. It transmits messages containing its identifier as an information content of the node (in the sense of *TAI*). Execution of the *TAI*-subroutine is interleaved with rounds that are reserved for the source. In one of this rounds, the source informs the nodes participating on *TAI* that an unmarked neighbor is selected. This notification is realized by a transmission of the identifier of a selected (unmarked) neighbor.
5. Selected unmarked neighbor sets its label to B and the unselected neighbor (if exists) to D.
6. In one of 4 rounds, each neighbor transmits its label in a round that is designated for its labels.
7. 2-neighbors of the source set the labels according to labels received in previous rounds.

It is easy to see that the initialization schema works properly even the source has less than 4 neighbors. All steps, except step 4, require constant number of rounds. The step 4 is accomplished in $O(\log N)$ rounds due to lemma 1. We summarize the time complexity

of this part that computes initialization information in the following lemma.

Lemma 3. *The initialization information can be computed in $O(\log N)$ rounds with the energetic complexity $O(\log N)$ transmissions.*

2.4 Dissemination of the source message

This part of broadcasting algorithm works in phases. The goal of each phase is to disseminate a source message to the nodes that are in neighborhood of nodes informed during the previous phase. Furthermore, newly informed nodes compute an auxiliary information that is used to arrange their transmissions in the next phase. The auxiliary information has a similar structure as the initialization information. Particularly, the auxiliary information is a label of a node. Each node can be marked by a label from the set $\{A, B, C, D, A^+, B^+, C^+, D^+, AB, BC, CD, AD\}$. The source message is disseminated from the source in a wave that has a shape of square. All nodes located on one side of square have the same label, however each side is marked by different label. The corner nodes are marked by a label from the set $\{AB, BC, CD, AD\}$. Intuitively, a label of a corner node expresses that the node belongs to two sides (directions of the broadcasting wave). We mark the nodes that are adjacent with corner nodes, by a label with + sign.

Initially, we change labels of nodes in the neighborhood of the source in such a way, that the label A is changed to A^+, B to B^+, C to C^+, and D to D^+. This transformation guarantees that the labels of nodes are compatible with the semantic description of labels stated above. In the first round of this part, neighbors and 2-neighbors of the source are considered as the nodes that were informed in the previous phase.

We assume that each transmitted message contains the source message and the number of actual round of executed phase. Each phase takes 5 communication rounds:

1. A node informed in the previous phase and marked by the label A, A^+, AB, AD, C, C^+, BC, or CD transmits a message containing its label.
2. A node informed in the previous phase and marked by the label B, B^+, C, or C^+ transmits a message containing its label.
3. The nodes that receive a label in the round 1 or 2 of the phase and are not yet informed, set the label to the received label. These nodes are referred to as newly informed nodes of given phase. Each newly informed node that received the label A^+, B^+, C^+, or D^+ in the round 1 or 2 of the given phase, transmits its label.
4. Newly informed nodes marked by label AB, AD, BC, or CD that received in the previous round the label A^+ or C^+ transmits their labels (received in round 1 or 2).
5. Newly informed nodes marked by label AB, AD, BC, or CD that received in the previous round the label B^+ or D^+ transmits their labels (received in round 1 or 2).

At the end of the phase, we change labels of some newly informed nodes. No communication is required in this step. First, the nodes with label A^+, B^+, C^+, or D^+ change the label to A, B, C, or D, respectively. Next, the nodes with label AB, AD, BC, or CD change the label to a label received in the round 3. Finally, if a node has received messages transmitted in the rounds 4 and 5, it sets its label to the received value and is considered as a newly informed node of the given phase. Note that the label received in the round 4 and in the round 5 is the same.

It is easy to see that the number of phases is limited by the eccentricity ecc of the source. Each phase takes constant number of round and thus the third part of algorithm is completed in $O(ecc)$ rounds. Each node transmits constantly many times. It implies that energetic complexity of this part is $O(n)$ transmissions.

2.5 Complexity of the broadcasting algorithm

The time and energetic complexity of designed broadcasting algorithm is summarized in the following theorem.

Theorem 1. *Consider a radio network such that its underlying reachability graph is a grid graph. There is a distributed deterministic algorithm that completes the broadcasting task in $O(ecc+\log N)$ rounds using total $O(n+\log N)$ transmissions, where n is the number of nodes and N is the unknown upper bound of identifiers in the network. Moreover, designed algorithm is asymptotically optimal.*

Proof. The first part of algorithm (selection of a neighbor of the source) takes $O(\log N)$ rounds and uses $O(\log N)$ transmissions. Time complexity of the second part (computation of the initial information) is $O(\log N)$ rounds. Energetic complexity of this part is $O(\log N)$ transmissions. Finally, the third part of algorithm that disseminates the source message, takes $O(ecc)$ rounds and uses $O(n)$ transmission. Therefore, the time complexity of the broadcasting algorithm is $O(ecc + \log N)$ rounds. Summing total number of transmissions in each part of algorithm we obtain

that at most $O(n + \log N)$ messages are transmitted by nodes during the execution of the algorithm.

Note that it is usually assumed that $N = O(n^p)$, for a constant $p > 0$. In grid graphs, it holds that $ecc \geq \sqrt{n} \geq \log n$ for sufficiently large n. Hence $O(\log N) = O(\log n) = O(ecc)$. In our setting, parameter N cannot be bounded in this way. Let fix a deterministic broadcasting algorithm. One can show that there is such an assignment of identifiers to the nodes of a grid radio network $G_{3,3}$ that the broadcasting task cannot be completed in less that $\Omega(\log N)$ rounds. The proof could be obtained by adaptation of the argument that was used in the proof of $\Omega(n\frac{\log n}{\log (n/D)})$ lower bound of the time complexity for the broadcasting task in [7]. Complete proof will appear in the full version of the paper [6]. □

2.6 Algorithm for acknowledged broadcasting

Note that the broadcasting algorithm presented in the previous section is not acknowledged, i.e. the source is not aware of the round when the broadcasting task is completed. Furthermore, the source is not able to compute the duration of the algorithm, because parameters of the radio network are unknown for the nodes. Presented principles of the constructed broadcasting algorithm allow to extend the algorithm to an algorithm completing the acknowledged broadcasting task. We present modified algorithm only briefly. The modification is based on the following. First, we add new labels A^*, B^*, C^*, and D^* to the set of labels that is used to mark the nodes. In each phase, new labels are used to mark one node, called a *progress node*, in each direction of the broadcasting wave. In particular, the progress nodes will form a cross with the center in the source. The nodes forming a limb of a cross are marked by the same label and each limb of a cross is marked by a different label. We append a new round to each phase of algorithm. In this round, each active progress node informs its neighboring progress node which is closer to the source, about the fact that the broadcasting task in the given direction is not yet accomplished. If an active progress node does not receive a message in this round during an appropriate phase, it becomes inactive. Since the nodes located on the border of grid do not inform new nodes, a progress node on this border does not receive a message in this round. It causes a chain of continuous deactivations of progress nodes. Finally, there must be a phase in which the source is notified (by silent) about completing broadcasting in the given direction. If the source is informed that disseminations of the source message are completed in all direction, it knows that all nodes are informed. In order to avoid interference during the last round, we have to schedule transmissions in appropriate manner. Let v to be a progress node that has been informed in the i-th phase (the number of the current phase must be included in each transmitted message). We define a number $P(v)$ as follows:

- $P(v) = (i \ mod \ 3) \ mod \ 4$, if v is marked by A^*
- $P(v) = (i \ mod \ 3 + 1) \ mod \ 4$, if v is marked by B^*
- $P(v) = (i \ mod \ 3 + 2) \ mod \ 4$, if v is marked by C^*
- $P(v) = (i \ mod \ 3 + 3) \ mod \ 4$, if v is marked by D^*

An active progress node v transmits a message in the last round of the j-th phase iff $j \equiv P(v) \ (mod \, 4)$.

It is easy to see that asymptotical time and energetic complexity are preserved by this modification.

3 Computation of grid coordinates

Since we consider radio networks with a regular topology, it could be assumed that the radio network is static. It means that the topology of the network remains unchanged for a long time period. This assumption heads towards the issue of computation of an communication structure for the collision-free communication. As we show later, the grid coordinates of nodes can serve as the basic information for a collision-free communication schema. In this section we present a distributed algorithm which computes grid coordinates of each node. The algorithm is a modification of the previously presented broadcasting algorithm. Particularly, it takes advantage of the auxiliary information computed during the third part of the broadcasting algorithm. We assume that the task of computation of grid coordinates is initiated by a distinguished node, called a *initiator*.

The algorithm consists of 3 parts. First two parts of the algorithm are identical to first two parts of the broadcasting algorithm. After this two part, an initialization information is computed. Now, we assign to the source coordinates $[0, 0]$ and to its neighbor marked by A^+ (B^+, C^+, D^+) coordinates $[0, 1]$ ($[-1, 0]$, $[0, -1]$, $[1, 0]$, respectively). It is easy to see that the assignment of coordinates is correct. Indeed, it follows from the way how the nodes are marked by labels during the second part of the algorithm. Similarly, we assign to a node marked by AB (BC, CD, AD) coordinates $[-1, 1]$ ($[-1, -1]$, $[1, -1]$, $[1, 1]$, respectively). Notice that in the broadcasting algorithm the labels of nodes store an information about direction in which the source message is disseminated. We shall use the sense of direction in order to compute grid coordinates of nodes according to information received from some their neighbors.

We modify content of messages sent in the third part of the broadcasting algorithm in the following way. Each message that is transmitted in the round 1

or 2 of the phase, contains coordinates of the sender. During execution of the third part of the algorithm, we preserve an invariant that each informed node (in sense of broadcasting algorithm) has already computed its grid coordinates. Thus the nodes transmitting in first two rounds of a phase have already computed their coordinates. Moreover, we modify messages sent in the round 4 and 5 of a phase. We attach grid coordinates received in one of first two rounds of the phase (in fact it happens in exactly one of those rounds) to the transmitted message. Finally at the end of the phase, the newly informed nodes compute their coordinates. Let $[x, y]$ be the coordinates which were included in a message received in the round 1, 2, 4, or 5 of a phase. We apply the following rules to set the coordinates of a newly informed node:

- label of node A or A^+: $[x, y + 1]$
- label of node B or B^+: $[x - 1, y]$
- label of node C or C^+: $[x, y - 1]$
- label of node D or D^+: $[x + 1, y]$
- label of node AB: $[x - 1, y + 1]$
- label of node BC: $[x - 1, y + 1]$
- label of node CD: $[x + 1, y - 1]$
- label of node AD: $[x + 1, y + 1]$.

Theorem 2. *Consider an unknown radio network such that its underlying reachability graph is a grid graph. There is a distributed deterministic algorithm that computes grid coordinates of each node in $O(ecc + \log N)$ rounds with total $O(n + \log N)$ transmissions, where n is the number of nodes and N is the unknown upper bound of identifiers in the network. Designed algorithm is asymptotically optimal.*

Proof. Correctness of the algorithm follows from the properties of the broadcasting algorithm and the rules for computation of coordinates of newly informed nodes. Since we do not allow spontaneous transmission (i.e. to participate on algorithm before receiving a message from other node), the task cannot be accomplished in better time (and energetic complexity) than the broadcasting task. It implies asymptotical optimality of designed algorithm. □

Note that we can design an algorithm in which the initiator of the computation is notified that the task is completed. It could be achieved by a similar modification of acknowledged broadcasting algorithm for radio networks with grid topology that is presented in the section 2.6

4 Collision-free communication mechanism

In this section, we discuss a collision-free communication mechanism for radio networks with grid topology. It is based on results concerning 2-distance coloring of grids that have been proposed by Fertin et al. in [5]. A 2-distance coloring of a graph is a proper coloring of vertices satisfying that no vertices in distance at most 2 have assigned the same color. Hence no vertex has in its neighborhood two vertices with the same color. Applying algorithm for computing grid coordinates, we may assume that each node is aware of its grid coordinates.

Definition 4. *Let $[x, y]$ be the coordinates of a node v. The number $TR(v) = (2x + y) \ mod 5$ is called a collision free number of the node v.*

Collision free mechanism is defined as follows. A node v is allowed to transmit a message in the i-th round iff $i \equiv TR(v) \ (mod 5)$. Correctness of this mechanism follows from that fact that $TR(v)$ corresponds to a 2-distance coloring of a grid with 5 colors, which is moreover shown to be optimal in [5]. Thus we can use algorithms that are not primary designed for the communication in radio networks. The slowdown caused by the presented mechanism is by a constant factor.

5 Conclusion

We discussed communication in radio networks in the case when underlying reachability graph is a grid graph. We presented asymptotically optimal broadcasting algorithm. Moreover, the energetic complexity is asymptotically optimal, too. The algorithm has been modified and we obtained two another algorithms. The first one realizes the task of acknowledged broadcasting. The second one solves the problem of computation of grid coordinates in an unknown grid network. The later algorithm can be applied to compute an initial information for a collision-free communication mechanism. Note that the algorithm for computation of grid coordinates of nodes can be extended to compute dimensions of the underlying grid graph. Another extension could be an algorithm that solves the task of assignment of compact identifiers to the nodes (i.e. the nodes have to be labelled by unique numbers from the set $1, \ldots, n$).

References

1. Bermond J.-C., Galtier J., Klasing R., Morales N., and Perennes S., Hardness and Approximation of Gathering in Static Radio Networks. Parallel Processing Letters, 16(2), 2006, 165–184
2. Chlebus B. S., Gasieniec L., Gibbons A., Pelc A., and Rytter W., Deterministic Broadcasting in Unknown Radio Networks. In SODA, 2000, 861–870

3. Czumaj A. and Rytter W., Broadcasting Algorithms in Radio Networks with Unknown Topology. In FOCS, IEEE Computer Society, 2003, 492–501
4. Diks K., Kranakis E., Krizanc D., and Pelc A., The Impact of Knowledge on Broadcasting Time in Radio Networks. In J. Nesetril, editor, ESA, Lecture Notes in Computer Science, 1643, Springer, 1999, 41–52
5. Fertin G., Godard E., and Raspaud A., Acyclic and k-Distance Coloring of the Grid. Inf. Process. Lett., 87(1), 2003, 51–58
6. Galčík F., Radio Communication in Grid Networks. Unpublished manuscript, 2007
7. Kowalski D. R. and Pelc A., Broadcasting in Undirected Ad Hoc Radio Networks. In PODC, 2003, 73–82
8. Kranakis E., Krizanc D., and Pelc A., Fault-Tolerant Broadcasting in Radio Networks. J. Algorithms, 39(1), 2001, 47–67
9. Pelc A., Handbook of Wireless Networks and Mobile Computing, chapter Broadcasting in Radio Networks, John Wiley & Sons, Inc., New York, NY, USA, 2002, 509–528

Automatically generated genetic algorithms for constrained mixed programming in materials science*

Martin Holeňa

Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2, 18207 Praha 8, Czech Republic
martin@cs.cas.cz, web:cs.cas.cz/~martin

Abstract. *The paper addresses key problems pertaining to the commonly used evolutionary approach to optimization problems in materials science. These are on the one hand the insufficient dealing in existing implementations of genetic algorithms with the constrained mixed programming problems, which play a crucial role in materials science, on the other hand the narrow scope of specific genetic algorithms developed for this area. The paper proposes an approach to mixed constrained programming problems based on formulating a separate linearly-constrained continuous optimization task for each combination of values of the discrete variables. Then, discrete optimization on the set of nonempty polyhedra describing the feasible solutions of those tasks is performed, followed by solving those tasks for each individual of the resulting population of polyhedra. To avoid computationally expensive checking of the nonemptiness of individual polyhedra, the set of polyhedra is first partitioned into equivalence classes such that only one representative from each class needs to be checked. Finally, the paper outlines a program generator generating problem-tailored genetic algorithms from descriptions of optimization tasks in a specific description language, into which the proposed approach to mixed linear programming has been incorporated.*

1 Introduction

In modern engineering, health care and science, a crucial role is played by the search for new materials, materials that are optimal from the point of view of some physical or chemical property or combination of properties, or from the point of view some desired functionality, such as catalytic or inhibitory effects in some chemical or biochemical reactions, a particular biological impact, etc. At the same time, the materials have to fulfil a number of restricting requirements, implied by the conditions in which they will be used, by cost reasons, as well as by general physical or chemical laws. Consequently, the search for such new materials entails a *complex optimization task* with the following features:

(i) *high dimensionality* (30-50 variables are not an exception);
(ii) *mixture* of *continuous* and *discrete* variables;
(iii) *constraints*;
(iv) *objective function* cannot be explicitly described, its values must be *obtained empirically*.

Commonly used optimization methods, such as the steepest descent, conjugate gradient methods or second order methods (e.g., Gauss-Newton or Levenberg-Marquardt) cannot be employed to this end. Indeed, to obtain sufficiently precise numerical estimates of gradients or second order derivatives of the empirical objective function, those methods need to evaluate the function in points some of which would have a smaller distance than is the measurement error. That is why *methods not requiring any derivatives* have been used to solve the above optimization task - both deterministic ones, in particular the simplex method and holographic strategy, and stochastic ones, such as simulated annealing, or genetic and other evolutionary algorithms. Especially *genetic algorithms (GA)* have become quite popular for the optimization tasks in materials science, mainly due to the possibility to establish a straightforward correspondence between multiple optimization paths followed by the algorithm and the technical realization of the way how the materials proposed by the algorithm are subsequently tested, e.g., channels of a high-throughput reactor, or assay plates.

Nevertheless, a lack of appropriate implementations still hinders genetic algorithms to be routinely used to this end. Generic GA implementations, such as the Matlab Genetic Algorithms and Direct Search Toolbox [9], do not sufficiently address all the above features (i.)-(iv.), in particular the mixed programming and the constrained programming are always addressed only quite superficially. In addition, they use a low-level coding of input variables by means of bit strings and real numbers, which is tedious, error prone, and difficult to understand for end users in materials science. This disadvantage can be avoided with GA developed specifically for this area, and several of them have really appeared in recent years [11, 14, 17]. However, none of those specific algorithms attempts to

* The research reported in this paper has been supported by the grant No. 201/05/0557 of the Grant Agency of the Czech Republic and partially supported by the Institutional Research Plan AV0Z10300504.

seriously tackle the problem of mixed constrained programming, and the experience gathered with them so far shows that they bring a problem of another kind: they are usable only for a narrow spectrum of particular optimization tasks and have to be reimplemented each time when different tasks within the same application area emerge.

This paper proposes an approach to constrained mixed programming problems in optimization tasks in material science that deals with them on a very general level. However, to avoid the disadvantages of generic GA implementations, and at the same time not to suffer from the narrow scope of specific GA, this approach has not been implemented in any particular algorithm. Instead, it has been incorporated into a program generator that generates problem-tailored GA from descriptions of optimization tasks in a specific description language developed to this end [5].

In the next section, theoretical principles of GA, and main approaches to using them for constrained programming are recalled. The approach proposed for optimization tasks in materials science is explained in Section 3. Finally, Section 4 sketches a prototype program generator that generates problem-tailored GA implementing this approach.

2 Genetic algorithms and their modifications for constraints

The term "genetic algorithms" refers to the fact that their particular way of incorporating random influences into the optimization process has been inspired by the *biological evolution of a genotype* [8, 12, 15]. Basically, that way consists in:

- randomly exchanging coordinates between two particular points in the input space of the objective function *(recombination, crossover)*,
- randomly modifying coordinates of a particular point in the input space of the objective function *(mutation)*,
- *selecting* the points for crossover and mutation according to a probability distribution, either uniform or skewed towards points at which the objective function takes high values (the latter being a probabilistic expression of the survival-of-the-fittest principle).

In the context of materials science, it is useful to differentiate between quantitative mutation, which modifies merely the proportions of substances already present in the material, and qualitative mutation, which enters new substances or removes present ones (Fig. 1).

Genetic algorithms have been originally introduced for unconstrained optimization. However, there have been repeated attempts to modify them for *constrained programming*. Basically, those attempts belong to one of (or combine several of) the following approaches [1–4, 7, 10, 13]:

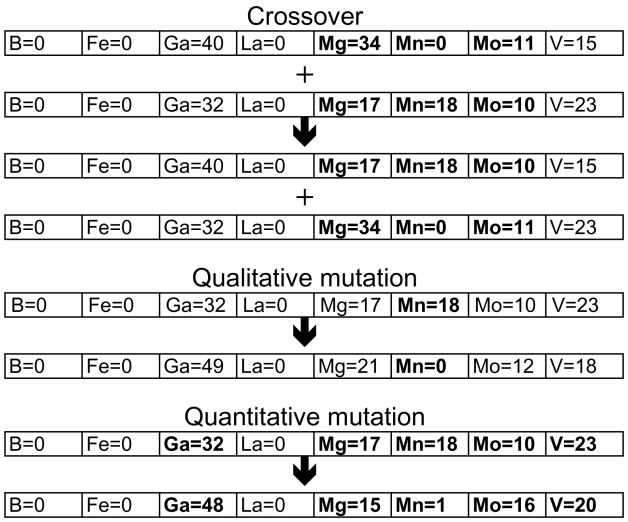


Fig. 1. Illustration of genetic operations used in materials science, the values in the examples are molar proportions (in %) of oxides of the indicated element in the material.

(i) To *ignore* offsprings infeasible with respect to constraints and not include them into the new population. Because in constrained programming, the global optimum frequently lies on a boundary determined by the constraints, ignoring infeasible offsprings may lead to discharging information on offsprings very close to that optimum. Moreover, for some genetic algorithms this approach can lead to the deadlock of not being able to find a whole population of feasible offsprings.

(ii) To modify the objective function by a superposition of a *penalty for infeasibility*. This works well if theoretical considerations allow an appropriate choice of the penalty function. If on the other hand some heuristic penalty function is used, then its values typically turn out to be either too small, which can allow the optimization paths to stay forever outside the feasibility area, or too large, thus suppressing the information on the value of the objective function for infeasible offsprings.

(iii) To *repair the infeasible offspring* through modifying it so that all constraints get fulfilled. This again can discard information on some offsprings close to the global optimum. Therefore, various modifications of the repair approach have been proposed that give up full feasibility, but preserve some information on the original offsprings, e.g., repairing only randomly selected offsprings

(with a prescribed probability), or using the original offspring together with the value of the objective function of its repaired version.

(iv) To add the feasibility/infeasibility as *another objective function*, thus transforming the constrained optimization task into a task of *optimization with respect to multiple goals*. Unfortunately, this new task is similarly far from the original purpose of genetic algorithms and other evolutionary methods, and similarly difficult for solving with them as the original one.

(v) To *modify the recombination and/or mutation operator* in such a way that it gets closed with respect to the set of feasible solutions. Hence, also a mutation of a point fulfilling all the constraints or a recombination of two such points has to fulfil them. An example of such an operator is the edge-recombination operator used for the traveling salesman problem [16]. Unfortunately, such a modification always requires enough knowledge about the particular constraints, therefore this approach can not be elaborated in a general setting.

The experience with those approaches shows that each of them has its specific problems, and non of them is an ultimate way of dealing with constraints, generally better than the others.

3 Proposed approach to constrained mixed programming

The proposed way of solving constrained mixed programming problems follows the above recalled approach (v) and is based on two specific features of such problems in the area of materials science:

(i) It is sufficient to consider only linear constraints. Even if the set of feasible solutions is not constrained linearly in reality, the finite measurement precision of the involved continuous variables always allows to constrain it piecewise linearly and to indicate the relevant linear piece with an additional discrete variable. Consequently, the set of feasible values of the continuous variables that form a solution together with a particular combination of values of the discrete variables is a polyhedron, though each such polyhedron can be empty, and each has its specific dimension, ranging from 1 (closed interval) to the number of continuous variables.

(ii) If a solution polyhedron is described with an inequality

$$P = \{x : Ax \leq b\}, \tag{1}$$

then its feasibility (i.e., $P \neq \emptyset$) is invariant with respect to any permutation of columns of A, as well as respect to any permutation of rows of (A b). Moreover, since:

- identity is also a permutation,
- each permutation has a unique inverse permutation,
- the composition of permutations is again a permutation,

the relation $\approx$ between solution polyhedra, defined for P and $P' = \{x : A'x \leq b'\}$, by means of

$$\begin{aligned} P \approx P' \text{ iff } &(A'b') \text{ can be obtained from} \\ &(Ab) \text{ through some permutation} \\ &\text{of columns of } A\text{, followed by} \\ &\text{some permutation of rows} \\ &\text{of the result and of } b \end{aligned} \tag{2}$$

is an equivalence on the set of solution polyhedra. Consequently, it partitions that set into disjoint equivalence classes, the number of which can be much lower than the number of polyhedra. In the optimization problems encountered in the testing of the approach, the number of solution polyhedra ranges between thousands and hundreds of thousands (Fig. 2), but the number of their equivalence classes ranges only between several and several dozens. Whereas a separate check of the nonemptiness of each polyhedron could prohibitively increase the computing time of the generated GA, forming the equivalence classes is fast, and then only one representative from each class needs to be checked.

Those features together with the fact that the constraints determine which combinations of values of dis-

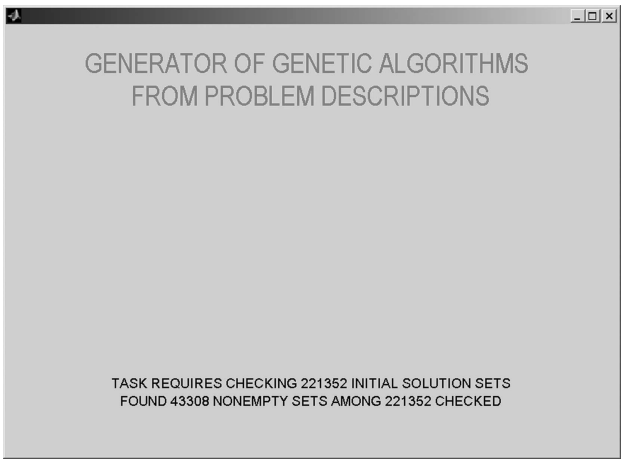


Fig. 2. Screen-shot of the graphical interface of a generated GA after all the polyhedral solutions sets for individual continuous optimization tasks have been checked for nonemptiness.

crete variables to consider, suggest the following procedure for dealing with constrained mixed optimization in the generated GA:

1. A separate continuous optimization problem is formulated for each combination of values of discrete variables that can be for some combination of continuous variables feasible with respect to the specified constraints.
2. The set of all solution polyhedra corresponding to the continuous optimization problems formulated in step 1, is partitioned according to the equivalence (2).
3. One representative polyhedron from each partition class is checked for nonemptiness, taking into account the discernibility (measurement precision) of considered variables.
4. On the set of nonempty polyhedra, discrete optimization is performed, using operations selection and mutation developed specifically to this end. In particular, selection is performed in the following way:
   - In the first generation, all polyhedra are selected according to the uniform distribution.
   - In the second and later generations, a prescribed proportion is selected according to an indicator of their importance due to the points from the earlier generations, and the rest is selected according to the uniform distribution.
   - As an indicator of the importance of a polyhedron due to the points from the earlier generations, the difference between the fitness of the point and the minimal fitness of points in the earlier generations is taken, summed over all points for which the discrete variables assume the combination of values corresponding to the polyhedron.
   - Each of the polyhedra forming the population obtained in this way corresponds to a subpopulation of combinations of values of continuous variables.
5. In each of the polyhedra found through the discrete optimization, a continuous optimization is performed. The combinations of values of continuous variables found in this way, combined with the combinations of values of discrete variables, corresponding to the respective polyhedra, form together the final population of solutions to the mixed optimization problem.

4 Implementation by means of a program generator

To be available for a possibly broad spectrum of problems in materials science, the proposed approach has not been incorporated into a particular GA implementation, but has been combined with a *program generator* that transforms a description of the optimization task to an executable program. A first prototype of such a generator has been developed at the Leibniz Institute for Catalysis (LIKat) in Berlin and is currently in the testing phase. It has been developed in Matlab; also the generated GA implementations are in Matlab and make use of its Genetic Algorithm and Direct Search Toolbox [9], as well as of the Multi-Parametric Toolbox from ETH Zurich [6]. Differently to a human programmer, a program generator needs the description to be expressed in a rigorously formal way, i.e., with some kind of a problem description language. For the area of materials science, such a language has been proposed in [5]. It allows expressing a broad variety of user requirements on the catalytic materials to be sought by the genetic algorithm, as well as on the algorithm itself (Fig. 3).

```
                    ⋮
FeedbackTable GlobalParameter vam
FeedbackField GlobalParameter fitness
vam_catalyst ComposedOf support InProportion support_fraction,
& support_dopants InProportion support_dopants_fraction, Pd InProportion
& Pd_fraction, Au InProportion Au_fraction, dopants InProportion
& dopants_fraction PreparedUsing  dopants_preparation
support_fraction FromInterval 87,100 WithPrecision 1
support_dopants_fraction FromInterval 0,10 WithPrecision 0.1
Pd_fraction OneOf 0.7,1
Au_fraction OneOf 0,0.2,0.4,0.6,0.8
dopants_fraction FromInterval 0,1.2 WithPrecision 0.01
dopants_preparation OneOf sequential, intermediate_drying
support_fraction + support_dopants_fraction + Pd_fraction + Au_fraction
& + dopants_fraction = 1
support OneOf siliperl, aerosil200, aerosil300, basisP25, zirkonix, TiO2BET
support_dopants ComposedOf number_of_support_dopants FromAmong
& B InProportion B_fraction, Ti InProportion Ti_fraction, Ce InProportion
& Ce_fraction, Fe InProportion Fe_fraction, Y InProportion Y_fraction,
& La InProportion La_fraction, Mo InProportion Mo_fraction, Cr InProportion
& Cr_fraction, Nb InProportion Nb_fraction
number_of_support_dopants OneOf 0,1,2
B_fraction FromInterval 1,10 WithPrecision 0.1
Ti_fraction FromInterval 1,10 WithPrecision 0.1
Ce_fraction FromInterval 1,10 WithPrecision 0.1
Fe_fraction FromInterval 1,10 WithPrecision 0.1
Y_fraction FromInterval 1,10 WithPrecision 0.1
La_fraction FromInterval 1,10 WithPrecision 0.1
Mo_fraction FromInterval 1,10 WithPrecision 0.1
Cr_fraction FromInterval 1,10 WithPrecision 0.1
Nb_fraction FromInterval 1,10 WithPrecision 0.1
B_fraction + Ti_fraction + Ce_fraction + Fe_fraction + Y_fraction +
& La_fraction + Mo_fraction + Cr_fraction + Nb_fraction =
& support_dopants_fraction
siliperl IsPrimitive
aerosil200 IsPrimitive
aerosil300 IsPrimitive
basisP25 IsPrimitive
zirkonix IsPrimitive
TiO2BET IsPrimitive
                    ⋮
```

Fig. 3. Example fragment of a problem description in the description language proposed in [5].

An overall scheme of this solution is depicted in Fig. 4. The program generator accepts text files with problem descriptions as input, and produces GA

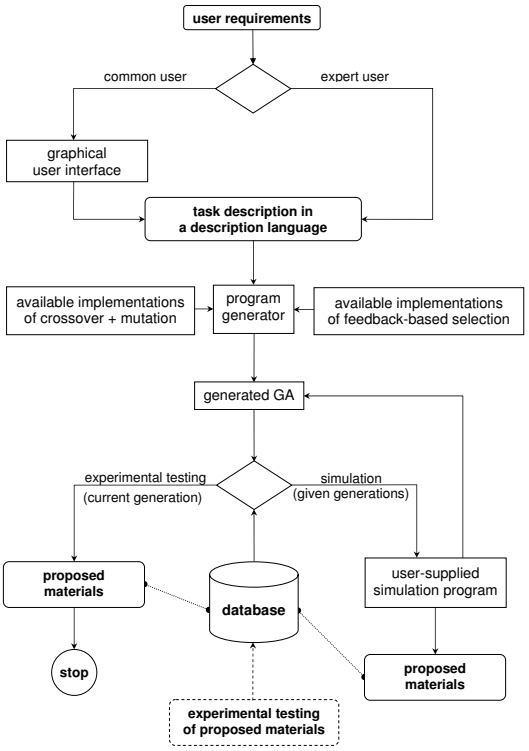


Fig. 4. Scheme of the proposed approach to generating problem-tailored genetic algorithms according to formal problem descriptions.

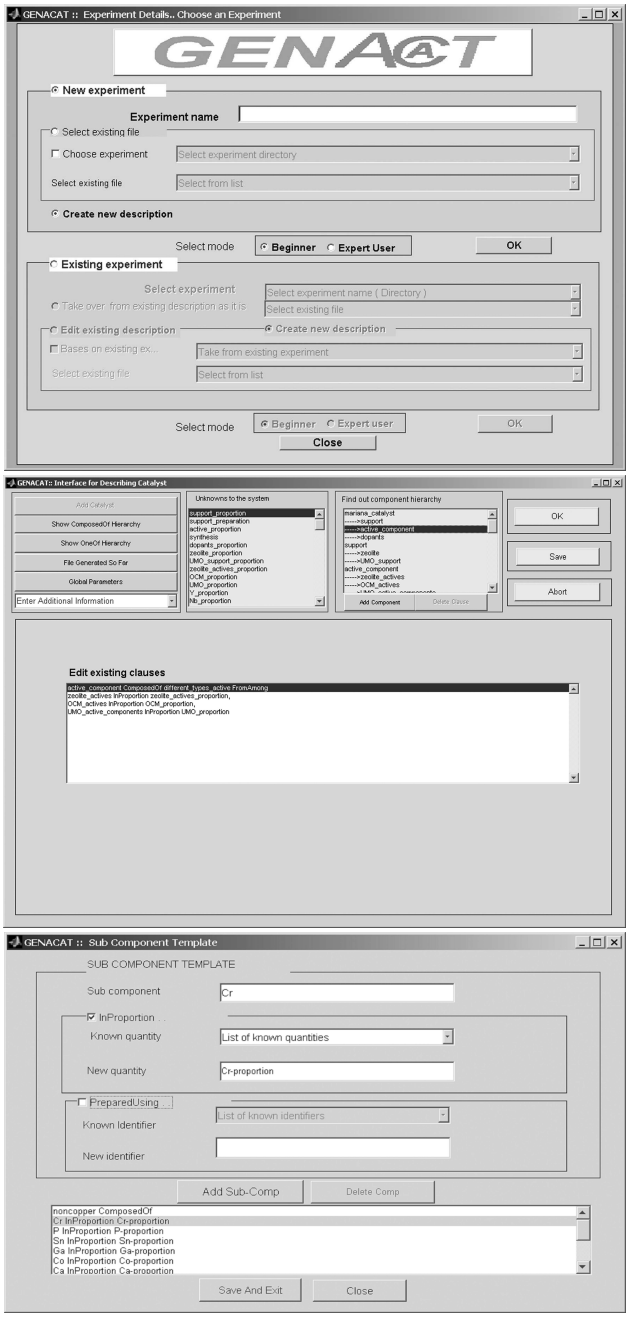


Fig. 5. Three example windows of the graphical interface allowing users to enter the information needed to create a problem description.

implementations as output. For the approach, it is immaterial how such an implementation looks like. It can be programmed in various languages; it can be a stand-alone program or can combine calls to generic GA software with parts implementing the functionality that the generic software does not cover. If the values of the objective function have to be obtained through experimental testing, then the GA implementation runs only once and then it exits. However, the approach previews also the possibility to obtain those values from some simulation program instead. Then the GA implementation alternates with that program for as many generations as desired.

Finally, the implementation places a graphical interface between the user and the program generator, the purpose of which is to remove from the users the necessity to write files with problem descriptions manually, and the necessity to understand the description language and its syntax. To this end, the interface provides a series of windows through which a user can enter all the information needed to create a complete problem description (Fig. 5). In addition, also the generated GA implementations contain a simple graphical interface, which shows the progress of the performed optimization, and allows deciding whether to run the implementation only once or which external program to use for simulation (Fig. 6).

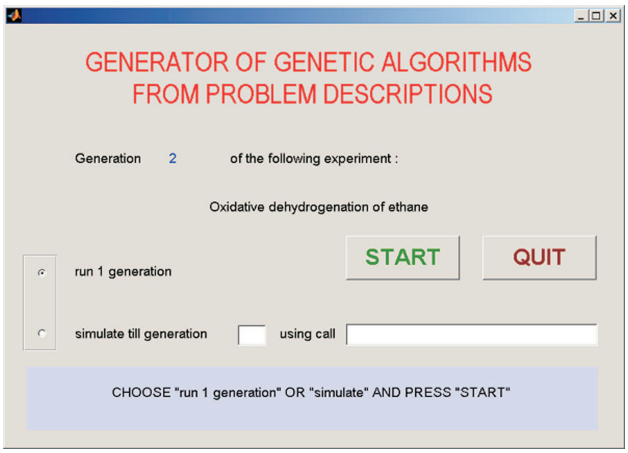


Fig. 6. Simple graphical user interface of the generated GA implementations.

5 Conclusion

The paper has presented solutions to two key problems encountered when using genetic algorithms for optimization tasks in materials science. First, it has proposed an approach to constrained mixed programming problems based on specific features of such problems in the area of materials science. Second, it has shown that it is possible to avoid repeated reimplementations of specific genetic algorithms in this area without having to resort to generic algorithms. To this end, a program generator has been used that generates problem tailored genetic algorithms from descriptions of optimization tasks, at the same time incorporating the proposed approach to constrained mixed programming into them. A first prototype of such a generator has been developed at LIKat Berlin and is currently in the testing phase.

References

1. Bunnag D. and Sun M., Genetic Algorithms for Constrained Global Optimization in Continuous Variables. Applied Mathematics and Computation, 171, 2005, 604–636
2. Chu P.C. and Beasley J.E., Constraint Handling in Genetic Algorithms: The Set Partitioning Problem. Journal of Heuristic, 4, 1998, 323–358
3. Fonseca C.M. and Fleming P.J., An Overview of Evolutionary Algorithms in Multiobjective Optimization. Evolutionary Computation, 3, 1995, 1–16
4. Fung R., Tang J., and Wang D., Extension of a Hybrid Genetic Algorithm for Nonlinear Programming Problems with Equality and Inequality Constraints. Computers and Operations Research, 29, 2002, 261–274
5. Holeňa M., Present Trends in the Application of Genetic Algorithms to Heterogeneous Catalysis. In A. Hagemeyer, P. Strasser, and A.F. Volpe, (eds), High-Throughput Screening in Chemical Catalysis, Wiley-WCH, Weinheim, 2004, 153–172
6. Kvasnica M., Grieder P., Baotić M., and Morari M., Multi-Parametric Toolbox (MPT). In Hybrid Systems: Computation and Control, Springer Verlag, Berlin, 2004, 449–462
7. Li H., Jiao Y.-C., and Wang Y., Integrating the Simplified Interpolation into the Genetic Algorithm for Constrained Optimization Problems. In Computational Intelligence and Security, Springer Verlag, Berlin, 2005, 247–254
8. Man K.F. , Tang K.S., and Kwong S., Genetic Algorithms. Concepts and Design. Springer Verlag, London, 1999
9. The Mathworks, Inc. Genetic Algorithm and Direct Search Toolbox, 2001
10. Michalewicz Z. and Schonauer M., Evolutionary Algorithms for Constrained Parameter Optimization Problems. Evolutionary Computation, 4, 1996, 1–32
11. Pereira R.M., Clerc F., Farusseng D., Waal J.C., and Maschmeyer T., Effect of Genetic Algorithm Parameters on the Optimization of Heterogeneous Catalysts. (QSAR) and Combinatorial Science, 24, 2005, 45–57
12. Reeves C.R. and Rowe J.E., Genetic Algorithms: Principles and Perspectives. Kluwer Academic Publishers, boston, 2003
13. Reid D.J., Genetic Algorithms in Constrained Optimization. Mathematical and Computer Modelling, 23, 1996, 87–111
14. Rodemerck U., Baerns M., and Holeňa M., Application of a Genetic Algorithm and a Neural Network for the Discovery and Optimization of New Solid Catalytic Materials. Applied Surface Science, 223, 2004, 168–174
15. Vose M.D., The Simple Genetic Algorithm. Foundations and Theory. MIT Press, Cambridge, 1999
16. Whitley D., Starkweather T., and Shanner D., The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination. In L. Davis, (ed.), Handbook of Genetic Algorithms, Van Nostrand Reinold, New York, 1991, 350–372
17. Wolf D., Buyevskaya O.V., and Baerns M., An Evolutionary Approach in the Combinatorial Selection and Optimization of Catalytic Materials. Applied Catalyst A: General, 200, 2000, 63–77

Využitie jednostranne fuzzy konceptových zväzov pri skúmaní sociálnych sietí*

Stanislav Krajči and Jana Krajčiová

[1] Ústav informatiky, PF UPJŠ Košice
stanislav.krajci@upjs.sk
[2] Osemročné gymnázium, Alejová 1, Košice
jana.krajciova@pobox.sk

Abstrakt *V tomto článku informujeme o našom experimente so sociálnou sieťou istej triedy študentov osemročného gymnázia. Každý žiak najprv charakterizoval svoj vzťah ku každému spolužiakovi hodnotou z istého rozsahu. Na takto získané relačné dáta sme aplikovali data-miningovú metódu jednostranne fuzzy konceptových zväzov, včítane príslušne modifikovaného Rice-ovho a Siff-ovho algoritmu. Pomocou neho získané zhluky sme sa v závere pokúsili interpretovať ako skupiny žiakov vnímaných svojimi spolužiakmi rovnakým spôsobom.*

1 Úvod

Sociálna sieť môže byť charakterizovaná ako systém vzájomných vzťahov medzi ľuďmi istej skupiny. Takáto sieť je často vyjadrená vo forme orientovaného grafu, ktorého vrcholy tvoria účastníci siete. V najjednoduchšom prípade prítomnosť (orientovanej) hrany znamená iba existenciu vzťahu osoby reprezentovanej začiatočným vrcholom tejto hrany k osobe reprezentovanej jej koncovým vrcholom. (Poznamenajme, že takýto vzťah nemusí byť vzájomný, orientovanosť hrán je preto pochopiteľná.) Takýto graf môže byť reprezentovaný klasickou reláciou: Ak B je množina zúčastnených osôb, tak naša relácia je istá podmnožina množiny $B \times B$. Ak je táto relácia znázornená vo forme tabuľky, množina B znamená jednak jej riadky (ako objekty) a jednak stĺpce (ako atribúty). Každá hodnota v tejto tabuľke potom znamená ohodnotenie vzťahu osoby zo stĺpca k osobe z riadku.

Toto je však ideálna situácia na aplikovanie nejakej data-miningovej metódy na získanie prípadných skrytých informácií z tejto tabuľky. V tomto článku sa sústredíme iba na jednu z nich, na *formálnu konceptovú analýzu* (čím, samozrejme, nevylučujeme použitie iných prístupov). Ako výsledok tak budeme očakávať zhluky osôb, ktoré sú považované inými osobami za podobné.

Klasická podoba formálnej konceptovej analýzy pracuje s dvoma hodnotami – áno, resp. nie. Existuje však mnoho situácií, keď tieto dve hodnoty nedokážu skúmané vzťahy plne popísať. V takom prípade potrebujeme na ich charakterizáciu viac hodnôt, tie budú hodnotami (farbami, váhami) hrán grafu popisujúceho sociálnu sieť. Hovoríme tu potom o *fuzzy relácii*, ktorá nie je ničím iným ako funkciou z množiny $B \times B$ do predpísanej množiny hodnôt (ktorá je často usporiadaná, ba dokonca lineárne). Ak tu chceme použiť formálnu konceptovú analýzu, musíme sa uchýliť k niektorej jej viachodnotovej verzii. V našom experimente používame tzv. *jednostranne fuzzy konceptový zväz*.

2 Experiment

Náš experiment bol vykonaný na Osemročnom gymnáziu, Alejová 1, Košice, kde ako učiteľka pracuje i spoluautorka tohto článku. Vybrali sme triedu *Sekunda*, ktorú dobre pozná, lebo bola skoro rok a pol jej triednou učiteľkou. Túto triedu navštevuje 34 okolo 12-ročných študentov, z toho 11 dievčat a 23 chlapcov.

Každý študent dostal za úlohu vyjadriť svoj vzťah ku každému spolužiakovi výberom jednej z nasledujúcich siedmich hodnôt (poznamenajme, že tento počet zodpovedá dobre známej Lickertovej stupnici):

- 3 – je to môj výborný priateľ
- 2 – je to môj kamarát
- 1 – je mi sympatický
- 0 – nie je mi ani sympatický, ale ani nesympatický
- -1 – nie je mi sympatický
- -2 – nemám ho rád
- -3 – neznášam ho

Všimnime si, že hodnoty sú stupňované, väčšia znamená lepší vzťah. Hodnoty boli vybraté v súlade s bežným chápaním celých čísel, teda tak, aby kladné vyjadrovali pozitívny vzťah, záporné negatívny a nula, prirodzene, neutrálny. Treba zdôrazniť, že tu nedorozumenie zo strany žiakov nehrozilo – v tomto veku už veľmi dobre rozumejú významu záporných čísel, navyše je táto trieda explicitne zameraná na matematiku. Žiakov sme motivovali prísľubom, že sa dozvedia (aspoň čiastkové) výsledky experimentu. Možno aj preto k ankete pristúpili veľmi zodpovedne, vypĺňali

* Tento článok je podporený grantom 1/3129/06 Slovenskej grantovej agentúry VEGA.

	name	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
1	Ch01	3	−2	1	1	3	0	−1	0	3	0	−2	2	−2	1	1	0	0	0	3	2	2	0	−1	−1	0	−1	0	0	0	0	3	0	0	0
2	Ch02	1	3	3	0	0	0	3	3	0	1	3	0	3	3	2	−1	2	1	−1	0	1	0	1	0	−2	2	−1	0	2	0	1	2	3	2
3	Ch03	0	3	3	2	−1	−2	3	2	1	3	3	1	2	3	2	−2	1	0	0	1	1	0	0	0	1	0	0	1	2	0	2	0	2	1
4	Ch04	−1	−1	0	3	−1	−2	−2	0	0	0	−1	2	−1	−3	1	−2	−2	0	−2	1	−1	0	−2	−1	3	−1	0	2	0	0	0	0	0	−2
5	Ch05	3	0	2	−1	3	0	−1	0	2	0	1	2	0	2	2	0	1	1	3	2	2	0	0	0	0	0	1	0	−2	2	0	1	0	2
6	D01	−1	−1	−1	−1	−1	3	−3	0	−1	0	0	0	−2	−2	1	3	1	2	−1	0	−1	2	2	2	0	2	1	0	2	2	0	1	0	2
7	Ch06	0	3	3	−1	−2	0	3	3	−3	0	3	0	3	3	2	0	1	1	−2	0	0	0	1	0	0	2	1	1	3	0	−1	2	3	1
8	Ch07	1	2	3	1	1	0	2	3	1	1	3	1	2	3	2	0	2	1	1	0	1	0	1	0	2	1	1	1	1	0	2	0	3	2
9	Ch08	3	1	2	2	3	−1	−1	−1	3	1	1	2	−3	−3	2	−1	2	1	3	1	3	0	1	0	−3	1	1	0	2	0	3	0	−2	1
10	Ch09	0	0	3	1	0	−2	0	2	1	3	0	1	−2	−1	1	−1	1	0	1	1	0	0	0	−1	0	0	1	1	1	0	0	−1	1	1
11	Ch10	−1	3	3	−2	−1	0	3	3	−3	0	3	0	3	3	3	−2	2	1	−2	0	0	0	1	0	−3	2	−1	0	2	0	−2	2	3	2
12	Ch11	1	0	1	2	2	0	−2	0	0	0	0	3	0	−2	1	0	−2	1	−1	3	0	0	−1	−2	1	−1	1	1	1	0	2	0	0	0
13	Ch12	−1	3	3	0	−1	−2	3	3	−3	−1	3	1	3	3	2	−2	2	1	0	1	1	−1	−1	−2	0	0	−2	1	2	0	−1	−2	3	2
14	Ch13	−1	3	2	−1	−1	−3	1	3	−1	−1	3	0	2	3	2	−2	3	0	−3	0	0	0	0	−2	0	0	0	0	0	0	0	−2	2	3
15	Ch14	−2	2	3	−1	−1	0	1	2	1	0	3	0	2	2	3	−1	3	1	0	0	1	0	0	−1	−1	0	1	1	3	0	0	0	2	3
16	D02	−1	0	−2	0	−1	3	0	−1	0	−1	0	0	−2	−3	1	3	2	2	−1	0	0	2	2	2	0	2	1	0	−1	2	0	2	0	2
17	Ch15	−3	−1	0	−3	−3	−1	−1	0	1	−2	0	0	−3	−3	3	−2	3	1	0	−3	0	−1	−1	0	−3	0	0	−1	−2	−1	−3	0	0	3
18	D03	−1	1	−1	0	−1	2	1	−1	0	0	3	0	1	−3	3	2	3	3	−1	0	0	0	2	2	0	2	3	0	1	1	2	3	−1	2
19	Ch16	3	0	0	−3	3	0	−1	−1	3	1	0	0	−1	−3	2	0	2	1	3	1	2	0	0	0	−3	0	1	1	2	0	3	1	−1	1
20	Ch17	0	−2	−1	1	1	0	−3	−1	−3	−2	−2	3	−3	−3	−3	0	−3	0	−2	3	−2	0	−1	−2	1	−1	1	−1	0	0	0	0	−1	−2
21	Ch18	3	1	1	1	3	−1	1	2	3	1	1	2	1	1	2	−1	1	0	3	2	3	0	0	0	0	0	1	0	2	0	3	1	1	0
22	D04	−1	0	−1	0	−1	0	0	−1	−2	0	−1	0	−3	−3	2	0	−1	1	−1	0	−2	3	2	1	0	2	1	0	0	1	0	1	−2	2
23	D05	−1	−1	−1	0	−1	1	0	−1	−3	0	0	0	−2	−3	2	1	2	2	−1	0	−1	2	3	1	0	3	2	0	−2	1	0	1	−1	2
24	D06	−1	0	0	0	−1	3	−3	0	0	0	0	0	0	−2	2	3	3	2	−1	0	0	2	2	3	0	2	2	0	−1	3	0	2	0	3
25	Ch19	−1	−3	−3	3	−1	−3	−3	−1	−3	−1	−3	1	−2	−3	−3	−3	−3	0	−3	1	−2	0	−3	−2	3	−3	−2	2	−3	−1	−1	−1	−1	−3
26	D07	−1	1	−1	0	−1	1	2	0	−3	0	3	0	1	−3	3	1	3	2	−1	0	0	2	3	1	0	3	2	0	2	1	0	1	0	2
27	D08	−2	−3	−1	0	−1	1	1	−2	0	0	−3	0	−2	−3	3	1	3	3	−1	0	0	0	2	2	0	2	3	0	1	1	2	3	−1	−1
28	Ch20	−1	−3	−2	3	0	−2	−2	−1	−3	0	−3	1	−2	−3	−1	−2	−2	0	−2	1	−2	0	−1	0	3	−1	0	3	1	0	0	1	0	0
29	Ch21	0	2	3	2	0	0	3	2	1	1	3	1	1	1	1	−1	3	1	−1	1	1	0	1	0	1	1	2	1	3	0	0	1	2	2
30	D09	−1	−1	−2	0	−1	3	−3	−1	0	0	0	0	−1	−2	2	2	1	2	−1	0	0	2	2	3	0	2	1	0	−1	3	0	1	0	2
31	Ch22	3	−1	2	1	3	0	0	0	3	0	−1	3	−2	−1	2	0	2	1	3	2	3	0	0	0	1	0	1	1	2	0	3	1	0	0
32	D10	−1	2	1	0	−1	2	2	1	0	0	3	0	−3	−2	2	1	3	2	−1	0	0	2	2	2	0	2	2	0	2	1	1	3	0	2
33	Ch23	1	3	3	1	−1	0	3	3	−3	1	3	1	1	3	2	0	3	1	−1	2	0	0	1	0	0	1	2	1	2	0	1	0	3	1
34	D11	0	0	1	−2	0	2	−2	1	0	0	2	0	1	0	3	2	3	1	−1	0	0	2	2	2	−3	2	1	1	1	1	0	1	1	3

ju v dobrej atmosfére, bez časového stresu i bez iných rušivých elementov.

Výsledky sú v nasledujúcej tabuľke. Stĺpec každej hodnoty znamená žiaka, ktorý hodnotil, jej riadok zodpovedá žiakovi, ktorý bol hodnotený. Treba povedať, že hodnoty na diagonále (t. j. samohodnotenia študentov) boli upravené na jednotnú, prirodzene najvyššiu, hodnotu (väčšina z nich ju naozaj použila, no niektorí túto položku vôbec nevyplnili a pár ich použilo nižšie (dokonca záporné) hodnoty). Všetky ostatné hodnoty však ostali, samozrejme, nezmenené. Plné mená žiakov sú dôverné, pohlavie vyplýva z ich krstného mena.

3 Jednostranne fuzzy konceptový zväz

Naša tabuľka dát je jednoducho objektovo-atribútový model: Riadky sú žiaci, čiže objekty nášho výskumu, stĺpce zodpovedajú hodnotiacim spolužiakom a môžeme ich chápať ako atribúty. Naše (ako sme už naznačili, psychologicky užitočné) hodnoty z množiny $\{-3,-2,-1,0,1,2,3\}$ možno ľahko transformovať (pomocou funkcie $x \mapsto \frac{x+3}{6}$) na hodnoty z množiny $\{0, \frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{5}{6}, 1\}$, čo je podmnožina klasického fuzzy intervalu $[0,1]$. Hodnoty v každom riadku zodpovedajú jednému objektu a môžu byť chápané ako jemu zodpovedajúca funkcia z množiny všetkých atribútov do $[0,1]$. Budeme teda pracovať s klasickými (tzv. *crisp*) podmnožinami objektov a fuzzy podmnožinami atribútov.

V článku [5] sme modifikovali klasickú binárnu formálnu konceptovú analýzu z [4] na tzv. *jednostranne fuzzy konceptový zväz*, ktorý pracuje práve s crisp podmnožinami objektov a fuzzy podmnožinami atribútov. Tento prístup je v podstate ekvivalentný tzv *škálovaniu* Gantera & Willeho ([4]), môžeme ho však považovať za jeho rýchlejšiu a efektívnejšiu verziu. (Poznamenajme, že podobné prístupy sa vyskytujú i v prácach Ben Yahiu a Jaouu v [3] a Bělohlávka a iných v [1].)

Pripomeňme základné pojmy z teórie jednostranne fuzzy konceptových zväzov, ako boli navrhnuté v našom článku [5]:

Nech A (ako atribúty) a B (ako objekty) sú neprázdne (obvykle konečné) množiny a nech R je fuzzy relácia na ich karteziánskom súčine, t. j. $R : A \times B \to [0,1]$. Túto reláciu potom môžeme chápať ako tabuľku s riadkami a stĺpcami zodpovedajúcimi objektom, resp. atribútom, hodnota $R(a,b)$ vyjadruje stupeň, v akom má objekt b atribút a.

Definujme zobrazenie $\uparrow : \mathscr{P}(B) \to [0,1]^A$, ktoré priradí každej množine objektov X fuzzy množinu atribútov $\uparrow(X)$, pričom hodnota v bode $a \in A$ bude

$$\uparrow(X)(a) = \inf\{R(a,b) : b \in X\}.$$

Táto funkcia teda priraďuje každému atribútu najväčšiu možnú hodnotu takú, aby každý objekt z X mal tento atribút aspoň v takomto stupni.

Symetricky definujme zobrazenie $\downarrow : [0,1]^A \to \mathscr{P}(B)$, ktoré priradí každej funkcii $f : A \to [0,1]$ množinu

$$\downarrow(f) = \{b \in B : (\forall a \in A) R(a,b) \geq f(a)\},$$

čiže všetky objekty majúce všetky atribúty v aspoň takom stupni, ako predpisuje funkcia f.

Ľahko vidieť, že tieto zobrazenia majú nasledujúce vlastnosti:

- Ak $X_1 \subseteq X_2$, tak $\uparrow(X_1) \geq \uparrow(X_2)$.
- Ak $f_1 \leq f_2$, tak $\downarrow(f_1) \supseteq \downarrow(f_2)$.
- $X \subseteq \downarrow(\uparrow(X))$.
- $f \leq \uparrow(\downarrow(f))$.

(Pod $f_1 \leq f_2$ pritom rozumieme, že pre všetky prvky x z definičného oboru (spoločného pre f_1 i f_2) platí $f_1(x) \leq f_2(x)$.)

Tieto vlastnosti sú ekvivalentné tvrdeniu, že pre každé $X \subseteq B$ a $f \in [0,1]^A$ platí

$$f \leq \uparrow(X), \quad \text{akk} \quad X \subseteq \downarrow(f)$$

(alebo, ekvivalentne, že štvorica $\langle \uparrow, \downarrow, \subseteq, \leq \rangle$ je Galoisova konexia).

Poznamenajme, že v dvojhodnotovom prípade, t. j. keď je obor hodnôt relácie R (najviac) dvojprvková množina $\{0,1\}$, znamená $R(a,b) = 1$, že atribút a je atribútom objektu b, a opačne, $R(a,b) = 0$ znamená, že a nie je atribútom b. V tomto zmysle je tento prístup zovšeobecnením klasického Ganterovho-Willeho prístupu.

Teraz definujme zobrazenie $\mathrm{cl} : \mathscr{P}(B) \to \mathscr{P}(B)$ ako zloženie zobrazení $\uparrow$ a $\downarrow$: pre všetky $X \subseteq B$ položme $\mathrm{cl}(X) = \downarrow(\uparrow(X))$.

Opäť ľahko vidieť, že cl je operátor uzáveru, čiže že sú splnené nasledujúce tri podmienky:

- $X \subseteq \mathrm{cl}(X)$.
- Ak $X_1 \subseteq X_2$, tak $\mathrm{cl}(X_1) \subseteq \mathrm{cl}(X_2)$.
- $\mathrm{cl}(X) = \mathrm{cl}(\mathrm{cl}(X))$.

Ako v klasickom prípade, i tu hrajú dôležitú rolu tie množiny objektov X, pre ktoré $X = \mathrm{cl}(X)$ (pretože vtedy platí $f = \uparrow(X)$ práve vtedy, keď $X = \downarrow(f)$). Takúto dvojicu $\langle X, \uparrow(X) \rangle$ nazývame *jednostranne fuzzy koncept* (lebo $\uparrow(X)$ je fuzzy množina, kým množina X je klasická). Potom X nazývame *extent* takéhoto konceptu a zodpovedajúcu fuzzy množinu $\uparrow(X)$ jeho *intent*. Kvôli možnosti jednoznačného vzájomného určenia oboch súradníc stačí uvažovať len jednu z nich, často je to prvá: Množina $\{X \in \mathscr{P}(B) : X = \mathrm{cl}(X)\}$ usporiadaná inklúziou je potom (úplný) zväz s operáciami $X_1 \wedge X_2 = X_1 \cap X_2$ a $X_1 \vee X_2 = \mathrm{cl}(X_1 \cup X_2)$. Tento zväz nazveme *jednostranne fuzzy konceptový zväz*.

4 Rice-ov a Siff-ov algoritmus

Použitie formálnej konceptovej analýzy (či už klasickej alebo viachodnotovej) má (podobne ako mnoho iných zhlukovacích metód) jednu veľkú nevýhodu – spravidla priveľké množstvo nájdených zhlukov. Stalo sa to i v našom prípade – jednostranne fuzzy konceptový zväz vzniknutý z našich dát obsahoval viac než 25000 zhlukov. Uvažovať o každom koncepte jednotlivo je pri takomto množstve zrejme nemožné. Je preto potrebné hľadať doplnkové metódy na redukciu tejto škaredej vlastnosti. Niektoré z nich vynašiel tím Radima Bělohlávka (napríklad [2]). Sú založené buď na využití nejakej dodatočnej informácie o objektoch alebo atribútoch (napr. ich usporiadanie alebo nejaká iná relácia), alebo na redukcii kardinality oboru hodnôt použitím tzv. *zdôrazňovačov pravdy*. V našom príklade však nemáme žiadnu dodatočnú informáciu tohto typu a nechceme ani umelo redukovať pomerne malý počet možných hodnôt (to by totiž znamenalo, že v ankete stačí respondentom poskytnúť menej alternatív). (Predsa sme však v tomto smere isté pokusy urobili, keď sme našich sedem hodnôt prirodzene zredukovali na dve, ako možné odpovede na otázku sympatie „+“ (3, 2 a 1) a „−“ (−3, −2, −1 a 0), avšak počet konceptov bol i naďalej priveľký, a to okolo 2000.)

Aby zhluky mohli byť pre používateľa aspoň v nejakom zmysle užitočné, musí ich byť buď dosť málo, alebo malá musí byť ich kardinalita. Na túto požiadavku existuje istá odpoveď, ktorá je na formálnej konceptovej analýze založená, avšak využíva i ďalšie, metrické vlastnosti. Je navrhnutá v našom článku [5] a teraz ju v krátkosti pripomenieme:

Definujme funkciu $\rho : \mathscr{P}(B) \times \mathscr{P}(B) \to \mathbb{R}$ tak, že pre $X_1, X_2 \subseteq B$ položme

$$\rho(X_1, X_2) = 1 - \frac{\sum_{a \in A} \min\{\uparrow(X_1)(a), \uparrow(X_2)(a)\}}{\sum_{a \in A} \max\{\uparrow(X_1)(a), \uparrow(X_2)(a)\}}.$$

Dá sa dokázať, že táto funkcia je na množine všetkých extentov $\{X \subseteq B : \mathrm{cl}(X) = X\}$ metrikou. Poznamenajme, že táto metrika je zovšeobecnením vzdialenostnej funkcie použitej Rice-om a Siff-om v [6], a to

$$\rho'(X_1, X_2) = 1 - \frac{|X_1 \cap X_2|}{|X_1 \cup X_2|}.$$

Teraz vezmeme hierarchický zhlukovací algoritmus z [6] a jednoducho nahradíme ich metriku ρ' našou ρ. Výsledný algoritmus môžeme vyjadriť nasledujúcim pseudokódom:

---

vstup A, B, R
$\mathcal{C} \leftarrow \mathcal{D} \leftarrow \{\mathrm{cl}(\{b\}) : b \in B\}$
kým ($|\mathcal{D}| > 1$) rob
{
  $m \leftarrow \min\{\rho(X_1, X_2) : X_1, X_2 \in \mathcal{D}, X_1 \neq X_2\}$
  $\mathcal{E} \leftarrow \{\langle X_1, X_2\rangle \in \mathcal{D} \times \mathcal{D} : \rho(X_1, X_2) = m\}$
  $\mathcal{V} \leftarrow \{X \in \mathcal{D} : (\exists Y \in \mathcal{D})\langle X, Y\rangle \in \mathcal{E}\}$
  $\mathcal{N} \leftarrow \{\mathrm{cl}(X_1 \cup X_2) : \langle X_1, X_2\rangle \in \mathcal{E}\}$
  $\mathcal{D} \leftarrow (\mathcal{D} \setminus \mathcal{V}) \cup \mathcal{N}$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{N}$
}
výstup $\mathcal{C}$

Premenná $\mathcal{D}$ je množina zhlukov-konceptov v aktuálnej iterácii, $\mathcal{C}$ je zjednotenie všetkých doterajších iterácií $\mathcal{D}$. Číslo m je najmenšia vzdialenosť dvojíc z $\mathcal{D}$. Množina $\mathcal{E}$ obsahuje „hrany“ – dvojice zhlukov z $\mathcal{D}$, ktorých vzdialenosť je práve m a premenná $\mathcal{V}$ obsahuje „vrcholy“ – konce takýchto hrán. Prvky množiny $\mathcal{N}$ sú nové zhluky. Nová iterácia $\mathcal{D}$ nahradzuje „staré“ zhluky z $\mathcal{V}$ „novými“ z $\mathcal{N}$.

Voľne povedané, v každom kroku spájame tie dva zhluky, ktoré sú najbližšie (v zmysle našej metriky) – z novej iterácie ich vyhodíme a namiesto nich do nej pridáme ich spojenie t. j. uzáver ich zjednotenia. Hoci tento algoritmus je v princípe exponenciálny, v prípade, keď v každej iterácii je najbližšie práve jedna dvojica (čo je zrejme pravda pre väčšinu reálnych dát), dostávame len okolo $2|B|$ zhlukov.

V našom prípade sme dostali nasledujúce zhluky. (Čísla v zátvorkách znamenajú kroky. Singletony (jednoprvkové množiny) sú vynechané pre ich zrejmú nezaujímavosť. Zhluky označené (0.*) sú ne-singletony, ktoré vznikli v 0. iterácii, sú uzávermi podčiarknutých objektov.)

(0.1) D03, D08
(0.2) D06, D01
(0.3) D06, D09
(0.4) D07, D05
(0.5) Ch11, Ch17
(0.6) Ch20, Ch19, Ch04
(1) rovnaký ako (0.3)
(2) D06, D01, D09
(3) D06, D01, D02, D09
(4) Ch10, Ch06, Ch02
(5) Ch23, Ch07
(6) rovnaký ako (4)
(7) D03, D10
(8) Ch18, Ch05
(9) Ch21, Ch14
(10) rovnaký ako (0.4)
(11) Ch08, Ch16
(12) D07, D10, D05, D04
(13) Ch18, Ch05, Ch22
(14) D03, D10, D08
(15) Ch23, Ch07, Ch03
(16) Ch08, Ch16, Ch18, Ch05, Ch22
(17) Ch12, Ch23, Ch07, Ch03
(18) Ch12, Ch23, Ch07, Ch03, Ch13

(19) D06, D07, D01, D10, D02, D05, D04, D09
(20) Ch20, Ch04
(21) D11, Ch14, Ch21
(22) Ch10, Ch12, Ch23, Ch07, Ch03, Ch13, Ch06, Ch02
(23) Ch09, D11, Ch07, Ch14, Ch21
(24) rovnaký ako (0.6)
(25) Ch09, D11, Ch11, Ch07, Ch14, Ch21
(26) D03, D06, D07, D01, D10, D02, D08, D05, D04, D09
(27) Ch09, D11, Ch08, Ch11, D10, Ch16, Ch07, Ch18, Ch05, Ch22, Ch14, Ch21
(28) Ch09, D03, D11, D06, D07, Ch08, Ch11, D01, D10, D02, D08, D05, Ch23, Ch16, Ch07, Ch18, Ch05, D04, Ch22, Ch14, Ch21, D09
(29) Ch10, Ch01, Ch12, Ch23, Ch07, Ch03, Ch13, Ch06, Ch14, Ch02, Ch21
(30) Ch09, D03, D11, D06, D07, Ch08, Ch11, D01, D10, D02, D08, D05, Ch23, Ch16, Ch07, Ch18, Ch05, Ch17, D04, Ch22, Ch14, Ch21, D09
(31) Ch09, D10, Ch10, Ch01, Ch12, Ch23, Ch07, Ch03, Ch13, Ch18, Ch05, Ch06, Ch14, Ch02, Ch15, Ch21
(32) všetci žiaci okrem Ch10, Ch01, Ch12, Ch13, Ch06 a Ch15
(33) všetci žiaci

Nasledujúci diagram zobrazuje tieto zhluky (s výnimkou posledných troch). Na ľavej strane sú všetky objekty, smerom doprava sú postupne spájané do väčších a väčších skupín. Tmavosivá farba je použitá na chlapcov a chlapčenské skupiny, svetlosivá znamená diavčatá a dievčenské skupiny. Zmiešané skupiny sú biele.

5 Interpretácia

Naša interpretácia je založená na jednoduchom pozorovaní vyjadrenom slovenským príslovím „Vrana k vrane sadá“, teda že ľudia s podobnými charakteristikami sú priatelia (ak, pravdaže, majú na vytvorenie priateľstva dostatok času a príležitostí, čo je však v našom prípade zrejme splnené). Môžeme teda s veľkou dávkou istoty dedukovať, že zhluky, ktoré vznikli na základe vzájomných hodnotení spolužiakov, ktorí sa navzájom už dostatočne dlho poznajú, pozostávajú z osôb s dobrým vzájomným vzťahom, pričom čím je zhluk menší, tým je ich priateľstvo silnejšie. Pokúsme sa preto zhodnotiť naše zhluky z tohto hľadiska:

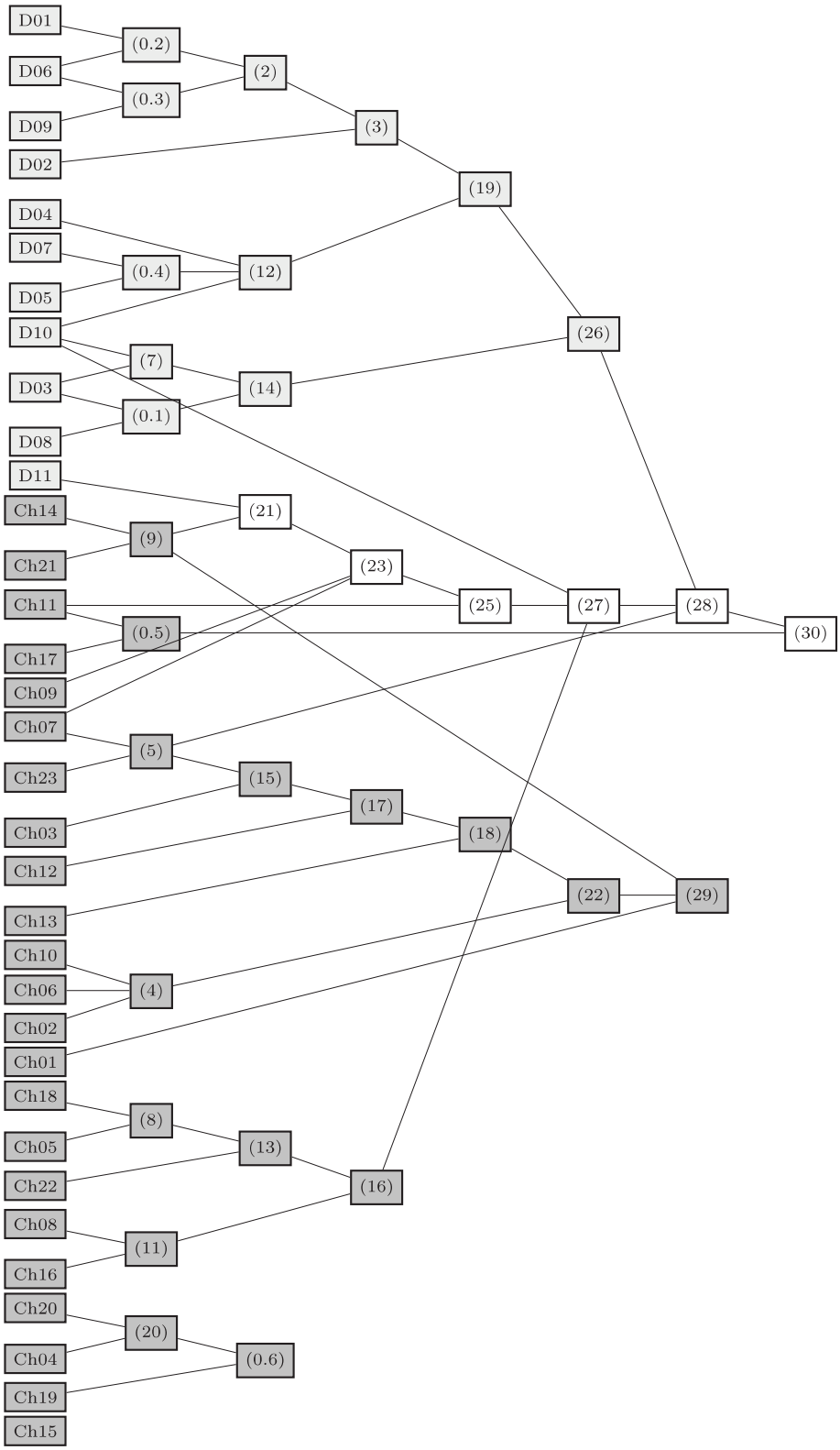
- Najviditeľnejšou črtou je dosť striktné delenie podľa pohlavia (existuje len 9 pohlavne zmiešaných zhlukov, 6 z nich je načrtnutých na diagrame). Toto pozorovanie korešponduje so všeobecne známym správaním 12-ročných detí. Jedinou výnimkou je D11, dievča s chlapčenskými záujmami typu karate.
- Veľmi zaujímavá skupina je (0.6), pozostávajúca z troch chlapcov, ktorí sú spolužiakmi v istom zmysle podceňovaní, pretože majú relatívne horší prospech a zároveň pochádzajú zo slabších sociálnych pomerov.
- Zhluk (31) (nie je na diagrame) je zaujímavý tým, že popri mnohých (hoci nie všetkých) chlapcoch obsahuje jediné dievča – D10, považované za krásku triedy, ktoré je navyše veľmi priateľské (ako vidieť, je spojivom dvoch menších dievčenských skupín).
- Ch15 je triedny exhibicionista. Sám často hovorí, že chce byť stredom pozornosti spolužiakov i učiteľov. To je pravdepodobne príčina, prečo nie je príliš obľúbený. Je iba v troch (na diagrame neznázornených) zhlukoch.
- Ako vo väčšine slovenských tried, aj tu žiaci sedia po dvojiciach. Tento rok im triedna učiteľka dala možnosť urobiť si tieto dvojice po svojom. Je zaujímavé, že sme v zhlukoch odhalili väčšinu z nich: D01 & D02, D06 & D09, D07 & D05, D03 & D08, D10 & D04, Ch14 & Ch21, Ch11 & Ch17, Ch07 & Ch23, Ch10 & Ch02, Ch04 & Ch19.
  Zaujímavé sú i dva páry Ch08 & Ch18 a Ch16 & Ch05. Pôvodné dvojice boli Ch08 & Ch16 and Ch18 & Ch05 (čiže podvojná výmena), ako to vidíme na diagrame. Chlapci však boli v tejto konfigurácii až príliš hluční, boli preto rozsadení.
  Zostávajúce páry D11 & Ch15 (vyššie spomenutí dvaja žiaci), Ch12 & Ch13, Ch22 & Ch01, Ch09 & Ch03 a Ch06 & Ch20 zostali neodhalené.

Samozrejme, vzniká prirodzená otázka, či existujú aj iné zaujímavé a podobne prirodzene interpretovateľné skupiny žiakov. Nepoznáme ich, nevylučujeme však, že existujú. Možno môžu byť odhalené nejakou inou zhlukovacou metódou.

6 Záver

V tomto článku sme popísali experiment s triedou žiakov istého košického gymnázia Po vyhodnotení ich vzájomných vzťahov sme uvažovali ich sociálnu sieť ako viachodnotový objektovo-atribútový model a aplikovali jednostrannú fuzzifikácie formálnej konceptovej analýzy. Počet vzniknutých konceptov – skupín žiakov – bol však priveľký, a preto v praxi nepoužiteľný. Redukovali sme ho preto, a to pomocou našej modifikácie Rice-ovho a Siff-ovho algoritmu, ktorá popri konceptovej analýze využíva aj isté metrické vlastnosti. Získali sme tak rozumný počet zhlukov a na naše milé prekvapenie sa ukázalo, že boli skutočne zaujímavé a ľahko interpretovateľné.

Samozrejme, tento postup môže byť použitý na ľubovoľnú relatívne uzavretú skupinu ľudí, v ktorej


D01
(0.2)
(2)
D06
(0.3)
D09
(3)
D02
(19)
D04
D07
(0.4)
(12)
D05
D10
(26)
(7)
D03
(14)
(0.1)
D08
D11
(21)
Ch14
(9)
(23)
Ch21
Ch11
(25)
(27)
(28)
(30)
(0.5)
Ch17
Ch09
Ch07
(5)
Ch23
(15)
(17)
Ch03
(18)
Ch12
(22)
(29)
Ch13
Ch10
Ch06
(4)
Ch02
Ch01
Ch18
(8)
Ch05
(13)
Ch22
(16)
Ch08
(11)
Ch16
Ch20
(20)
(0.6)
Ch04
Ch19
Ch15

každý pozná každého. Po tomto experimente sme presvedčení, že takáto aplikácia formálnej konceptovej analýzy môže pomôcť manažérovi skupiny (napr. triednemu učiteľovi) lepšie pochopiť štruktúru zverenej sociálnej siete.

Reference

1. Bělohlávek R., Sklenář V., Zacpal J., Crisply Generated Fuzzy Concepts. In: B. Ganter and R. Godin (Eds.), ICFCA 2005, Lecture Notes in Computer Science 3403, Springer-Verlag, Berlin/Heidelberg, 2005, 268–283
2. Bělohlávek R., Vychodil V., Reducing the Size of Fuzzy Concept Lattices by Hedges. In: FUZZ-IEEE 2005, The IEEE International Conference on Fuzzy Systems, May 22-25, 2005, Reno (Nevada, USA), (proceedings on CD), 663–668, abstract in printed proceedings, 44, ISBN 0-7803-9158-6
3. Ben Yahia S., Jaoua A., Discovering Knowledge from Fuzzy Concept Lattice. In: Kandel A., Last M., Bunke H., Data Mining and Computational Intelligence, Physica-Verlag, 2001, 169–190
4. Ganter B., Wille R., Formal Concept Analysis, Mathematical Foundation, Springer Verlag 1999, ISBN 3-540-62771-5
5. Krajči S., Cluster Based Efficient Generation of Fuzzy Concepts. Neural Network World 13, 5, 2003, 521–530
6. Rice M.D., Siff, M., Clusters, Concepts, and Pseudometrics, Electr. Notes Theor. Comput. Sci. 40, 2000

Indexing XML data – the state of the art*

Michal Krátký[1] and Radim Bača[1]

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava–Poruba, Czech Republic
{michal.kratky,radim.baca,vaclav.snasel}@vsb.cz

Abstract. *XML (Extensible Mark-up Language) has recently been embraced as a new approach to data modeling. Nowadays, more and more information is formated as semi-structured data, e.g. articles in a digital library, documents on the web and so on. Implementation of an efficient system enabling storage and querying of XML documents requires development of new techniques. The indexing of an XML document is enabled by providing an efficient evaluation of a user query. XML query languages, such as XPath or XQuery, apply a form of path expressions for composing more general queries.*
In this article, we present the up-to-date state in the area of indexing XML data. A classification of approaches is depicted and some important approaches are described. We are interested in the efficiency of proposed approaches and their ability to manage large XML documents.

Key words: *indexing XML data, XPath, XQuery*

1 Introduction

The mark-up language XML (*Extensible Mark-up Language*) [27] has recently been embraced as a new approach to data modeling [22]. A *well-formed* XML document or a set of documents is an XML database and the associated DTD or schema specified in *XML Schema* language [30] is its database schema. Implementation of a system enabling us to store and query XML documents efficiently (so-called *native XML databases*) requires a development of new techniques [22].

An XML document is usually modeled as a graph of the nodes which correspond to XML elements and attributes. The graph is usually a tree (without `IDREF` or `IDREFS` attributes). A number of special query languages like *XPath* [29] and *XQuery* [28] have been developed to obtain specified data from an XML database. Most XML query languages are based on the XPath language. The language applies *regular path expressions* (*RPEs*) for composing paths in the XML tree. This path is a sequence of steps describing how to get a set of result nodes from a set of input nodes (a set of *context nodes*).

* Work is partially supported by Grant of GACR No. 201/03/0912.

The basic XPath query step in non-reduced notation is `axis::tag[filter]`, which provides by evaluation on node u an answer a set of nodes u':

- relation given by the `axes` contains (u, u'),
- tag for u' is `tag`,
- the condition assigned by `filter` assumes the value TRUE on u'.

The important subset of the language is RPE including two axes `/` and `//`. However, there are other axes generated by the language. For example, `//books/book [author='John Smyth']/title` indicates siblings axes. The language allows to generate common used queries as `//books/book [author= 'John Smith']//book// author` or `/books/book [title= 'The XML Book']/ author`.

As proposed in [4], we can classify RPEs depending on their structure into *simple paths* and *branching queries*. *Branching queries* can be modeled as a tree with more than one branch and *simple path queries* correspond to a single path without conditions. Another classification criterion of XML queries can be defined according to the initial node of the query, thus we recognize *total matching queries* starting in the root node and *partial matching queries* starting in an internal node. If a query matches the string content of a node, we call it a *content-based query*.

There is a lot of approaches to indexing XML data. Each database conference includes an XML track or a workshop. There is a lot of articles in journals as well. It is hard to understand which approach is the best, or, which approach brought a new idea in this problem. This paper tries to clarify an up-to-date state-of-the-art in XML indexing area.

In Section 2, we propose truncated state-of-the-art. We observe that there are two major methods to indexing XML data different to each other, element-based and path-based approaches. Therefore, in Section 3, we described two models of an XML document for these two methods. In Sections 4 and 5, representatives are detailed described. We put emphasis on advantages and disadvantages of these representatives. In Section 6, we show an evaluation.

2 Related work

In recent years many approaches to indexing XML data have appeared. In [4], authors propose a classification of approaches to indexing XML data. The classification includes: element-based, path-based and sequence-based approaches. We adopt this classification.

One way how to index XML data is to summarize all nodes with the same labeled path (e.g. *DataGuide* [23], *Index Fabric* [8], *APEX* [7]). We call this one the *summary index*. DataGuide is an early work which indexes only paths starting in the root node. It only supports total matching simple path queries which do not contain wildcards and descendant axis. Index Fabric extends the data structure called *Patricia trie*, where the summary index and frequently used paths (*refined paths*) are stored. It is not able to efficiently process more complicated queries, which are not prepared in the index. Such inefficiency improves *APEX* which utilizes data-mining algorithm retrieving *refined paths*. APEX is able to proceed unprepared queries more efficiently. However, branching and content-based queries require additional processing.

A simple way to index an XML document is to store each node as a tuple with some document order information. We call these approaches *element-based*. The proposed element-based approaches, *XPA* [11], *XISS* [20], e.g. Zhang et al. [32], can easily solve ancestor-descendant and parent-child relation between two sets of nodes. For better query support XPA can utilize the multi-dimensional data structures and Dietz labeling scheme, so it can find all nodes in relation with one node using single range query. The work proposed by Zhang et al. and *XISS* apply the inverted list for retrieving sets of ancestors and descendants according to a node name and resolve their relationship (make a *structural join*). [32] introduces MPMGJN algorithm which outperform the relational processing of the XML query. Also other approaches [1, 3, 14] improve the structural join since performance of query processing in element-based approaches relies on effectiveness of structural join. An algorithm proposed in [1] introduces *in-memory stack* and improve structural join in that way each node in joined sets is handled only once. Also their algorithm have better results outside RDBMS. *XR-tree* [14] is a special data structure which can even skip the nodes having no matches in ancestor-descendant relationship. The structural join is an extensive operation because the size of the intermediate result sets can be much larger then the size of the result set in the end. The advantage of element-based approaches is that they can process branching queries, partial matching queries and content-based queries without altering the algorithm's performance.

Other approaches try to decrease the number of structural joins and it is the goal of path-based approaches as well (e.g. *Blas* [5], [13], *XRel* [21], *ViST* [31]). Blas indexes suffix paths of an XML document and allows us to retrieve results for a simple path query without expensive joins. XML query is split into paths with length one or more, and results for each path are joined together as in element-based approaches. A disadvantage of the Blas method is that it indexes only paths beginning with `descendant-or-self` axis (abbreviated by `//`) and containing a child axis in every location step of the path. The approach described in [13] works similar to the Blas method. It is a combination of the element-based and path-based methods, where path indexing is used for data set preprocessing before structural joins are applied. The best results are obtained when processing simple path queries. The ViST method converts an XML file into a sequence of pairs, where each pair contains a path in the XML tree. Therefore, the ViST method is sometimes referred to as a *sequence-based* approach. Sequences of the various documents are put together and evaluated by using a labeling scheme. The query is also mapped to the sequence and ViST looks for non-contiguous subsequence matches. Although no join is necessary the result can contain false hits so additional processing of the result set is necessary. XRel is work which use labeled paths only for decrease number of structural joins between tables. In [6] authors propose a path-based approach as well, however, each path of all subtrees is stored.

Recently a lot of works have been introduced for improving efficiency of XML-to-relational mapping [2, 9, 10, 19]. They use an XML-shredding [26] system which decompose XML data into relations and process queries with RDBMS. Most of above mentioned element-based and path-based approaches [11, 1, 20, 32, 5, 21] use some kind of an XML-shredding as a core of their functionality and they can be processed on RDBMS. Relational approaches translate XML queries into SQL queries which use usually self-joins (edge XML-shredding) or even compare unindexed values in relation. When an XML Schema is available an XML-shredding based on inlining [25] can be used for a better performance of SQL queries, but expensive structural joins are still there and number of relations grows rapidly with number of different XML Schemas. More complex content-based or branching queries cannot be easily handled without joining many tables together and this issue limits them for processing large collections. The performance is mainly dependent on a type of the RDBMS applied for processing this type of SQL queries. The single advantage of

these approaches, which can be crucial for some database vendors, is possibility to realize them on RDBMS. Some of these "relational" approaches [11, 21, 32] can be implemented more efficiently only by utilizing multidimensional data structures. So we understand XML processing on relational databases as a different problem, which cannot be compared with more efficient native XML processing.

We summarize often cited approaches which we have mentioned above.

- Approaches which in some way extend a summary index (e.g. *DataGuide*, *Index Fabric*, *APEX*) are able to process only simple and refined queries efficiently. Inefficiency occurs when processing branching and content-based queries. Content-based queries are quite important because we are usually concerned more with the element content then with the structure itself.
- Element-based approaches (e.g. Zhang et al., *XISS*, *XPA*) are able to process all types of queries with the regular effectivity, however, due to fact that they are based on structural joins their level of efficiency is not very high. The structural join is a very expensive operation, as we will demonstrate in our experiments. The time required for query processing increases dramatically with the number of structural joins.
- Path-based approaches (e.g. Blas, [13]) try to decrease the number of structural joins. The Blas is only efficient for simple queries with child axes (abbreviated by `/`) and the number of structural joins increases with the number of branches and `//` axes. The second path-based approach [13] is more about decreasing the number of nodes in a structural join. The best results are again achieved only for simple queries.
- Sequence-based approach *ViST* processes query without a structural join, however, the result can contain false hits, so the query result requires additional processing.

3 A model

Let $\mathbb{X}.Tree$ be the XML tree of an $\mathbb{X}$ XML document. We describe models for both element-based and path-based approaches.

3.1 A model of an XML document for element-based approaches

The element-based approaches store all nodes of $\mathbb{X}.Tree$ in a single relation. The relation scheme is in a form $(id, parentId, tag, value)$ where id corresponds to some numbering scheme, $parentId$ store link to the parent node, tag is a tag of node and $value$ is content of node. In XPath accelerator [11] they use the Dietz numbering scheme, where id of node is represented by preorder and postorder value. The $parentId$ then store only preorder value of the parent node. This numbering scheme allows us to resolve all XPath axes. Let v and v' be nodes of XML tree. Then

- v' is descendant of $v \quad \Leftrightarrow \quad pre(v) < pre(v') \ \wedge \ post(v') < post(v)$,
- v' is following of $v \quad \Leftrightarrow \quad pre(v) > pre(v') \ \wedge \ post(v') < post(v)$.

Similarly, relations *ancestor* and *preceding* can be evaluated between any two nodes. $ParentId$ then allows us to resolve *parent*, *child*, $following-sibling$ and $preceding-sibling$ axes. XPath accelerator also store the indication if the node is attribute or not.

There are other numbering schemes applied in element-based approaches. In XISS [20], authors introduced the interval numbering scheme to be equivalent with the proposed Dietz numbering scheme.

3.2 A model of an XML document for path-based approaches

A path, p, is a sequence $(n_0, n_1, \ldots, n_l)$ of nodes from the root node to a leaf. The length of the path is l. For each p path, there is a labelled path as a sequence of $(t_0, t_1, \ldots, t_l)$, where $t_i, 0 \leq i \leq l$, is the tag of n_i. Let us consider the lp attribute of the p path including the labelled path of the p path. There are differences between path-based approaches. Path-based approach [16] stores all root-to-leaf paths, approach[21] stores paths to all nodes of an XML tree, whereas approach [6] includes each path of all subtrees. For a content-based query, some path-based approaches [16, 6] include a string value, an element content or an attribute value of the last element to each path. Let $\mathbb{X}.P^p$ be a set of all root-to-leaf paths in an $\mathbb{X}.Tree$. Let $\mathbb{X}.P^{lp}$ be a set of all labelled root-to-leaf paths in an $\mathbb{X}.Tree$.

Example 1. In Figure 1(b) we can observe an $\mathbb{X}.Tree$ of the XML document from Figure 1(a). $\mathbb{X}.P^p = \{$(0,1,2,'u'), (0,1,3,'v'), (0,4,5,6,'w'), (0,4,5,7,'x'), (0,8,9,'y'), (0,8,10,'z')$\}$, $\mathbb{X}.P^{lp} = \{(a,b,c), (a,b,d), (a,b,f,c), (a,b,f,d)\}$.

4 Element-based approaches

In this Section, we will describe XPA as the representative of element-based approaches. XPA index is

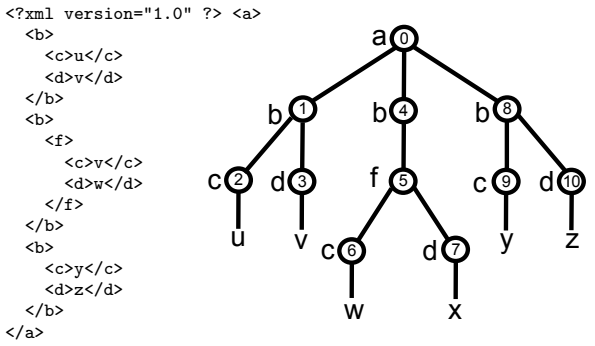


Fig. 1. (a) An XML Document $\mathbb{X}$ (b) $\mathbb{X}.Tree$ of the document.

done after we map the whole XML document into 5-dimensional space. We resolve *location steps* of XPath query step by step. The *location step* consists of name of the axis, name of the node (*nodeName*) and predicate. The predicate is optional but in the case there is some predicate we have to solve *axis::nodeName* part and predicate separately and then we have to union the results. We designed implementation of XPA so that it is possible to handle nested predicates. Solving *axis::nodeName* part of one *location step* is realized using query upon 5-dimensional space.

In Figure 2, an evaluation of four major axis for content node is shown. When the index applies R-trees or other multi-dimensional data structure retrieving of all nodes inside 5-dimensional cube can be performed by a single range query.

XPath query is evaluated from one context node v_c. XPath query consists of a sequence of *location steps*. Query processing is done in these phases:

1. We obtain a set of nodes S_1 as a result of evaluation of the first *location step* from context node v_c. We set $i = 1$.
2. The set S_i is established as a set of context nodes for the following step.
3. We evaluate $(i + 1)$th *location step* for every context node from the set S_i and the result is a set of nodes S_{i+1}. We increment i by one.
4. Phases 2 and 3 are repeated until the last *location step* of XPath query is evaluated.
5. Set of nodes S_i is the result of the XPath query.

That means running many range queries during every phase 3. With increasing number of *location steps* the execution time of the query increases as well. Size of the set S_i which is created during each *location step* may be much larger then the size of the XPath query result. Such inefficiency leads to unnecessary execution time overhead.

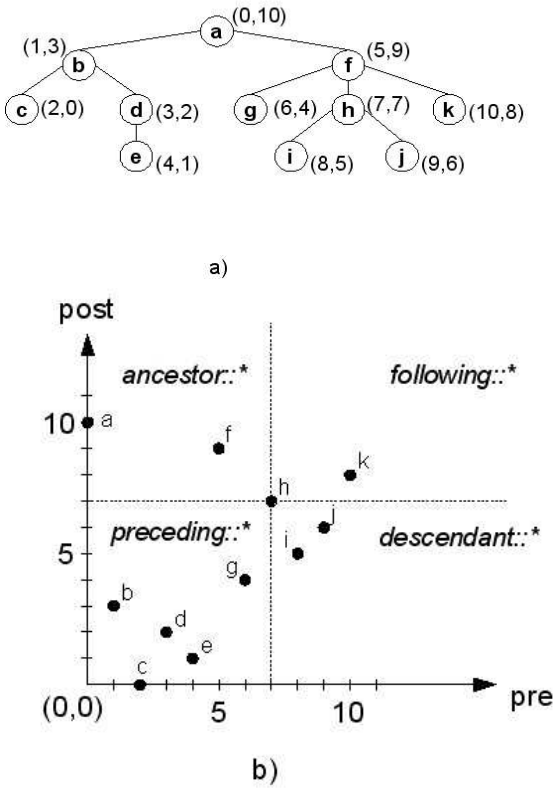


Fig. 2. (a) Evaluation of an XML tree with pairs $(pre(v), post(v))$. (b) Node distribution in $pre/post$ plane and four major axes for a context node h.

5 Path-based approaches

We consider two path-based approaches: a relational approach [6] and multi-dimensional approach [16]. Other path-based approaches are similar to these methods, e.g. [21] is similar to [6].

Various numbering schemes are applied in path-based approaches. A global numbering scheme is applied in approaches [16, 6, 21]. In Figure 1(b), we observe the global ordering of an XML tree. There is a local numbering scheme where a node of a path is evaluated by a number which is unique among its siblings. *Dewey Order* [26] is an example of the numbering scheme. This numbering scheme is more suitable for both update and insert operations in an XML document. Obviously, this numbering scheme is possible to apply for path-based approaches depicted above.

In work [6], a relation scheme (`HeadId`, `Labeled path`, `String value`, `Path`) is defined. In Table 1, we observe an example of the relation for the XML document from Figure 1(a). Authors propose two indices

HeadId	Labeled path	String value	Path
0	a	null	
0	ba	null	1
0	ba	null	4
0	ba	null	8
0	cba	null	1,2
0	cba	u	1,2
0	dba	null	1,3
0	dba	v	1,3
0	fba	null	5,4
0	cfba	null	6,5,4
	...		
5	f	null	
5	cf	null	6
5	cf	w	6
5	df	null	7
5	df	x	7
	...		

Table 1. An example of the relation for an XML document from Figure 1(a).

id_p, id_{lp}	Path	id_t
0,0	0,1,2	id_T('u')
1,1	0,1,3	id_T('v')
2,2	0,4,5,6	id_T('w')
3,3	0,4,5,7	id_T('x')
4,0	0,8,9	id_T('y')
5,1	0,8,10	id_T('z')

Table 2. Multidimensional points for an XML document from Figure 1(a).

ROOTPATHS and DATAPATHS. The ROOTPATHS index includes a concatenation of `StringValue` and reversed `Labeled path` for each root-to-leaf path and paths are retrieved for a query. The DATAPATHS index includes a concatenation of `StringValue`, `HeadId` and reversed `Labeled path` for each path of all subtrees and paths are retrieved for a query. Each attribute is indexed by a B-tree, consequently simple-path queries are processed in one index search.

In work [16], we define a multi-dimensional point $(id_p, id_{lp}, id_{n_0}, id_{n_1}, \ldots, id_{n_l}, id_t)$ for each root-to-leaf path. One point built for each leaf of the tree is stored in a multidimensional data structure [24], e.g. R-tree [12]. In Table 1(a), points for an XML document from Figure 1(a) are put forward.

Although there are some differences, common issues for path-joining remain. In the case of the first approach, there is a redundancy, e.g. the same reversed labelled paths appear in many records. Authors propose that this problem is possible to solve by a compression. However, the depicted redundancy does not appear in the case of the second approach. Querying a path's node is processed by the multi-dimensional range query [24] in this case. Multidimensional forests [17] and Signature multidimensional data structures [18] are applied for a handling various path length and more efficient range queries, respectively. When a twig query is considered, the path-join is a common operation for all path-based approaches.

6 An evaluation

Although, basic idea of element-based approaches is the same, each new paper often overcomes the efficiency of these approaches. In [11] author compares XPA with a simple element-based approach. XPA is more efficient than the proposed index. However, the overhead of structural joins remains for all these approaches. In [13] authors applied path-based features for an element-based method and their method is more efficient than other element-based approaches. When real data collections are considered, it seems that path-based approaches overcome element-based approaches. In [15], authors compared MDX with XISS and XPA. They put forward that MDX is more than 10× more efficient when simple-path and content-based queries are considered. It seems that a problem of path-based approaches appear when twig queries with many twigs are processed. However, this feature has not been published yet.

7 Conclusion

In this article, we present the up-to-date state in the area of indexing XML data. A classification of approaches is depicted and some important approaches are described. We are interested in the efficiency of proposed approaches and their ability to manage large XML documents.

References

1. Al-Khalifa S., Jagadish H.V., and Koudas N., Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In Proceedings of ICDE'02, 2002
2. Amer-Yahia S., Du F., and Freire J., A Comprehensive Solution to the XML-to-Relational Mapping Problem. In Proceedings of WIDM 2004, New York, USA, ACM Press, 2004, 31–38
3. Bruno N., Srivastava D., and Koudas N., Holistic Twig Joins: Optimal XML Pattern Matching. In Proceedings of SIGMOD Conference, 2002, 310–321
4. Catania B., Maddalena A., and Vakali A., XML Document Indexes: A Classification. IEEE Internet Computing, 9, 5, 2005, 64–71

5. Chen Y., Davidson S.B., and Zheng Y., Blas: an Efficient xpath Processing System. In Proceedings of ACM SIGMOD 2004, New York, NY, USA, 2004, 47–58
6. Chen Z., Korn G., Koudas F., Shanmugasundaram N., and Srivastava J., Index Structures for Matching XML Twigs Using Relational Query Processors. In Proceedings of ICDE 05, IEEE Computer Society, 2005, 1273–1273
7. Chung C.-W., Min J.-K., and Shim K., Apex: An Adaptive Path Index for XML Data. In Proceedings of ACM SIGMOD 2002, New York, NY, USA, 2002, 121–132
8. Cooper B., Sample N., Franklin M.J., Hjaltason G.R., and Shadmon M., A Fast Index for Semistructured Data. In Proceedings of VLDB'01, 2001, 341–350
9. DeHaan D., Toman D., Consens M.P., and Özsu M.T., A Comprehensive XQuery to SQL Translation Using Dynamic Interval Encoding. In Proceedings of the 2003 ACM SIGMOD, New York, USA, ACM Press, 2003, 623–634
10. Georgiadis, H. and Vassalos V., Improving the Efficiency of XPath Execution on Relational Systems. In Proceedings of EDBT 2006, Springer–Verlag, Jan 2006, 570–587
11. Grust T., Accelerating XPath Location Steps. In Proceedings of the 2002 ACM SIGMOD, Madison, USA, ACM Press, June 4-6, 2002
12. Guttman A., R-Trees: A Dynamic Index Structure for Spatial Searching. In Proceedings of ACM SIGMOD 1984, Boston, USA, June 1984, 47–57
13. Hanyu Li W.H., Li Lee M., A Path-Based Labeling Scheme for Efficient Structural Join. In Proceedings of XSym 2005, Springer–Verlag, 2005, 34–48
14. Jiang H., Lu H., Wang W., and Ooi B., XR-Tree: Indexing XML Data for Efficient Structural Join. In Proceedings of ICDE, 2003, India, IEEE, 2003
15. Krátký M., Bača R., and Snášel V., On the Efficient Processing Regular Path Expressions of an Enormous Volume XML Data. In Accepted at DEXA 2007, Regensburg, Germany, 2007
16. Krátký M., Pokorný J., and Snášel V., Implementation of XPath Axes in the Multi-dimensional Approach to Indexing XML Data. In Current Trends in Database Technology, EDBT 2004, Springer–Verlag, 3268, 2004
17. Krátký M., Skopal T., and Snášel V., Multidimensional Term Indexing for Efficient Processing of Complex Queries. Kybernetika, Journal, 40, 3, 2004, 381–396
18. Krátký M., Snášel V., Zezula P., and Pokorný J., Efficient Processing of Narrow Range Queries in the R-Tree. In Proceedings of IDEAS 2006, IEEE CS Press, 2006
19. Krishnamurthy R., Kaushik R., and Naughton J.F., Efficient XML-to-SQL Query Translation: Where to Add the Intelligence? In Proceedings of the 30th VLDB Conference, 2004
20. Li Q. and Moon B., Indexing and Querying XML Data for Regular Path Expressions. In Proceedings of 27th International Conference on VLDB'01, 2001
21. Yoshikawa T.S.M., Amagasa T. and Uemura S., Xrel: a Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases. ACM Trans. Inter. Tech., 1, 1, 2001, 110–141
22. Pokorný J., XML: a Challenge for Databases? Kluwer Academic Publishers, Boston, 2001, 147–164
23. Goldman J.W.R., DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In Proceedings of VLDB, 1997, 436–445
24. Samet H., *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006
25. Shanmugasundaram J. and at al., A General Technique for Querying XML Documents Using a Relational Database System. SIGMOD Rec., 30, 2001, 20–26
26. Tatarinov I. and at al., Storing and Querying Ordered XML Using a Relational Database System. In Proceedings of the ACM SIGMOD 2002, New York, NY, USA, ACM Press, 2002, 204–215
27. W3 Consortium. Extensible Markup Language (XML) 1.0, W3C Recommendation, 10 February 1998, `http://www.w3.org /TR/REC-xml`.
28. W3 Consortium. XQuery 1.0: An XML Query Language, W3C Working Draft, 12 November 2003, `http://www.w3.org/TR/xquery/`.
29. W3 Consortium. XML Path Language (XPath) Version 2.0, W3C Working Draft, 15 November 2002, `http://www.w3.org/TR/xpath20/`.
30. W3 Consortium. XML Schema Part 1: Structure, W3C Recommendation, 2 May 2001, `http://www.w3.org/TR/xmlschema-1/`.
31. Wang H., Park S., Fan W., and Yu P.S., ViST: a Dynamic Index Method for Querying XML Data by Tree Structures. In Proceedings of the ACM SIGMOD 2003, ACM Press, 2003, 110–121
32. Zhang C., Naughton J., DeWitt D., Luo Q., and Lohman G., On Supporting Containment Queries in Relational Database Management Systems. In Proceedings of the ACM SIGMOD 2001, New York, USA, ACM Press, 2001, 425–436

Vliv nastavení slovníku na účinnost komprese malých souborů*

Jan Lánský and Michal Žemlička

Univerzita Karlova, Matematicko-fyzikální fakulta, Malostranské nám. 25, 118 00, Praha 1
zizelevak@gmail.com, michal.zemlicka@mff.cuni.cz

Abstrakt *Při kompresi velkých kolekcí malých textových souborů, jako jsou například zprávy elektronické pošty, články z novin a časopisů nebo webové stránky, zjišťujeme, že mnohé kompresní techniky zde nebývají tak úspěšné jako na větších dokumentech. Rozhodli jsme se proto nalézt vhodné modifikace nebo nastavení parametrů existujících textových kompresních metod, které by úspěšně pracovaly v prostředí, kde je nutné uchovávat velké množství samostatně přístupných malých souborů, například ve webovém vyhledávači. Jedním z faktorů ovlivňující chování testovaných kompresních metod je počáteční nastavení jejich slovníku. Při kompresi velkých souborů je význam tohoto nastavení velmi malý, a proto bývá autory programů využívajících těchto metod počáteční nastavení slovníku zjednodušeno na minimum — bývají prázdné. Dle našich měření je význam vhodného počátečního naplnění slovníku při kompresi malých souborů značný.*

1 Motivace

Vývojáři obvykle optimalizují textové kompresní metody na velké soubory nebo jejich kolekce. Domníváme se, že je zajímavé se zabývat i kompresí velkých kolekcí malých souborů, kde můžeme k jednotlivým souborům přistupovat samostatně, tedy bez použití `tar` pro konverzi kolekce do jednoho velkého souboru či obdobných technik. Například v prostředí internetu je nutné přenášet po síti od serveru k uživateli konkrétní webové stránky podle přání uživatele. Pomocí komprese bychom mohli snížit nároky systému na diskový prostor. Mnoho kompresních metod vyžaduje nějakou minimální velikost souboru určeného ke kompresi, aby došlo k rozumnému zmenšení velikosti komprimovaného souboru. V případě některých textových kompresních metod je tato minimální velikost souboru dána nutností přenášet slovník základních kompresních jednotek (slabik, slov) mezi kodérem a dekodérem. Předpokládáme, že při zpracování kolekcí malých, individuálně přístupných dokumentů je možné výrazně zlepšit kompresní poměr, vytvoříme-li slovník častých kompresních elementů, které se vyskytují ve velké části dokumentů. Tento slovník častých elementů se pak využije pro vlastní inicializaci slovníku elementů užitých v různých kompresních metodách.

V tomto článku se zaměříme na slovní a slabikové kompresní metody a na rozdílné způsoby inicializace jejich slovníků. Nastavení algoritmů jsme testovali na třech různých jazycích: angličtině, češtině a němčině, reprezentujících tři rozdílné třídy jazyků. Jejich nejdůležitější vlastnosti shrnuje tento přehled:

angličtina – slova mají několik málo gramatických tvarů a jsou poměrně krátká; v jejich psané podobě je těžké rozeznat hranice slabik;

čeština – slova mají velmi mnoho gramatických tvarů a jsou delší; v jejich psané podobě je snadné rozeznat hranice slabik;

němčina – slova jsou často tvořena složením jiných slov (mohou tedy být i velmi dlouhá) a mají o trochu více gramatických tvarů než v angličtině; hranice slabik jsou poměrně snadno rozpoznatelné;

2 Příbuzné práce

V této sekci se zaměříme na práce pojednávající o kompresi malých textových souborů a dále o alternativních přístupech k řešení problému přenosu slovníku mezi kodérem a dekodérem.

2.1 Komprese malých souborů

Význam vhodné volby slovníku pro kompresi textu nad velkou abecedou byl zmíněn například v [10]. Zvláště na velmi malých souborech je tento efekt velmi výrazný. V práci [8] byla zmíněna důležitost problému různého nastavení slovníku pro různé jazyky.

V práci [16] byl představen algoritmus pro kompresi krátkých textových zpráv (SMS), který měl tak nízkou časovou a prostorovou náročnost, aby byl vhodný pro použití v mobilních zařízeních. Tato metoda používala statistický kontextový model pro jednotlivé symboly a následně pak aritmetické kódování. Model použitý v této metodě byl statický a byl nacvičen na vzorcích textových dat. Tato metoda byla testována na datech o velikosti 50 - 250B, tedy na menších souborech než je zaměření naší práce.

Práce [7] se zabývá kompresní metodou založenou na využití posloupností kontextových stromů, kde každý strom je rozšířením předchozího stromu. Při kó-

* Práce byla částečně podporována Národním programem výzkumu v rámci projektu Informační společnosti 1ET100300517 a Grantovou agenturou Univerzity Karlovy v ramci projektu Slabikova komprese (číslo 1607 v sekci A).

dování symbolu se začíná hledat v prvním kontextovém stromě. Pokud zde symbol není nalezen, pomocí escape sekvence se přepneme do následujícího stromu. Stromy se tvoří podobně jako pracuje například metoda PPM [12], ale s tím rozdílem, že stromy jsou vytvořeny staticky na základě vzorků textových dat. Tato práce byla také zaměřena na kompresi velmi malých souborů, testovány byly i soubory textové. Tato metoda dávala zajímavé výsledky pouze pro data do velikosti 200B, pro data větší velikosti nebyla příliš účinná.

2.2 Komprese slovníku

V článku [8] je popsána metoda, která využívá sdílení slovníku častých elementů mezi kodérem a dekodérem. Přenášeny jsou pouze ty elementy (slabiky, slova), které nejsou v tomto sdíleném slovníku.

Alternativním přístupem může být zakódování celé množiny slov či slabik a její přenos mezi kodérem a dekodérem jako součástí komprimovaného souboru. V článku [9] je navržena metoda TD3 pracující na principu efektivního kódování datové struktury trie, pomocí které je slovník elementů reprezentován. Inspirací pro tuto metodu byl dobře známý princip přední komprese. Metoda TD3 dává dobré výsledky na větších souborech (zhruba od 20 kB), ale na velmi malých souborech tvoří zakódovaný slovník značnou část kódované zprávy.

Existují některé další metody pro kompresi množiny řetězců jako celku, které jsou navrženy pro kompresi řetězců nad velmi omezenou abecedou, například v článku [11] je použito jen 26 písmen malé abecedy. V článku [5] je popsána metoda kombinující kompresi předpon a přípon, které byla navržena pro slovníky dlouhých elementů, tedy pro případ slabik není vhodná.

3 Komprese textových dokumentů

Při kompresi textů můžeme používat jako základní jednotku znaky, slova, či slabiky. Většina metod byla vyvinuta pro kompresi po znacích, pak byla upravena na slova, případně i na slabiky.

Při adaptaci klasických znakových metod na slovní nebo slabikové metody se musí většinou podstatně modifikovat datové struktury, aby byly schopny pracovat místo s 256 znaky s předem neurčeným a navíc vysokým počtem slov či slabik. Při kompresi nad velkou abecedou musí kodér informovat dekodér, jaké prvky obsahuje abeceda, kterou používá. Nejčastěji se tento problém řeší přidáním zakódované abecedy k vlastnímu zakódovanému dokumentu [9].

3.1 Komprese po slovech

Při použití slovních kompresních metod [21] je nutné rozdělit dokument na posloupnost slov a neslov. Jako slova se nejčastěji označují řetězce písmen a číslic, neslovy jsou pak řetězce zbylých znaků. Při dělení dokumentu se hledají maximální alfanumerické řetězce, které se označují za jednotlivá slova. Řetězce znaků, které zůstanou mezi slovy, se prohlásí za jednotlivá neslova.

Můžeme tedy vycházet z předpokladu, že v dokumentu se pravidelně střídají slova a neslova. Dále se používá heuristika, že slovo bývá obvykle následováno speciálním typem neslova – "mezerou". Můžeme si tedy dovolit neslova ("mezery") vynechat a nekódovat. Správná dekomprese se zaručí tím, že pokud dekódujeme posloupnost dvou po sobě jdoucích slov automaticky mezi ně vložíme mezeru, která byla při kompresi vynechána. Pokud po slovu nenásleduje mezera, je nutné tuto skutečnost zakódovat pomocí speciálního symbolu.

Pro praktické použití se neuvažují neomezeně dlouhá slova a neslova, ale jejich délka se omezuje nějakou rozumnou konstantou. Příliš dlouhé řetězce se rozdělují, a aby byl zachován model střídání slov a neslov, musíme mezi ně vložit řetězec nulové délky opačného typu (prázdné slovo, neslovo). Například pokud rozdělíme slovo na dvě slova, musíme mezi ně vložit prázdné neslovo.

3.2 Komprese po slabikách

Použijeme-li slabikové kompresní metody, musíme slova dále dělit na slabiky. Přestože slabika je logickou jednotkou, ze kterých se slova skutečně skládají, rozdělit slovo na slabiky není vždy snadné. Budiž nám útěchou, že pro potřeby komprese není nezbytné, aby dělení slov na slabiky bylo vždy z lingvistického hlediska zcela korektní; stačí se tomuto ideálu jen přiblížit. Důležité je, aby z námi vytvořených slabik či "slabik" bylo možné zrekonstruovat původní text.

Slabiku jsme definovali jako posloupnost hlásek, která obsahuje právě jednu podposloupnost samohlásek. Z definice vyplývá, že počet maximálních podposloupností samohlásek ve slově se rovná počtu slabik ve slově. Například slovo *famous* obsahuje dvě sekvence samohlásek *a* a *ou*, tedy toto slovo je tvořeno dvěma slabikami: *fa* a *mous*. Slovo *pour* obsahuje pouze jednu maximální podposloupnost samohlásek *ou*, tedy celé slovo je tvořeno jen jednou slabikou.

Formální definice pojmů písmeno, samohláska, souhláska, slovo, slabiky a jazyk a jednotlivé algoritmy dělení slov na slabiky jsou podrobně popsány v článku [8]. Odlišný přístup k dělení slov na slabiky (za pomocí genetických algoritmů) lze nalézt v [19].

Protože vliv algoritmu dělení slov na slabiky na dosažený kompresní poměr není příliš významný, rozhodly jsme se použít pro češtinu a němčinu algoritmus P_{UML} a pro angličtinu algoritmus P_{UL}.

Tyto algoritmy rozdělí slova na bloky samohlásek a souhlásek. Bloky samohlásek tvoří základy slabik a bloky souhlásek se k těmto základům slabik přidávají. Algoritmus P_{UML} rozdělí blok souhlásek na dvě poloviny rovnoměrně mezi sousední bloky samohlásek (v případě liché velikosti bloku souhlásek mírně preferuje levý blok samohlásek). Algoritmus P_{UL} přiřadí celý blok souhlásek k levému bloku samohlásek.

4 Kompresní metody

Uvažujme rozšířenou kategorizaci slov, kdy místo dělení textu na slova a neslova budeme dělit text na slova velká (psaná velkými písmeny), malá (psaná malými písmeny), smíšená (začíná velkým písmenem a pokračuje malými písmeny), numerická (obsahuje pouze číslice) a speciální (obsahuje ostatní znaky – odpovídá neslovům z dvouprvkového dělení). V naší práci vycházíme z předpokladu, že text je strukturován do vět a lze jej popsat těmito pravidly: Věta obvykle začíná smíšeným slovem (první písmeno je velké, zbylá malá) a končí speciálním slovem (neslovem v klasickém pojetí), které obsahuje tečku. Obvykle se dále ve větě pravidelně střídají malá a speciální slova. Začíná-li věta velkým slovem, pak se obvykle dále v ní pravidelně střídají velká a speciální slova.

Po rozdělení slov na slabiky nastává s tímto modelem problém. Každé slovo má jiný počet slabik. Zatímco malé slovo je většinou následováno speciálním slovem, tak malá slabika může být následována jak malou slabikou, tak speciální slabikou.

Kódování použité abecedy elementů probíhá takto: Pro každý jazyk máme slovník častých elementů (slabik či slov). Tímto slovníkem častých elementů můžeme naplnit při inicializaci slovního nebo slabikového algoritmu jeho slovník. Potom nám stačí mezi kodérem a dekodérem přenášet pouze informaci o elementech, které se v dokumentu vyskytly a zároveň se nenachází ve slovníku častých elementů. Toto vylepšení je nejvíce užitečné pro malé soubory, na větších souborech se jeho vliv stává zanedbatelným. Konstrukce slovníku častých elementů je hlavní náplní tohoto článku a je popsána v sekci 5.

4.1 LZWL

Algoritmus LZW [20] je slovníkovou kompresí pracující se znaky. Slabikovou verzi této metody značíme LZWL. Algoritmus LZWL může pracovat jak nad slabikami získanými libovolným algoritmem dělícím slova na slabiky, tak i nad slovy.

```
01  Initialize dictionary with empty syllable
    and frequent syllables of given language
02  OLDSTRING = empty syllable
03  NEWSTRING = empty syllable
04  SYLLABLE = empty syllable
05  WHILE not end of input stream OR SYLLABLE is
                                        not empty
06    IF NEWSTRING + SYLLABLE is in the dictionary
07      NEWSTRING = NEWSTRING + SYLLABLE
08      SYLLABLE = next input syllable
09    ELSE
10      IF NEWSTRING is empty syllable
11        output the code of empty syllable
12        encode SYLLABLE by character-by-character
            coding method and output this code
13        add SYLLABLE to the dictionary
14        SYLLABLE = empty syllable
15      ELSE
16        output the code for NEWSTRING
17        IF OLDSTRING is not empty syllable
18          FIRSTSYLLABLE = first syllable of
                                        NEWSTRING
19          add OLDSTRING and FIRSTSYLLABLE to the
                                        dictionary
20        ENDIF
21      ENDIF
22      OLDSTRING = NEWSTRING
23      NEWSTRING = empty syllable
24    ENDIF
25  ENDWHILE
```

Obrázek 1. LZWL – komprese.

Nejdříve připomeňme původní verzi LZW. Metoda využívá slovník frází reprezentovaný datovou strukturou trie. Fráze jsou číslovány přirozenými čísly v pořadí, jak byly vloženy do slovníku.

Slovník je na začátku naplněn všemi znaky z abecedy (znakové sady). V každém dalším kroku se ve slovníku hledá nejdelší řetězec S odpovídající prefixu nezpracované části vstupu. Pozice S ve slovníku je následně zapsána na výstup. Do slovníku je vložena nová fráze složená z S a jednoho znaku bezprostředně za S. Aktuální pozice ve vstupním souboru se posune o délku S vpřed.

Dekódování je vcelku jednoduché: dostáváme čísla frází, které jsou ve slovníku. Z dekódované frází postupně rekonstruujeme slovník tak, jak to bylo při kódování. Přijde-li číslo, jemuž ve slovníku žádná fráze neodpovídá, použijeme poslední dekódovanou frázi prodlouženou o její první znak. Je to totiž jediný možný případ, kdy může přijít kód, k němuž ještě ve slovníku není zcela definovaná položka (nově přidávaná fráze je vlastně předposlední fráze doplněná o první znak naposledy dekódované fráze – proto je slovník

u dekodéru o jednu položku zpožděn proti slovníku u kodéru).

Slabiková verze (obr. 1) pracuje nad abecedou slabik. Slovník inicializujeme prázdnou slabikou a všemi slabikami z databáze frekventovaných slabik daného jazyka. Nalezení řetězce NEWSTRING, zakódování jeho čísla i jeho prodloužení je obdobné jako ve verzi pro znaky – jen s tím rozdílem, že řetězec NEWSTRING není tvořen znaky, ale slabikami. Může se stát, že potřebujeme zakódovat slabiku SYLLABLE, která ještě není ve slovníku. Pak uvedeme prázdnou slabiku, a za ni zapíšeme nově přidávanou slabiku znak po znaku.

```
01  Initialize data structures
02  WHILE not end of input stream
03    SYLLABLE = next input syllable
04    EXPECTEDTYPE = expected type of SYLLABLE
05      TYPE = type of SYLLABLE
06      IF EXPECTEDTYPE != TYPE
07        output escape sequence for correct type
08      ENDIF
09      IF SYLLABLE is unknown
10        CODE = escape code for new node in TYPE
                                        Huffman tree
11        output CODE
12        encode SYLLABLE by unknown-syllable
            coding algorithm and output this code
13        insert SYLLABLE to the TYPE Huffman
                                                tree
14      ELSE
15        CODE = code of SYLLABLE in TYPE Huffman
                                                tree
16        output CODE
17      ENDIF
18      increment weight of SYLLABLE
19      if necessary reorganize the TYPE Huffman
                                                tree
20  ENDWHILE
```

Obrázek 2. HuffSyllable – komprese.

4.2 HuffSyllable (HS)

HuffSyllable (obr. 2), prvně publikovaný v článku [8], je statistická kompresní metoda využívající adaptivní Huffmanovo kódování. Tato metoda je částečně inspirována algoritmem HuffWord [21]. Pomocí metody HuffSyllable lze dokument kódovat po elementech typu slabika nebo slovo.

Pro každý typ elementů (malý, velký, smíšený, číselný a speciální) je postaven adaptivní Huffmanův strom [6], pomocí kterého se kódují elementy daného typu. V inicializační fázi naplníme strom malých slabik typickými slabikami daného jazyka.

V každém kroku algoritmu je odhadnut typ elementu, který má následovat. V případě, že následující element má jiný typ než jsme odhadli, je použit kód espace sekvence pro správný typ elementu. Následně se element zakóduje za použití stromu elementů svého typu. U daného elementu se zvýší četnost o jedna. Odhad typu elementu, který má následovat, se provádí na základě již zkomprimované části dokumentu.

5 Inicializace kompresních slovníků

Vytvoření slovníku často používaných slabik pro daný jazyk je poměrně pracné. Ten pak můžeme použít jako počáteční slovník pro různé kompresní metody. Počáteční nastavení slovníku slabikové komprese má zásadní vliv na její účinnost: Je-li ve slovníku příliš mnoho slabik, může se stát, že většinu z nich ke kompresi nevyužijeme. Kódová slova použitých slabik tak mohou být zbytečně dlouhá. Opačný případ nastává, ponecháme-li slovník na začátku prázdný. Každá slabika pak musí být při svém prvním výskytu draze kódována znak po znaku. Optimální nastavení kompresního slovníku je někde uprostřed.

Pro různé jazyky a různé algoritmy dělení slov na slabiky dostaneme odlišné slovníky. Pravidla pro přidání slabiky do slovníku jsou však jednotná. Slabiky přidané do počátečního slovníku musí být pro daný jazyk a dělení slov na slabiky typické. Jak rozhodnout, kolik slabik do počátečního slovníku zařadit? Zdá se, že tato dvě kritéria pro výběr slabik do počátečního slovníku jsou rozumná:

- Kumulativní kritérium – podíl výskytu dané slabiky na celkovém počtu slabik v kolekci.
- Výskytové (appearance) kritérium – V kolika dokumentech se slabika vyskytuje alespoň jednou.

Kumulativní kritérium bylo použito v [8]. Slovníky zde byly naplněny slabikami vyskytujícími se v dané kolekci s pravděpodobností větší než 1/65000. Takto vytvořený slovník budeme značit C65.

Mohlo se stát, že slovník obsahoval slabiky vyskytující se jen v málo dokumentech, ale hojně.

Domníváme se, že použití výskytového kritéria je v daném případě výhodnější, neboť častěji šetří definice slabik. Vytvořili jsme různé počáteční slovníky pro různé frekvence výskytů (značíme je Axx, kde xx reprezentuje minimální procento dokumentů, v nichž se slabiky vyskytly). Tyto slovníky obsahují slabiky vyskytující se nejméně v 5% (A05), 20% (A20), 40% (A40), 60% (A60), 80% (A80) nebo 100% (A100) dokumentů.

Abychom mohli vytvořit rozumný počáteční slovník slabik, potřebujeme mít dostatečně velkou kolekci dokumentů svým obsahem typických pro daný jazyk.

Jak dokumenty, tak celá kolekce by měly být co do velikosti střední až velké, abychom co nejlépe vyhověli výskytovému kritériu. Nevhodná volba testovacích dokumentů může způsobit, že se do slovníku dostanou jinak vzácně se vyskytující slabiky, což nevhodně zvětší vytvářený slovník. Také se může stát, že se do slovníku nedostanou jinak velmi časté slabiky pro daný jazyk typické. Význam správného počátečního nastavení roste se zmenšující se velikostí komprimovaných souborů. Experimentálně jsme ověřili, že rozdíly v efektivitě komprese mohou být výrazně ovlivněny trénovací množinou.

Použitím sady dokumentů, které chceme komprimovat, pro inicializaci slovníku můžeme (v případě nastavení A05 či A20) zlepšit dosažený kompresní poměr až o 10 %.

Pro tvorbu slovníků a pro získání statistik pro všechny tři jazyky jsme použili tři sady dokumentů – jednu pro češtinu (obsahovala 69 souborů o úhrnné velikosti 15 MB; zdroj: [2]), jednu pro angličtinu (obsahující 334 souborů o úhrnné velikosti 144MB; zdroj: [4]), a jednu pro němčinu (obsahující 100 souborů o úhrnné velikosti 40MB; zdroj: [4]). Velikosti slovníků jsou v Tabulce 1.

Inicializace	Angličtina		Čeština		Němčna	
slovníku	slova	slab.	slova	slab.	slova	slab.
A05	458	151	701	114	688	166
A20	174	84	152	64	163	79
A40	80	53	49	40	63	48
A60	41	35	21	26	27	32
A80	17	22	8	15	11	18
A100	1	2	1	2	1	3

Tabulka 1. Počáteční velikosti slovníků (v KB).

6 Pokusy a výsledky

6.1 Cíl

Metody komprese textu bývají srovnávány na kolekcích velmi velkých dokumentů. Setkali jsme se i s kolekcemi, kde tyto velké soubory vznikly spojením mnoha souborů menších. To je výhodné pro kompaktní archivy, ale ne pro práci s dynamicky se měnícími či náhodně přistupovanými daty, jak je tomu v případě webovských stránek (kde různé stránky se různě mění v čase) nebo v případě novinových článků (kdy některé články mohou být zajímavé dlouho, zatímco jiné články jsou zajímavé nejvýš pouhý den).

Takovéto dokumenty bývají obvykle malé: průměrná velikost webovských stránek je dle zdroje od 5–10 KB ([15]) do 10–20 KB ([14]). Novinové článku v Pražském závislostním korpusu [3] jsou velké asi 1,5 KB a soubory z Kalifornského zákoníku jsou dlouhé v průměru 8KB. Domníváme se, že má smysl měřit i na takovýchto kolekcích.

	Metoda	LZWL		HuffSyl	
Jazyk	Slovník	slabiky	slova	slabiky	slova
CZ	C65	5.14	5.27	4.44	4.46
CZ	A05	5.40	**5.56**	4.42	—
CZ	A20	5.27	5.81	**4.35**	**4.34**
CZ	A40	5.07	5.79	4.39	4.73
CZ	A60	**4.99**	5.78	4.46	4.93
CZ	A80	5.03	5.73	4.64	5.15
CZ	A100	5.25	5.64	5.49	5.56
EN	C65	2.99	2.91	3.23	2.44
EN	A05	3.26	**2.88**	3.36	—
EN	A20	3.09	2.90	3.24	**2.35**
EN	A40	2.99	2.94	**3.23**	2.46
EN	A60	2.97	2.95	**3.23**	2.58
EN	A80	**2.92**	2.97	3.26	2.71
EN	A100	3.03	2.93	3.62	2.93
GE	A05	4.91	4.99	4.06	**3.54**
GE	A20	4.65	4.85	**3.92**	3.60
GE	A40	4.50	4.68	3.98	3.77
GE	A60	4.52	4.64	4.03	4.17
GE	A80	**4.45**	**4.61**	4.17	4.07
GE	A100	4.66	4.66	4.88	4.53

Poznámka: Inicializace slovníku A05 pro češtinu a angličtinu se ukázala jako časově neúměrně náročná, a proto byla příslušná měření vyřazena.

Tabulka 2. Kompresní poměr (v porovnání s původní velikostí souboru) v bitech na znak pro různá naplnění slovníku.

6.2 Zkušební data

Pro testy jsme použili tři sady neformátovaných dokumentů. První sada obsahovala 1000 novinových článků v češtině o průměrné velikosti 1,5KB. Tyto články jsme náhodně vybrali z PDT. Druhá sada obsahovala 1000 zákonů v angličtině o průměrné velikosti 8 KB. tyto dokumenty byly náhodně vybrány z kalifornského zákoníku. Třetí sada obsahovala pouze 89 novinových článků v němčině o průměrné velikosti 3 KB. Tyto články jsme získali z www.sueddeutsche.de.

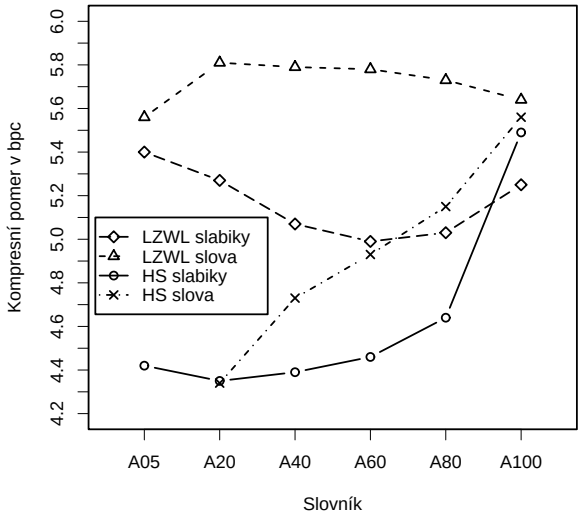
6.3 Výsledky

Porovnejme kompresní poměry v bitech na znak získané různými kompresními metodami pro anglické (obr. 4), české (obr. 3) a německé (obr. 5) texty s různým nastavením slovníků (tab. 2) s nejlepším nastavením pro daný jazyk a kompresní metodu (tab. 3).

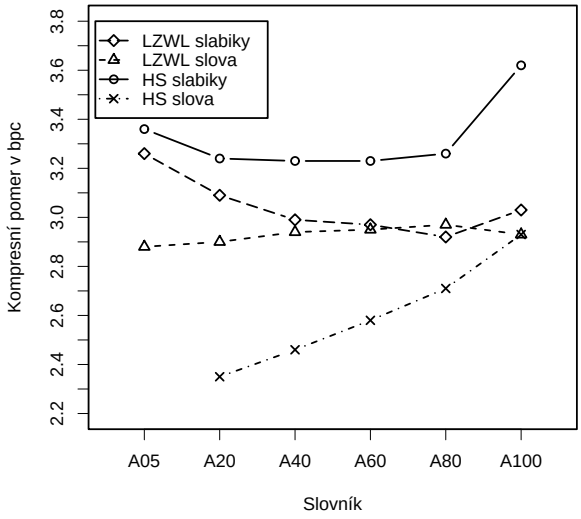
Čeština je bohatá na slova střední délky. Mnohá slova se vyskytují ve více tvarech. Slabiková komprese

Metoda\Jazyk	anglický	český	německý
LZWL(slabiky)	2.92	4.99	4.45
LZWL(slova)	2.88	5.56	4.61
compress 4.0 [18]	3.65	5.43	4.80
HS(slabiky)	3.23	4.35	3.92
HS(slova)	**2.35**	**4.34**	**3.54**
ACM(slova) [13]	2.56	4.78	4.03
FGK [6]	4.58	5.15	4.77
bzip2 [17]	2.42	4.69	3.96

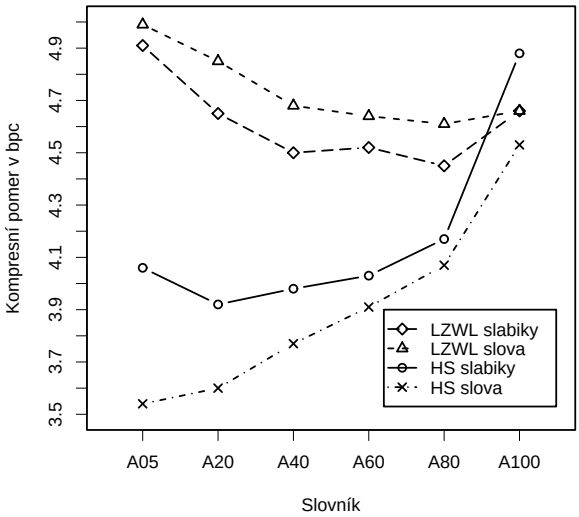
Tabulka 3. Porovnání kompresních poměrů v bitech na znak pro české, anglické a německé dokumenty.



Obrázek 3. Porovnání kompresních poměrů v bitech na znak (nižší hodnota je lepší) pro dokumenty v češtině.



Obrázek 4. Porovnání kompresních poměrů v bitech na znak (menší hodnota je lepší) pro dokumenty v angličtině.



Obrázek 5. Porovnání kompresních poměrů v bitech na znak (nižší hodnota je lepší) pro německé dokumenty.

se proto pro češtinu jeví jako výhodnější pro obě metody (LZWL a HuffSyllable) s jedinou vyjímkou (HuffSyllable a slovník A20), kdy slova jako základní kompresní jednotka vycházejí nepatrně lépe než slabiky.

Angličtina má mnoho velmi krátkých slov. Slova se vyskytují jen v několika málo tvarech. Pro kompresi anglických textů se proto výrazně lépe hodí slovní metody. V případě LZWL je rozdíl menší (v případě slovníku A80 vycházejí dokonce slabiky lépe).

V němčině se setkáváme i s velmi dlouhými slovy. Slova se používají jen v několika málo tvarech. Srovnání slabikové a slovní komprese vychází mezi češtinou a angličtinou: Pro LZWL vycházejí lépe jako základní jednotka slabiky, pro HuffSylable zas slova.

Obecně platí, že pro slabiky se hodí inicializovat slovník pouze velmi frekventovanými slabikami, zatímco pro slova se vyplatí naplnit na začátku slovník více. Důvodem je vysoká cena kódování nově přidávaných slov.

Nejvýhodnějšími nastaveními (z hlediska dosahovaného stupně komprese) pro HuffSyllable jsou A05 a A20, zatímco LZWL vychází nejvýhodněji s A60 či A80.

Naše měření ukázala (tab. 3), že nejlepšího průměrného kompresního poměru bylo dosaženo pro slovní verzi algoritmu HuffSyllable se slovníkem inicializovaným dle A20 pro angličtinu a češtinu a dle A05 pro němčinu. Pro češtinu je také možné použít slabikovou verzi HuffSyllable, která dosahuje srovnatelných výsledků.

7 Závěr

Testovali jsme několik kompresních metod (compress, bzip, upravené (pro slova) aritmetické kódování, FGK, HuffSyllable a LZWL pro slova a slabiky) na třech sadách menších dokumentů – po jedné pro každý z jazyků čeština, angličtina a němčina.

Nejlepších výsledků jsme dosáhli pro slovní verzi HuffSyllable, kdy pro češtinu a angličtinu je nejvýhodnější inicializovat slovník dle A20, pro němčinu pak dle A05. Pro češtinu se velmi slibnou zdá i slabiková verze HuffSyllable, která dosahovala vyšší účinnosti než její slovní obdoba – jedinou vyjímkou bylo nastavení A20, ale rozdíl byl nevýznamný.

Dospěli jsme k závěru, že chceme-li komprimovat velkou kolekci menších dokumentů, může být užitečné vyzkoušet různé metody s různými počátečními nastaveními. Nejúčinnější metoda může záviset na jazyce komprimovaných dokumentů a na jejich průměrné velikosti a struktuře. Výsledky mohou být výrazně ovlivněny volbou počátečního naplnění slovníků.

V našem dalším výzkumu jsme se proto rozhodli porovnat více kompresních metod. Také bychom rádi vyzkoušeli, jak účinné budu testované metody na specifických dokumentech jako jsou zprávy elektronické pošty či webovské stránky.

Reference

1. California law. `http://www.leginfo.ca.gov/calaw.html`.
2. eknihy. `http://go.to/eknihy`
3. The Prague Dependency Treebank. `http://ufal.mff.cuni.cz/pdt/`
4. Project Gutenberg. `http://www.promo.net/pg`.
5. Aoe J.-I., Morimoto K., Shishibori M., and Park K.H., A Trie Compaction Algorithm for a Large Set of Keys. IEEE Transactions on Knowledge and Data Engineering, 08, 3, 1996, 476–491
6. Knuth D.E., Dynamic Huffman Coding. J. of Algorithms, 6, 1985, 163–180
7. Korodi G., Rissanen J., and Tabus I., Lossless Data Compression Using Optimal Tree Machines. In Data Compression Conference, Los Alamitos, CA, USA, IEEE CS Press, 2005, 348–357
8. Lánský J. and Žemlička M., Text Compression: Syllables. In K. Richta, V. Snášel, and J. Pokorný, (eds), DATESO 2005, Prague, Czech Technical University, 2005, 32–45
9. Lánský J. and Žemlička M., Compression of a Dictionary. In V. Snášel, K. Richta, and J. Pokorný, (eds), DATESO 2006, Prague, Czech Technical University, 2006, 11–20
10. Lánský J. and Žemlička M.: Compression of Small Text Files Using Syllables. In Data Compression Conference, 2006, 458
11. Maly K., Compressed Tries. Commun. ACM, 19, 7, 1976, 409–415
12. Moffat A., Implementing the ppm Data Compression Scheme. IEEE Transactions on Communications 38, 1990, 1917–1921
13. Moffat A., Neal R.M., and Witten I.H., Arithmetic Coding Revisited. ACM Transactions on Information Systems, 16, 1998, 256–294
14. O'Neill E.T., Lavoie B.F., and Bennett, R., Trends in the Evolution of the Public Web: 1998–2002. D-Lib Magazine, 9, 4, Apr. 2003
15. Petrakis E.G.M., Searching the Web. `http://www.intelligence.tuc.gr/~petrakis/courses/multimedia/web.pdf`
16. Rein S., Guhmann C., and Fitzek F.H.P., Low-Complexity Compression of Short Messages. In Data Compression Conference, Los Alamitos, CA, USA, IEEE CS Press, 2006, 123–132
17. Seward J., The bzip2 and libbzip2 official home page. `http://sources.redhat.com/bzip2/` as visited on 6th February 2005.
18. Thomas S.W., McKie J., Davies S., Turkowski K., Woods J.A., and Orost J.W., Compress (Version 4.0) Program and Documentation, 1985
19. Üçoluk G. and Toroslu I.H., A Genetic Algorithm Approach for Verification of the Syllable-Based Text Compression Technique. Journal of Information Science, 23, 5, 1997, 365–372
20. Welch T.A., A Technique for High Performance Data Compression. IEEE Computer, 17, 6, 1984, 8–19
21. Witten I.H., Moffat A., and Bell T.C., Managing Gigabytes: Compressing and Indexing Documents and Images. Van Nostrand Reinhold, 1994

Expresivita plánovania založeného na STRIPS a HTN*

Marián Lekavý and Pavol Návrat

Slovenská technická univerzita, Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava, Slovensko
lekavy@fiit.stuba.sk, navrat@fiit.stuba.sk

Abstrakt *Všeobecne sa predpokladá, že expresivita plánovania založeného na operátoroch typu STRIPS je nižšia, než expresivita plánovania založeného na hierarchickej sieti úloh (Hierarchical Task Network, HTN). To by znamenalo, že plánovač založený na HTN dokáže vo všeobecnosti vyriešiť viac domén než plánovač založený na STRIPS. Tento článok ukazuje, že obidva prístupy, tak ako sa prakticky používajú, majú rovnakú expresivitu a dokážu vyriešiť všetky domény riešiteľné Turingovym strojom s konečnou páskou (tzn. riešiteľné bežným počítačom).*

1 Úvod

Dva najznámejšie a najpoužívanejšie prístupy k doménovo nezávislému symbolickému plánovaniu sú plánovanie typu STRIPS (založené na operátoroch) a plánovanie typu HTN (založené na hierarchickej dekompozícii). Plánovanie typu STRIPS je staršie a založené na vytváraní plánu ako reťazca akcií, z ktorých každá má svoju predpodmienku a výsledky. Plánovanie typu HTN je založené na hierarchickej dekompozícii - plánovač začína s počiatočnou úlohou a postupne ju dekomponuje na primitívnejšie úlohy podľa informácií, ktoré má plánovač k dispozícii. Primitívne úlohy, ktoré nie je možné ďalej dekomponovať, tvoria konečný plán.

HTN vzniklo ako rozšírenie „klasického“ plánovania typu STRIPS a dovoľuje plánovaču využívať dodatočné informácie o hierarchickej dekompozícii.

Keďže je hierarchická dekompozícia rozšírením plánovania založeného na operátoroch, vyvstáva otázka expresivity: Je expresivita plánovania typu HTN väčšia než expresivita plánovania typu STRIPS? Dokáže plánovanie typu HTN vyriešiť viac domén než plánovanie typu STRIPS?

Táto otázka bola v minulosti zodpovedaná pozitívne [2]. Tento dôkaz bol však založený na predpoklade, že plánovač HTN môže použiť na značkovanie úloh nekonečnú množinu symbolov. Pre teoretický model HTN to platí, avšak tento predpoklad nemôže splniť žiadny prakticky použiteľný plánovač implementovaný na bežnom počítači.

Tento článok ukazuje, že expresivita plánovania typu STRIPS a plánovania typu HTN sú identické za predpokladu, že použijeme ľubovoľné obmedzenie, ktoré spôsobí že bude proces plánovania pomocou HTN rozhodnuteľný. Tento predpoklad nie je veľmi obmedzujúci, keďže každá implementácia plánovania typu HTN používa takéto obmedzenie pre ukončenie výpočtu v konečnom čase (hoci aj neúspešné).

V tejto kapitole poskytneme stručný prehľad plánovania typu STRIPS a HTN. Kapitola 2 pojednáva o expresivite oboch prístupov. V kapitole 3 ukážeme, že STRIPS dokáže emulovať Turingov stroj s konečnou páskou bez zmeny časovej zložitosti. Kapitola 4 potom ukazuje jednoduchú konverziu domén typu HTN na domény typu STRIPS.

1.1 STRIPS

Základný princíp STRIPS [3] a plánovania typu STRIPS je nájdenie postupnosti akcií, ktorá zmení počiatočný stav sveta na koncový stav sveta. Stav sveta je vyjadrený množinou literálov. Plánovač do plánu postupne pridáva akcie, ktorými sa snaží dosiahnuť korektnú transformáciu počiatočného stavu na koncový.

Plánovanie STRIPS je založené na operátoroch v tvare $Op = (pre, del, add)$, kde pre je predpodmienka ktorá musí byť splnená bezprostredne pred tým ako sa operátor použije, add / del sú množiny literálov, ktoré sú po ukončení operátora pridané/vymazané do/zo stavu sveta. Operátor, ktorý sa vykoná (pridá sa do plánu) sa nazýva akcia.

Od vzniku STRIPS (pred 35 rokmi) vzniklo veľa plánovačov používajúcich myšlienku STRIPS. Väčšina plánovačov sa neobmedzuje na základný formalizmus STRIPS a má rôzne rozšírenia ako zdroje, paralelné vykonávanie, senzorické a koordinačné akcie, podmienené a náhodné akcie atď.

Základný algoritmus plánovania typu STRIPS je založený na sekvenčnom pridávaní akcií do plánu. Ak začína z počiatočného stavu, hovoríme o doprednom reťazení, ak začína z koncového stavu, ide o spätné reťazenie. Ak pridávame akcie na ľubovoľné miesto v pláne, potom hovoríme o prehľadávaní priestoru plánov. Plán sa vytvára len na základe znalostí o predpodmienkach a výsledkoch operátorov a na základe aktuálneho stavu sveta. Plánovač nedostáva žiadne ďalšie informácie, čo je hlavným rozdielom oproti prístupoch typu HTN o ktorých píše ďalšia kapitola.

* Táto práca bola podporená SPVV 1025/04; APVT 51-024604; VG 1/3102/06.

1.2 Hierarchická sieť úloh (HTN)

HTN[7] (Hierarchical Task Network) a prístupy typu HTN sú založené na manuálnej hierarchickej dekompozícii problémovej domény. Plánovač dostáva doménové znalosti vyjadrené ako možné dekompozície úloh na podúlohy. Úlohy môžu byť primitívne (priamo vykonateľné) alebo zložené. Zložené úlohy musia byť ďalej dekomponované na podúlohy. Pre každú zloženú úlohu je jeden alebo niekoľko zoznamov úloh, na ktoré môže byť dekomponovaná. Tento zoznam úloh spolu s ďalšími obmedzeniami (ako napríklad poradie úloh, viazanie premenných alebo vzájomné vylučovanie) sa nazýva sieť úloh.

Hľadanie plánu začína jednou alebo niekoľkými počiatočnými úlohami, ktoré sa postupne dekomponujú na jednoduchšie, až kým všetky úlohy nie sú dekomponované na primitívne. Ak dekompozícia nie je možná (napríklad kvôli nezlučiteľným obmedzeniam), plánovač sa vráti a použije inú dekompozíciu.

Dekompozícia môže byť úplne usporiadaná, ale existujú aj plánovače, ktoré povoľujú prekrývanie podúloh rôznych úloh (napríklad SHOP2 [6]).

Na HTN sa môžeme tiež pozerať ako na rozšírenie plánovania založeného na akciách o gramatiku, ktorá osekáva priestor možných plánov [4].

Základný algoritmus HTN je nasledujúci:

1. Vlož do plánu počiatočné úlohy.
2. Ak plán obsahuje len primitívne úlohy, úspech.
3. Vyber z plánu jednu zloženú úlohu.
4. Nahraď vybranú úlohu jej podúlohami podľa nejakej siete úloh.
5. Vyrieš interakcie a konflikty v pláne. Ak to nie je možné, vráť sa a použi inú dekompozíciu.
6. Pokračuj krokom 2.

Plánovače typu HTN, podobne ako plánovače typu STRIPS, zavádzajú rôzne rozšírenia ako správa zdrojov, paralelné vykonávanie, senzorické a koordinačné akcie. Plánovače typu HTN majú významnú úlohu v plánovaní pre multiagentové systémy. Plánovače ako STEAM [8] alebo PGP/GPGP [5] poskytujú mechanizmy koordinácie a vyjednávania, umožňujúc tak agentom spolupracovať na splnení spoločných úloh. STEAM poskytuje tímové úlohy, ktoré sa ďalej dekomponujú na úlohy pre jednotlivé agenty. Oproti tomu pri PGP má každý agent svoju vlastnú počiatočnú úlohu ktorú má splniť a časti stromov dekompozície jednotlivých agentov sa môžu prekrývať (obrázok 1).

2 Expresivita

Prevláda všeobecný názor, že expresivita plánovania typu HTN je väčšia ako expresivita plánovania typu STRIPS. Pod pojmom expresivita rozumieme množinu domén, ktoré dokáže plánovací systém vyriešiť.

Existuje dôkaz [2] založený na transformácii plánovacieho problému na gramatiku, ktorý ukazuje, že plánovanie typu STRIPS zodpovedá regulárnym gramatikám a plánovanie typu HTN zodpovedá bezkontextovým gramatikám. Tým potom ukazuje, že plánovanie typu HTN pokrýva širšiu triedu domén.

Hlavný rozdiel, ktorý spôsobuje tento výsledok, je fakt, že teoretický model HTN používa na značkovanie úloh nekonečnú množinu symbolov, takže dokáže pre mnohé domény vytvoriť nekonečný priestor plánov. Na druhej strane má STRIPS a plánovanie typu STRIPS konečný priestor plánov, ak nepoužíva niektoré rozšírenia, ktoré sa zvyčajne považujú za neštandardné. Niektoré plánovače typu STRIPS (napríklad FHP [9]) rozširujú základný formalizmus o funkčné symboly, čím umožňujú vyjadriť aj nerozhodnuteľné problémy.

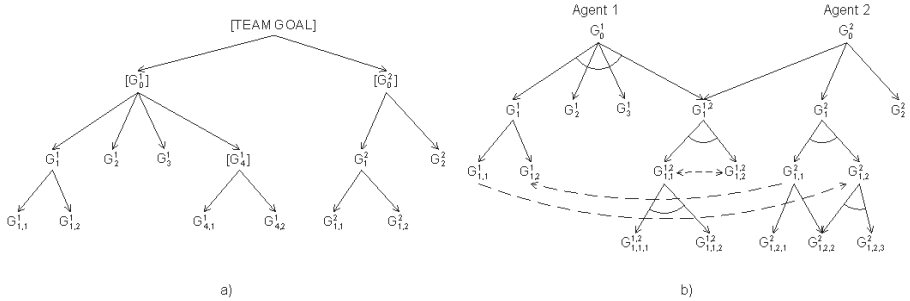
Teoretický model HTN je pochopiteľne expresívnejší ako základné plánovanie typu STRIPS. Teoretický model HTN však na druhej strane nie je použiteľný v praxi, pretože je nerozhodnuteľný, a to aj pri veľmi silných obmedzeniach. HTN je vo všeobecnosti nerozhodnuteľné, ak sieť úloh môže obsahovať dve zložené úlohy bez určenia ich poradia, a to aj keď nie sú povolené premenné [2].

Keďže je teoretický model HTN nerozhodnuteľný, výpočet môže trvať neobmedzený čas a nemôžeme dopredu ani spoľahlivo predpovedať čas ukončenia. To znamená, že teoretický model nie je (a nemôže byť) prakticky použiteľný. V praxi sa používajú rôzne modifikácie, ktoré obmedzujú priestor plánov na konečný a menia problém na rozhodnuteľný [2].

Najpoužívanejšie obmedzenia pre HTN sú:

1. Obmedzenie dĺžky plánu. Keď je konečná maximálna dĺžka plánu, potom priestor možných plánov je tiež konečný, keďže pri pridávaní úloh do plánu si vyberáme z konečného množstva možností.
2. Obmedzenie úloh na acyklické. Každá úloha môže byť dekomponovaná len do konečnej hĺbky, ktorá je menšia než je celkový počet úloh.
3. Obmedzenie siete úloh, aby boli úplne usporiadané. Úlohy sú plnené sekvenčne jedna po druhej, takže podúlohy rôznych úloh sa nemôžu v čase prekrývať.

Ľubovoľné z týchto obmedzení je postačujúce, aby bolo plánovanie typu HTN rozhodnuteľné, a teda prakticky použiteľné. Všetky doterajšie plánovače založené na HTN používajú aspoň jedno z týchto obmedzení (alebo ich malé modifikácie). To je dôvod, prečo je vhodnejšie používať pojem „plánovanie typu HTN“ pre plánovanie založené na modeli HTN používajúce



Obrázok 1. Príklad dekompozičného stromu pre STEAM (a) a PGP (b).

jedno z týchto obmedzení a nie pre teoretický model HTN bez obmedzení.

Pomenovacia konvencia. *Pojem "plánovanie typu HTN" bude používaný pre plánovanie založené na modeli HTN a používajúce obmedzenie, ktoré spôsobí, že priestor plánov bude konečný a plánovací problém rozhodnuteľný.*

V tomto článku dodržiavame túto pomenovaciu konvenciu.

Veta 1. *Každú doménu typu HTN je možné vyjadriť ako doménu typu STRIPS. Každú doménu typu STRIPS je možné vyjadriť ako doménu typu HTN. Z toho vyplýva, že expresivita plánovania typu STRIPS je rovnaká ako expresivita plánovania typu HTN.*

Dôkaz (náčrt). Priestor plánov domény plánovania typu HTN je konečný. Z toho vyplýva, že stavový priestor tejto domény je konečný. Pre konečný stavový priestor môžeme vytvoriť doménu typu STRIPS jednoduchým vymenovaním všetkých stavových prechodov ako akcií STRIPS. Z toho vyplýva, že expresivita plánovania typu STRIPS nie je menšia ako expresivita plánovania typu HTN. Druhá polovica dôkazu, ukazujúca že expresivita plánovania typu HTN nie je menšia než expresivita plánovania typu STRIPS, je konštruktívna, je možné ju nájsť v [2] a je založená na transformácii domény typu STRIPS na plytkú doménu typu HTN s hĺbkou dekompozície 0. □

Vymenovanie všetkých stavov v doméne nie je veľmi praktické a môže viesť k exponenciálnemu množstvu akcií. V ďalších kapitolách ukážeme, ako transformovať doménu typu HTN na doménu typu STRIPS v polynomiálnom čase, použijúc STRIPS na emuláciu dekompozície v HTN.

3 STRIPS ako Turingov stroj s konečnou páskou

V predchádzajúcej kapitole sme ukázali, že expresivita plánovania typu STRIPS a typu HTN je rovnaká. V tejto kapitole uvádzame jednoduchý spôsob konštrukcie Turingovho stroja s konečnou páskou za použitia STRIPS. Takto ukážeme, že expresivita STRIPS (a teda aj plánovania typu HTN) je rovnaká ako expresivita Turingovho stroja s konečnou páskou.

Turingov stroj je šestica:

$$M = (Q, \Gamma, b, \delta, q_0, F) \tag{1}$$

kde Q je konečná množina stavov, Γ je konečná množina symbolov pásky, $b \in \Gamma$ je prázdny symbol, $\delta = Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ je prechodová funkcia (L a R sú symboly posunutia doľava a doprava), q_0 je počiatočný stav a $F \subseteq Q$ je množina koncových stavov.

V emulácii pomocou STRIPS je množina stavov Q reprezentovaná množinou konštánt C_Q. Γ je reprezentovaná množinou konštánt C_Γ. Aktuálny stav je vyjadrený literálom $state(q)$, pozícia čítacej hlavy literálom $position(p)$ a symbol na páske na pozícii p je vyjadrený literálom $symbol(\gamma, p)$. Prechodová funkcia je definovaná literálmi $transition(q, g, q', g', m)$, kde $q \in C_Q$ a $g \in C_\Gamma$ sú pôvodný stav a symbol na páske, $q' \in C_Q$ a $g' \in C_\Gamma$ sú nový stav a nový symbol na páske a $m \in \{LEFT, RIGHT\}$ je smer pohybu čítacej hlavy. Nasledujúce operátory kódujú Turingov stroj s konečnou páskou:

```
operator: stateTransition
pre: state(q), position(p), symbol(g, p),
transition(q, g, q', g', m),
   translate
del: state(q), symbol(g, p), translate
add: state(q'), symbol(g', p), move(m)

operator: moveHeadLeft
pre: move(LEFT), position(from), leftOf(to, from)
del: move(LEFT), position(from)
add: position(to), translate

operator: moveHeadRight
pre: move(RIGHT), position(from), leftOf(from, to)
del: move(RIGHT), position(from)
add: position(to), translate

operator: finish
```

```
pre: state(q), final(q), translate
del: state(q), translate
add: stop
```

Môžeme vidieť, že stroj pracuje v dvoch krokoch: zmena stavu a pohyb hlavy. Stroj zastaví, ak je dosiahnutý jeden z koncových stavov označených literálom $final(q)$.

Ešte pred začiatkom odvodzovania pomocou STRIPS musíme „vytvoriť“ štruktúru pásky pridaním literálov $leftOf(left, right)$ pre každé dve susediace políčka pásky. Ak by sme použili rozšírenie STRIPS dovoľujúce aritmetické operátory, mohli by sme na pohyb hlavy namiesto toho použiť operátory + a −.

Reprezentácia pásky pre pásku dĺžky n používa $n - 1$ literálov. Na druhej strane obsah pamäte stroja je tiež uložený v podobe literálov, takže n pamäťových miest je reprezentovaných $2n - 1$ literálmi, čo oproti Turingovmu stroju zvyšuje pamäťovú náročnosť iba konštantne.

Emulovaný Turingov stroj s konečnou páskou je deterministický, ak pre každú dvojicu (q, g) existuje najviac jeden literál $transition(q, g, q', p', m)$. V opačnom prípade je stroj nedeterministický.

Mohli by sme stroju ešte pridať vstup, ktorý by bol reprezentovaný rovnako ako páska. Toto by ale samozrejme nezmenilo expresivitu.

Veta 2. *Vyššie zadefinovaná doména pre STRIPS emuluje Turingov stroj s konečnou páskou. Časová zložitosť emulácie je konštantne vyššia ako zložitosť emulovaného stroja.*

Dôkaz (náčrt). Počiatočná množina literálov je $state(q_0)$ spolu so zakódovaním Turingovho stroja (uvedeným vyššie). Podmienka koncového stavu pre plánovač STRIPS je nastavená na *stop*. Ľahko vidno, že ak sa Turingov stroj zastaví, plánovač STRIPS vytvorí plán vedúci z počiatočného do koncového stavu a každé dva nasledujúce kroky tohto plánu (stateTransition, moveHeadLeft/Right) zodpovedajú jednému kroku Turingovho stroja. Ak Turingov stroj s konečnou páskou nezastaví, nebude nájdený žiadny plán. Ak je emulovaný stroj deterministický, potom je v každom stave vykonateľná práve jedna akcia a proces odvodzovania plánu je deterministický. Ak je emulovaný stroj nedeterministický, plánovač STRIPS vyberie v každom stave na vykonanie jednu akciu, rovnako ako musí jednu akciu vybrať aj Turingov stroj. Ak toto rozhodnutie nevedie ku konečnému stavu, plánovač sa vráti a systematicky prehľadáva všetky alternatívy až kým nedosiahne koncový stav alebo mu neostávajú ďalšie alternatívy (čiže Turingov stroj s konečnou páskou nezastaví). □

Turingov stroj s *ne*konečnou páskou je dôležitý teoretický koncept. Na druhej strane, počítače ktoré v praxi používame majú len konečnú pamäť a môžu byť simulované Turingovym strojom s konečnou páskou.

Rovnosť expresivity STRIPS a Turingovho stroja s konečnou páskou má dôležitý dôsledok. Znamená, že STRIPS dokáže vyjadriť všetky problémy riešiteľné počítačom. Vyjadrenie pomocou domény STRIPS môže byť samozrejme v mnohých prípadoch veľmi umelé a nešikovné a výpočtová zložitosť môže byť oveľa väčšia. Napriek tomu by však vyjadrovacia sila STRIPS nemala byť podceňovaná.

4 STRIPS ako emulátor HTN

Predchádzajúca kapitola ukazuje, že je možné vyjadriť Turingov stroj s konečnou páskou ako doménu STRIPS. Spolu s možnosťou vyjadriť doménu typu HTN (s obmedzeniami zaručujúcimi rozhodnuteľnosť) pomocou Turingovho stroja s konečnou páskou to znamená, že STRIPS dokáže vyjadriť ľubovoľnú doménu typu HTN.

V tejto kapitole uvádzame konverziu domény typu HTN na doménu typu STRIPS v polynomiálnom čase. Konverzia je založená na emulácii dekompozície úloh pomocou odvodzovania plánu v STRIPS. Výsledný plán v STRIPS potom predstavuje poradie dekompozície.

Povedzme, že máme sieť úloh $n = (A, \{B, C\})$, ktorá hovorí, že úlohu A môžeme dekomponovať na B a C. Vytvoríme dva operátory A_{start}, A_{stop} a operátory pre B a C. A_{start} pridáva literály $(B_{A_{init}}, C_{A_{init}})$, ktoré povoľujú vykonanie B a C. B môže byť znovu zložená úloha, takže tiež pozostáva z operátorov B_{start}, B_{stop}, ktoré dovoľujú jej ďalšiu dekompozíciu. Ak je B primitívna úloha, pozostáva len z jedného operátora B. Po tom, ako je B spracovaná (operátor B_{stop} alebo B skončil), pridá literál $B_{A_{finish}}$, ktorý je časťou predpodmienky operátora A_{stop}. A_{stop} sa takto vykoná jedine ak sú všetky podúlohy A plne dekomponované alebo primitívne. Počiatočný stav plánovača STRIPS obsahuje len literál S_{init}, ktorý povoľuje začiatok počiatočnej úlohy S (prípadne niekoľko literálov ak je viac počiatočných úloh). Koncový stav obsahuje literál S_{finish}, ktorý je dosiahnutý po úplnej dekompozícii počiatočnej úlohy S a všetkých jej podúloh. Koncový stav môže navyše obsahovať literály pridané v úlohách, ktoré sú označené ako cieľové úlohy.

Ak je povolené prekrývanie úloh, musíme zabrániť situácii, kedy by ukončená podúloha povolila skončenie rodičovskej úlohy z inej siete úloh. Musíme preto vytvoriť rôzne operátory pre podúlohu, ktorá sa nachádza vo viacerých sieťach úloh. Z rovnakého dôvodu musíme vytvoriť rôzne operátory aj pre rodičovskú úlohu, ktorá sa nachádza vo viacerých sieťach úloh. Úloha, ktorá je rodičovskou úlohou v n sieťach úloh

a podúlohou v m sieťach úloh, je teda v konečnom dôsledku reprezentovaná $m * n$ operátormi.

Algoritmus na konverziu HTN na STRIPS pre domény s acyklickým grafom dekompozície je vyjadrený nasledujúcim pseudokódom. Používame skrátenú notáciu operátora $Op = (pre, del, add)$, kde *pre*, *del* a *add* sú predpodmienka a množiny zmazaných a pridaných literálov operátora Op. Prehľad zodpovedajúcich konceptov domény typu HTN a typu STRIPS po konverzii je v tabuľke 1.

```
pre každú úlohu A
   ak A je dekompozícia nejakej úlohy B,
   pre každé B
      ak A je primitívna
         pridaj operátor
            A_B = (A.pre ∪ {A_Binit}, A.del ∪
            {A_Binit}, A.add ∪ {A_Bfinish})
      inak
         pre každú sieť úloh n_i = (A, {C_j | j})
         pridaj operátory
            A_iB_start = (A.pre ∪ {A_Binit},
            {A_Binit}, {C_jA_init})
            A_iB_stop = ({C_jA_finish}, A.del ∪
            {C_jA_finish}, A.add ∪ {A_Bfinish})
   inak
      ak A je primitívna
         pridaj operátor
            A = (A.pre ∪ {A_init}, A.del ∪
            {A_init}, A.add ∪ {A_finish})
      inak
         pre každú sieť úloh n_i = (A, {C_j | j})
         pridaj operátory
            A_i_start = (A.pre ∪ {A_init}, {A_init},
            {C_jA_init})
            A_i_stop = ({C_jA_finish}, A.del ∪
            {C_jA_finish}, A.add ∪ {A_finish})
pre každú počiatočnú úlohu S
   pridaj do počiatočného stavu literál S_init
   pridaj do koncového stavu literál S_finish
```

V prípade potreby je možné predávať parametre (premenné alebo konštanty) z rodičovskej úlohy A na jej podúlohu C pridaním parametrov do literálu $C_{A_{init}}$. Jednoduchým pridaním literálov do operátorov môžeme vyjadriť aj ďalšie obmedzenia ako poradie úloh alebo vzájomné vylučovanie. Ak by sme napríklad chceli aby bola úloha B vykonaná až po úlohe A, jednoducho pridáme literál do množiny *add* operátora pre A a ten istý literál aj do množiny *pre* operátora pre B. To zabráni vykonaniu B pred skončením A.

Veta 3. *Vyššie uvedený algoritmus konvertuje acyklickú doménu typu HTN na ekvivalentnú doménu typu STRIPS. Časová zložitosť tejto domény typu STRIPS je konštantne vyššia než zložitosť pôvodnej domény typu HTN.*

Dôkaz (náčrt). Odvodenie plánu vo výslednej doméne typu STRIPS kopíruje dekompozíciu pôvodnej domény typu HTN. Pre každú dekompozíciu nejakej úlohy A (tzn. výber vhodnej siete úloh a pridanie jej podúloh $\{B_j\}$ do plánu) je v odvodení STRIPS práve jedna akcia A_{start}, jedna akcia $B_{j\,A}$ pre každú primitívnu úlohu z $\{B_j\}$, jeden literál $B_{j\,A_{start}}$ pre každú zloženú úlohu z $\{B_j\}$ a jedna akcia A_{stop}. Akcie reprezentujúce podúlohy A nie sú nikdy vykonané pred A_{start} alebo po A_{stop}. Plánovací algoritmus STRIPS si vyberá akcie z možností len vtedy, keď by aj plánovač typu HTN vyberal úlohu na dekomponovanie a sieť úloh, ktorá sa má na túto dekompozíciu použiť. Znamená to teda, že máme presne jednu akciu STRIPS pre každý krok plánovača typu HTN a presne jedno rozhodnutie plánovača STRIPS pre každé rozhodnutie plánovača typu HTN. □

Ak chceme použiť cyklickú dekompozíciu úloh (teda úloha môže byť po niekoľkých krokoch dekomponovaná sama na seba), musíme obmedziť maximálnu hĺbku dekompozície alebo musí byť doména úplne usporiadaná (kapitola 2 Expresivita), aby sme dostali rozhodnuteľnú doménu.

Pre plne usporiadanú doménu jednoducho operátorom pridáme obmedzenia na poradie vykonávania.

Ak chceme obmedziť maximálnu hĺbku dekompozície a mať pri tom cyklickú a nie plne usporiadanú doménu, musíme značkovať akcie, aby sme v cyklickej dekompozícii rozoznali dekompozície tej istej úlohy pomocou tej istej siete úloh. Toto môžeme dosiahnuť vytvorením postupnosti značkovacích symbolov (podobne ako pri konečnej páske v predchádzajúcej kapitole), ktoré potom použijeme na označenie akcií reprezentujúcich jednu dekompozíciu. Po každej akcii A_{start} jednoducho pridáme inkrementujúcu akciu (podobne ako moveHeadRight v predchádzajúcej kapitole). Hodnota „počítadla“ bude parametrom A_{start} a bude prenášaná medzi operátormi reprezentujúcimi tú istú sieť úloh pomocou literálov $B_{A_{init}}$ a $B_{A_{finish}}$. Táto konečná množina značkovacích symbolov je ekvivalentná značkovacím symbolom používaným plánovačmi typu HTN.

Mnohé plánovače (či už založené na HTN alebo STRIPS) umožňujú rôzne rozšírenia, ako je správa zdrojov, premenné a aritmetické operátory, paralelné vykonávanie alebo plánovanie s neurčitosťou. Podľa vety 2 môžu byť všetky rozšírenia plánovania typu HTN (za predpokladu že ostane rozhodnuteľné) transformované do STRIPS, hoci aj za cenu zvýšenia výpočtovej zložitosti. Na druhej strane je však väčšina rozšírení spoločná pre oba prístupy, takže je možné pozmeniť konverzný algoritmus z tejto kapitoly tak, aby mal výsledný plánovací proces rovnakú zložitosť.

HTN	STRIPS
primitívna úloha A	operátor A
zložená úloha A	operátory $A_{i_{start}}$, $A_{i_{stop}}$
sieť úloh $n_i = (A, \{C_j \mid j\})$	operátory $A_{i_{start}}$, $A_{i_{stop}}$; $C_{jA_{i_{start}}}$, $C_{jA_{i_{stop}}}$ alebo C_{jA_i}
pridanie úlohy A ako podúlohy pre úlohu B	pridanie literálu $A_{B_{init}}$ do aktuálneho stavu
výber jednej z možných dekompozícií A	výber jedného z aplikovateľných operátorov $A_{iB_{start}}$ s literálom $A_{B_{init}}$ v predpodmienke
dekomponovanie A s použitím siete úloh $n_i = (A, \{C_j \mid j\})$	vykonanie postupnosti operátorov $A_{iB_{start}}$; $C_{jA_{start}}$, $C_{jA_{stop}}$ alebo C_{jA}; $A_{iB_{stop}}$

Tabuľka 1. Zodpovedajúce koncepty HTN a emulácie HTN pomocou STRIPS.

5 Zhodnotenie

Koncept HTN nie je o nič viac (a o nič menej), než umožnenie používateľovi poskytnúť plánovaču dodatočné heuristické informácie o tom, ako má vytvárať plán. Nezväčšuje však priestor riešiteľných domén.

Napriek tomu je HTN veľmi užitočný a používateľsky príjemný koncept, čo potvrdzuje aj veľké množstvo jeho praktických použití.

Tento článok ukazuje, že plánovanie typu STRIPS môže byť použité pre tie isté domény ako plánovanie typu HTN (s obmedzeniami zaručujúcimi rozhodnuteľnosť), teda že expresivita oboch prístupov je rovnaká. Navyše je možné v polynomiálnom čase konvertovať domény typu HTN na domény typu STRIPS a späť, takže je možné teoretické výsledky pre plánovanie typu STRIPS využiť aj pre plánovanie typu HTN a naopak.

Tento článok tiež ukazuje, že expresivita STRIPS je rovnaká ako expresivita Turingovho stroja s konečnou páskou, čo znamená že všetky problémy riešiteľné (bežným) počítačom sú tiež riešiteľné pomocou STRIPS. Toto je skôr teoretický výsledok než prakticky použiteľná konverzia. Určuje však hornú aj dolnú hranicu expresivity plánovania typu STRIPS a plánovania typu HTN. Navyše je možné využiť formalizmus Turingových strojov pre výsledky zložitosti domén typu STRIPS a typu HTN.

Referencie

1. Bylander T., The Computational Complexity of Propositional STRIPS Planning. Artificial Intelligence 69, Elsevier Science Publishers Ltd., Essex, UK, 1994, 161–204
2. Erol K., Nau D., and Hendler J., HTN planning: Complexity and Expressivity. Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94) AAAI Press, Menlo Park, USA, 1994
3. Fikes R.E., Nilsson N.J., STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence 2, Elsevier Science Publishers Ltd., Essex, UK, 1971, 189–208
4. Kambhampati S., Mali A., and Srivastava B., Hybrid Planning in Partially Hierarchical Domains. Proceedings of the National Conference on Artificial Intelligence (AAAI), 1998
5. Lesser V.R., et al., Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. Autonomous Agents and Multi-Agent Systems. Kluwer Academic Publishers, 9, 1, 2004 87–143
6. Nau D.S., Au T.-C., Ilghami O., Kuter U., Murdock W., Wu D., and Yaman F., SHOP2: An HTN Planning System. Journal of Artificial Intelligence Research 20, 2003, 379–404
7. Sacerdoti E.D., A Structure for Plans and Behavior. Elsevier-North Holland, 1977
8. Tambe M., Towards Flexible Teamwork. Journal of Artificial Intelligence Research. 7, 1997, 83–124
9. Zalaket J., and Camilleri G., FHP: Functional Heuristic Planning. 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES 2004), Wellington, New Zealand, Springer-Verlag, 2004

O neopraviteľnosti algoritmu $\mathbf{DBI_{basic}}^{\star}$

Richard Ostertág and Peter Košinár

Katedra informatiky, Fakulta matematiky, fyziky a informatiky, Univerzity Komenského
Mlynská dolina, 842 48 Bratislava
{ostertag,kosinar}@dcs.fmph.uniba.sk
http://www.dcs.fmph.uniba.sk

Abstrakt *V článku analyzujeme algoritmus DBI_{basic} pre overovanie RSA podpisov v dávke [1]. Tento algoritmus si kladie za cieľ nielen overiť korektnosť všetkých podpisov, ale v prípade, že je v dávke najviac jeden zlý podpis, identifikovať aj jeho pozíciu. Ako bolo ukázané v [2,3], pôvodný algoritmus nie je korektný. Ukážeme, že ani riešenie naznačené v [2,3] nie je korektné. Nakoniec, napriek našej pôvodnej snahe algoritmus opraviť, ukážeme, že to nie je možné dosiahnuť úpravou exponentov, ani pri povolení väčšieho počtu iterácií nezávislom na dĺžke vstupnej dávky.*

1 Úvod a základné pojmy

V dnešnej dobe sa čoraz viac stretávame s digitálnymi podpismi. Používajú sa pri elektronickom nakupovaní, platení, a iných finančných transakciách, pri komunikácii so štátnou správou, pri hlasovaní, pri archivovaní dokumentov, a tak ďalej. Zatiaľ čo digitálny podpis sa vytvorí väčšinou iba raz pri podpise správy majiteľom privátneho kľúča, overovanie podpisov prebieha v mnohých prípadoch opakovane (kýmkoľvek, kto pozná verejný kľúč). Dokonca takéto overovanie môže prebiehať v dávkach, keď sa overuje veľké množstvo podpisov vytvorených jedným subjektom (napríklad pri pravidelnom overovaní platnosti záznamov v digitálnom archíve). Tento problém formálne popíšeme v nasledujúcej časti.

1.1 Overovanie podpisov v dávke

Vstupom je postupnosť (dávka) n usporiadaných dvojíc $(m_1, s_1), \ldots, (m_n, s_n)$ predstavujúcich správu m a jej digitálny podpis s. Všetky podpisy sú vytvorené jediným subjektom. Úlohou je overiť, či je každý podpis korektný. V prípade, že algoritmus zistí prítomnosť nekorektných podpisov, môžeme ešte chcieť určiť aj ich pozíciu v dávke. Takto zadaná úloha má samozrejme zmysel iba vtedy, pokiaľ jej riešenie je efektívnejšie ako postupné overovanie všetkých podpisov.

1.2 RSA

V tomto článku budeme používať digitálne podpisy na báze RSA. Konkrétna inštancia RSA je daná:

1. Verejným modulom N v tvare $N = pq$, kde p a q sú veľké prvočísla.
2. Verejným exponentom e, kde e je nesúdeliteľné s $\phi(N) = (p-1)(q-1)$.
3. Súkromným exponentom d, ktorý je zvolený tak, aby platilo $de \equiv 1 (\mathrm{mod}\ \phi(N))$.

Digitálny podpis s správy m sa vypočíta pomocou súkromného exponentu a verejného modulu ako $s = m^d \bmod N$. Potom digitálny podpis s je korektným podpisom správy m práve vtedy, keď platí:

$$m \equiv s^e (\mathrm{mod}\ N) \ . \tag{1}$$

Podpis je teda možné overiť použitím verejného exponentu a modulu. Pri praktickom nasadení digitálnych podpisov na báze RSA sa pri podpisovaní neumocňuje celá správa m ale iba jej odtlačok (hash) $H(m)$. Teda počíta sa $s = H(m)^d \bmod N$ a overuje sa, či $H(m) \equiv s^e (\mathrm{mod}\ N)$. Prípadne[1] sa ešte používa zarovnanie (padding) ako ochrana pred „jednoduchými" správami, ktoré sa ľahko dešifrujú (napríklad $m=0$). Bez ujmy na všeobecnosti a zjednodušenie zápisov budeme v ďalšom namiesto $H(m)$ písať iba m. Môžeme sa na to pozerať tak, že vlastne nepodpisujeme správy ale iba ich odtlačky (čo sa aj v skutočnosti robí).

1.3 Generický test

Ľahko sa dá nahliadnuť, že pokiaľ by overovaná dávka $(m_1, s_1), (m_2, s_2), \ldots, (m_n, s_n)$ obsahovala iba korektné podpisy, tak platí nasledovný vzťah [4]:

$$\prod_{i=1}^{n} m_i \equiv \left(\prod_{i=1}^{n} s_i \right)^e \bmod N \ . \tag{2}$$

Toto tvrdenie však vo všeobecnosti opačným smerom neplatí. Napríklad stačí, ak v ľubovoľnej korektnej dávke prehodíme medzi dvomi rôznymi správami ich podpisy a (2) bude platiť, ale dávka bude obsahovať práve dva nekorektné podpisy.

Ďalším cieľom bolo znížiť pravdepodobnosť omylu, ak sa aplikuje predchádzajúca úvaha opačným smerom. Na tento účel bol v [4] zavedený generický test.

* Táto práca vznikla s prispením grantu Univer. Komenského č. UK/409/2007 a VEGA grantu č. 1/3106/06.

[1] Pozri napríklad RSAES-OAEP.

Definícia 1. *Generický test je pravdepodobnostný polynomiálny algoritmus, ktorého vstupom je dávka* $x = (m_1, s_1), (m_2, s_2), \ldots, (m_n, s_n)$ *a „bezpečnostný" parameter* l*. Ak vstupná dávka* x *neobsahuje nekorektné podpisy, potom výstup algoritmu musí byť* ***true****. Ak* x *obsahuje aspoň jeden nekorektný podpis, potom výstup algoritmu je správne* ***false****, alebo chybne* ***true*** *s pravdepodobnosťou chyby najviac* 2^{-l}*.*

Definícia generického testu je schématicky znázornená v algoritme 1.

Algoritmus 1 Generický test

Vstup: $x = (m_1, s_1), (m_2, s_2), \ldots, (m_n, s_n)$
Vstup: $l \in N$
Výstup: **true** ak sú všetky podpisy korektné.
Výstup: **false** ak je v x aspoň jeden nekorektný podpis (pravdepodobnosť omylu je v tomto prípade 2^{-l}).

function GenericTest($x; l$)

Pri reálnom nasadení sa generický test implementuje rôznymi algoritmami [5,6,7]. Jednou z možností je napríklad Small Exponent Test (pozri algoritmus 2). Tento test umocňuje m_i a s_i z výrazu (2) na náhodne zvolené malé l bitové exponenty x_i.

Algoritmus 2 Generický test – malé exponenty

1: **function** SmallExponentTest($x; l$)
2: náhodne vyber $x_1, x_2, \ldots, x_n \in \{0,1\}^l$
3: vypočítaj:

$$M \leftarrow \prod_{i=1}^{n} m_i^{x_i} \pmod N, \quad S \leftarrow \prod_{i=1}^{n} s_i^{x_i} \pmod N$$

4: **if** $M \not\equiv S^e \pmod N$ **then return false**
5: **return true**
6: **end function**

Pokiaľ je verejný exponent e malý (v binárnom zápise má malý počet jednotiek, napríklad 3, 5, 35 alebo $2^{16} + 1 = 65537$), tak je výhodnejšie testovať každý podpis samostatne [2]. Small Exponent Test a generické testy vo všeobecnosti prinášajú vyšší výkon oproti testovaniu každého podpisu samostatne, keď je verejný exponent e veľký.

Ďalšou možnosťou ako implementovať spomínaný generický test je Random Subset Test (pozri algoritmus 3). V tomto prípade sa generický test realizuje pomocou l krát opakovaného testovania výrazu (2) na zakaždým náhodne zvolenej podmnožine vstupnej dávky.

Algoritmus 3 Generický test – náhodné podmnožiny

1: **function** RandomSubsetTest($x; l$)
2: **repeat** l **times**
3: náhodne vyber $I \subseteq \{1, 2, \ldots, n\}$
4: vypočítaj:

$$M \leftarrow \prod_{i \in I} m_i \pmod N, \quad S \leftarrow \prod_{i \in I} s_i \pmod N$$

5: **if** $M \not\equiv S^e \pmod N$ **then return false**
6: **end repeat**
7: **return true**
8: **end function**

2 Algoritmus DBI

Autori v článku [1] navrhli nový spôsob identifikácie nekorektných podpisov v dávke. Najprv prezentujú metódu $\mathrm{DBI}_{\mathrm{basic}}$, ktorá dokáže určiť pozíciu jediného nekorektného podpisu v dávke. Vstupom tohto algoritmu je dávka x a už spomínaný „bezpečnostný" parameter l. Výstupom je **true**, keď sú všetky podpisy v dávke korektné. Algoritmus vráti **false**, ak v dávke je viac ako jeden nekorektný podpis. Pokiaľ je v dávke iba jeden nekorektný podpis (m_k, s_k) tak je výstupom jeho index k (pozri algoritmus 4).

Algoritmus 4 Division Based Identifier – Basic

Vstup: $x = (m_1, s_1), (m_2, s_2), \ldots, (m_n, s_n)$
Vstup: $l \in N$
Výstup: **true** ak sú všetky podpisy korektné.
Výstup: Index k jediného nekorektného podpisu.
Výstup: **false** ak je v x viac ako jeden nekorektný podpis.

1: **function** DBI($x; l$)
2: **if** GenericTest($x; l$) **then return true**
3: $M \leftarrow \prod_{i=1}^{n} m_i$, $S \leftarrow \prod_{i=1}^{n} s_i \pmod N$
4: $M' \leftarrow \prod_{i=1}^{n} m_i^i$, $S' \leftarrow \prod_{i=1}^{n} s_i^i \pmod N$
5: hľadaj od 1 po n vrátane prvé také k, že:

$$\left(\frac{S^e}{M}\right)^k \equiv \frac{S'^e}{M'} \pmod N \tag{3}$$

6: **if** k neexistuje **then return false**
7: **if** GenericTest($x \setminus (m_k, s_k); l$) **then return** k
8: **return false**
9: **end function**

Algoritmus 4 sme oproti [1] bez ujmy na všeobecnosti spresnili v kroku 5 tak, aby hľadal k presne uvedeným spôsobom a to postupne v cykle od 1 po n. Táto úprava nám neskôr sprehľadní dôkaz vety 3.

Okrem toho autori v článku [1] rozširujú algoritmus $\mathrm{DBI}_{\mathrm{basic}}$ na DBI_{α}, ktorý identifikuje všetky nekorektné podpisy v dávkach bez obmedzenia na najviac

jeden nekorektný podpis v dávke. V našom článku sa vzhľadom na očakávané negatívne výsledky sústredíme iba na analýzu algoritmu $\mathrm{DBI}_{\mathrm{basic}}$, ktorý je základom článku [1] a z ktorého vychádza aj algoritmus DBI_{α}. Preto DBI_{α} ani neuvádzame a v ďalšom pre zjednodušenie označenia budeme v súlade s algoritmom 4 namiesto $\mathrm{DBI}_{\mathrm{basic}}$ písať už iba DBI.

2.1 Pôvodný dôkaz korektnosti DBI

V článku [1] sa dokazuje korektnosť algoritmu DBI nasledovnou vetou 1. (Upozoňujeme, že veta neplatí, ako ukážeme hneď po uvedení pôvodného nekorektného dôkazu.)

Veta 1. *Nech $x = (m_1, s_1), \ldots, (m_n, s_n)$ je dávka s práve jedným nekorektným podpisom. Ak celé číslo $1 \leq k \leq n$ spĺňa rovnosť (3), tak potom (m_k, s_k) je nekorektný podpis v dávke.*

Dôkaz. Nech j je index nekorektného podpisu v dávke x. Keďže všetky podpisy v dávke okrem (m_j, s_j) spĺňajú rovnosť (1), môžeme zjednodušiť zlomky:

$$\frac{\left(\prod_{i=1}^{n} s_i\right)^e}{\prod_{i=1}^{n} m_i} \text{ a } \frac{\left(\prod_{i=1}^{n} s_i^i\right)^e}{\prod_{i=1}^{n} m_i^i}$$

nasledovným spôsobom:

$$\begin{aligned}\frac{\left(\prod_{i=1}^{n} s_i\right)^e}{\prod_{i=1}^{n} m_i} &\equiv \frac{s_1^e}{m_1}\frac{s_2^e}{m_2}\cdots\frac{s_j^e}{m_j}\cdots\frac{s_n^e}{m_n}\\ &\equiv \frac{s_j^e}{m_j} \pmod N,\\ \frac{\left(\prod_{i=1}^{n} s_i^i\right)^e}{\prod_{i=1}^{n} m_i^i} &\equiv \frac{s_1^e}{m_1}\frac{s_2^{2e}}{m_2^2}\cdots\frac{s_j^{je}}{m_j^j}\cdots\frac{s_n^{ne}}{m_n^n}\\ &\equiv \frac{s_j^{je}}{m_j^j} \pmod N.\end{aligned}$$

Substituovaním $\frac{s_j^e}{m_j}$ a $\frac{s_j^{je}}{m_j^j}$ za $\frac{\left(\prod_{i=1}^{n} s_i\right)^e}{\prod_{i=1}^{n} m_i}$ a $\frac{\left(\prod_{i=1}^{n} s_i^i\right)^e}{\prod_{i=1}^{n} m_i^i}$ v rovnosti (3) dostávame rovnosť:

$$\left(\frac{s_j^e}{m_j}\right)^k \equiv \left(\frac{s_j^e}{m_j}\right)^j \pmod N. \tag{4}$$

Z toho podľa [1] vyplýva, že k je rovné j, čo je index nekorektného podpisu. □

2.2 Analýza DBI podľa [2]

Pri dôkaze korektnosti algoritmu DBI autori v [1] považujú za triviálne to, že vlastnosť (4) pre $k \neq j$ neplatí. Toto tvrdenie však nie je pravdivé. V [2,3] je skonštruovaná dávka s práve jedným zlým podpisom, ktorý algoritmus DBI nie je schopný určiť. Využíva sa pritom práve neplatnosť uvedeného tvrdenia. Konštrukcia je nasledovná:

Definícia 2. *Nech x je dávka. Potom x označíme pojmom korektná dávka práve vtedy, ak sú v x všetky podpisy korektné.*

Definícia 3. *Nech $x = (m_1, s_1), \ldots, (m_n, s_n)$ je dávka. Potom symbolom $x|^j$ označme dávku, ktorá vznikne z dávky x nahradením dvojice (m_j, s_j) novou dvojicou $(m_j, -s_j \bmod N)$.*

Veta 2. *Nech $x = (m_1, s_1), \ldots, (m_n, s_n)$ je korektná dávka. Potom dávka $x' = x|^j$ obsahuje práve jeden nekorektný podpis.*

Dôkaz. Keďže verejný exponent RSA je nepárne číslo, platí nasledovné:

$$(-s_j)^e \equiv -(s_j^e) \equiv -m_j \not\equiv m_j \pmod N.$$

Preto v dávke x' je práve jeden nekorektný podpis. □

Veta 3. *Nech $x = (m_1, s_1), \ldots, (m_n, s_n)$ je korektná dávka s $n \geq 4$ podpismi. Zvoľme si ľubovoľné párne $4 \leq j \leq n$. Potom DBI nedokáže v $x' = x|^j$ správne určiť pozíciu nekorektného podpisu.*

Dôkaz. Na základe vety 2, dávka x' obsahuje práve jeden nekorektný podpis (a ten je na párnom mieste j). Napriek tomu pri výpočte DBI na dávke x' je rovnosť (3) splnená pre každé párne $1 \leq k \leq n$:

$$\begin{aligned}\left(\frac{S^e}{M}\right)^k &\equiv \left(\frac{(-s_j)^e \prod_{i\in\{1,\ldots,n\}-\{j\}} s_i^e}{m_j \prod_{i\in\{1,\ldots,n\}-\{j\}} m_i}\right)^k\\ &\equiv \left(\frac{(-s_j)^e}{m_j}\right)^k \equiv \left(\frac{s_j^e}{m_j}\right)^k \equiv 1 \pmod N\\ \frac{S'^e}{M'} &\equiv \frac{((-s_j)^j)^e \prod_{i\in\{1,\ldots,n\}-\{j\}} (s_i^i)^e}{m_j^j \prod_{i\in\{1,\ldots,n\}-\{j\}} m_i^i}\\ &\equiv \frac{((-s_j)^j)^e}{m_j^j} \equiv \frac{(s_j^j)^e}{m_j^j} \equiv 1 \pmod N\end{aligned}$$

Pri zjednodušovaní ľavej a pravej strany bol využitý fakt, že k a j sú párne. Keďže ľavá aj pravá strana sa rovnajú, tak DBI identifikuje ako index nekorektného podpisu prvú párnu hodnotu k, ktorú bude testovať. Na základe nášho spresnenia (bez ujmy na všeobecnosti) algoritmu 4 to bude $k = 2$. Avšak pozícia jediného nekorektného podpisu je $j \geq 4$, teda $k \neq j$. Preto generický test na $x' \setminus (m_k, s_k)$ vráti **false** akoby dávka x' obsahovala viac ako jeden nekorektný podpis.

Algoritmus DBI teda nie je schopný určiť pozíciu nekorektného podpisu a navyše sa domnieva, že v dávke je viac ako jeden nekorektný podpis. □

Vo vete 3 sme na rozdiel od jej pôvodného znenia v [2], použili naše definície 2 a 3, ktoré budeme

využívať aj neskôr. Zároveň sme vetu mierne upravili tak, aby na rozdiel od pôvodného znenia nevznikal triviálny problém s prípadom, keď sa j zvolí rovné 2. Vtedy by algoritmus 4 náhodou správne určil pozíciu nekorektného podpisu v rozpore s tvrdením pôvodnej vety.

2.3 Modifikácia DBI podľa [2]

V článku [2] je navrhnutý aj nenáročný spôsob ako zamedziť problému s párnymi indexmi. Navrhované riešenie je používať iba nepárne exponenty pri výpočte hodnôt S' a M'. Teda riadok 4 v algoritme 4 sa zmení na:

$$M'' \leftarrow \prod_{i=1}^{n} m_i^{2i-1}, \;\; S'' \leftarrow \prod_{i=1}^{n} s_i^{2i-1} \pmod N$$

a zároveň vlastnosť (3) sa zmení na:

$$\left(\frac{S}{M}^{e}\right)^{2k-1} \equiv \frac{S''^{e}}{M''} \pmod N. \tag{5}$$

Autor v článku [2] ďalej píše, že korektnosť a bezpečnosť takto upraveného algoritmu DBI treba ešte podrobne analyzovať.

3 Naše pozorovania

Zatiaľ sme pomlčali o základnom probléme v algoritme DBI a doteraz vedených dôkazoch. Problém spočíva v tom, že množina $\{0, 1, \ldots, n-1\}$ s operáciami sčítania a násobenia modulo $n = pq$ netvorí pole. Problém spôsobuje operácia násobenia, ktorá nemá inverzný prvok pre každý násobok p alebo q. Preto sa nedá uvažovať o delení nad celou touto množinou. Konkrétne by sa nedali vypočítať zlomky v tvare $\frac{x}{y}$, kde x nie je násobok y a y je násobok p alebo q (teda aj 0).

Týmto problémom sa však v článku nebudeme zaoberať, lebo ukážeme, že aj keď zakážeme správy, ktoré sú násobkom p alebo q, tak aj tak algoritmus 4 nie je korektný. Preto v ďalšom texte predpokladáme, že všetky dávky $(m_1, s_1), \ldots, (m_n, s_n)$, ktoré budeme používať, majú všetky m_i nesúdeliteľné s p a q.

3.1 Analýza modifikácie DBI

Ukážeme, že ani úprava algoritmu 4 podľa článku [2] nezaručuje jeho korektnosť.

Veta 4. *Nech $x = (m_1, s_1), \ldots, (m_n, s_n)$ je korektná dávka s $n \geq 2$ podpismi. Zvoľme si ľubovoľné $2 \leq j \leq n$. Potom DBI upravené podľa [2] nedokáže v $x' = x|^j$ správne určiť pozíciu nekorektného podpisu.*

Dôkaz. Na základe vety 2, dávka x' obsahuje práve jeden nekorektný podpis. Napriek tomu pri výpočte upraveného DBI na dávke x' je rovnosť (5) splnená pre každé $1 \leq k \leq n$:

$$\begin{aligned}
\left(\frac{S}{M}^{e}\right)^{2k-1} &\equiv \left(\frac{(-s_j)^e \prod_{i\in\{1,\ldots,n\}-\{j\}} s_i^e}{m_j \prod_{i\in\{1,\ldots,n\}-\{j\}} m_i}\right)^{2k-1} \\
&\equiv \left(\frac{(-s_j)^e}{m_j}\right)^{2k-1} \\
&\equiv \left(\frac{-s_j^e}{m_j}\right)^{2k-1} \equiv -1 \pmod N \\
\frac{S''^{e}}{M''} &\equiv \frac{((-s_j)^{2j-1})^e \prod_{i\in\{1,\ldots,n\}-\{j\}} (s_i^{2i-1})^e}{m_j^{2j-1} \prod_{i\in\{1,\ldots,n\}-\{j\}} m_i^{2i-1}} \\
&\equiv \frac{((-s_j)^{2j-1})^e}{m_j^{2j-1}} \\
&\equiv \frac{-(s_j^{2j-1})^e}{m_j^{2j-1}} \equiv -1 \pmod N
\end{aligned}$$

Pri zjednodušovaní ľavej a pravej strany bol využitý fakt, že $2k-1$ a $2j-1$ sú nepárne exponenty. Keďže ľavá aj pravá strana sa rovnajú, tak DBI identifikuje ako index nekorektného podpisu prvú hodnotu k, ktorú bude testovať. Na základe nášho spresnenia (bez ujmy na všeobecnosti) algoritmu 4 to bude $k = 1$. Avšak pozícia jediného nekorektného podpisu je $j \geq 2$, teda $k \neq j$. Preto generický test na $x' \setminus (m_k, s_k)$ vráti **false** akoby dávka x' obsahovala viac ako jeden nekorektný podpis.

Algoritmus DBI upravený podľa návrhu z [2] tiež nie je schopný určiť pozíciu nekorektného podpisu a navyše sa domnieva, že v dávke je viac ako jeden nekorektný podpis. □

3.2 Zovšeobecnenie DBI

Vzhľadom na to, že ani úprava z [2] nezabezpečila DBI jeho korektnosť, rozhodli sme sa pôvodný algoritmus očistiť a zovšeobecniť tak, aby sme ho mohli ľahšie analyzovať.

Hlavným cieľom pôvodného algoritmu je určiť pozíciu jediného nekorektného podpisu v dávke. Preto budeme rovno požadovať, aby dávka na vstupe, mala práve jeden nekorektný podpis. Na základe tohto predpokladu môžeme z algoritmu vypustiť úvodné volanie generického testu za účelom overenia, či je dávka korektná. Taktiež môžeme vynechať záverečné volanie generické testu, ktorého účelom bolo zistiť, či v dávke nie je viac nekorektných podpisov.

Prvým zovšeobecnením, ktoré nám umožní jednoducho a bez opakovaných úprav algoritmu meniť exponenty, ktoré sa budú používať pri umocňovaní jednotlivých m_i a s_i, bude zavedenie pomocnej funkcie GetExp (pozri algoritmus 6).

Druhé zovšeobecnenie posilní algoritmus DBI tým, že ak pomocou jednej postupnosti exponentov nie je možné jednoznačne určiť, kde sa nachádza nekorektný podpis, tak DBI môže opakovane testovať vstupnú dávku s inou postupnosťou exponentov. S tým súvisí aj úprava testu z riadku 5 algoritmu 4. Už nebudeme hľadať prvé také k, ktoré spĺňa rovnosť (3), ale všetky také k. Počet týchto indexov sa potom budeme snažiť redukovať v ďalšej iterácii zovšeobecneného algoritmu s ďalšou postupnosťou exponentov, až pokiaľ nám nezostane jediný index.

Pre prehľadnosť vyjmeme funkcionalitu redukovania počtu indexov z hlavného algoritmu 5 (GDBI) a vytvoríme z nej samostatný algoritmus 7 (Reduce).

Algoritmus 5 Zovšeobecnenie DBI

Vstup: Dávka $x = (m_1, s_1), (m_2, s_2), \ldots, (m_n, s_n)$ s práve jedným nekorektným podpisom.
Výstup: Index jediného nekorektného podpisu v x.

```
1: function GDBI(x)
2:     i ← 1
3:     T ← 1, 2, ..., n
4:     while |T| ≠ 1 do        // |T| je dĺžka postupnosti T
5:         E ← GetExp(i;|T|), i ← i + 1
6:         T ← Reduce(x;T;E)
7:     end while
8:     return T[1]                    // vráť prvý prvok T
9: end function
```

Pri algoritme GDBI si v premennej i počítame poradie aktuálnej iterácie. Túto informáciu odovzdávame funkcii GetExp, aby (ak to potrebuje) mohla pre každú iteráciu generovať inú postupnosť. Algoritmus nemusí skončiť, pokiaľ sa nevhodnou voľbou exponentov stane, že postupnosť indexov T sa prestane redukovať[2]. Postupnosť T sa nemôže úplne vyprázdniť lebo rovnosť z riadku 6 algoritmu 7 bude platiť minimálne pre nekorektný podpis.

V hlavnom algoritme 5 sa volá pomocná funkcia GetExp. Ak by sme použili implementáciu z jej riadku 2, a obmedzili by sme sa iba na jednu iteráciu, dostali by sme pôvodný algoritmus DBI. Ak by sme použili implementáciu z riadku 5, a obmedzili by sme sa iba na jednu iteráciu, dostali by sme algoritmus DBI podľa úpravy z článku [2].

Funkcia Reduce neprechádza všetky prvky dávky x ale iba tie, ktorých index je v postupnosti I. Prvok (m_j, s_j), ktorý je na i-tom mieste v postupnosti I (teda $I[i] = j$) bude umocnený na exponent $E[i]$.

[2] Napríklad ak by funkcia GetExp zakaždým vrátila postupnosť samých núl.

Algoritmus 6 Výpočet exponentov

Vstup: Poradie iterácie i.
Vstup: Počet testovaných podpisov n.
Výstup: Postupnosť dĺžky n exponentov pre testovanie n podpisov v i-tej iterácii.

```
1: function GetExp(i; n)      // verzia podľa článku [1]
2:     return 1, 2, ..., n
3: end function
4: function GetExp(i; n)      // verzia podľa článku [2]
5:     return 1, 3, ..., 2n − 1
6: end function
7: function GetExp(i; n)      // využíva aj argument i
8:     return 1 + i, 2 + i, ..., n + i
9: end function
```

Algoritmus 7 Redukcia postupnosti indexov

Vstup: Dávka $x = (m_1, s_1), (m_2, s_2), \ldots, (m_n, s_n)$ s práve jedným nekorektným podpisom.
Vstup: Postupnosť testovaných indexov $I = i_1, i_2, \ldots, i_c$, obsahujúca aj index nekorektného podpisu.
Vstup: Postupnosť exponentov $E = e_1, e_2, \ldots, e_c$.
Výstup: Zredukovaná postupnosť indexov (podpostupnosť I bez indexov, o ktorých sme zistili, že príslušný podpis je korektný), obsahujúca aj index nekorektného podpisu.

```
1: function Reduce(x; I; E)
2:     M ← ∏_{i=1}^{c} m_{I[i]}, S ← ∏_{i=1}^{c} s_{I[i]} (mod N)
3:     M' ← ∏_{i=1}^{c} m_{I[i]}^{E[i]}, S' ← ∏_{i=1}^{c} s_{I[i]}^{E[i]} (mod N)
4:     T ← ε                          // prázdna postupnosť
5:     for i ← 1, c do
6:         if (S^e / M)^{E[i]} ≡ S'^e / M' (mod N) then
7:             T ← T, I[i]          // pridaj na koniec postup.
8:         end if
9:     end for
10:    return T
11: end function
```

3.3 Situácia je zlá

Najprv ukážeme, že pôvodný algoritmus je neopraviteľný, nech už sa zvolia exponenty pre umocňovanie akokoľvek ale pevne pre každé n.

Veta 5. *Nech funkcia* GetExp *je zvolená ľubovoľne ale pevne. Potom existuje dávka, na ktorej algoritmus 5 (*GDBI*) neskončí po prvom zavolaní* Reduce.

Dôkaz. Nech $x = (m_1, s_1), (m_2, s_2), (m_3, s_3)$ je korektná dávka. Nech $\mathrm{GetExp}(1; 3) = e_1, e_2, e_3$. Na základe dirichletovho princípu môžeme nájsť také $i \neq j \in \{1, 2, 3\}$, že e_i a e_j majú rovnakú paritu. Bez ujmy na všeobecnosti nech $i = 1$ a $j = 2$. Nech $x' = x|^2$. Na základe vety 2 táto dávka obsahuje práve jeden nekorektný podpis.

Ukážeme, že GDBI(x') neskončí po prvom zavolaní REDUCE. Presnejšie, že po prvej iterácii bude T obsahovať index 1 aj 2. Na to potrebujeme ukázať, že REDUCE($x'; 1, 2, 3; e_1, e_2, e_3$) obsahuje 1 aj 2.

Vzhľadom na to, že $E[i] = e_i$ a $I[i] = i$ a konštrukciu x', bude cyklus v algoritme 7 vyzerať nasledovne:

```
1: T ← ε                                  // prázdna postupnosť
2: for i ← 1, 3 do
3:     if (−1)^{e_i} ≡ (−1)^{e_2} (mod N) then
4:         T ← T, i                       // pridaj na koniec postup.
5:     end if
6: end for
```

Je zrejmé, že podmienka v cykle je splnená pre $i = 2$. Keďže e_1 ma rovnakú paritu ako e_2, bude podmienka splnená aj pre $i = 1$. Preto po skončení cyklu bude postupnosť T obsahovať aspoň indexy 1 a 2 a to je aj postupnosť, ktorú funkcia REDUCE vráti. □

Dôsledkom tejto vety je, že pôvodný algoritmus 4 (DBI) nie je opraviteľný len „lepšou“ voľbou exponentov. Vzhľadom na existenciu hypotézy[3], že algoritmus by mohol byť korektný, ak by mal možnosť testovať dávku na viacerých postupnostiach exponentov, rozhodli sme sa analyzovať aj túto alternatívu.

3.4 Situácia je ešte horšia

Ukážeme, že algoritmus nie je možné opraviť ani povolením ľubovoľného ale pevne zvoleného počtu zavolaní REDUCE. Pre tento účel algoritmus 5 prepíšeme do ekvivalentného algoritmu 8, kde namiesto iterácie budeme používať rekurziu. Počet zavolaní REDUCE v algoritme GDBI sa zhoduje s počtom zavolaní REDUCE v algoritme RGDBI.

Algoritmus 8 Rekurzívna verzia GDBI

Vstup: Dávka $x = (m_1, s_1), (m_2, s_2), \ldots, (m_n, s_n)$ s práve jedným nekorektným podpisom.
Vstup: Interný parameter i. Pokiaľ tento parameter chýba, má preddefinovanú hodnotu 1.
Výstup: Index jediného nekorektného podpisu v x.

```
1: function RGDBI(x; i ← 1)
2:     if |x|=1 then return 1
3:     E ← GETEXP(i;|x|)
4:     T ← REDUCE(x;1,2,...,|x|;E)
5:     y ← ε                              // prázdna postupnosť
6:     for j ← 1, |T| do
7:         y ← y, x[T[j]]
8:     end for
9:     return T[RGDBI(y;i+1)]
10: end function
```

[3] Hypotéza bola, že pri povolení dvoch iterácií by mohlo pomôcť GETEXP implementované podľa riadku 8.

Definícia 4. *Nech $n \geq 1$ a nech $1 \leq j \leq n$. Potom symbol $X|_n^j$ definujeme nasledovne:*

$$X|_n^j = \underbrace{(1,1), \ldots, (1,1)}_{n\times}|^j.$$

Dávka $X|_n^j$ obsahuje práve jeden nekorektný podpis, lebo $1 \equiv 1^e \pmod N$. Nasledujúce tvrdenie by sa dalo dokázať aj bez využitia špeciálneho tvaru korektných podpisov v $X|_n^j$. Len dôkaz lemy 2 by bol zložitejší (pri konštruovaní x' z $X|_l^b$).

Definícia 5. *Označme $P(k)$ nasledovné tvrdenie:*
$\exists v_k\ \forall v \geq v_k\ \forall$GETEXP $\forall i \geq 1\ \exists j$*:*
RGDBI($X|_v^j; i$) *spraví viac než k volaní* REDUCE.

Lema 1. *Tvrdenie $P(1)$ je pravdivé.*

Dôkaz. Nech $v_1 = 3$. Potom ukážeme, že pre ľubovoľné ale pevne dané GETEXP a $\forall v \geq v_1\ \forall i \geq 1$ existuje j také, že RGDBI($X|_v^j; i$) spraví aspoň dve volania funkcie REDUCE.

Postupnosť $E \leftarrow$ GETEXP(i;v) môžeme rozdeliť na podpostupnosť párnych a podpostupnosť nepárnych exponentov. Aspoň jedna z nich bude mať aspoň dva prvky. Označme j index akéhokoľvek prvku tejto podpostupnosti v postupnosti E.

Potom $x' = X|_v^j$ je dávka, na ktorej RGDBI($x'; i$) spraví aspoň dve volania REDUCE.

Po prvej redukcii bude mať postupnosť T vďaka voľbe j a konštrukcii x' dĺžku aspoň dva. Preto príde k ďalšiemu zavolaniu funkcie REDUCE. Podrobnosti dôkazu sú analogické s vetou 5. □

Lema 2. *Tvrdenie $P(k) \Rightarrow P(k+1)$ je pravdivé.*

Dôkaz. Nech $v_{k+1} = 2v_k - 1$. Potom ukážeme, že pre ľubovoľné ale pevne dané GETEXP a $\forall v \geq v_{k+1}\ \forall i \geq 1$ existuje j, že RGDBI($X|_v^j; i$) spraví viac než $k+1$ volaní REDUCE.

Postupnosť $E \leftarrow$ GETEXP($i; v$) môžeme rozdeliť na podpostupnosť párnych a podpostupnosť nepárnych exponentov. Aspoň jedna z nich bude mať aspoň v_k prvkov. Označme $l \geq v_k$ dĺžku tejto podpostupnosti a j_a index a-teho prvku tejto podpostupnosti v postupnosti E.

Nech $X|_l^b$ je dávka, pre ktorú na základe platnosti $P(k)$ spraví RGDBI($X|_l^b; i+1$) viac než k volaní REDUCE.

Potom $x' = X|_v^{j_b}$ je dávka na ktorej RGDBI($x'; i$) spraví viac než $k+1$ volaní REDUCE.

Tvrdenie vyplýva z toho, že po prvom zavolaní funkcie REDUCE(x';1,2,...,v;E) v RGDBI(x';i) bude $y = X|_l^b$. Preto nasledujúce volanie T[RGDBI(y;i+1)] spraví viac než k volaní REDUCE. □

Lema 3. $\forall k \geq 1\, \exists v_k\, \forall v \geq v_k\, \forall \textsc{GetExp}\, \forall i \geq 1\, \exists j$: $\textsc{rGDBI}(X|_v^j; i)$ *spraví viac než* k *volaní* REDUCE.

Dôkaz. (Pomocou matematickej indukcie.)
Báza indukcie: Lema 1.
Indukčný predpoklad: Nech platí $P(k)$.
Indukčný krok: Lema 2. □

Veta 6. *Nech funkcia* GETEXP *je zvolená ľubovoľne ale pevne. Ďalej nech* $k \geq 1$ *je ľubovoľne zvolené celé číslo. Potom existuje dávka, na ktorej algoritmus 8 (*RGDBI*) spraví viac než* k *volaní* REDUCE.

Dôkaz. Na základe lemy 3 máme pre každú voľbu k a GETEXP garantovanú existenciu nekonečne veľa dávok x dĺžky aspoň v_k s práve jedným nekorektným podpisom na ktorých $\textsc{rGDBI}(x; 1)$ spraví viac než k volaní REDUCE. □

4 Záver

V článku sme ukázali[4], že algoritmus $\mathrm{DBI}_{\mathrm{basic}}$ prezentovaný v článku [1] nie je možné opraviť ani inou vhodnejšou voľbou použitých exponentov (ako sa pokúšal článok [2]), ani povolením väčšieho počtu iterácií nezávislého od dĺžky vstupnej dávky. Ďalším pokračovaním práce by mohlo byť analyzovanie prípadu, keď bude povolené, aby počet iterácií závisel od dĺžky vstupnej dávky. Otázkou však ostáva, či takýto algoritmus by ešte stále bol opravou pôvodného algoritmu $\mathrm{DBI}_{\mathrm{basic}}$, alebo by to už bol iný algoritmus.

Referencie

1. Lee S., Cho S., Choi J., and Cho Y., Efficient Identification of Bad Signatures in RSA-Type Batch Signature. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E89-A, 2006 74–80
2. Stanek M., Overovanie RSA podpisov v dávke — takto nie. In: Mikulášská kryptobesídka, Trusted Network Solutions, 2006, 33–37 ISBN 978-80-903083-7-6
3. Stanek M., Attacking LCCC Batch Verification of RSA Signatures. Cryptology ePrint Archive, Report 2006/111 (2006) http://eprint.iacr.org/, (to appear in: International Journal of Network Security, 6, 3, 2008, 255–257)
4. Pastuszak J., Michatek D., Pieprzyk J., and Seberry J., Identification of Bad Signatures in Batches. In: PKC: Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptography. Volume 1751 of Lecture Notes in Computer Science., London, UK, Springer-Verlag, 2000, 28–45
5. Bellare M., Garay J.A., and Rabin T., Batch Verification with Applications to Cryptography and Checking. In: LATIN: Proceedings of the Third Latin American Symposium on Theoretical Informatics. Volume 1380 of Lecture Notes in Computer Science., London, UK, Springer-Verlag, 1998, 170–191
6. Bellare M., Garay J.A., and Rabin T., Fast Batch Verification for Modular Exponentiation and Digital Signatures. In Nyberg K., ed.: Advances in Cryptology – EUROCRYPT. Volume 1403 of Lecture Notes in Computer Science., Springer-Verlag, 1998, 236–250
7. Boyd C. and Pavlovski C., Attacking and Repairing Batch Verification Schemes. In Okamoto, T., ed.: Advances in Cryptology – ASIACRYPT. Volume 1976 of Lecture Notes in Computer Science., London, UK, Springer-Verlag, 2000, 58–71

[4] Napriek našej pôvodnej snahe algoritmus opraviť.

Distribuovaná tvorba hierarchie konceptov z textových dokumentov v gridovom prostredí*

Martin Sarnovský, Peter Butka, and Erik Tkáč

Katedra kybernetiky a umelej inteligencie, Fakulta elektrotechniky a informatiky
Technická univerzita v Košiciach, Letná 9, 04200 Košice, Slovensko
{martin.sarnovsky,peter.butka}@tuke.sk, erik.tkac@gmail.com

Abstrakt *Tento príspevok sa zaoberá integrovaním algoritmu vytvárania hierarchie konceptuálnych modelov z textových dokumentov do prostredia Gridu z dôvodu značnej výpočtovej náročnosti daného algoritmu. Algoritmus je integrovaný do prostredia systému GridMiner. Hierarchie konceptuálnych modelov sú vytvárané pomocou hybridnej metódy kombinujúcej jednostranne fuzzifikovanú verziu formálnej konceptuálnej analýzy (FCA) a zhlukovacích algoritmov pre redukciu zložitosti problému. Algoritmus pracuje s množinou textových dokumentov, ktoré sa v prvom kroku rozdelia pomocou zhlukovacieho algoritmu GHSOM (Growing Hierarchical Self-Organizing Maps) na menšie podmnožiny zhlukov. V ďalšom kroku sa výpočet jednotlivých máp rozdistribuuje medzi pracovné uzly. Pracovný uzol predstavuje základnú výpočtovú jednotku, ktorá môže byť reprezentovaná vo forme vlákna v prípade paralelnej verzie algoritmu alebo ako gridový uzol v distribuovanej verzii. Na takto vytvorené distribuované mapy sa aplikuje algoritmus FCA (upravený algoritmus generovania horných susedov), ktorý vytvorí lokálny jednostranne fuzzy konceptový zväz pre každý zhluk na jednotlivých mapách. Jednotlivé konceptové zväzy sú následne spájané do výslednej hierarchie konceptov aglomeratívnym zhlukovacím algoritmom na základe podobnosti ich popisov. Dôraz tohto príspevku je kladený na stránku zrýchlenia pomocou paralelizácie a aplikovaním na Gride. Celý proces bol testovaný na dvoch množinách dokumentov - články denníka Times a štandardná databáza Reuters-21578.*

1 Úvod

Formálna konceptová analýza (FCA, [1]) poskytuje formálny rozbor tabuľky dát a umožňuje identifikáciu zhlukov podobných objektov (reprezentujúcich koncepty) z formálneho konceptuálneho pohľadu. Výstupom tejto metódy sú netriviálne informácie o vstupe v zásade dvoch typov - konceptový zväz a atribútové implikácie. Koncept je vlastne zhluk "podobných" objektov (táto relácia je založená na prítomnosti rovnakých resp. podobných hodnôt atribútov objektov). Koncepty je možné následne hierarchicky zoradiť do podoby zväzu. Štandardné použitie FCA je založené na binárnej objektovo-atribútovej tabuľke – "crisp" prípad.

Pri použití tohto postupu v textoch je problémom nutnosť použiť vektorovú reprezentáciu dokumentov pomocou váh jednotlivých termov, teda objektovo atribútová tabuľka (v tomto prípade tzv. dokumentovo-termová matica) obsahuje reálne hodnoty (v našom prípade to boli hodnoty z intervalu [0,1]). Tu je potrebná fuzzifikácia minimálne na strane atribútov. Jeden prístup ku tzv. jednostrannej fuzzifikácii je možné nájsť v [2]. Konceptový zväz vytvorený na základe takéhoto vstupu je potom jednostranne fuzzy konceptový zväz. Výpočtová zložitosť tvorby formálnych konceptových zväzov s veľkých vstupných kontextov je značná, preto má zmysel hľadať metódy pre efektívnejšie generovanie konceptov, napr. kombináciou s inými algoritmami a použitím heuristík. V spomínanej práci autor navrhol aj možnosť redukcie budovania zväzu použitím aglomeratívneho zhlukovania v procese hľadania zmysluplných konceptov, definuje potrebné zobrazenia a vzdialenostnú funkciu pre tento prípad.

Ďalší prístup k redukcii problému budovania konceptuálnych modelov z dokumentov môže byť založený na dekompozícii problému spracovania množiny dokumentov [3]. Tento prístup buduje hierarchiu lokálne vytvorených konceptových zväzov resp. ich vyextrahovaných popisov. Vstupujúca množina dokumentov je na začiatku dekomponovaná použitím rozdeľujúceho zhlukovacieho algoritmu na menšie množiny navzájom podobných dokumentov. Následne sú pre každý klaster vybudované pomocou FCA lokálne modely, ktoré sú potom kombinované do jednoduchej hierarchie konceptov použitím aglomeratívneho zhlukovania na základe podobnosti ich popisov. Pre experimenty bol v procese dekompozície použitý algoritmus GH-

* Práca prezentovaná v tomto príspevku vznikla za podpory Vedeckej grantovej agentúry Ministerstva školstva SR a Slovenskej akadémie vied v rámci projektu VEGA č.1/4074/07 s názvom "Metódy anotovania, vyhľadávania, tvorby a sprístupňovania znalostí s využitím metadát pre sémantický popis znalostí", za podpory Agentúry na podporu výskumu a vývoja na základe zmlúv č. RPEU-0011-06 (projekt PoZnaŤ) a č. APVV-0391-06 (projekt SEMCO-WS), a za podpory Kultúrnej a edukačnej grantovej agentúry Ministerstva školstva SR v rámci projektu č.3/3124/05 s názvom "Virtuálne laboratórium manažmentu dodávateľsko odberateľských reťazcov".

SOM [4], pre budovanie lokálnych FCA modelov bol použitý algoritmus generovania horných susedov (ako bol definovaný v [5], avšak generujúci koncepty jednostranne fuzzy konceptového zväzu), tieto boli popísané charakteristickými termami a jednoduchý aglomeratívny zhlukovací algoritmus bol použitý pre budovanie hierarchie konceptových zväzov pomocou metriky založenej na týchto charakteristických popisoch. Popísaná metóda bola testovaná na menšej množine textových dokumentov v anglickom jazyku (Times 60 - novinové články, 420 dokumentov z obdobia šesťdesiatych rokov minulého storočia). Pre prácu z dokumentmi bola použitá nami navrhnutá knižnica JBOWL [6] pre spracovanie a reprezentáciu textových dokumentov.

Tento príspevok sa zaoberá integrovaním tohto algoritmu vytvárania hierarchie konceptuálnych modelov z textových dokumentov do prostredia Gridu z dôvodu jeho značnej výpočtovej náročnosti. Algoritmus je integrovaný do prostredia systému GridMiner, pracuje s množinou textových dokumentov, ktoré sa v prvom kroku rozdelia pomocou zhlukovacieho algoritmu GHSOM (Growing Hierarchical Self-Organizing Maps) na menšie podmnožiny zhlukov. V ďalšom kroku sa výpočet jednotlivých máp rozdistribuuje medzi pracovné uzly. Pracovný uzol predstavuje základnú výpočtovú jednotku, ktorá môže byť reprezentovaná vo forme vlákna v prípade paralelnej verzie algoritmu alebo ako gridový uzol v distribuovanej verzii. Na takto vytvorené distribuované mapy sa aplikuje algoritmus FCA (upravený algoritmus generovania horných susedov), ktorý vytvorí lokálny jednostranne fuzzy konceptový zväz pre každý zhluk na jednotlivých mapách. Jednotlivé konceptové zväzy sú následne spájané do výslednej hierarchie konceptov aglomeratívnym zhlukovacím algoritmom na základe podobnosti ich popisov. Dôraz tohto príspevku je kladený na stránku zrýchlenia pomocou paralelizácie a aplikovaním na Gride. Celý proces bol testovaný na dvoch množinách dokumentov - články denníka Times a štandardná databáza Reuters-21578.

V nasledujúcej kapitole sú popísané základné myšlienky a postup algoritmu pôvodnej sekvenčnej verzie (viď. [3]) potrebné pre ďalšie časti príspevku. Následne je popísaná myšlienka paralelizácie a distribúcie výpočtov za účelov dosiahnutia zlepšenia časovej efektívnosti. Nasleduje stručný popis experimentov, krátke zhrnutie a myšlienky pre ďalšiu prácu sa nachádzajú v závere príspevku.

2 Popis sekvenčnej verzie algoritmu

V tejto kapitole je stručne popísaný algoritmus v jeho sekvenčnej podobe, ktorej distribúcia bude popísaná v ďalších kapitolách.

2.1 Jednostranne fuzzy konceptový zväz

Majme neprázdnu množinu atribútov A, neprázdnu množinu objektov B a fuzzy reláciu R, pricom $R : A \times B \rightarrow [0,1]$. Túto reláciu potom reprezentuje tabulka dát, kde $R(a,b)$ vyjadruje hodnotu atribútu a objektu b. V našom prípade objekt je dokument a atribút je term, resp. miera vyjadrujúca výskyt daného termu (slova) v dokumente.

Definujme zobrazenie $\tau : P(B) \rightarrow [0,1]^A$, ktoré priradí každej množine X prvkov z B nejakú funkciu $\tau(X)$, hodnotou ktorej v bode $a \in A$ je:

$$\tau(X)(a) = \min\{R(a,b) : b \in X\}, \tag{1}$$

t.j. táto funkcia priradí každému atribútu aspon takú hodnotu, akú minimálne dosahujú všetky objekty z X.

Definujme dalej zobrazenie $\sigma : [0,1]^A \rightarrow P(B)$, ktoré priradí každej funkcii $f : A \rightarrow [0,1]$ množinu:

$$\sigma(f) = \{b \in B : (\forall\, a \in A)\, R(a,b) \geq f(a)\}, \tag{2}$$

t.j. objekty ktoré "dominujú"danej funkcii f v každom atribúte.

Je lahké ukázat, že tieto zobrazenia majú nasledovné vlastnosti:

$$X_1 \subseteq X_2 \rightarrow \tau(X_1) \geq \tau(X_1) \tag{3}$$
$$f_1 \leq f_2 \rightarrow \sigma(f_1) \supseteq \tau(f_2) \tag{4}$$
$$X \subseteq \sigma(\tau(X)) \tag{5}$$
$$f \leq \tau(\sigma(f)) \tag{6}$$

Potom dvojica zobrazení $< \tau, \sigma >$, ktorá splna tieto podmienky, tvorí tzv. Galoisovu konexiu.

Definujme zobrazenie $cl : P(B) \rightarrow P(B)$ ako kompozíciu zobrazení τ a σ, t.j. nech pre každú množinu X je $cl(X) = \tau(\sigma(X))$. Z predchádzajúcich vlastností spájaných zobrazení je možné lahko vidiet, že cl je uzáverovým operátorom, t.j. splna nasledovné podmienky:

$$X \subseteq cl(X) \tag{7}$$
$$X_1 \subseteq X_2 \rightarrow cl(X_1) \subseteq cl(X_2) \tag{8}$$
$$cl(X) = cl(cl(X)) \tag{9}$$

Hlavnú úlohu pri identifikácii konceptov (rovnako ako v "crisp"prípade) hrajú také podmnožiny objektov $X \subseteq B$, pre ktoré je $X = cl(X)$. Potom pár $< X, \tau(X) >$ sa nazýva *fuzzy koncept*, pricom X je extentom a $\tau(X)$ intentom daného konceptu. Potom množina L všetkých extentov, t.j. $L = \{X \in P(B) : X = cl(X)\}$, usporiadaná inklúziou je zväz, v ktorom sú operácie definované nasledovne: $X_1 \wedge X_2 = X_1 \cap X_2$ a $X_1 \vee X_2 = cl(X_1 \cup X_2)$. Tento zväz potom nazývame *jednostranne fuzzy konceptovým zväzom.*

2.2 Navrhnutý algoritmus redukcie problému

Veľkosť problému v doméne textových dokumentov nás privádza k myšlienke redukcie na sub-problémy. Jednou z možností je zakomponovanie redukcie pomocou rozdeľujúcej zhlukovacej procedúry. Potom riešenie budovania konceptuálneho modelu pozostáva z nasledujúcich krokov:

- Zhlukovacia fáza - predzhlukovanie vstupnej množiny dokumentov, nájdenie zaujímavých podskupín podobných dokumentov
- Fáza budovania lokálnych modelov - každý zhluk je nezávislou trénovacou množinou, pre každý z nich sa vytvorí lokálny konceptuálny model využitím fuzzy FCA prístupu
- Fáza spájania - množina lokálnych modelov sa v tomto kroku spája do jedného veľkého modelu, vzťahujúceho sa k celej testovanej množine

Vo fáze zhlukovania je použitý hierarchický algoritmus na báze samo-organizujúcich sa máp (SOM) - GHSOM (Growing Hierarchical SOM). Tento kombinuje adaptáciu SOM-ov v rámci jednej mapy (Growing Grid resp. GSOM) a hierarchického rozkladu na systém podmáp (Hierarchical SOM). Znamená to, že každá vrstva hierarchickej štruktúry pozostáva z množiny nezávislých máp, ktoré adaptujú svoju veľkosť s ohľadom na požiadavky vstupných dát (príkladov prislúchajúcich danej mape). Viac o týchto algoritmoch a príbuzných prácach je možné nájsť v [4]. Hlavným cieľom je však rozdelenie na relevantné podmnožiny podobných dokumentov.

Na vzniknuté zhluky (podmnožiny vstupnej množiny dokumentov) sa vo fáze budovania lokálnych modelov aplikuje metóda tvorby lokálnych hierarchií s využitím FCA (použitý jednostranne fuzzy konceptový zväz). Každý koncept je charakterizovaný svojim extentom a intentom. Extent predstavuje objekty, v našom prípade názvy dokumentov a intent predstavuje atribúty, v našom prípade váhy termov pre príslušné dokumenty získané vo fáze predspracovania. Teda koncept v našom prípade možno chápať aj ako množinu textových dokumentov, charakterizovanú minimálnymi hodnotami váh termov.

Na vytvorenie zväzu sme použili algoritmus založený na generovaní horných susedov, ako je uvedený v [3] resp. [5]. Tento algoritmus je vzhľadom ku navrhnutej distribúcii nemenný, výstupom sú konceptové zväzy pre každý "listový"zhluk hierarchie GHSOM-u.

V záverečnej fáze sa vzniknuté lokálne hierarchie spájajú do výslednej hierarchie. Z jednotlivých lokálnych hierarchií konceptov sa získajú termy, ktoré daná hierarchia obsahuje a zoradia sa podľa miery, do akej sú jednotlivé termy významné, resp. charakteristické pre danú hierarchiu. Miera dôležitosti termu pre konkrétny konceptový zväz sa určí na základe pozície konceptu, ktorý je charakterizovaný týmto termom, t.j. term patrí do intentu daného konceptu. Čím vyššie sa koncept v hierarchii nachádza, tým viac dokumentov charakterizuje, a teda termy, ktoré tvoria intent daného konceptu budú mať vyššiu dôležitosť. Pre každú lokálnu hierarchiu sa potom vytvorí "uzol", ktorý obsahuje zoznam obsiahnutých dokumentov, zoznam charakteristických termov zoradený zostupne podľa váhy, vektor váh jednotlivých termov, normovaný vektor váh termov. Jednotlivé uzly sa ďalej porovnávajú na základe váh charakteristických termov, preto je nutné získané vektory váh termov normovať, aby sa zanedbali rozdiely v množstve dokumentov pre jednotlivé uzly. Po ich získaní pre všetky lokálne hierarchie sú jednotlivé uzly porovnávané práve na základe získaných vektorov. Následne sú uzly na základe podobnosti zlučované. Na určenie podobnosti dvoch uzlov sa používa nasledovná funkcia podobnosti. Uzly U_1 a U_2 sú reprezentované vektormi normovaných váh termov $U_i = \langle u_{i1}, u_{i2}, ..., u_{in} \rangle$, kde n je počet termov v kolekcii dát. Potom funkcia podobnosti medzi nimi má tvar:

$$sim(U_1, U_2) = \frac{\sum_X \frac{min(U_{1j}, U_{2j})}{max(U_{1j}, U_{2j})}}{|X|}, \qquad (10)$$

kde X je množina všetkých tých termov t_j, pre ktoré platí $max(U1[j], U2[j]), t_j \in X$. Na samotný proces zlučovania na základe uvedenej podobnosti bola použitá metóda spájania na princípe aglomeratívneho zhlukovania. Metóda je založená na hľadaní najvyššej podobnosti medzi dvojicami uzlov. Po nájdení uzlov s najvyššou hodnotou podobnosti sa tieto spoja do nadradeného uzla. Následne sú tieto uzly odstránené zo zonamu uzlov a novovytvorený uzol je pridaný. Celý proces sa opakuje až kým v zozname uzlov nezostane len jeden uzol.

Podobnosť medzi jednotlivými dokumentmi sa určuje na základe ich normovaných váhových vektorov. Spoločný uzol z podobných uzlov sa vytvára zlúčením množín dokumentov a sčítaním vektorov váh jednotlivých termov. Výsledná hierarchia potom pozostáva z uzlov obsahujúcich zoznam dokumentov vo vnútri uzla a zostupne zoradený zoznam charakteristických termov uzla. Každý uzol obsahuje odkaz na nadradený uzol a zoznam odkazov na podradené uzly. Uzly na najspodnejšej úrovni hierarchie navyše obsahujú odkaz na príslušnú lokálnu štruktúru konceptov.

Nakoľko cieľom je najmä popis distribúcie výpočtov na gride (paralelizácia procesu), podrobnejšie informácie k vytváraniu popisov, príklady a detailnejší popis algoritmov a výsledkov sekvenčnej verzie nájdete v [3].

3 Grid

Grid je vo všeobecnosti technológia, ktorá umožňuje z geograficky distribuovaných výpočtových a pamäťových zdrojov vytvoriť univerzálny výpočtový systém s extrémne veľkým výkonom a pamäťou [7]. Systém má črty globálneho celosvetového počítača, kde komponenty sú spojené sieťou Internet. Pre užívateľov sa javí ako obyčajná pracovná stanica; v skutočnosti niektoré časti užívateľskej úlohy sa riešia na rôzne vzdialených miestach počítačového systému. Výhodou Gridu je vysoká efektívnosť využitia združených technických kapacít a tvorivého potenciálu užívateľov.

Gridové počítanie je vhodné použiť na masívne výpočtové problémy využitím zdrojov (výkon CPU, diskový priestor) veľkého počtu pripojených počítačov, ktoré svoje zdroje nevyužívajú na 100%. Grid pozostáva z definícií protokolov, služieb a pokrýva aj otázku bezpečnosti zdieľania zdrojov. Protokoly sú postavené na základe súčasných internetových protokolov, ktoré sú doplnené a rozšírené. Zavádza sa pojem virtuálna organizácia. Ide o organizáciu, ktorá združuje zdroje, ktoré poskytujú skutočné organizácie. Samozrejme tieto organizácie určujú, ktoré zdroje a poskytnú a za akých podmienok ich sprístupnia virtuálnej organizácii.

Základom paralelnej a distribuovanej verzie algoritmu je implementácia knižnice pre spracovanie textových dokumentov JBowl do prostredia pre dolovanie v dátach na Gride GridMiner. Takto navrhnutý systém umožňuje paralelizáciu celkového procesu, nakoľko algoritmy môžu v tomto prostredí bežať paralelne, čím sa zvyšuje efektivita a rýchlosť výpočtu.

V tomto riešení sme sa zamerali na algoritmus možnosť paralelnej tvorby lokálnych modelov. Boli navrhnuté dva typy algoritmu. Paralelný algoritmus je navrhnutý pre viacprocesorové systémy. Distribuovaný algoritmus je určený pre gridové prostredie. Oba algoritmy implementujú aj optimalizáciu rozdelenia GSOM máp medzi pracovné uzly.

4 Implementácia

V oboch navrhnutých riešeniach využívame princíp rozdelenia problému na menšie časti. Základná výpočtová alebo vykonávacia jednotka pre obidva prístupy (paralelizácia, distribúcia) je pracovný uzol. Pracovný uzol predstavuje v rámci paralelizácie jedno pracovné vlákno. Pri distribuovanej verzii algoritmu predstavuje pracovný uzol gridový uzol. Pričom uzol môže byť reprezentovaný vo forme PC, pracovnej stanice, klastra a pod.

Následne je potrebné určiť čo jednotlivé pracovné uzly budú vykonávať. Ako východiskový bod som si určil možnosti pridelenia jednotlivých konceptových zväzov. Z tohto hľadiska môžeme algoritmus rozdeliť na dve základné vetvy - prideľovanie zhlukov samostatne alebo prideľovanie zhlukov v rámci GSOM mapy. Faktor, ktorý do značnej miery ovplyvňuje rýchlosť výpočtu je vstupná GHSOM mapa. V závislosti od jej štruktúry sa mení zložitosť výpočtu konceptových zväzov. Ďalším dôležitým faktorom sú dolované dokumenty. V prípade rozsiahlych dokumentov, ktoré obsahujú značný počet termov, časová náročnosť narastá.

4.1 GridMiner

GridMiner je servisovo založená softvérová architektúra pre distribuovaný datamining v Grid prostredí [8]. Je postavený na Globus Toolkit 3, čo predstavuje štandard v oblasti gridových služieb. Cieľom systému GridMiner je poskytnúť užívateľovi nástroj pre proces objavovania znalostí v distribuovanom gridovom prostredí. Systém taktiež poskytuje grafické používateľské rozhranie, ktoré ukrýva komplexnosť Gridu, ale stále poskytuje možnosti na interferenciu počas vykonávania, kontrolu úloh a vizualizáciu výsledkov.

4.2 JBOWL

JBowl (Java Bag-Of-Worlds Library) je softwarový systém, ktorý bol vyvinutý v prostredí Java ako Open source pre podporu získavania znalostí a text mining [6]. Systém poskytuje jednoduchú rozšíriteľnosť a modulárnu konštrukciu pre predbežné spracovanie indexovania a ďalšieho výskumu veľkých textových súborov. Medzi charakteristické vlastnosti Jbowlu môžeme zahrnúť:

- Schopnosť efektívne predspracovávať veľké súbory textových dokumentov s flexibilnou množinou dostupných techník prespracovania
- Jednotlivé predspracujúce techniky dobre adaptovať na rôznych typoch a formátoch textu (napr. zrozumiteľný text, HTML alebo XML)
- Predstaviteľný súbor textov v rozličných jazykoch, napr. angličtina a slovenčina ako veľmi odlišné druhy jazykov vyžadujú rozdielne prístupy vo fáze predspracovania
- Podpora pre indexáciu a objavovanie v tomto súbore textu (a experimenty s rôznymi technikami objavovania znalostí)
- Má dobre navrhnuté rozhranie pre znalostné štruktúry ako ontológie, kontrolované slovníky alebo WordNet

4.3 Paralelný algoritmus

Tento algoritmus bol navrhnutý pre viacprocesorové systémy. Ako už bolo spomenuté základ algoritmu je

postavený na rozdelení GSOM máp medzi pracovné uzly. Pracovný uzol je v tomto prípade reprezentovaný vláknom. Vstupom do tohto procesu sú dve štruktúry – dokumenty rozdelené zhlukovacím algoritmom (GHSOM mapa) a vektory dokumentov. Algoritmus využíva knižnicu JBowl, v ktorej je implementovaný algoritmus FCA. Tento algoritmus, ktorý sekvenčne generuje lokálne konceptové zväzy, sme rozšírili o triedu vykonávajúcu paralelné generovanie zväzov.

4.4 Paralelný algoritmus

Algoritmus implementuje knižnicu JBowl v rámci systému GridMiner. Algoritmus umožňuje využívať triedy knižnice JBowl pre celý proces dolovania dokumentov od predspracovania, zhlukovania, generovania konceptových zväzov až po vytvorenie výsledného modelu. Zvolený algoritmus je teda implementovaný ako gridová služba, využívajúca "workflow engineïmplementovaný v systéme GridMiner. Mechanizmus distribúcie rozdelí dáta rovnomerne medzi dostupné pracovné stanice, na ktorých sa spustia inštancie tejto služby.

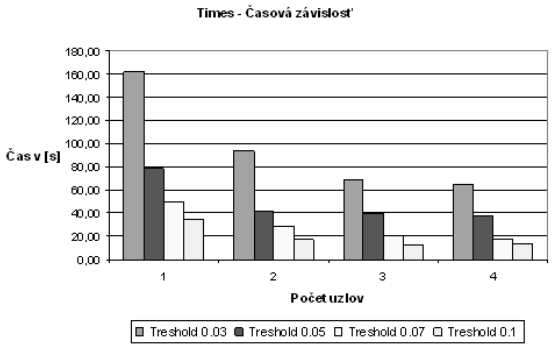
5 Experimenty

5.1 Dátové množiny a popis architektúr

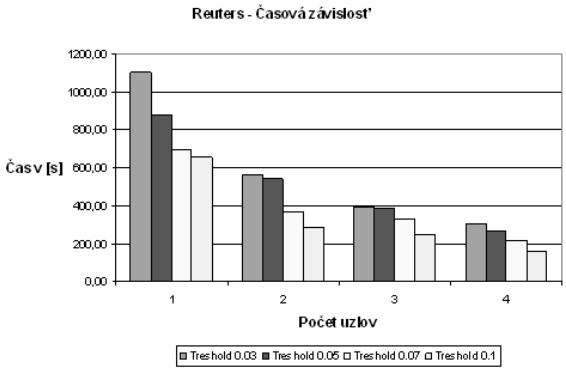
V tejto práci sú použité dve kolekcie dokumentov. Prvou z nich je TIMES 60. Je to kolekcia dát ktorú tvoria články denníka Times zo šesťdesiatych rokov minulého storočia. Obsahuje 420 dokumentov. Textové dokumenty sú článkami s rôznymi vojnovými a politickými témami mnohých krajín sveta ako je Rusko, Veľká Británia, Francúzsko, Malajzia, Egypt, Sýria, atď. Dátová množina bola pomocou zhlukovacieho algoritmu GHSOM rozdelená na 11 podmáp (GSOM máp).

Kolekcia dát Reuters je štandardný voľne dostupný korpus textových dokumentov. Obsahuje 12,902 dokumentov. Každý dokument je novým článkom s nejakou témou, najčastejšie ekonomického charakteru. Pre účel tejto diplomovej práce boli vybrané dokumenty z roku 1987. Kategórie jednotlivých článkov boli priradené ručne. Počet dokumentov je 7769, počet termov 2401. Množina bola algoritmom GHSOM rozdelená na celkovo 286 podmáp.

Experimenty boli vykonané na dvoch architektúrach, v závislosti na verzii algoritmu. Paralelný algoritmus bol vykonávaný na tejto konfigurácii počítača: Dual Core AMD Opteron (tm) 265, 1808.343MHz, 2GB ECC RAM, 3x250GB HDD. Distribuovaný algoritmus sa vykonával na viacerých pracovných staniciach. Vo všeobecnosti sa jednalo o pracovné stanice Sun s rôznymi typmi procesorov a rôznou veľkosťou pamäte.



Obrázok 1. Výsledky paralelnej verzie na dátach Times.



Obrázok 2. Výsledky paralelnej verzie na dátach Reuters.

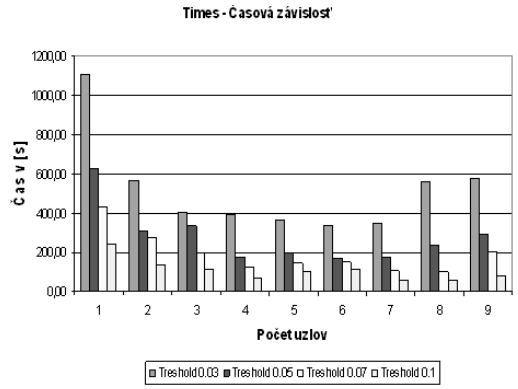
5.2 Experimenty na paralelnej verzii

Experiment bol zameraný na zistenie časovej náročnosti generovania konceptových zväzov v závislosti od počtu uzlov (v tomto prípade počtu vlákien). Pracovali sme s dátovými množinami Times60 a Reuters. K dispozícii sme mali štvor-jadrovú architektúru (2 x dvojjadrový procesor).

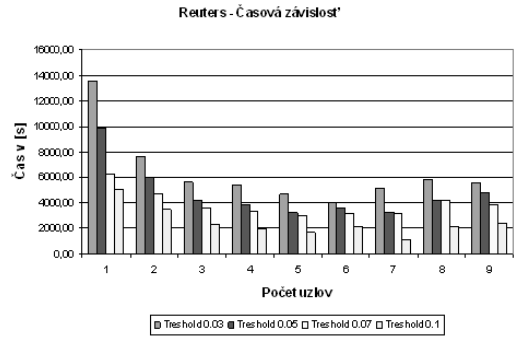
Hodnoty parametra treshold sme zvolili 0.03, 0.05, 0.07 a 0.1. Pre každú hodnotu parametra treshold boli konceptové zväzy generované tri krát a určili sme priemernú hodnotu z týchto meraní. Tieto testy boli vykonané aj pre optimalizované prideľovanie GSOM máp. Počet všetkých meraní pre paralelný algoritmus bol 192. Výsledky sú zhrnuté v nasledujúcich grafoch.

5.3 Experimenty na distribuovanej verzii

Distribuovaná verzia algoritmu je implementovaná v prostredí GridMiner. Experimenty prebehli na rovnakých dátach a s rovnakými nastaveniami ako v prípade paralelnej verzie algoritmu. K dispozícii bolo celkovo deväť pracovných staníc. Jednotlivé experimenty som vykonal iba raz pre štyri hodnoty parametra treshold 0.03, 0.05, 0.07 a 0.1.



Obrázok 3. Výsledky distribuovanej verzie na dátach Times.



Obrázok 4. Výsledky distribuovanej verzie na dátach Reuters.

6 Záver

V tejto práci je predstavená myšlienka možnosti paralelizácie tvorby lokálnych FCA modelov a návrh paralelného a distribuovaného algoritmu pre ich tvorbu a kombináciu. V experimentoch sme porovnávali závislosť počtu pracovných uzlov od výpočtového času generovania výsledného modelu. Z experimentov vyplýva, že paralelný aj distribuovaný algoritmus dosahovali značnú úsporu výpočtového času, už pri malom počte pracovných uzlov. Distribuovaný algoritmus však dosahoval pri väčšom počte pracovných uzlov zhoršenie výpočtového času. Bolo to spôsobené rôznou výpočtovou silou pracovných uzlov v rámci Gridu.

Podstatnejšie zefektívnenie distribuovanej verzie algoritmu by priniesla optimalizácia, kde by sa jednotlivé zhluky alebo GSOM mapy prideľovali uzlom dynamicky. Teda podľa výkonu pracovného uzla. Pracovný uzol by nespracovával množinu zhlukov alebo GSOM máp, ale spracoval by iba jeden prvok (zhluk alebo mapu). Následne až po vygenerovaní FCA modelu pre daný prvok by dostal ďalšie dáta na spracovanie. Týmto spôsobom by sa eliminoval vplyv slabších pracovných uzlov. Výkonovo silnejšie pracovné uzly by spracovali viac dát a teda celý výpočet by trval kratší čas ako v prípade rovnomernej distribúcie medzi pracovné uzly.

Referencie

1. Ganter B. and Wille R., Formal Concept Analysis. Mathematical Foundations, Springer Verlag, 1997
2. Krajči S., Cluster Based Efficient Generation of Fuzzy Concepts. Neural Network World, 13, 5, 2003, 521–530
3. Butka P., Jednoduchá metóda kombinácie lokálnych modelov pri budovaní hierarchie konceptov z textových dokumentov. Znalosti 2007, Proceedings of the 6th Annual Conference, VŠB-TU Ostrava, Czech Republic, 2007, 167–178
4. Dittenbach M., Merkl D., and Rauber A.: Using Growing Hierarchical Self-Organizing Maps for Document Classification. Proceedings of European Symposium on Artificial Neural Networks, ESANN 2000, Bruges, 2000, 7–12
5. Bělohlávek R., Konceptuální svazy a formální konceptuální analýza. Znalosti 2004: Proceedings of the 3rd Annual Conference, Brno, Czech Republic, 2004, 66–84
6. Bednar P., Butka P., and Paralic J., Java Library for Support of Text Mining and Retrieval. Znalosti 2005: Proceeding of the 4th Annual Conference, Stará Lesná, Slovakia, 2005, 162–169
7. Foster I. and Kesselman C. Globus: A Metacomputing Infrastructure Toolkit. The International Journal of Supercomputer Applications and High Performance Computing, 11, 2, 1997, 115–128
8. Brezany I., Janciak A., and Sarnovsky M., Text Mining within the GridMiner Framework. 2nd Dialogue Workshop, Edinburg 2006

Testing different evolutionary neural networks for autonomous robot control*

Stanislav Slušný, Petra Vidnerová, and Roman Neruda

Institute of Computer Science, Academy of Science of the Czech Republic
Pod Vodárenskou věží 2, Prague 8, Czech Republic
slusny@cs.cas.cz

Abstract. *The design of intelligent embodied agents represents one of the key research topics of today's artificial intelligence. The goal of this work is to study emergence of intelligent behavior within a simple intelligent agent. Cognitive agent functions are realized by mechanisms based on neural networks and evolutionary algorithms. The evolutionary algorithm is responsible for the adaptation of a neural network parameters based on the performance of the embodied agent in a simulated environment.*
The evolutionary learning is realized for several architectures of neural networks, namely the feed-forward multilayer perceptron network, the recurrent Elmans neural network, and the radial basis function network.
In experiments, we demonstrate the performance of evolutionary algorithm in the problem of agent learning where it is not possible to use traditional supervised learning techniques. A comparison of different networks architectures is also presented and discussed.

1 Introduction

One of the main goals of Artificial Intelligence (AI) is to gain insight into natural intelligence through a synthetic approach, by generating and analyzing artificial intelligent behavior. In order to glean an understanding of a phenomenon as complex as natural intelligence, we need to study complex behavior in complex environments.

In contrast to traditional systems, reactive and behavior based systems have placed agents with low levels of cognitive complexity into complex, noisy and uncertain environments. One of the many characteristics of intelligence is that it arises as a result of an agent's interaction with complex environments. Thus, one approach to develop autonomous intelligent agents, called *evolutionary robotics*, is through a self-organization process based on artificial evolution. Its main advantage is that it is an ideal framework for synthesizing agents whose behavior emerge from a large number of interactions among their constituent parts [10].

In the following sections we introduce multilayer perceptron networks (MLP), Elman's networks (ELM) and radial basis function networks (RBF). Then we take a look at Khepera robots and related simulation software. In the following section we present two experiments with Khepera robots. In both of them, the artificial evolution is guiding the self-organization process. In the first experiment we expect an emergence of behavior that guarantees full maze exploration. The second experiment shows the ability to train the robot to discriminate between walls and cylinders. In the last section we draw some conclusions and present directions for our future work.

* This work was fully supported by Charles University Grant Agency, project No: GAUK 7637/2007.

2 Neural networks

Neural networks are popular in robotics from various reasons. They provide straightforward mapping from input signals to output signals, several levels of adaptation and they are robust to noise.

2.1 Multilayer perceptron networks

A multilayer feedforward neural network is an interconnected network of simple computing units called neurons which are ordered in layers, starting from the input layer and ending with the output layer [5]. Between these two layers there can be a number of hidden layers. Connections in this kind of networks only go forward from one layer to the next. The output $y(x)$ of a neuron is defined in equation (1):

$$y(x) = g\left(\sum_{i=1}^{n} w_i x_i\right), \tag{1}$$

where x is the neuron with n input dendrites (x_0 ... x_n), one output axon $y(x)$, w_0 ... w_n are weights and $g : \Re \rightarrow \Re$ is the activation function. We have used one of the most common activation functions, the logistic sigmoid function (2):

$$\sigma(\xi) = 1/(1 + e^{-\xi t}), \tag{2}$$

where t determines its steepness.

In our approach, the evolutionary algorithm is responsible for weights modification, the architecture of the network is determined in advance and does not undergo the evolutionary process.

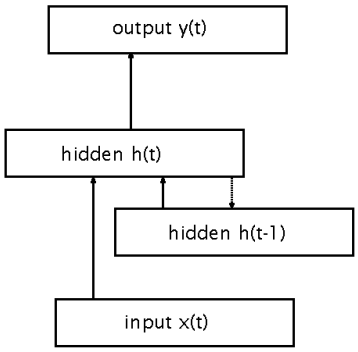


Fig. 1. Scheme of layers in the Elman network architecture.

2.2 Recurrent neural networks

In recurrent neural networks, besides the feedforward connections, there are additional recurrent connections that go in the opposite direction. These networks are often used for time series processing because the recurrent connection can work as a memory for previous time steps. In the Elman [3] architecture, the recurrent connections explicitly hold a copy (memory) of the hidden units activations at the previous time step. Since hidden units encode their own previous states, this network can detect and reproduce long sequences in time. The scheme how the Elman network works is like this (also cf. Fig. 1):

- Compute hidden unit activations using net input from input units and from the copy layer.
- Compute output unit activations as usual based on the hidden layer.
- Copy new hidden unit activations to the copy layer.

2.3 Radial basis function networks

An RBF neural network represents a relatively new neural network architecture. In contrast with the multilayer perceptrons the RBF network contains local units, which was motivated by the presence of many local response units in human brain. Other motivation came from numerical mathematics, radial basis functions were first introduced as a solution of real multivariate interpolation problems [13].

The RBF network is a feed-forward neural network with one hidden layer of RBF units and a linear output layer (see Fig. 2). By the RBF unit we mean a neuron with n real inputs $\boldsymbol{x}$ and one real output y, realizing a radial basis function φ, such as Gaussian:

$$y(\boldsymbol{x}) = \varphi\left(\frac{\parallel \boldsymbol{x} - \boldsymbol{c} \parallel}{b}\right). \tag{3}$$

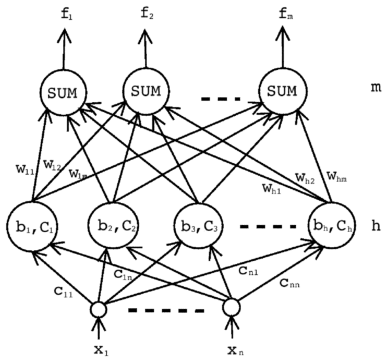


Fig. 2. RBF network architecture.

The network realizes the function:

$$f_s(\boldsymbol{x}) = \sum_{j=1}^{h} w_{js}\varphi\left(\frac{\parallel \boldsymbol{x} - \boldsymbol{c}_j \parallel}{b_j}\right), \tag{4}$$

where f_s is the output of the s-th output unit.

There is a variety of algorithms for RBF network learning, in our previous work we studied their behavior and possibilities of their combinations [8, 9].

The learning algorithm that we use for RBF networks was motivated by the commonly used three-step learning [9]. Parameters of RBF network are divided into three groups: centers, widths of the hidden units, and output weights. Each group is then trained separately. The centers of hidden units are found by clustering (k-means algorithm) and the widths are fixed so as the areas of importance belonging to individual units cover the whole input space. Finally, the output weights are found by EA. The advantage of such approach is the lower number of parameters to be optimized by EA , i.e. smaller length of individual.

3 Evolutionary learning algorithms for robotics

3.1 The Khepera robot

Khepera [7] is a miniature mobile robot with a diameter of 70 mm and a weight of 80 g. The robot is supported by two lateral wheels that can rotate in both directions and two rigid pivots in the front and in the back. The sensory system employs eight "active infrared light" sensors distributed around the body, six on one side and two on other side. In "active mode" these sensors emit a ray of infrared light and measure the amount of reflected light. The closer they are to a surface, the higher is the amount of infrared light measured. The Khepera sensors can detect a white paper at a maximum distance of approximately 5 cm.

In a typical setup, the controller mechanism of the robot is connected to the eight infrared sensors as input and its two outputs represent information about the left and right wheel power. For a neural network we typically consider architectures with eight input neurons, two output neurons and a single layer of neurons, mostly five or ten hidden neurons is considered in this paper. It is difficult to train such a network by traditional supervised learning algorithms since they require instant feedback in each step, which is not the case for evolution of behavior. Here we typically can evaluate each run of a robot as a good or bad one, but it is impossible to assess each one move as good or bad. Thus, the evolutionary algorithm represent one of the few possibilities how to train the network.

3.2 Evolutionary algorithm

The evolutionary algorithms (EA) [6, 4] represent a stochastic search technique used to find approximate solutions to optimization and search problems. They use techniques inspired by evolutionary biology such as mutation, selection, and crossover. The EA typically works with a population of *individuals* representing abstract representations of feasible solutions. Each individual is assigned a *fitness* that is a measure of how good solution it represents. The better the solution is, the higher the fitness value it gets. The population evolves towards better solutions. The evolution starts from a population of completely random individuals and iterates in generations. In each generation, the fitness of each individual is evaluated. Individuals are stochastically selected from the current population (based on their fitness), and modified by means of operators *mutation* and *crossover* to form a new population. The new population is then used in the next iteration of the algorithm.

3.3 Evolutionary network learning

Various architectures of neural networks used as robot controllers are encoded in order to use them the evolutionary algorithm. The encoded vector is represented as a floating-point encoded vector of real parameters determining the network weights.

Typical evolutionary operators for this case have been used, namely the uniform crossover and the mutation which performs a slight additive change in the parameter value. The rate of these operators is quite big, ensuring the exploration capabilities of the evolutionary learning. A standard roulette-wheel selection is used together with a small elitist rate parameter. Detailed discussions about the fitness function are presented in the next section.

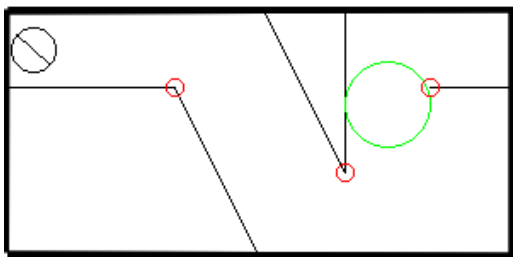


Fig. 3. In the maze exploration task, agent is rewarded for passing through the zone, which can not be sensed. The zone is drawn as the bigger circle, the smaller circle represents the Khepera robot. The training environment is of 60x30 cm.

4 Experiments

4.1 Setup

Although evolution on real robots is feasible, serial evaluation of individuals on a single physical robot might require quite a long time. One of the widely used simulation software (for Khepera robots) is the Yaks simulator [2], which is freely available. Simulation consists of predefined number of discrete steps, each single step corresponds to 100 ms.

To evaluate the individual, simulation is launched several times. Individual runs are called "trials". In each trial, neural network is constructed from the chromosome, environment is initialized and the robot is put into randomly chosen starting location. The inputs of neural networks are interconnected with robot's sensors and outputs with robot's motors. The robot is then left to "live" in the simulated environment for some (fixed) time period, fully controlled by neural network. As soon as the robot hits the wall or obstacle, simulation is stopped. Depending on how well the robot is performing, the individual is evaluated by value, which we call "trial score". The higher the trial score, the more successful robot in executing the task in a particular trial. The fitness value is then obtained by summing up all trial scores.

4.2 Maze exploration

In this experiment, the agent is put in the maze of 60x30 cm. The agent's task is to fully explore the maze. Fitness evaluation consists of four trials, individual trials differ by agent's starting location. Agent is left to live in the environment for 250 simulation steps.

The three-component $T_{k,j}$ motivates agent to learn to move and to avoid obstacles:

$$T_{k,j} = V_{k,j}(1 - \sqrt{\Delta V_{k,j}})(1 - i_{k,j}). \qquad (5)$$

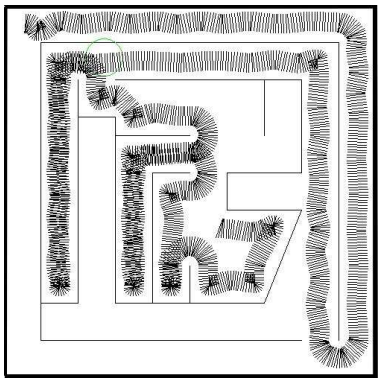


Fig. 4. The agent is put in the bigger maze of 100x100 cm. Agent's strategy is to follow wall on it's left side.

First component $V_{k,j}$ is computed by summing absolute values of motor speed in the k-th simulation step and j-th trial, generating value between 0 and 1. The second component $(1 - \sqrt{\Delta V_{k,j}})$ encourages the two wheels to rotate in the same direction. The last component $(1-i_{k,j})$ encourage obstacle avoidance. The value $i_{k,j}$ of the most active sensor in k-th simulation step and j-th trial provides a conservative measure of how close the robot is to an object. The closer it is to an object, the higher the measured value in range from 0 to 1. Thus, $T_{k,j}$ is in range from 0 to 1, too.

In the j-th trial, score S_j is computed by summing normalized trial gains $T_{k,j}$ in each simulation step.

$$S_j = \sum_{k=1}^{250} \frac{T_{k,j}}{250} \tag{6}$$

To stimulate maze exploration, agent is rewarded, when it passes through the zone. The zone is randomly located area, which can not be sensed by an agent. Therefore, Δ_j is 1, if agent passed through the zone in j-th trial and 0 otherwise. The fitness value is then computed as follows:

$$Fitness = \sum_{j=1}^{4} (S_j + \Delta_j) \tag{7}$$

Successful individuals, which pass through the zone in each trial, will have fitness value in range from 4 to 5. The fractional part of the fitness value reflects the speed of the agent and it's ability to avoid obstacles.

4.3 Results

All the networks included in the tests were able to learn the task of finding a random zone from all four positions. The resulting best fitness values (cf. Tab. 1) are all in the range of 4.3–4.4 and they differ only in the order of few per cent. It can be seen that the MLP networks perform slightly better, RBF networks are in the middle, while recurrent networks are a bit worse in terms of the best fitness achieved. According to their general performance, which takes into account ten different EA runs, the situation changes slightly. In general, the networks can be divided into two categories. The first one represents networks that performed well in each experiment in a consistent manner, i.e. every run of the evolutionary algorithm out of the ten random populations ended in finding a successful network that was able to find the zone from each trial. MLP networks and recurrent networks with 5 units fall into this group. The second group has in fact a smaller trial rate because, typically, one out of ten runs of EA did not produced the optimal solution. The observance of average and standard deviation values in Tab. 1 shows this clearly. This might still be caused by the less-efficient EA performance for RBF and Elman networks.

The important thing is to test the quality of the obtained solution is tested in a different arena, where a bigger maze is utilized (Fig. 4). Each of the architectures is capable of efficient space exploration behavior that has emerged during the learning to find random zone positions. The above mentioned figure shows that the robot trained in a quite simple arena and endowed by relatively small network of 5–10 units is capable to navigate in a very complex environment.

4.4 Walls and cylinders experiment

Following experiment is based on the experiment carried out by Nolfi [11, 12]. The task is to discriminate between the sensory patterns produced by the walls

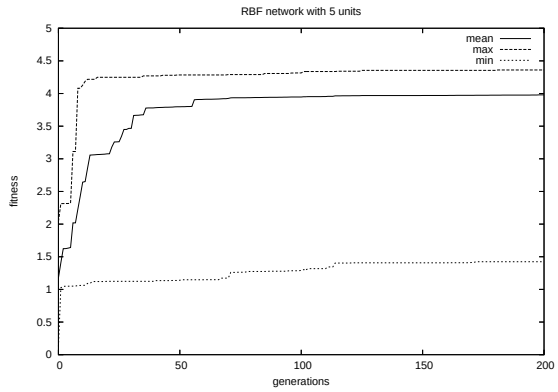


Fig. 5. Plot of fitness curves in consecutive populations (maximal, minimal, and average individual) for a typical EA run (one of ten) training the RBF network with 5 units.

Network type	Maze exploration				Wall and cylinder			
	mean	std	min	max	mean	std	min	max
MLP 5 units	4.29	0.08	4.20	4.44	2326.1	57.8	2185.5	2390.0
MLP 10 units	4.32	0.07	4.24	4.46	2331.4	86.6	2089.0	2391.5
ELM 5 units	4.24	0.06	4.14	4.33	2250.8	147.7	1954.5	2382.5
ELM 10 units	3.97	0.70	2.24	4.34	2027.8	204.3	1609.5	2301.5
RBF 5 units	3.98	0.90	1.42	4.36	1986.6	230.2	1604.0	2343.0
RBF 10 units	4.00	0.97	1.23	4.38	2079.4	94.5	2077.5	2359.5

Table 1. Comparison of the fitness values achieved by different types of network in the experiments.

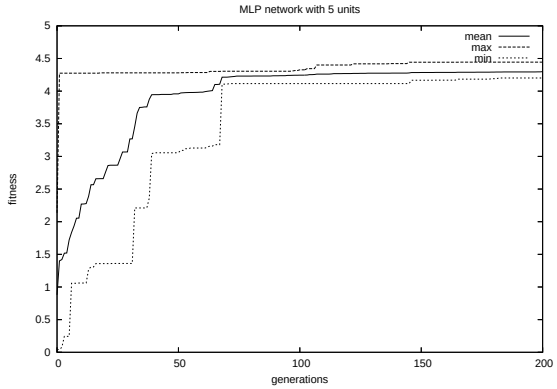


Fig. 6. Plot of fitness curves in consecutive populations (maximal, minimal, and average individual) for a typical EA run (one of ten) training the MLP network with 5 units.

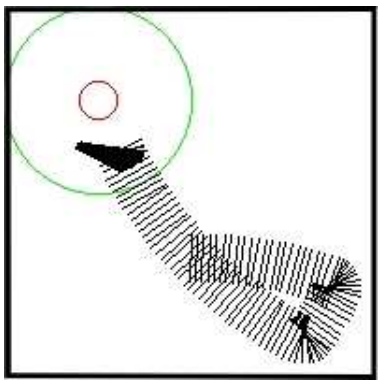


Fig. 7. Trajectory of an agent doing the Walls and cylinders task. The small circle represents the searched target cylinder. The agent is rewarded in the zone represented by a bigger circle. It is able to discriminate between wall and cylinder, and after discovering the cylinder it stays in it's vicinity.

and small cylinders. As noted in [10], passive networks (i.e. networks which are passively exposed to a set of sensory patterns without being able to interact with the external environment through motor action), are mostly unable to discriminate between different objects. However, this problem can easily be solved by agents that are left free to move in the environment.

The agent is allowed to sense the world by only six frontal infrared sensors, which provide it with only limited information about environment. Fitness evaluation consists of five trials, individual trials differ by agent's starting location. Agent is left to live in the environment for 500 simulation steps. In each simulation step, trial score is increased by 1, if robot is near the cylinder, or 0.5, if robot is near the wall. The fitness value is then obtained by summing up all trial scores. Environment is the arena of 40x40 cm surrounded by walls.

4.5 Results

It may seem surprising that even this more complicated task was solved quite easily by relatively simple network architectures. The images of walls and cylinders are overlapping a lot in the input space determined by the sensors.

The results in terms of best individuals are again quite comparable for different architectures with reasonable network sizes. The differences are more pronounced than in the case of the previous task though. Again, the MLP is the overall winner mainly when considering the overall performance averaged over ten runs of EA. The behavior of EA for Elman and RBF networks was less consistent, there were again several runs that obviously got stuck in local extrema (cf. Tab. 1).

We should emphasize the difference between fitness functions in both experiment. The fitness function used in the first experiment rewards robot for single actions, whereas in the second experiment, we describe only desired behavior.

All network architectures produced similar behavior. Robot was exploring the environment by doing arc movements and after discovering target, it started to move there and back and remained in it's vicinity.

5 Conclusions

The main goal of this paper was to demonstrate the ability of neural networks trained by evolutionary algorithm to achieve non-trivial robotic tasks. There have been two experiments carried out with three types of neural networks and different number of units.

For the maze exploration experiment the results are encouraging, a neural network of any of the three types is able to develop the exploration behavior. The trained network is able to control the robot in the previously unseen environment. Typical behavioral patterns, like following the right wall have been developed, which in turn resulted in the very efficient exploration of an unknown maze. The best results achieved by any of the network architectures are quite comparable, with simpler perceptron networks (such as the 5-hidden unit perceptron) marginally outperforming Elman and RBF networks.

In the second experiment it has been demonstrated that the above mentioned approach is able to take advantage of the embodied nature of agents in order to tell walls from cylindrical targets. Due to the sensor limitations of the agent, this task requires a synchronized use of a suitable position change and simple pattern recognition. This problem is obviously more difficult than the maze exploration, nevertheless, most of the neural architectures were able to locate and identify the round target regardless of its position.

The results reported above represent just a few steps in the journey toward more autonomous and adaptive robotic agents. The robots are able to learn simple behavior by evolutionary algorithm only by rewarding the good ones, and without explicitly specifying particular actions. The next step is to extend this approach for more complicated actions and compound behaviors. This can be probably realized by incremental learning one network a sequence of several tasks. Another—maybe a more promising approach—is to try to build a higher level architecture (like a type of a Brooks subsumption architecture [1]) which would have a control over switching simpler tasks realized by specialized networks. Ideally, this higher control structure is also evolved adaptively without the need to explicitly hardwire it in advance. The last direction of our future work is the extension of this methodology to the field of collective behavior.

References

1. Brooks, R.A., A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, **RA-2**, 1986, 14–23
2. Carlsson J. and Ziemke T., YAKS - Yet Another Khepera Simulator. In: S. Ruckert, Witkowski (eds.) Autonomous Minirobots for Research and Entertainment Proceedings of the Fifth International Heinz Nixdorf Symposium, HNI-Verlagsschriftenreihe, Paderborn, Germany, 2001
3. Elman, J., Finding Structure in Time. Cognitive Science, 14, 1990, 179-214
4. Fogel D.B., Evolutionary Computation: The Fossil Record. MIT-IEEE Press. 1998
5. Haykin S., Neural Networks: A Comprehensive Foundation, (2nd ed). Prentice Hall, 1998
6. Holland J., Adaptation In Natural and Artificial Systems (reprint ed). MIT Press, 1992
7. Khepera II web page documentation. `http://www.k-team.com`
8. Neruda R. and Kudová P., Hybrid Learning of RBF Networks. newblock Neural Networks World, 12, 2002, 573–585
9. Neruda R. and Kudová, P.: Learning Methods for RBF Neural Networks. Future Generations of Computer Systems, 21, 2005, 1131–1142
10. Nolfi S., Floreano D., Evolutionary Robotics — The Biology, Intelligence and Technology of Self-Organizing Machines. The MIT Press, 2000
11. Nolfi S., Adaptation as a More Powerful Tool than Decomposition and Integration. In T. Fogarty and G. Venturini (eds.), Proceedings of the Workshop on Evolutionary Computing and Machine Learning, 13th International Conference on Machine Learning, Bari, 1996
12. Nolfi S., The Power and Limits of Reactive Agents. Technical Report. Rome, Institute of Psychology, National Research Council, 1999
13. Powel M., Radial Basis Functions for Multivariable Interpolation: A Review. In: IMA Conference on Algorithms for the Approximation of Functions and Data, RMCS, Shrivenham, England, 1985, 143–167

Learning with regularization networks mixtures*

Petra Vidnerová-Kudová

Institute of Computer Science, Academy of Science of the Czech Republic, v.v.i.
Pod Vodárenskou věží 2, Prague 8, Czech Republic
petra@cs.cas.cz

Abstract. *In this paper we propose a supervised learning model – regularization networks mixtures (RNMs). It is organized as an two-stage architecture. The unsupervised first component realizes partitioning of the input space. The second component consists of a bundle of regularization networks, each trained for one part of the input space. The performance of RNMs is demonstrated on experiments. We show that the model has much lower time requirements but generalization capability comparable to a single regularization network.*

1 Introduction

Learning from data attracted an attention of many scientists during the last years. The amount of data produced in various areas of human activity is rapidly increasing and the need for efficient learning algorithm arises in many applications.

The goal of learning is to find a concise description of the data given as a sample of limited size. In this paper we deal with the *supervised learning*, where the data is a sample of input-output patterns (called training set). Thus a concise description of the data is typically a function that can produce the output, given the input. Supervised learning covers wide range of tasks, including classification of handwritten digits, prediction of stock market share values, and weather forecasting, etc.

An important feature of the learning algorithm is a *generalization ability*. By generalization we mean that the mapping found by the learning algorithm maps correctly not only inputs included in the training set, but also gives correct answers for input points that were not seen before.

In this work we focus on one particular learning algorithm, called *regularization network* (RN). Regularization networks are derived from regularization theory that is based on the idea of simultaneous minimization of error on the training set and regularization term, reflecting our knowledge about a given problem. RNs benefit from very good theoretical background [2, 10, 15, 3]. Our previous work was focused on experimental study of RNs in order to verify their practical applicability and compare them to other learning methods, such as to similar, but computationally cheaper, RBF networks [3, 11]. See [5, 4, 7, 6]. Regularization networks proved to be a competitive learning method in terms of accuracy and generalization error. However, their applicability is limited by high time and space requirements on bigger data sets.

In this paper we focus both on accuracy and efficiency. Our main goal is to decrease the time and space requirements of RN learning. We propose a two-stage architecture that divides the data into clusters and performs learning independently on each of them.

In the next section we briefly describe the derivation of RN. In section 3 we describe the model of regularization networks mixtures in more detail. Section 4 presents the results of our experiments. We show that such approach significantly reduces the time requirements of RN learning.

2 Regularization networks

To describe the derivation of RNs, we will formulate the problem of supervised learning as a function approximation problem. We are given a set of examples (pairs)

$$\{(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}\}_{i=1}^N$$

that was obtained by random sampling of some real function f, generally in the presence of noise. Our goal is to recover the function f from data, or find the best estimate of it.

Note that it is not necessary that the function exactly interpolates all the given data points, but we need a function with a good *generalization*.

It is easy to see that the problem is generally ill-posed. There are many functions interpolating the given data points, but not all of them also exhibit the generalization ability (see Figure 1). Therefore it is necessary to consider some a priori knowledge or assumption about the function f. Typically, it is assumed that the function is smooth, does not oscillate too much, etc.

* This work was supported by the GA ČR grant 201/05/0557 and by the Institutional Research Plan AV0Z10300504 "Computer Science for the Information Society: Models, Algorithms, Applications".

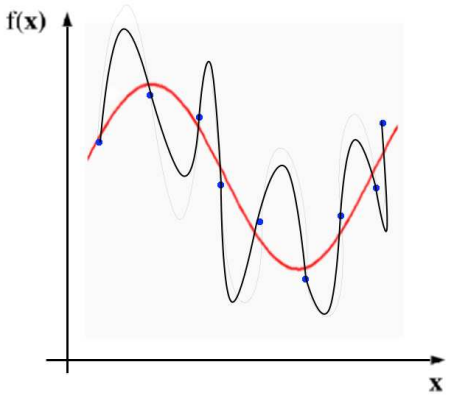


Fig. 1. The problem of learning from examples.

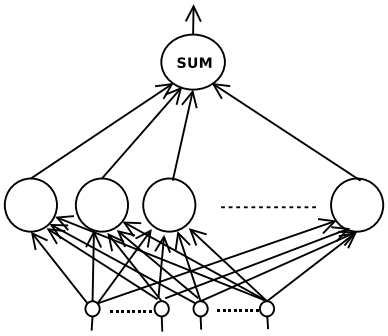


Fig. 2. Regularization network.

Then we are looking for a function minimizing the functional containing both the data term and smoothness information [14]

$$H[f] = \sum_{i=1}^{N} (f(\boldsymbol{x}_i) - y_i)^2 + \gamma \Phi[f], \tag{1}$$

where Φ is called a *stabilizer* and $\gamma > 0$ is *the regularization parameter* controlling the trade-off between the closeness to data and the smoothness of the solution.

Poggio, Girrosi, and Jones [2] used the stabilizer

$$\Phi[f] = \int_{R^d} d\boldsymbol{s} \frac{|\tilde{f}(\boldsymbol{s})|^2}{\tilde{G}(\boldsymbol{s})}, \tag{2}$$

where $\tilde{f}$ indicates the Fourier transform of f , $\tilde{G}$ is some positive function that goes to zero for $||s|| \to \infty$, i.e. $1/\tilde{G}$ is a high-pass filter.

Under slight assumptions on $\tilde{G}$ it can be shown that the minimum of the functional (2) has the form of linear combination of basis functions G

$$f(x) = \sum_{i=1}^{N} w_i G(\boldsymbol{x} - \boldsymbol{x}_i) + \sum_{\alpha=1}^{k} d_\alpha \psi_\alpha(\boldsymbol{x}), \tag{3}$$

where $\{\psi_\alpha\}_{\alpha=1}^{k}$ is a basis of the k-dimensional null space $\mathcal{N}$ of the functional Φ. Coefficients d_α and w_i depend on the data and satisfy the following linear system:

$$(G + \gamma I)\boldsymbol{w} + \Psi^T \boldsymbol{d} = \boldsymbol{y} \tag{4}$$

$$\Psi \boldsymbol{w} = 0 \tag{5}$$

where I is the identity matrix, and $\boldsymbol{y} = (y_1, \ldots, y_N)$, $\boldsymbol{w} = (w_1, \ldots, w_N)$, $\boldsymbol{d} = (d_1, \ldots, d_k)$, $G_{ij} = G(\boldsymbol{x}_i - \boldsymbol{x}_j)$, and $\Psi_{\alpha i} = \psi_\alpha(\boldsymbol{x}_i)$.

For positive semi-definite function G the null space is empty, for conditionally positive semi-definite function G the basis of the null space is a set of polynomials, however in practical applications the polynomial term is omitted. Then the function f (3) is a linear combination of basis functions G and can be represented by a neural network with one hidden layer (see Figure 2). We call such a network *regularization network* (RN).

Input: Data set $\{\boldsymbol{x_i}, y_i\}_{i=1}^{N} \subseteq R^n \times R$
Output: Regularization network.

1. Set the centers of kernels:

$$\forall i \in \{1, \ldots, N\} : \boldsymbol{c_i} \leftarrow \boldsymbol{x_i}$$

2. Compute the values of weights:

$$(G + \gamma I)\boldsymbol{w} = \boldsymbol{y},$$

Fig. 3. RN learning algorithm.

The learning algorithm for RN (see Fig. 3) is straightforward. It fixes the second arguments of the basis functions (called centers) to the given data points, and computes the output weights w_i as a solution of linear system $(G + \gamma I)\boldsymbol{w} = \boldsymbol{y}$. The regularization parameter γ and the possible parameters of basis function G (such as a width of the Gaussian function) are estimated by cross-validation.

The key point of the algorithm is the second step, i.e. linear system solving. Efficient numerical algorithms exist [1, 16, 13, 8], but note that the size of matrix G is $N \times N$, where N is the number of data points. So the size of the matrix, as well as time and space requirements of the learning, increases with increasing size of data.

To reduce the time and space requirements of learning, we propose to use an *divide et impera* approach that we describe in the following section.

3 Mixtures of regularization networks

As we have discussed in the previous section, the size of the linear system (4) is one of the main limiting factors of the RN learning algorithm. To overcome this

difficulty we propose a learning model based on regularization networks and mixtures of experts paradigm.

It is designed as a two-stage architecture. The first stage component consist of clustering unit that performs partitioning of the input space into disjoint, or possibly overlapping, parts. The second stage component consists of a set of regularization networks.

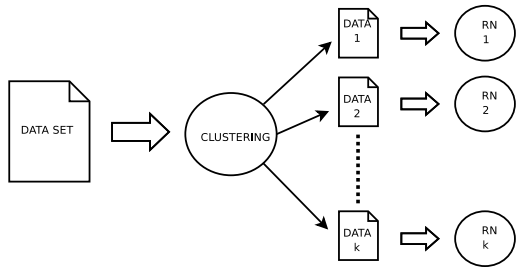


Fig. 4. Regularization networks mixture: learning phase.

The learning with regularization networks mixtures is described by Figure 4. It consists of two steps. First, unsupervised learning is performed by the clustering unit. For this step we use k-means clustering algorithm (but other methods, such as self-organizing maps, hierarchical clustering, may be used as well). Then, the data are divided into k clusters and for each cluster one regularization network is created and trained.

Evaluation phase is described in Figure 5. When a new input vector is presented, the clustering unit calculates its distance from each cluster center. The network corresponding to the nearest cluster is selected and used to evaluate the input vector.

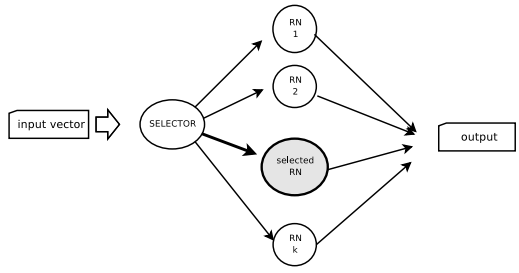


Fig. 5. Regularization networks mixture: evaluation phase.

Learning with regularization networks mixtures benefits from smaller time and space requirements than learning with a single regularization network. This is caused by the fact that one large linear system is replaced by k smaller ones. The question to answer is whether such approach preserves the quality of the learning, i.e. accuracy of the solution and generalization ability. To give an answer is the goal of the experiments described in the following section.

4 Experimental results

The aim of our experiments was to demonstrate the performance of regularization networks mixtures and to compare them to a single regularization network. For this purpose we have chosen the Proben1 data repository containing both approximation and classification tasks, listed in Table 1. Each task is present in three variants corresponding to three different partitioning into training and test data. We refer to this variants with suffix 1,2, or 3 (e.g. cancer1, cancer2, cancer3). More details of the individual data sets, their source, and qualities can be found in [12].

Task name	n	m	N_{train}	N_{test}	**Type**
building	14	3	3156	1052	class
cancer	9	2	525	174	class
card	51	2	518	172	class
diabetes	8	2	576	192	class
flare	24	3	800	266	approx
gene	120	3	2382	793	class
glass	9	6	161	53	class
heartac	35	1	228	75	approx
hearta	35	1	690	230	approx
heartc	35	2	228	75	class
heart	35	2	690	230	class
horse	58	3	273	91	class
mushroom	125	2	6093	2031	class
soybean	82	19	513	170	class
thyroid	21	3	5400	1800	class

Table 1. Overview of Proben1 tasks. Number of inputs (n), number of outputs (m), number of samples in training and testing sets (N_{train},N_{test}). Type of task: approximation or classification.

We work with distinct data sets for training and testing, referred to as the *training set* and the *test set*. The learning algorithm is run on the training set, including the cross-validation. The test set is never used during the learning phase, it is only used for the evaluation of error of the resulting network.

For the data set $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$ and the network representing a function f, the normalized error is computed as follows:

$$E = 100\frac{1}{Nm}\sum_{i=1}^{N}||\boldsymbol{y}_i - f(\boldsymbol{x}_i)||^2, \qquad (6)$$

where $||\cdot||$ denotes the Euclidean norm.

Task	RN	MRN	MORN
building1	0.426	**0.420**	0.434
building2	**0.221**	0.225	0.224
building3	**0.214**	0.226	0.227
cancer1	1.717	1.758	**1.169**
cancer2	**2.995**	3.630	3.531
cancer3	2.850	4.470	**2.774**
card1	**9.769**	12.038	11.308
card2	**12.598**	13.498	13.322
card3	**12.897**	14.372	14.665
diabetes1	**15.920**	16.849	16.818
diabetes2	**16.972**	18.030	17.516
diabetes3	15.994	16.116	**15.805**
flare1	0.546	**0.545**	0.546
flare2	**0.269**	0.279	0.280
flare3	**0.337**	0.358	0.357
gene1	**30.315**	30.319	**30.315**
gene2	**29.622**	29.623	**29.622**
gene3	**30.050**	30.087	**30.050**
glass1	6.899	**6.766**	6.775
glass2	**7.803**	8.499	8.527
glass3	**7.268**	7.887	7.934
heart1	**13.712**	14.326	14.522
heart2	**13.839**	14.542	14.917
heart3	**15.950**	17.042	16.761
hearta1	**4.355**	4.366	4.403
hearta2	**4.129**	4.356	4.305
hearta3	**4.494**	4.814	4.773
heartac1	**3.048**	3.825	4.313
heartac2	**3.886**	6.919	6.781
heartac3	**5.160**	5.820	5.920
heartc1	**15.943**	16.732	16.500
heartc2	**6.473**	8.328	7.968
heartc3	**13.229**	13.560	13.474
horse1	**12.091**	14.449	14.678
horse2	**15.363**	17.529	16.807
horse3	**13.581**	14.083	14.781
mushroom1	**0.000**	**0.000**	**0.000**
mushroom2	**0.001**	**0.001**	**0.001**
mushroom3	**0.000**	**0.000**	**0.000**
soybean1	0.664	0.677	**0.659**
soybean2	0.491	0.466	**0.462**
soybean3	**0.586**	0.630	0.609
thyroid1	**3.007**	3.104	3.109
thyroid2	**3.067**	3.190	3.190
thyroid3	3.194	**2.999**	3.000

Table 2. Comparison of errors on the test set obtained by RN, Mixture of RNs, and Mixture of overlapping RNs.

Task	RN	RNM	
		sum	max
building1	120	16	10
building2	140	16	11
building3	118	18	12
cancer1	1	1	1
cancer2	1	1	1
cancer3	1	1	1
card1	2	1	1
card2	2	1	1
card3	1	1	1
diabetes1	1	1	1
diabetes2	1	1	1
diabetes3	1	1	1
flare1	3	1	1
flare2	4	1	1
flare3	3	2	1
gene1	95	11	3
gene2	91	10	4
gene3	70	15	6
glass1	1	1	1
glass2	1	1	1
glass3	1	1	1
heart1	3	1	1
heart2	2	1	1
heart3	3	1	1
hearta1	2	1	1
hearta2	3	1	1
hearta3	2	1	1
heartac1	1	1	1
heartac2	1	1	1
heartac3	1	1	1
heartc1	1	1	1
heartc2	1	1	1
heartc3	1	1	1
horse1	1	1	1
horse2	1	1	1
horse3	1	1	1
mushroom1	846	123	78
mushroom2	945	118	78
mushroom3	1049	134	94
soybean1	2	1	1
soybean2	2	1	1
soybean3	1	1	1
thyroid1	518	133	117
thyroid2	614	136	116
thyroid3	615	127	112

Table 3. Time comparison. Times rounded up to seconds.

The experiments have been performed in the system Bang [9], the standard numerical library LAPACK [8] was used to solve linear least square problems (step 2 in RN learning algorithm).

We have run a single regularization network (RN), mixture of regularization networks (MRN), and mixture of regularization network with overlapping clusters (MORN) (the clusters created by the first step are slightly overlapping to ensure good approximation along the splits) on all tasks from Proben1 repository. Gaussian function (which is possitive-definite) was used as a basis function. In both mixtures, clustering was performed using k-means algorithm, number of clusters was 4. Table 2 compares the errors achieved on the test set.

We can see that the single network achieved the best results on 36 tasks from 45, however the differences in the error are not significant almost in all cases. The expected improvement in case of overlapping clusters was observed only in 60% cases, so further experiments will be needed to identify the optimal overlapping rate.

Table 3 lists the times needed for evaluation of the RN learning algorithm (Figure 3) by RN and RN mixtures. In case of RN mixtures we included both sum and maximum of times over all regularization networks (the maximum represent an estimate of time needed by parallel computation). On the bigger tasks using RN mixtures significantly reduced the time requirements.

5 Conclusion

In order to reduce space and time requirements of learning with regularization networks, we have designed a two-stage architecture – regularization network mixtures. We have tested the RN mixtures on benchmark data sets and compared them to single regularization networks. We have shown that the time and space requirements can be significantly reduced by the use of RN mixtures with the reasonable increase of error on the test set.

Our future work will be focused on the improving of clustering phase. Firstly, we would like to obtain clusters which have similar sizes. This will ensure that time requirements of individual regularization networks will not differ too much. Secondly, we would like to further improve the approximation accuracy by tuning the cluster overlap.

References

1. E. Björk, Numerical Methods for Least Square Problems. SIAM, Philadelphia, 1996
2. F. Girosi, M. Jones, and T. Poggio, Regularization Theory and Neural Networks Architectures. Neural Computation, 2, 7, 1995, 219–269
3. S. Haykin, Neural Networks: a Comprehensive Foundation. Tom Robins, 2nd edition, 1999
4. P. Kudová, Learning with Kernel Based Regularization Networks. In Information Technologies - Applications and Theory, Košice, Prírodovedecká fakulta Univerzity Pavla Jozefa Šafárika, 2005, 83–92
5. P. Kudová and R. Neruda, Kernel Based Learning Methods: Regularization Networks and RBF Networks. In Lawrence N. Winkler J., Niranjan M., (ed.), Deterministic and Statistical Methods in Machine Learning, Berlin, Springer-Verlag, 2005, 124–136
6. P. Kudová and T. Šámalová, Product Kernel Regularization Networks. In Ribeiro B., Albrecht R.F., Dobnikar A., Pearson D.W., and Steele N.C., (eds), Adaptive and Natural Computing Algorithms, Wien, Springer Verlag, 2005, 433–436
7. P. Kudová and T. Šámalová, Sum and Product Kernel Regularization Networks. In L. Rutkowski, R. Tadeusiewicz, L.A. Zadeh, and J. Zurada, (eds), Artificial Intelligence and Soft Computing, Lecture Notes in Artificial Intelligence, Berlin, Springer-Verlag, 2006, 56–65
8. LAPACK. Linear algebra package, `http://www.netlib.org/lapack/`.
9. R. Neruda, P. Krušina, P. Kudová, P. Rydvan, and G. Beuster, Bang 3: A Computational Multi-Agent System. In Zhong N., Bradshaw J., Pal S.K., Talia D., Liu J., and Cerrone N., (eds), Intelligent Agent Technology, Piscataway, IEEE, 2004, 563–564
10. T. Poggio and S. Smale, The Mathematics of Learning: Dealing with Data. Notices of the AMS, 50, 5, 2003, 536–544
11. M.J.D Powel, Radial Basis Functions for Multivariable Interpolation: A Review. In IMA Conference on Algorithms for the Approximation of Functions and Data, RMCS, Shrivenham, England, 1985, 143–167
12. L. Prechelt, PROBEN1 – a Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms. Technical Report 21/94, Universitaet Karlsruhe, 9 1994
13. W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, Numerical Recipes: The Art of Scientific Computing. Cambridge University Press, 1986
14. A.N. Tikhonov and V.Y. Arsenin, Solutions of Illposed Problems. W.H. Winston, Washington, D.C, 1977
15. G. Wahba, Spline Models for Observational Data. Series in Applied Mathematics, 59, 1990
16. J. H. Wilkinson and C. Reinsch, Handbook for Automatic Computation, volume 2 of Linear Algebra. Springer, Berlin, 1971

Dynamizace gridu*

Zuzana Vlčková and Leo Galamboš

Univerzita Karlova v Praze, Matematicko-fyzikální fakulta, Katedra softwarového inženýrství
Malostranské nám. 25, 118 00 Praha 1, Česká republika
Zuzana.Vlckova@gmail.com, Leo.Galambos@mff.cuni.cz

Abstrakt *V současnosti je kladen velký důraz na rychlost aktualizace indexů nad velkými množinami dat, např. nad všemi stránkami Internetu. Tento článek se věnuje problematice vytváření invertovaných indexů, dynamizaci statických struktur a nakonec distribuci dynamického indexu na víc uzlů. V každém kroku vylepšování těchto datových struktur popíšeme, jakým způsobem je potřeba tyto změny provést a jaké jsou výhody (popř. nevýhody) nově vzniklých datových struktur. Cílem článku je experimentálně ukázat, že aktualizace dynamického invertovaného indexu rozmístěného na více uzlech je řádově rychlejší než aktualizace statického invertovaného indexu a můžeme ušetřit až desítky procent operací* read *a* write.

1 Úvod

Vyhledávače jsou systémy, které se snaží na uložených datech získávat odpovědi na dotazy typu „Ve kterém dokumentu se nachází slova...“. My se budeme věnovat vyhledávačům nad stránkami internetu, jejichž význam stále roste [13] a je důležité, aby byla optimalizována rychlost vyhledávání a množství spotřebovaných prostředků.

Po zadání dotazu vrátí vyhledávač dokumenty, které jsou seřazeny podle určitého kritéria (většinou se používá nějaká ohodnocovací funkce, např. PageRank u Google [3],[9]).

Vyhledávače používají tzv. index [4] kvůli optimalizaci rychlosti u vyhledávání relevantních dokumentů po zadání dotazu. Bez něj by musel vyhledávač pokaždé projít všechny dokumenty, což by zbytečně zabralo spoustu času a výpočetní síly. Podle [12] je na internetu víc než 2,5 miliard dostupných stránek. Není možné, aby nad takto velkým počtem dokumentů fungoval vyhledávač bez indexu, protože by vyhledávání procházením všech dokumentů trvalo řádově hodiny. Cena, kterou zaplatíme za rychlost vyhledávání v indexu (řádově vteřiny) je ta, že musíme pravidelně tento index aktualizovat a právě tato operace může na dlouhý čas znemožnit efektivní práci s indexem.

* Práce byla podpořena projektem 1ET100300419 programu Informační společnost (Tématického programu II Národního programu výzkumu v ČR: Inteligentní modely, algoritmy, metody a nástroje pro vytváření sémantického webu).

Postupně budeme budovat indexovací algoritmus vyhledávače a optimalizovat ho. Sekce 2 popisuje postup vytvoření invertovaného indexu. V sekci č. 3 se budeme věnovat dynamizaci statické datové struktury a vlastnostem dynamických struktur. Sekce 4 popisuje použité datové struktury a jejich vlastnosti tak, jak jsou použity v EGOTHORu (viz. [7]). V sekci 5 popisujeme další možnost jak zlepšit vlastnosti aktualizace indexu, a to distribucí datových struktur na různé uzly. V sekci 6 se věnujeme simulačnímu programu a zkoumaným veličinám. Sekce 7 popisuje výsledky simulačního programu.

2 Invertovaný index

Konstrukci invertovaného indexu lze vysvětlit ve dvou krocích [6]: Nejdřív vytvoříme dopředný index, který obsahuje všechna významová slova dokumentu a pak jej zkonvertujeme na invertovaný index. Invertovaný index tedy mapuje všechna tato slova na pozice v jednotlivých dokumentech vstupní množiny souborů dat. Tato datová struktura umožňuje fulltextové vyhledávání, proto je využívána v systémech zpracovávajících data, jako např. ve vyhledávačech. V praxi se ale invertovaný soubor vytváří rovnou (bez použití dopředného indexu).

Celá operace vytváření indexu může zabrat spoustu času – k vybudování invertovaného indexu je nutné projít všechny dokumenty a najít v nich všechna významová slova. Proto nechceme tuto strukturu vytvářet pokaždé znovu.

Naším cílem je optimalizovat dobu, pamět a výpočetní zdroje, které potřebujeme, abychom udržovali index aktuální. Toho dosáhneme dynamizací této datové struktury (sekce 3) a následně ještě rozdělením nové struktury na víc uzlů a rozdělení aktualizace mezi ně.

3 Dynamizace

Indexy, vytvořené nad statickými datovými strukturami, mají mnoho výhod, ale také několik zásadních nevýhod. Umožňují rychlé vyhledávání a jednoduché vytváření struktury. Na druhou stranu do nich nelze jednoduše vkládat nebo z nich odstraňovat velké

množství prvků, protože tyto operace mohou vyžadovat celkové přestavění datové struktury. U indexů jsou vkládání, aktualizace a mazání záznamů nejčastějšími operacemi. Jedním z řešení je využití dynamizace.

Dynamizace pozůstává z tzv. dekompozice. Během této fáze rozdělíme statickou datovou strukturu do různě velkých menších jednotek. Základní myšlenkou je fakt, že každé přirozené číslo lze jednoznačně vyjádřit v soustavě s libovolnou bází. Pro jednoduchost implementace zvolíme binární reprezentaci.

Mějme datovou strukturu s n elementy. Pokud je n_i i-tá cifra v binárním zápisu čísla n, můžeme n zapsat jako $\sum_i 2^i * n_i$. Pro každou nenulovou cifru na i-tém místě bude mít nová množina velikost 2^i, jinak bude množina prázdná. Každá z těchto nově vytvořených množin má stejné vlastnosti jako původní statická struktura. Operace na dynamických datových strukturách vyžadují projít maximálně $log(n)$ těchto podmnožin. Obecně musíme provést o $O(log(n))$ operací víc než v původní množině [10], [11]. Tuto cenu zaplatíme za možnost vkládat nebo mazat prvky struktury.

4 Správa indexů

Index bude reprezentován následujícím způsobem:

Nad statickým indexem, reprezentujícím invertovaný index pro všechny dokumenty, tzv. barelem, bude vystavěna dynamická datová struktura, kterou nazveme tanker (podle [7]).

Tanker je tedy dynamický index, který je složen z barelů. Obecně barely představují jednotlivé dokumenty. U každého barelu B si pamatujeme jeho kapacitu $B.capacity$ (maximální možný počet dokumentů, které reprezentuje), velikost $B.size$ (aktuální počet dokumentů v barelu), a pak pro každý dokument bit *deleted*, který určuje, zda je smazán nebo aktivní. Značení B_i reprezentuje i-tý barel.

Algoritmus 1 Do tankeru vkládáme barel B

1: najdeme minimální index x: $|B| + \sum_{i=0}^{x} |B_i| \leq 2^x$
2: sloučíme všechny barely s indexem $i \leq x$ s novým barelem B a nahradíme ním barel B_x
3: u barelů s indexem $i < x$ nastavíme $B_i = \emptyset$

V tankeru máme uloženy ukazatele na všechny barely, které obsahuje index největšího barelu $tanker.r$. U těchto barelů vyžadujeme, aby byla splněna podmínka vzniklá dekompozicí během dynamizace, a to:

$$\forall i : B_i \neq \emptyset \rightarrow 2^{i-3} < |B_i| \,\&B.size \leq 2^i \qquad (1)$$

kde B_i je i-tý barel tankeru a $|B_i|$ označuje velikost i-tého barelu ($B_i.size$).

Operace insert (popsána algoritmem 1) vkládá barel B do tankeru. Nejdřív najdeme barely, které lze sloučit s nově vkládaným barelem a vložit je na volné místo s nejnižším indexem tak, aby zůstala zachována podmínka 1.

Algoritmus 2 Metoda modify dle [8]

{Krok 1:}
odstraníme dokumenty s příznakem *deleted*
{Krok 2:}
vytvoříme novou strukturu N:
{(tyto barely odstraníme ze struktury N)}
$Y = Y \cup B_{ins}$
$\forall i : |B_i| == 0 \rightarrow N_i = \emptyset$
if $|B_i| > 2^{i-3}$ **then**
  $N_i = B_i$
end if
if $N_{i-1} \geq 2^{i-2}$ **then**
  $N_i = N_{i-1}$; $N_{i-1} = B_i$
else
  $Y = B_i + N_{i-1}$
  if $|Y| \geq 2^{i-2}$ **then**
    $N_i = Y$; $N_{i-1} = \emptyset$
  else
    $N_i = \emptyset$; $N_{i-1} = Y$
  end if
end if
{Krok 3:}
odstraníme příliš malé barely a vložíme přidávaný barel B_{ins}
$Y = \cup\{B_i : |B_i| > 2^{i-3}\}$
$N.insert\,(Y)$ (do N vložíme funkcí insert - viz. alg. 1) barel Y
{Krok 4:}
$B = N$, původní B uvolníme

Při mazání dokumentu nastavíme jeho flag *deleted* na true. Pokud je v barelu hodně dokumentů označených jako smazané, může být podmínka 1 porušena, musíme spustit restrukturalizaci tankeru. Funkce modify se skládá ze čtyř hlavních kroků:

1. nejdřív odstraníme dokumenty, u kterých je nastaven příznak *deleted*
2. vytvoříme novou strukturu N, která bude obsahovat pouze platné dokumenty a která zároveň splňuje podmínky velikostí barelů (podmínka 1)
3. ani po těchto krocích nemusí kvůli odstraňování smazaných dokumentů struktura splňovat podmínku, že pro každý její barel B_i platí $|B_i| > 2^{i-3}$. Proto v dalším kroku sloučíme všechny takové barely spolu s barelem, který máme do struktury vložit do pomocného barelu Y, odstraníme barely ze struktury N a pomocí funkce insert tam vložíme barel Y.
4. v posledním kroku provedeme samotnou aktualizaci a strukturu B nahradíme novou strukturou N. Strukturu B odstraníme. Protože jediná operace, která zapisuje do struktury B, obsahující barely, je $B = N$, re-

strukturalizace může probíhat během normálního provozu indexu.

Postup této operace je popsán v algoritmu 2. Tuto funkci můžeme využít také pro dávkování příkazů insert a delete.

V indexech vyhledávačů je nejčastější operaci update, což se někdy řeší dvojící operací delete a insert [2], [5]. Proto vyžadujeme, aby modifikační funkce dokázala pracovat i s tímto faktorem a umět odstranit a vložit dokumenty, kterých se týká aktualizace.

5 Distribuce indexu

V dalším kroku náš algoritmus aktualizace indexu ještě vylepšíme tím, že použijeme víc tankerů, které rozmístíme na několik různých uzlů. Proto bude aktualizace probíhat na menších dynamických strukturách (tankery budou mít na stejném počtu dokumentů menší velikost, v závislosti na počtu tankerů). Barely v tankerech budou rozděleny rovnoměrně.

Je několik možností, jak vybírat tanker, ke kterému bude přidán barel s aktualizovanými dokumenty (sekce 7.2). Tankery můžeme vybrat náhodně, můžeme je pravidelně střídat a vždy vybrat následující, další možností je vybrat tanker s nejmenším počtem barelů nebo nejmenším počtem dokumentů. My budeme používat metodu výběru následujícího tankeru, v sekci 7 pak tuto metodu srovnáme s ostatními (obr. 6, 8, 7).

Celý algoritmus aktualizace indexu je popsán v algoritmu (alg. 3).

Algoritmus 3 Metoda modify v distribuovaném prostředí

for all i:=0...iter **do**
{v každém kroku iterace}
1: pro každý tanker provedeme aktualizaci popsanou v algoritmu 2
2: vybereme tanker, ke kterému přidáme barel obsahující aktualizované dokumenty a provedeme na něm znovu funkci modify
end for

6 Simulace

Naším cílem je ukázat, že dynamizace výrazně zlepšuje vlastnosti invertovaného indexu. Výsledky jsou popsány v [7] a ukazují, že tento způsob aktualizace indexu lze úspěšně použít u indexů kde není možné během aktualizace indexu přerušit nabízené služby. V tomto článku je také ukázáno, že dynamickou správou indexů lze ušetřit desítky procent času (přesné číslo závisí na ostatních vstupních parametrech) v porovnání s kompletním vybudováním indexu v každém kroku. Na druhou stranu za toto zrychlení zaplatíme zpomalením vyhledávání.

Simulační program nepracuje přímo s dokumenty, ale jen s počty (např. datová strutura pro barel obsahuje počet dokumentů a počet smazaných dokumentů).

Abychom simulovali distribuci Page Ranku [3], [9], používáme rozdělení ohodnocení stránek podle tzv. Power-law zákona [1]. Dokumenty jsou rozděleny do deseti kategorií podle svého ohodnocení. Kvůli jednodušší implementaci jsme použili exponenciální rozdělení se základem 2, proto je počet dokumentů v sousedních kategoriích vždy v poměru 2^i ke 2^{i+1} (1:2). U nově vytvořeného barelu tedy dokumenty dosahují rozdělení podle Power-law zákona. U starších barelů se poměry mezi počty dokumenů v jednotlivých kategoriích mění podle počtu dokumentů, které jsou označeny jako smazané.

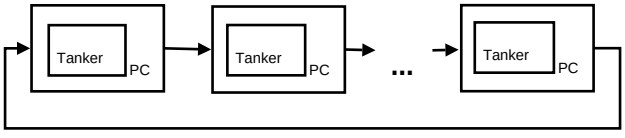
Náš simulační program dostane jako parametry tato čísla: n - log počtu dokumentů při založení tankeru, t - počet tankerů, *percent* - procentuální podíl dokumentů, které budou v každém kroku označeny jako smazané, *hit* - počet dokumentů vyhovujících zadanému dotazu, které vrátí vyhledávač při zpracovávaní dotazu.

Vytvoříme t tankerů. V každém tankeru bude n barelů, které budou na začátku až na poslední prázdné. V n-tém tankeru bude barel s 2^n dokumenty. Počet smazaných dokumentů bude 0, kapacita barelu bude rovna její inicializační velikosti.

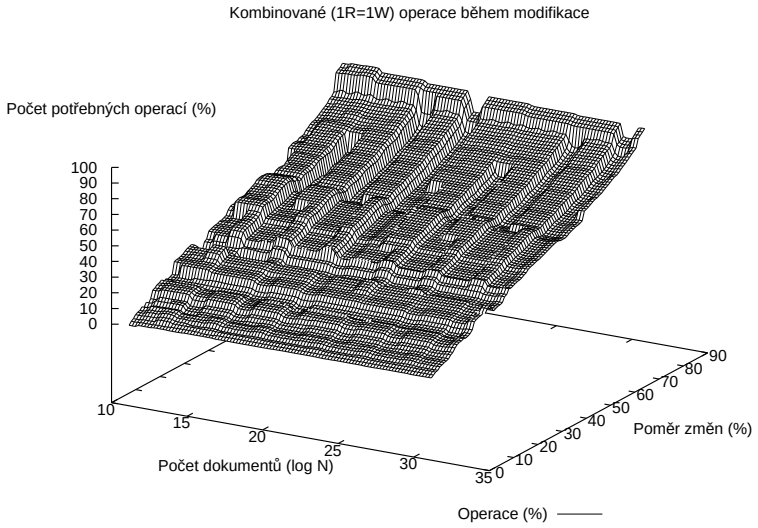
Pak simulujeme aktualizaci invertovaného indexu tím, že v cyklu procházíme všechny tankery a na každém provedeme funkci *modify*, která označí v závislosti na zadaném parametru přibližně *percent* procent nesmazaných dokumentů jako smazané a provede algoritmus 2 popsaný v sekci 4. Během této aktualizace nevkládáme smazané dokumenty zpátky do tankerů. Z těchto dokumentů vytvoříme jeden barel, který pak přidáme do vybraného tankeru.

Pak vybereme tanker, ke kterému přidáme barel s těmito novými dokumenty a provedeme ještě jednou modifikaci funkcí *modify*. Jako metodu výběru tankeru zvolíme následující tanker (podle svého *id*) - je znázorněna na obr. 1.

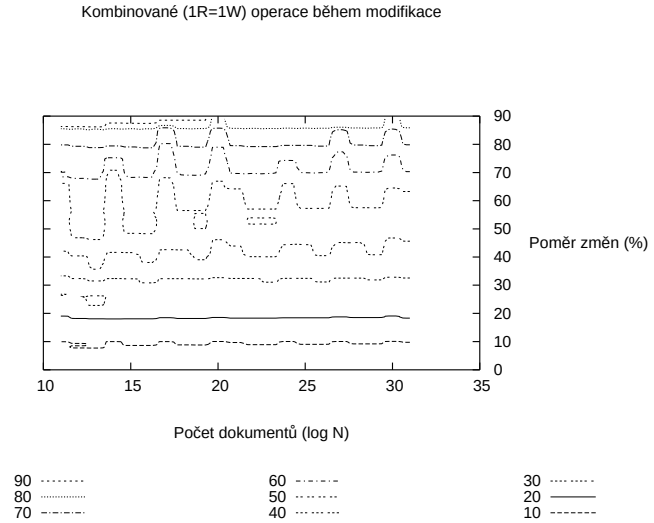
Tento krok představuje samotnou aktualizaci dokumentů. Místo toho, abychom u dokumentů, kde se



Obrázek 1. Metoda výběru následujícího tankeru.



Obrázek 2. Součet operací *read* + *write* pro 1 tanker.



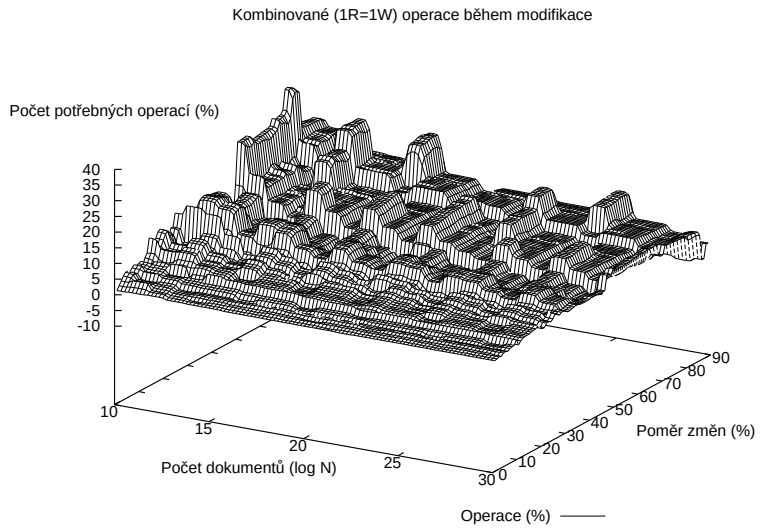
Obrázek 3. Jeden tanker - vrstevnicový graf operací *read* + *write*.

změnil obsah, jenom přepsali původní data, tyto dokumenty odstraníme a pak je znovu vložíme s aktuálním obsahem.

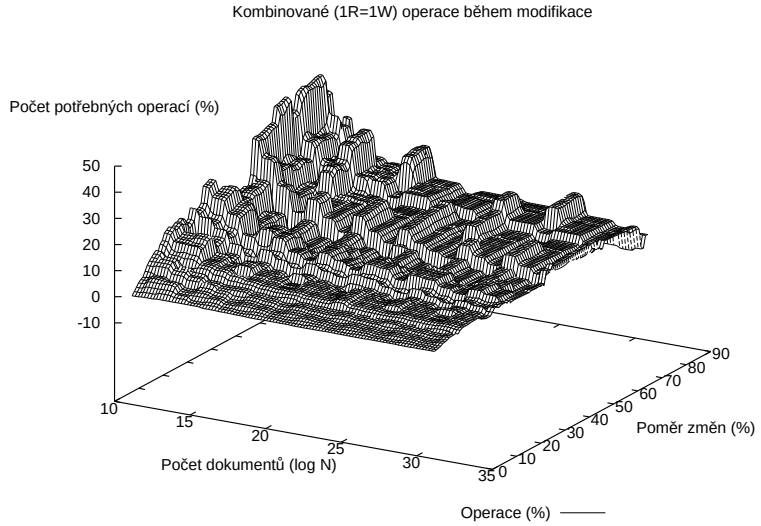
Celá simulace probíhá v *iter* iteracích. Během iterací počítáme jednotlivé veličiny, které nás zajimají, abychom z nich pak spočetli průměrné, případně maximální hodnoty. Postupně tyto sledované veličiny popíšeme: *capacity* - součet kapacit všech barelů tankeru, *capacityInHigh* - kapacita barelu s nejvyšším číslem, *occInHigh* - poměr velikosti a kapacity barelu s nejvyšším číslem, *top()* - vrátí počet operací *read* potřebných pro dosažení *hit* hitů, *topAll()* - vrací počet operací *read* potřebných pro dosažení *hit* hitů v každém barelu, *topBiggest()* - vrací počet operací *read* nutných pro dosažení *hit* hitů v největším barelu, *topFromBiggest()* - vrací počet operací *read* potřebných pro dosažení *hit* hitů v tankeru. Začínáme v největším barelu a pokračujeme k barelům s menším počtem dokumentů.

7 Experimenty

Na obrázku 2 je počet operací *read* a *write* během aktualizace na dokumentech s jedním tankerem. Tento tanker měl původně 2^n dokumentů, kde n určuje osa x



Obrázek 4. Metoda next: rozdíl počtu operací u jednoho a 8 tankerů.



Obrázek 5. Metoda next: rozdíl počtu operací u jednoho a 32 tankerů.

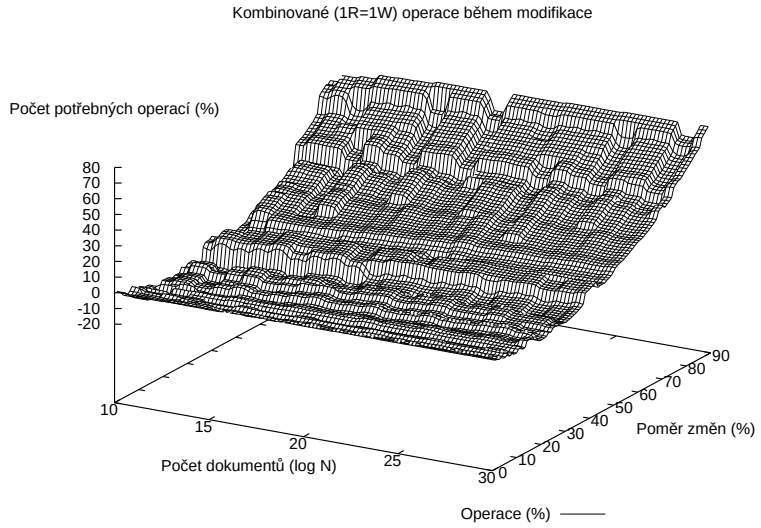
tohoto grafu. Osa y s označením Poměr změn % určuje parametr *procent* popsaný v předešlé sekci - tj. procentuální počet dokumentů, který bude změněn během jednoho kroku (tj. vyřazen a pak znovu vložen). Na ose z je výsledek podílu D/S, kde D je součet počtu operací *read* a *write* v aktualizaci naší dynamické struktury a S je počet operací *read* a *write*, které je potřeba při celkové přestavbě indexu (parametr *procent* je roven 100%). Na obrázku 3 jsou tytéž veličiny vyjádřeny jako vrstevnicový graf.

7.1 Počet tankerů

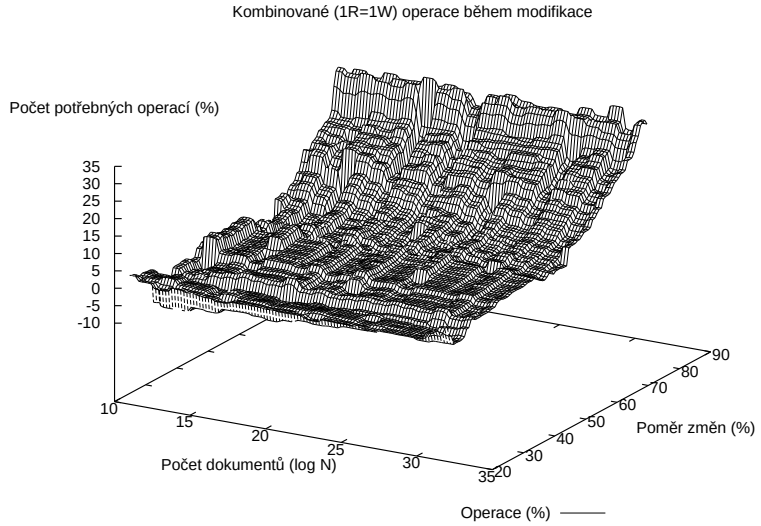
V závislosti na počtu tankerů, které obsahují dynamický invertovaný index, můžeme pozorovat snížený počet operací *read* a *write*, potřebných na provedení aktualizace tohoto indexu. Čím více tankerů použijeme, tím menší jsou dynamické struktury na nich a právě udržování těchto dynamických struktur je pak méně náročné na čtení nebo zápisy do paměti.

Obrázek 4 ukazuje rozdíl v počtu operací s použitím jednoho a 8 tankerů, obrázek 5 zobrazuje rozdíl mezi jedním a 32 tankery.

Počet IO operací u jednoho tankeru může být v porovnáním s 8mi tankery zvýšen u tankerů s menším počtem dokumentů o 30%, v průměru je tato hodnota skoro 20%. Pokud porovnáme počet operací *read* a *write* u 32 tankerů a jednoho tankeru, nastane u jednoho tankeru zvýšení až o přibližně 25%.



Obrázek 6. Srovnání metod u výběru dalšího tankeru během aktualizace: rozdíl operací *read+write* mezi metodou „následující“ a „minimální počet barelů“.



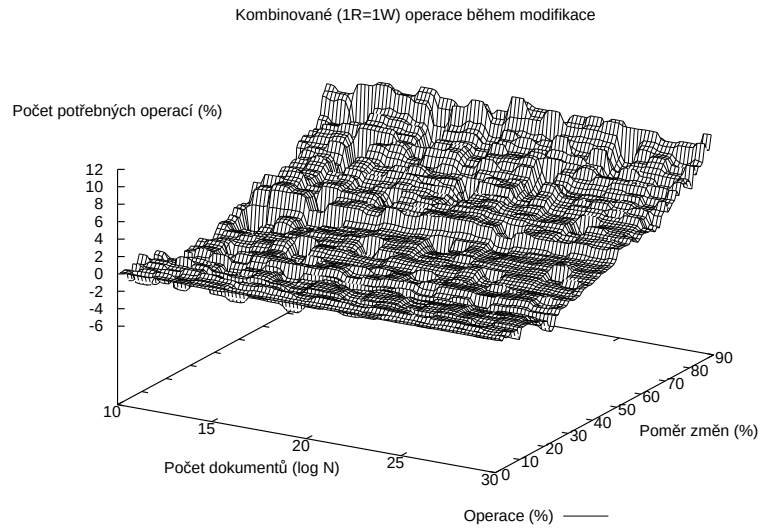
Obrázek 7. Srovnání metod u výběru dalšího tankeru během aktualizace: rozdíl operací read+write mezi metodou „následující“ a „minimální velikost barelů“.

7.2 Metoda výběru tankeru

V této části vzájemně srovnáme jednotlivé metody výběru tankeru u distribuovaného algoritmu modify (alg. 3).

Na obrázcích 6, 7 a 8 je zobrazen rozdíl mezi počtem všech IO operací s použitím metody „následující tanker“ a počtem IO operací metod: „minimální počet barelů“, „minimální velikost barelů“ a „náhodný výběr tankeru“.

Tyto grafy (obr. 6, 7 a 8) vznikly simulací počtu operací *read* a *write* s následujícími vstupními parametry: 2^{10} až 2^{30} dokumentů v barelu během inicializace, parametr *procent* je z rozmezí 1-90, počet tankerů je 8, počet iterací 1000, počet dotazů: 100 a jako odpověď na dotaz požadujeme 10 záznamů. Nejlepší výsledky dává metoda „minimální počet barelů“. Ve srovnání s ní dává metoda „následující“ až o 40% operací víc. Výrazné zlepšení je vidět i u metody „minimální velikost barelů“. Ta potřebuje se stejnými vstupními parametry v průměru o 20% operací méně než metoda



Obrázek 8. Srovnání metod u výběru dalšího tankeru během aktualizace: rozdíl operací read+write mezi metodou „následující“ a „náhodný výběr“.

„následující“. Výsledky metody „náhodný výběr“ jsou srovnatelné s metodou „následující“.

Z grafů můžeme vyčíst, že tyto rozdíly mezi metodami se projevují hlavně u větších počtů změn. Vidíme, že je lepší udržovat co nejmenší počet barelů v tankeru, protože restrukturalizace je pak rychlejší a potřebuje méně operací *read* a *write*. Tento fakt se dá vysvětlit tím, jak funguje algoritmus modify (viz. alg. 2), který prochází barely a tudíž jeho rychlost a počet operací záleží právě na počtu barelů.

8 Shrnutí

Jednou z nejdůležitějších vlastností indexu nad velkým počtem dokumentů je kromě rychlosti vyhledávání rychlost aktualizace. Použití dynamických struktur nám umožňuje z aktuálních dat generovat pomocnou datovou strukturu a tou pak nahradit původní invertovaný index, takže během aktualizace lze stále používat původní index.

Experimentálně jsme dokázali, že se zvyšováním počtu tankerů u distribuce invertovaného indexu se v řádech desítek procent v závislosti na použité metodě výběru tankeru, kam se budou přidávat aktualizované dokumenty, snižuje počet operací *read* a *write*, nutných pro aktualizaci dynamického invertovaného indexu. Toto zlepšení je u vybírání následujícího tankeru 10-15%, metoda „minimální počet barelů“ tato čísla ještě zvětšuje o další desítky procent (se vstupními parametry z příkladu uvedeném v předešlé sekci to je o 30% méně operací zápisu a čtení).

Reference

1. Adamic L.A. and Huberman B.A., Power-Law Distribution of the World Wide Web. Science, 2000
2. Apache, Jakarta project: Lucene. `http://jakarta.apache.org`
3. Arasu A., Novak J., Tomkins A., and Tomlin J., PageRank Computation and the Structure of the Web: Experiments and Algorithms. In The Eleventh International WWW Conference, New York, May 2002. ACM Press
4. Clarke C. and Cormack G., Dynamic Inverted Indexes for a Distributed Full-Text Retrieval System. TechRep MT-95-01, University of Waterloo, February 1995
5. EGOTHOR, JAVA IR system. `http://www.egothor.org/`
6. Fox E.A., Harman D.K., Baeza-Yates R., and Lee W.C., Inverted Files. In Information Retrieval: Data Structures and Algorithms, Prentice-Hall, 28–43
7. Galamboš L., Dynamic Inverted Index Maintenance. International Journal of Computer Science, 1, 2006
8. Galamboš L., Dynamization in IR Systems. Intelligent Information Systems, 2004, 297–310
9. Langville A.N., Meyer C.D., Deeper Inside PageRank. Internet Math., 1, 3, 2004, 335–380
10. Mehlhorn K., Data Structures and Algorithms 3. An EATCS Series, Springer, 3, 1984
11. Mehlhorn K., Lower Bounds on the Eficiency of Transforming Static Data Structures into Dynamic Data Structures. Math. Systems Theory, 15, 1981, 1–16
12. The size of the World Wide Web `http://www.pandia.com/sew/383-web-size.html`. 2007.
13. Yaghob J. and Zavoral F.: Semantic Web Infrastructure Using DataPile. The 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Itelligent Agent Technology.