Department of Computer Science
Faculty of Science
Pavol Jozef Šafárik University

Gabriela Andrejková
Rastislav Lencses (Eds)

ITAT 2002 Information Technologies - Applications and Theory

Workshop on Theory and Practice of Information Technologies, Proceedings
Malinô Brdo, Slovakia, September 2002

Proceedings

CONTENT

Invited Lectures:

Contributed Papers:

PREFACE

The 2nd Workshop ITAT 2002, was held in Malinô Brdo, Slovakia, September 12 - 15, 2002. This volume contains all contributed papers as well as 4 invited papers presented at the workshop.

The program committee of ITAT 2002 consisted of: Peter Vojtáš (chair), Gabriela Andrejková, Rastislav Lencses and Stanislav Krajči.

It is customary in speaking of a scientific conference to designated by an acronym of its full name. It is a measure of the standing of such an event how well the general public understands the meaning of the acronym. Acronyms like SOFSEM, MFCS, ICALP are very well known in the IT community. ITAT is not so lucky - it is an acronym of the one year old workshop and it is because of this that we start by providing some basic information. In addition to expanding the acronym to the full name **I**nformation **T**echnologies - **A**pplications and **T**heory it seems appropriate to provide more details. This is best done - especialy when strating a new tradition - by repeating answers of the five classic questions: WHY, WHO, HOW, WHERE and WHAT?

WHY did we organize ITAT 2001 and ITAT 2002? More precisely - what did we expect the workshop to look like? To be quite brief, we wanted to create an opportunity for researchers from the Slovakia, Czech Republic and Poland to meet in a nice place where they would be free from ordinary distractions and could discuss their work on - hopefully - the leading edge of computer science. This should promote closer contacts and cooperation in future.

WHO was invited to participate in the workshop ITAT 2002? Instead of precise formal selection criteria the organizers invited people with whom and whose work they were already acquainted. One might say that participants of this initial workshop were selected with a view to forming the program committee for future years.

HOW was the whole event organized? The program itself consisted of lectures and short contributions where the length was suited to the subject. The presentations took up mornings and evenings while the middle part of each day was devoted to getting acquainted with the beautiful surrounding mountains. The selection of papers for the present proceedings volume was made by the programming committee. One drawback of this mechanism was that it left no room for language editing; the language is therefore fully the responsibility of the authors. Thanks to the following referees: G. Andrejková, S. Krajči, R. Lencses, J. Vinař and P. Vojtáš.

WHERE did the workshop take place? The venue of the workshop was the Majeková Cottage, Malinô Brdo in the foothills of the Malá Fatra mountain range. The place was selected with a view to opportunities for rest and recreation and also because the weather there is usually nice in September. We hope that the participants will agree that it was a suitable choice for this as well as the coming workshops.

WHAT were the subjects dealt with at the workshop ITAT 2002? It is not suprising - considering the extreme variety of subjects addressed by theoretical computer science - that the subjects of presented papers fall into quite a lot of subject categories.

In formal languages and automata there were two invited papers: M. Platek and M. Procházka discussed the relaxations and restrictions of word order in dependency grammars and T. Holán and M. Plátek devoted his paper to DR-parsing and DR-analysis.

One invited paper given by J. Hric was devoted to the design patterns applied to functional programming and the invited paper given by J. Kohoutková described the use of hypertext presentation of relational database structures.

Six contributed papers were devoted to what is now generally known as neural networs and their applications. In contrast, nine papers dealt with subjects from the more general surroundings of information technology: information systems, database systems, compression methods, mobile agents a so on.

On the whole we feel that this "number one" workshop was a success and hope that it will lead to a whole series of annual ITAT workshops in the coming years.

Košice, 2002

G. Andrejková, R. Lencses

DR-parsing a DR-analýza

Tomáš Holan a Martin Plátek
Matematicko-fyzikální fakulta
Karlova Univerzita Praha, Česká Republika
e-mail: holan@ksvi.ms.mff.cuni.cz
e-mail: platek@ksi.ms.mff.cuni.cz

Abstrakt

Příspěvek se zabývá syntaktickou analýzou podle závislostních gramatik s rozvolněným slovosledem. Je předložen postup, který nejprve počítá tzv. DR-parsing, tj. množinu položek, která odpovídá dané větě a dané gramatice. Na základě DR-parsingu jsou postupně počítány jednotlivé syntaktické stromy, tzv. DR-stromy, které obsahují informace o historii vypouštění a přepisování. Jsou diskutovány význam a výpočetní složitost předloženého postupu.

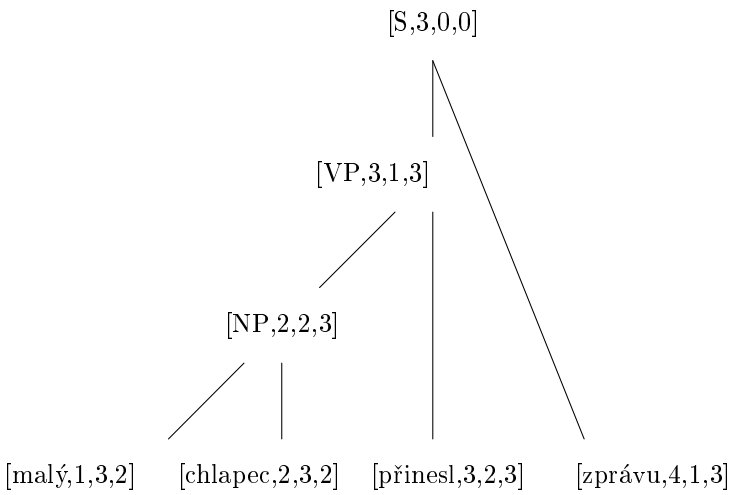
1 Úvod a základní pojmy

Tento příspěvek těží z disertační práce T.Holana [4] a navazuje na práce [3] a [5]. Vychází z pojmu DR-strom. Sám název DR-stromu vychází ze slov *delete* – *vypustit* a *rewrite* – *přepsat* a odráží představu práce automatu se dvěma hlavami nad danou větou.

Představujeme si, že věta je uložena na pásce (či spíše lineárním seznamu) a automat má dvě hlavy, které se mohou po této pásce pohybovat. První hlava je *vypouštěcí* — ta dokáže z pásky/věty úplně vypustit políčko/slovo, na kterém právě stojí. Druhá hlava je přepisovací — tato hlava může změnit obsah navštíveného políčka. Automat pracuje v krocích a v každém kroku rozmístí hlavy na pásce tak, aby stály na slovech, mezi nimiž má být vyznačena závislost — vypouštěcí hlava na závislém slově a přepisovací hlava na příslušném slově řídícím. Krok práce automatu se provede tím, že automat vypustí slovo pod vypouštěcí hlavou a případně přepíše slovo pod přepisovací hlavou.

V DR-stromu přepsání znázorňuje svislá hrana (přepsané slovo zůstává na svém místě) a vypuštění odpovídá šikmá hrana, shodná s hranou v odpovídajícím závislostním stromě.

Připustíme i druhou variantu kroku spočívající pouze v přepsání symbolu pod přepisovací hranou, aniž by nějaký symbol byl vypuštěn. Takovým případům odpovídají v DR-stromu uzly s jedinou dcerou. Tato dcera je k uzlu připojena svislou, přepisovací hranou.



Obrázek 1: DR-strom strom T_{11} nad větou „Malý chlapec přinesl zprávu“

Definice 1.1 (DR-strom) Nechť $w = a_1 \dots a_n$ je věta a A, K jsou konečné množiny symbolů (slov), kde $a_1, \dots a_n \in A$. Řekneme, že strom T je *DR-strom nad větou w (s terminály z A a kategoriemi z K)*, jestliže libovolný uzel stromu T je čtveřice tvaru $u = [A_u, i_u, j_u, d_u]$ s následujícími vlastnostmi:

- $A_u \in (A \cup K)$; A_u nazýváme *symbol* uzlu.
- $i_u \in \{1 \dots n\}$; i_u nazýváme *horizontální index* uzlu, udává vztah uzlu k pozici ve větě.
- pro každé $i \in \{1 \dots n\}$ existuje právě jeden list $u = [A_u, i_u, j_u, d_u]$ stromu T takový, že $i_u = i$ a $A_u = a_i$;
- j_u je přirozené číslo nebo 0; $j_u = 0$ právě když $d_u = 0$. V tom případě je uzel $[A_u, i_u, 0, 0]$ kořenem stromu T. Číslo j_u nazýváme *vertikální index* uzlu, udává vzdálenost uzlu od kořene stromu měřenou v počtu hran.
- $d_u \in \{0 \dots n\}$;

  d_u nazýváme *dominační index* uzlu u, udává horizontální index bezprostředně nadřízeného uzlu nebo v případě $d_u = 0$ jeho (nadřízeného uzlu) absenci.
- Je-li $u = [A_u, i_u, j_u, d_u]$ a $d_u \neq 0$, potom existuje právě jeden uzel v tvaru $v = [A_v, d_u, j_u - 1, d_v]$; dvojice (u, v) tvoří hranu stromu T, orientovanou od listu ke kořenu.

  Je-li $d_u = i_u$, hovoříme o hraně *svislé (přepisovací)*.

  Je-li $d_u \neq i_u$, hovoříme o hraně *šikmé (vypouštěcí)*. V tomto případě existuje také (pro nějaký symbol A_x) uzel x tvaru $x = [A_x, d_u, j_u, d_u]$ stromu T (vede-li do uzlu v šikmá hrana, vede do něj právě jedna svislá hrana).

Je-li $d_u > i_u$, hovoříme o R-hraně, je-li $d_u < i_u$, hovoříme o L-hraně.

- je-li $u = [A_u, i_u, j_u, d_u]$ uzel, potom existuje nejvýše jeden uzel v tvaru $v = [A_v, i_v, j_u + 1, i_u]$, takový, že $i_v \neq i_u$ (do každého uzlu vede nejvýše jedna šikmá hrana).
- jsou-li $u = [A_u, i, j_u, d_u]$ a $v = [A_v, i, j_v, d_v]$ uzly stromu T (se stejným horizontálním indexem), potom platí
  - $j_u = j_v \Rightarrow u = v$
    hodnota horizontálního indexu spolu s hodnotou vertikálního indexu jednoznačně určují uzel stromu T.
  - $d_u \neq i \Rightarrow j_v \geq j_u$, $d_v \neq i \Rightarrow j_v \leq j_u$
    pro každé i existuje nejvýše jeden uzel s horizontálním indexem i, z něhož vede šikmá hrana. Je to uzel, který má ze všech uzlů s horizontálním indexem i nejmenší hodnotu vertikálního indexu (nejmenší vzdálenost od kořene).
  - $j_v > j_u \Rightarrow$ existuje uzel $v' = [A_{v'}, i, j_v - 1, i]$.

Definice 1.2 (Pokrytí uzlu DR-stromu) Nechť T je DR-strom a nechť u je uzel stromu T. $Cov(u, T)$ označíme množinu horizontálních indexů všech uzlů stromu T, ze kterých vede cesta do u. Uvažujeme i prázdnou cestu a tedy $Cov(u,T)$ vždy obsahuje také horizontální index uzlu u. Řekneme, že $Cov(u, T)$ je *pokrytí uzlu u (podle stromu T)*.

Definice 1.3 (Díra (v pokrytí) DR-stromu) Nechť T je DR-strom nad větou $w = a_1 \ldots a_n$, nechť u je uzel stromu T a nechť $Cov(u, T) = \{i_1, i_2, \ldots, i_m\}$, kde $i_1 < i_2 < \ldots < i_{m-1} < i_m$. Řekneme, že dvojice (i_j, i_{j+1}) tvoří *díru v* $Cov(u, T)$, jestliže $1 \leq j < m$, a zároveň $i_{j+1} - i_j > 1$.

Budeme říkat, že T je DR-projektivní, pokud žádný z jeho uzlů nemá v pokrytí díru. V opačném případě říkáme, že T není DR-projektivní.

2 D-gramatiky a syntaktická analýza

Definice 2.1 (D-gramatika) *D-gramatika (Dependency grammar)* je čtveřice $G = (A, N, S, P)$, kde A je konečná množina terminálů, N je konečná množina neterminálů,

$S \subseteq (A \cup N)$ je množina počátečních symbolů a P je množina přepisovacích pravidel dvou typů:

$A \rightarrow_X BC$, kde $A, B, C \in (A \cup N)$, X je uvedeno jako index pravidla, $X \in \{L, R\}$.

$A \rightarrow B$, kde $A, B \in (A \cup N)$.

Písmena L resp. R v indexu pravidla znamenají, že první resp. druhý symbol pravé strany pravidla je *řídící*; druhý resp. první symbol pravé strany pravidla je potom *závislý*. Má-li pravidlo na pravé straně pouze jeden symbol, považujeme tento symbol za *řídící*.

Pravidlo má následující význam (pro redukci): závislý symbol bude vymazán (obsahuje-li pravá strana pravidla nějaký závislý symbol) a řídící symbol bude přepsán (nahrazen) symbolem z levé strany pravidla. Vezměme pravidla tvaru

(1a): $A \to_L BC$, (1b): $A \to_R BC$.

Redukci řetězu w podle pravidla (1a) lze uplatnit, když pro libovolný souvýskyt symbolů B, C ve w, symbol B předchází (ne nutně bezprostředně) symbol C. Redukce podle (1a) znamená, že symbol B je přepsán symbolem A a symbol C je z řetězu vypuštěn.

Za stejných předpokladů lze redukovat w podle pravidla (1b). Redukce podle (1b) znamená přepsat symbol C na A a symbol B z w vypustit.

Definice 2.2 (Gramatika rozpoznává větu) D-gramatika $G = (A, N, S, P)$ rozpoznává větu $w \in A^+$, lze-li větu w opakovanou redukcí pravidly gramatiky G přepsat na větu sestávající z jediného symbolu patřícího do množiny S počátečních symbolů gramatiky G.

Definice 2.3 (Gramatika rozpoznává jazyk) Řekneme, že D-gramatika G rozpoznává jazyk L, obsahuje-li L právě všechny věty rozpoznávané gramatikou G. Jazyk rozpoznávaný gramatikou G budeme značit $L(G)$.

Definice 2.4 (DR-strom podle D-gramatiky) Nechť w je věta a T je DR-strom nad větou w. Dále nechť G je D-gramatika.

Řekneme, že T je *DR-strom podle D-gramatiky G nad větou w*, jestliže pro každý uzel u tvaru $u = [A, i, j_u, d_u]$ stromu T platí:

1. má-li u jedinou dceru $v = [B, j, j_v, d_v]$

   (z definice DR-stromu plyne, že potom musí platit $d_v = i = j$, $j_v = j_u + 1$),

   potom gramatika G obsahuje pravidlo $A \to B$

2. má-li u dvě dcery $v = [B, j, j_v, d_v]$ a $w = [C, k, j_w, d_w]$ takové, že $j < k$

   (z definice DR-stromu potom plyne $j_v = j_w = j_u + 1$, $d_v = d_w = i$ a buď $i = j$, nebo $i = k$),

   potom gramatika G obsahuje pravidlo $A \to_x BC$, kde $x = L$, pokud $i = j$, a $x = R$, pokud $i = k$.

3. je-li u kořen stromu T

   (z definice DR-stromu potom plyne $j_u = 0$, $d_u = 0$),

   potom symbol A je prvkem množiny počátečních symbolů gramatiky G.

Poznámka 2.5 *D-gramatika G rozpoznává větu w právě tehdy, když existuje DR-strom podle D-gramatiky G nad větou w. Této ekvivalence budeme dále využívat.*

Definice 2.6 (DR-analýza) Nechť $G = (A, N, S, P)$ je D-gramatika. Řekneme, že $DR(G)$ je *DR-analýzou podle G*, jestliže platí $DR(G) = \{T |\ ex.\ w \in A^+, T\ je\ DR - strom\ nad\ w\ podle\ G\}$.

Pro každé $w \in A^+$ vezmeme $DR(w, G) = \{T |\ T\ je\ DR - strom\ nad\ w\ podle\ G\}$. Budeme říkat, že $DR(w, G)$ je DR-analýzou w podle G.

2.1 Parsing, složitost

V této partii se budeme věnovat výpočtu DR-analýzy podle dané D-gramatiky a podle jistých topologických omezení. Zavedeme pojem *parsing* a budeme se zabývat složitostí jeho výpočtu. Dále ukážeme postup, jak pomocí parsingu postupně nacházet DR-stromy patřící do hledané DR-analýzy. Omezíme se pouze na D-gramatiky *neobsahující unární pravidla*. Uvedené definice, tvrzení a postupy by bylo možno s určitou modifikací vyslovit i pro gramatiky s unárními pravidly, k našemu cíli, kterým je výpočet analýz vět přirozeného jazyka, je však nepotřebujeme a takové definice by byly méně přehledné. V této souvislosti budeme v následujícím textu hovořit o DR-stromech resp. DR-analýzách bez unárních uzlů a o D-gramatikách bez unárních pravidel.

Označení 2.7 Zápisem $\langle i, j\rangle$, kde $i \leq j$, budeme označovat množinu $\{i, i+1, \ldots, j\}$.

Definice 2.8 (Položka) *Položkou* nazveme $(2k+2)$-tici $P = [A, h, i_1, \ldots, i_{2k}]$, kde $i_1 \leq i_2 < i_3 \leq i_4 \ldots i_{2k-1} \leq i_{2k}$ a $h \in \langle i_1, i_2\rangle \cup \langle i_3, i_4\rangle \cup \ldots \cup \langle i_{2k-1}, i_{2k}\rangle$.

A budeme nazývat *symbol položky*, h *horizontální index položky*, množinu $\langle i_1, i_2\rangle \cup \langle i_3, i_4\rangle \cup \ldots \cup \langle i_{2k-1}, i_{2k}\rangle$ *pokrytí položky*. Pokrytí položky P budeme zkráceně zapisovat jako $Cov(P)$. O hodnotě $2k$ budeme hovořit jako o *velikosti položky*.

Definice 2.9 (Položka DR-stromu) Nechť T je DR-strom bez unárních uzlů.

Řekneme, že položka $P = [A_P, h_P, i_1, \ldots, i_{2k}]$ je *položkou DR-stromu* T, pokud DR-strom T obsahuje uzel $u = [A, h, v, d]$ takový, že $A_P = A$, $h_P = h$ a $Cov(P) = Cov(u, T)$.

Tvrzení 2.10 *DR strom bez unárních uzlů je určen množinou svých položek.*

Důkaz: Nechť je dána množina položek nějakého DR-stromu T nad větou $w = a_1 \ldots a_n$. Ukážeme postup, jímž lze z takové množiny určit všechny uzly a hrany DR-stromu T.

Začneme položkou s největším pokrytím. Existuje jediná taková položka a tato položka bude mít tvar $P = [A_P, h_P, 1, n]$, kde symbol A_P musí patřit mezi počáteční symboly gramatiky a číslo n odpovídá délce věty w (toto číslo bychom v případě potřeby mohli určit z celkového počtu položek $2n-1$ pomocí známého vztahu mezi počtem uzlů a počtem listů binárního stromu).

Této položce bude odpovídat uzel tvaru $[A_P, h_P, 0, 0]$ — kořen stromu T.

Nyní popíšeme odvozovací krok:

Předpokládáme, že k položce P jsme již určili odpovídající uzel stromu u_P.

Je-li $|Cov(P)| > 1$, potom najdeme dvě položky $Q = [A_Q, h_Q, \ldots]$ a $R = [A_R, h_R, \ldots]$ různé od P takové, že platí $Cov(Q) \cup Cov(R) = Cov(P)$. V množině položek stromu T musí existovat právě jedna taková dvojice položek a navíc pro ni musí platit

$$Cov(Q) \cap Cov(R) = \emptyset.$$

Každá z položek Q a R bude odpovídat jednomu uzlu stromu,

$$u_Q = [A_Q, h_Q, v_Q, d_Q] \text{ a } u_R = [A_R, h_R, v_R, d_R].$$

Symboly a horizontální indexy těchto uzlů budou převzaty z příslušných položek. Dominační indexy obou uzlů u_Q a u_R budou rovny h_P, vertikální indexy budou o 1 větší než vertikální index uzlu u_P, tedy $v_P + 1$.

Musí platit $h_Q = h_P$ nebo $h_R = h_P$. V prvním případě povede z u_Q do u_P vertikální hrana a z u_R do u_P šikmá hrana, v druhém případě naopak.

Na obě položky (Q a R) a oba uzly (u_Q a u_R) opět aplikujeme právě popsaný odvozovací krok.

Popsaný odvozovací krok pro každou položku a jí odpovídající uzel DR-stromu T určí celý podstrom tohoto uzlu. Provedeme-li tento postup pro položku odpovídající kořeni stromu, získáme celý strom T. Q.E.D.

Poznámka 2.11 *Opačné tvrzení — že DR strom bez unárních uzlů určuje svou množinu položek — je triviálním důsledkem definice položky DR-stromu. Vidíme vzájemně jednoznačný vztah mezi DR-stromem bez unárních uzlů a množinou jeho položek.*

Poznámka 2.12 *Pokud by strom T obsahoval unární uzly, existovalo by v množině položek stromu T více položek se stejným pokrytím i horizontálními indexy, takže uvedená tvrzení by nebyla platná.*

Definice 2.13 (Položka podle gramatiky nad větou) Nechť G je D-gramatika bez unárních pravidel a nechť $w = a_1 \dots a_n$ je věta.

Řekneme, že položka $P = [A_P, h_P, i_1, \dots, i_{2k}]$ je *položkou podle gramatiky G nad větou w*, je-li P položkou nějakého DR-stromu podle gramatiky G nad větou w.

Poznámka 2.14 *DR-strom podle gramatiky bez unárních pravidel je DR-stromem bez unárních uzlů.*

Definice 2.15 (Položka analýzy nad větou) Nechť $DR(G)$ je DR-analýza podle závislostní gramatiky bez unárních pravidel G a nechť $w = a_1 \dots a_n$ je věta.

Řekneme, že položka $P = [A_P, h_P, i_1, \dots, i_{2k}]$ je *položkou analýzy DR nad větou w*, je-li *položkou nějakého DR-stromu $T \in DR(w, G)$*.

Nyní zavedeme pojem parsing odpovídající výsledné množině syntaktické analýzy zdola.

Definice 2.16 (DR-parsing) Nechť G je D-gramatika bez unárních pravidel a $w = a_1 \dots a_n$ je věta. *DR-parsingem podle gramatiky G nad větou w* nazveme nejmenší množinu M položek takovou, že

- $[a_i, i, i, i] \in M$
  — tj. pro každé slovo věty w obsahuje M položku DR-stromu nad touto větou odpovídající jeho listu

- jsou-li $P, Q \in M$, $P = [B, i_P, i_1, \ldots, i_{2k_P}]$, $Q = [C, i_Q, j_1, \ldots, j_{2k_Q}]$ dvě položky takové, že $Cov(P) \cap Cov(Q) = \emptyset$, platí-li $i_P < i_Q$ a gramatika G obsahuje pravidlo $A \rightarrow_L BC$ resp. $A \rightarrow_R BC$,

  potom M obsahuje i položku $R = [A, i_R, l_1, \ldots, l_{2k_R}]$, kde $Cov(R) = Cov(P) \cup Cov(Q)$ a $i_R = i_P$ resp. $i_R = i_Q$.

  Říkáme, že položky P a Q tvoří *rozklad položky* R v parsingu M.

Prvky M budeme nazývat *položky parsingu M.*

Na parsing můžeme uplatnit následující omezení (neprojektivity).

Definice 2.17 Nechť G je D-gramatika bez unárních pravidel, $w = a_1 \ldots a_n$ je věta a M je parsing podle gramatiky G nad větou w. Dále nechť $d \geq 0$ je celé číslo. Množinu

$$P = \{[A_P, i_P, i_1, \ldots, i_{2k_P}] | P \subset M,\ k_P \leq d + 1\}$$

nazveme *parsing s počtem děr nepřevyšujícím d.*

Tvrzení 2.18 *Nechť $G = (N, T, St, Pr)$ je D-gramatika bez unárních pravidel, $w = a_1 \ldots a_n$ věta, $d \geq 0$ celé číslo, a M je parsing podle gramatiky G nad větou w s počtem děr, který nepřevyšuje d. Pak M má polynomiální velikost a lze jej určit s polynomiální složitostí, časovou i prostorovou vzhledem k délce věty n i vzhledem k velikosti gramatiky G.*

Důkaz: Z předpokladů plyne, že velikosti položek nejsou větší než $2d+2$. Položek s touto vlastností jistě nemůže existovat více než $|N \cup T| \times n \times n^{2d+2}$, tedy polynomiální počet vzhledem k délce věty i velikosti gramatiky.

Složitost výpočtu je shora omezena například složitostí jednoduchého algoritmu, který vychází od n položek $Pi = [a_i, i, i, i]$ odpovídajících listům DR-stromů a postupně pro všechny dvojice položek zkouší, lze-li jejich spojením na základě jejich symbolů, horizontálních indexů, pravidel gramatiky, pokrytí a předepsané restrikce počtu děr v pokrytí odvodit jinou položku. Pokud ano, zkontroluje ještě, zda se odvozená položka neshoduje s některou položkou odvozenou již dříve.

Je-li cílový počet položek omezen konstantou $C = |N \cup T| \times n^{2d+2}$, nebude tento proces trvat déle než C^3 kroků, kde za elementární kroky považujeme zjištění, zda lze ze dvou daných položek odvodit další položku a test shodnosti dvou položek — a má tedy také polynomiální složitost vzhledem k délce věty n i velikosti gramatiky G. Q.E.D.

Tvrzení 2.19 *Nechť G je D-gramatika bez unárních pravidel, nechť $w = a_1 \ldots a_n$ je věta, nechť M je DR-parsing podle gramatiky G nad větou w. Pak M obsahuje všechny položky DR-analýzy podle gramatiky G nad větou w.*

Důkaz: Nechť $DR(G)$ je DR-analýza podle G. Zvolme libovolný $T \in DR(w, G)$, ukážeme, že M obsahuje všechny položky stromu T.

Postupujme indukcí podle velikosti pokrytí uzlů stromu T.

Položky s jednoprvkovým pokrytím odpovídají listům stromu a musí mít tvar $P_i = [a_i, i, i, i]$. Takové položky parsing podle gramatiky G nad větou w obsahuje z definice.

Nyní předpokládejme, že již víme, že M obsahuje všechny položky stromu T s pokrytím velikosti nejvýše $k-1$, kde $k > 1$.

Uzly s pokrytím velikosti k nejsou listy (protože $k > 1$). Takový uzel u musí tedy mít dvě dcery (protože G neobsahuje unární pravidla), pokrytí těchto dcer jsou neprázdná a disjunktní, tedy menší než k, tedy podle indukčního předpokladu již víme, že parsing M obsahuje položky odpovídající těmto dcerám. Podle definice parsingu potom parsing musí obsahovat i položku odpovídající uzlu u. Q.E.D.

Poznámka 2.20 *Viděli jsme, že množina položek DR-stromu jednoznačně určuje DR-strom. Na druhé straně daný parsing sice obsahuje sjednocení množin položek všech stromů určité DR-analýzy nad určitou větou, ale toto sjednocení, bez znalosti gramatiky, nemusí postačovat k rekonstrukci původních množin položek jednotlivých stromů.*

Označení 2.21 Nechť G je D-gramatika bez unárních pravidel, w je věta a $M = \{P_1, \ldots, P_n\}$ je DR-parsing podle gramatiky G nad w. Předpokládejme zároveň, že jednotlivé položky parsingu M jsou pevně očíslovány. Řekneme, že položka P_i je *odvozená*, pokud platí $|Cov(P_i)| > 1$. S jakoukoliv odvozenou položkou P_i musí M obsahovat také jednu nebo více dvojic položek (P_j, P_k) tvořících *rozklad položky P_i v parsingu M.* Rozklady téže položky lze uspořádat lexikograficky podle velikostí indexů, první z nich označíme za *první rozklad položky P_i* a ke každému rozkladu kromě *posledního* dokážeme nalézt *následující rozklad.*

Označení 2.22 Nechť G je D-gramatika bez unárních pravidel, w je věta a $M = \{P_1, \ldots, P_n\}$ je parsing podle gramatiky G nad w a $P \in M$ je položka. Definujme množinu položek $M(P)$ jako nejmenší množinu takovou, která obsahuje položku P a která zároveň s každou odvozenou položkou obsahuje i položky tvořící její první rozklad v parsingu M. DR-strom, který k této množině položek sestrojíme postupem popsaným v důkazu Tvrzení 2.10 nazveme *první podstrom položky P v parsingu M.*

Tvrzení 2.23 *Nechť $w = a_1 \ldots a_n$ je věta, G je D-gramatika bez unárních pravidel, $M = \{P_1, P_2, \ldots, P_m\}$ je parsing podle gramatiky G nad větou w a nechť $DR(G)$ je DR-analýza podle G. Potom lze polynomiálním počtem kroků vzhledem k m a n určit, že množina $DR(w, G)$ je prázdná nebo nalézt první DR-strom $T \in DR(w, G)$.*

Důkaz: Určení, zda množina $DR(w, G)$ je prázdná, je ekvivalentní zjištění, zda parsing M obsahuje alespoň jednu položku P_{root} takovou, že $Cov(P_{root}) = \{1, \ldots, n\}$ a její symbol je jedním z počátečních symbolů. Pokud ano, hledaný DR-strom bude *první podstrom položky P_{root}.* Q.E.D.

Tvrzení 2.24 *Nechť $w = a_1 \ldots a_n$ je věta, G je D-gramatika bez unárních pravidel, $M = \{P_1, P_2, \ldots, P_m\}$ je parsing podle gramatiky G nad větou w, $DR(G)$ je DR-analýza podle G a $T_s \in DR(w, G)$ je DR-strom nad w podle analýzy DR. Pak existuje uspořádání U na množině $DR(w, G)$ a algoritmus, který po polynomiálně omezeném počtu*

kroků vzhledem k hodnotám m a n vydá další strom T_{s+1} z $DR(w,G)$ podle uspořádání U nebo zjistí, že takový strom již neexistuje.

Důkaz: Ukážeme způsob, jak nalézt další strom; uspořádání U bude dáno pořadím, v němž algoritmus nachází jednotlivé stromy.

Další strom nalezneme následujícím algoritmem:

1. Uzly stromu T_s očíslujeme podle jejich pozice ve stromě průchodem pre-order, tj. vždy nejprve matky, potom celý levý podstrom a potom celý pravý podstrom. Jako P_{root} označíme položku odpovídající kořeni T_s.

2. Procházíme uzly, které nejsou listy, v klesajícím pořadí podle očíslování, pro každý uzel určíme odpovídající položku $P_i \in M$, podle dcer tohoto uzlu a jim odpovídajících položek určíme (P_j, P_k) rozklad položky P_i a zkoumáme, zda pro tento uzel a tento rozklad existuje následující rozklad podle parsingu M.

3. Nalezneme-li uzel u, kde k odpovídající položce P_i a jejímu rozkladu (P_j, P_k) existuje následující rozklad $(P_{j'}, P_{k'})$, potom

   (a) změníme podstrom uzlu u tak, že jeho dcery budou uzly odpovídající položkám $P_{j'}$ a $P_{k'}$ a jejich podstromy budou první podstromy položek $P_{j'}$ a $P_{k'}$.

   (b) Dále pro každý uzel v v upraveném stromě, který ve svém levém podstromu obsahuje uzel u, určíme použitý rozklad (P_{j_u}, P_{k_u}) a jeho pravý podstrom nahradíme prvním podstromem položky P_{k_u}.

4. Nenalezneme-li žádný uzel u, pro který by existoval další rozklad, vyčerpali jsme všechny stromy s kořenem odpovídajícím položce P_{root}. Hledáme tedy další položku pokrývající celou větu a pro ni určíme první podstrom.

5. Neexistuje-li již žádná další položka pokrývající celou větu, neexistuje další DR-strom $T_{s+1} \in DR(w,G)$.

Považujme zjištění, zda lze ze dvou daných položek odvodit třetí danou položku, za elementární krok. Algoritmus prochází nejvýše $(2n-1)$ uzlů, pro každý z nich prohlíží nejvýše $m(m-1)/2$ dvojic položek, aby zjistil, zda existuje následující rozklad příslušného uzlu, tedy $(2n-1)m(m-1)/2$ kroků. Nejvýše stejně dlouho trvá nalezení prvního rozkladu všech položek v novém podstromu a vytvoření prvních podstromů všech pravých dcer matky uzlu u (protože pro každý uzel bude algoritmus první podstrom vytvářet nejvýše jednou).

Složitost algoritmu tedy není řádově větší než nm^2. Q.E.D.

2.2 Závěrečná poznámka

Předchozí dvě tvrzení říkají, že ačkoliv složitost syntaktické analýzy i v případě omezené DR-neprojektivity může být více než exponenciální vzhledem k délce věty, lze tuto analýzu provádět tak, že v polynomiálně omezeném čase rozhodneme, zda existuje nějaký strom nad větou podle dané analýzy a potom postupně, opět v polynomiálně omezených časech, hledáme po jednom další stromy až do celkového počtu. Podobné úvahy, jaké zde byly popsány, využíval první autor při iplementaci prostředí pro vývoj syntaktických analyzátorů, viz [4].

2.3 Poděkování

Děkujeme za technickou pomoc Milanu Fučíkovi. Tento příspěvek je podporován z prostředků grantu od Grantové Agentury České Republiky č. 201/02/1456.

Odkazy

[1] M.I.Beleckij: Beskontekstnye i dominacionnye grammatiki i svjazannye s nimi algoritmičeskije problemy, Kibernetika, 1967, No 4,pp. 90-97

[2] K. Sikkel.: *Parsing Schemata - a framework for specification and analysis of parsing algorithms.,* Texts in Theoretical Computer Science - An EATCS Series. ISBN 3-540-61650-0, Springer-Verlag, Berlin/Heidelberg/New York.

[3] Tomáš Holan, Vladislav Kuboň, Karel Oliva, Martin Plátek,: *Two Useful Measures of Word Order Complexity,* In: Processing of Dependency-based Grammars: Proceedings of the Workshop, pp. 21-28, ACL, Montreal, 1998

[4] Tomáš Holan: *Nástroj pro vývoj závislostních analyzátorů přirozených jazyků s volným slovosledem,* PhD thesis, Charles University, Prague, 2001

[5] Plátek, M., Holan,T., Kuboň, V., Oliva, K.: *Word-Order Relaxation and Restrictions within a Dependency Grammar.* In: ITAT'2001: Information Technologies - Applications and Theory, Workshop on Theory and Practice of Informations Technologies, Zuberec, Slovakia, 2001, pp. 5 – 25

Design Patterns applied to Functional Programming

Jan Hric
Dept. of Theoretical Computer Science
Charles University, Faculty of Mathematics and Physics
Malostranské náměstí 25
118 00 Praha 1
jan.hric@mff.cuni.cz

October 16, 2002

Abstract

Design patterns are well-known technique used in a development of object-oriented systems for reusing solutions of typical problems. In the paper we compare design patterns with development techniques used in functional programming. We describe ideas of some new patterns as well as an analogy of some known OO patterns in functional programming.

1 Introduction

Design patterns [GHJ], [BMR] are used as standard solutions of typical problems of an object-oriented design. Some problems are language independent and so they are relevant also in a different context of functional programming (FP). We took problems and their corresponding patterns from literature and look for corresponding patterns in functional programming.

Declarative programming provides support which is not available in object-oriented languages. Polymorphic functions and data structures and functional parameters are basic examples of such a support in both functional and logic languages. We chose the functional language Haskell for this paper because it has some other features compared to the logic language Mercury. We suppose that patterns can be transferred also to logic programming.

The paper concentrates on transferring patterns. Knowledge of particular design patterns and functional programming is an advantage during reading.

As noted in [Pr], the classification of patterns is of a little help for program developers. They need solutions for their problems. Thus we describe patterns in a new context and do not analyse their relations.

1.1 Level of patterns

High-level architecture of a program is independent on an implementation language. Thus high-level architectural patterns [BMR] can be used analogically in different languages (the patterns Blackboard, Microkernel). We are mostly interested in a lower level of patterns.

Low-level patterns, called programming idioms, are usually language specific. Therefore they can not be used as a source for a transfer. Moreover, some patterns in one language disappear in another language due to different possibilities of languages.

1.2 Comparison of OOP and functional programming

An object is the core entity in OOP. An object has a state and a composed interface and it associates data and functions. In functional programming there is no such a universal entity. So various means are used to describe design patterns, especially data structures, higher-order functions, type classes and modules. There is also a difference in a granularity of objects and data structures. One data structure usually corresponds to many interconnected objects.

The nonexistence of a state means that many patterns devoted to a processing or synchronization of states of one or more objects are not usable. The architecture of such programs is different and a problem formulated in the context of OOP disappears or must be reformulated for other entities than objects. One basic characteristic of pure functional languages is a referential transparency. Thus each function must get all data which are needed for computing of an output value. Therefore a (representation of a) state must be given in input data.

A direct reformulation of patterns in a functional programming sometimes gives too specific solutions. Such solutions can be generalised for other data structures or types.

1.3 Hook and template in funcional programming

The idea behind many patterns is a decoupling. A hook part which should be flexible is hidden from the rest of the system and is called only through a template part. Possibilities of an actual implementation follows.

We do not have objects and their virtual methods in functional programming as an universal way of late binding. Patterns must be implemented using other low-level principles.

First possibility is to use functional parameters in higher-order functions. An appropriate code for a hook is explicitly given as a parameter. This method is probably the most universal one, as we can pass a tuple of data structures and function. Second possibility is to use parametric polymorphism. Data structures and functions can be polymorphic and thus independent on a particular type

of a parameter. Third possibility is to use type classes and allow to select the particular operations during a (re)compilation. The last possibility is to use modules and abstract data types. Cooperating functions of a pattern are grouped together in the last two cases. Also some special features as extensible records can be suitable for a pattern description.

More OOP patterns will be reduced to the same or similar FP pattern. This is possible, as we can look at some patterns from different points of view.

The same program can be written using different programming styles. There are for instance continuation passing style, monadic style or compositional programming using combinators in functional programming. Such styles can use specific low-level patterns, which are not analysed here. Styles correspond to frameworks in some sense. There are special features and usual ways of combining parts in both cases.

Patterns can be aimed also at special domains. Hot spot identification combined with essential construction principles is suggested for a development of domain-specific patterns [Pr]. Combinators for specific domain are such (low-lever) patterns in functional programming.

2 Patterns

We take patterns from [GHJ] and look for corresponding ideas in a functional programming. Patterns described there are more general and less object-oriented in comparison with [BMR]. Some patterns solve problems too specific for object-oriented programming, especially questions of a state manipulation and synchronization in a wide sense.

The first three subsections describe structural, behavioral and creational patterns according to [GHJ]. For each pattern we describe an original central idea [HDP] in an object-oriented programming and then we start to analyse its functional twins.

2.1 Structural patterns

2.1.1 Adapter

The adapter pattern converts an interface of a class into another interface expected by a client.

This idea can be used for functions and for data structures. In the first case the interface of a function is its type. Each use of the pattern means to write an adapter function, which transforms the original adaptee function to a new one. The functions *flip*, *curry* and *uncurry* are examples of the pattern. Instead of the original incompatible function f we call the compatible function (*adapter* f) in the same context.

```
flip       :: (a->b->c) -> b -> a -> c
flip f x y  = f y x
```

```
curry      :: ((a,b) -> c) -> (a->b->c)
curry f x y = f (x,y)
uncurry    :: (a->b->c) -> ((a,b) -> c)
uncurry f p = f (fst p) (snd p)
```

In the second case of data structures, the adapter is a function which converts a data structure to another structure.

Other usage of adapter pattern change of data structure to some standard form. As an example, a linearization to a list is possible for any container structure with elements of a single type. So elements from any container can be processed using the same manner. One type of container are of general n-ary tries with inner nodes of some type a. Their definition and the function $listify$ for their linearization follows.

```
data Tree a = Node a [Tree a]
listifyNT (Node x Sub) =
  a: concat(map listifyNT Sub) where
        concat [] = []
        concat (xs:xss) = xs++concat xss
```

The auxiliary function *concat* concatenates all results from subtrees to a single list. The order of elements in the result is specified by the implementation of $listify$.

2.1.2 Composite

The composite pattern composes objects into tree structures to represent part-whole hierarchies. The pattern corresponds to a definition of a new type constructor $Tree$. Composite structures use a type $Tree\ a$ instead of a. Trees can be binary, n-ary etc.

The n-ary trees were introduced in the previous pattern Adapter and their structure is suitable for Composite pattern. The simpliest single node tree $Node\ x\ []$ can be used instead of an element x. If in an application data of the composite type $Tree\ a$ is used instead of data of a type a then also the function working with the data must be changed. All calls to some function f working with the type a are changed to the call $mapTree\ f$, where the new function $mapTree$ is

```
mapNT f (Node a Subtrees) =
        Node (f a) (map (mapNT f) Subtrees)  where
  map f []     = []
  map f (x:xs) = f x : map f xs
```

The function does not change the structure of a tree and performs the operation f on all elements. Another functions for working with the tree structure must be also given. The patterns visitor can be used for them.

2.1.3 Decorator

A decorator enable to attach additional responsibilities to an object dynamically.

A possible reformulation in a functional programming is that we want to extend the behaviour of a function for a given data structure. A simple approach is to give

a function higher-order parameter f, which describes how the data structure should be processed. This solution has a disadvantage that the parameter describes the whole processing but is not extensible. Using the idea of continuations, the extensible solution is to use the parameter f with a hole – another functional parameter g. The latter function g describes only the additional processing and is substituted to the identity function *id* when nothing new is needed.

In the following the examples show how to create a decorated structure and how to create a decorated function.

```
decorated_x = decorate2 (decorate1 x)

extended_processing =
        \x -> post_decorator (
                basic_decorator (
                  pre_decorator x))

decorator f_decor x = f_decor x
```

The use of the decorator pattern is the call *decorator id x* on the places where the value x is used. The "empty" decorator *id* can be then changed to appropriate processing as *basic_decorator* from example or (*post_decorator . id . pre_decorator*). The data x can be changed to *decorated_x* in a similar manner. Decorator functions can have its own parameters.

One note concerning a type of results. We suppose the same type of results for calls with an additional functionality and without it. So the type of results must be extensible and we must understand the semantics of results if we want to use them. Extensible structures in this sense are data structures as lists, trees etc. The semantics of old and new functionality can be captured in a lookup list. Each new functionality adds one (or more) key - value pair to the result. Other means for extensible data structures as extensible records (TREX) in Hugs implementation [Hs] are available.

We need not understand the results when they are not processed or are processed uniformly. The results given directly to output are an example of the former case.

2.1.4 Proxy

The proxy patterns provide a surrogate or placeholder for another object to control access to it. There is a more specific pattern concerning data structures in functional programming. Instead of using data directly we use the name of data. For instance we can use the name of a vertex in a graph to hide an actual data about the vertex. The data represented by the name may be a subject of change independently from the names. Some look-up function must be called dynamically to get an actual data for the given name.

2.2 Behavioral patterns

2.2.1 Chain of responsibility

In this pattern all possible receivers of an request are chained. It means that a sender is not coupled with the receiver(s) and possibly more objects can handle the request.

A possible implementation is to pass an argument corresponding to a request to a list of functions. Each function is a handler and it returns the result or some special value meaning a request was not handled. The type *Maybe a* parameterized with the type *a* of result can be used for extended results.

```
data Maybe a = Nothing
             | Just a
```

The return value of the whole processing is extracted from the sequence of results of handling functions. The examples of extracting functions are given in the figure. The function *chain*1 returns only the first valid value (using monadic sequencing) if it exists and *chainall* returns all valid results in a list.

```
chainall :: [a->Maybe b] -> a -> Maybe b
chainall fs x =
  filter (/=Nothing) (map (\f -> f x) x)

chain1 :: [a -> Maybe b] -> a -> Maybe b
chain1 [] x = Nothing
chain1 (f:fs) x = case f x of
                Nothing -> chain1 fs x
                Just x  -> Just x
```

In both cases all handling functions have the same type. The result value *Nothing* means that no function was able to process the data.

If the handling functions are not of the same type then the technique similar to continuations can be used. Each handler function has an additional parameter for a continuation function. The continuation is called with the original argument only when the current handler was not able to process the data. This corresponds to processing according to the *chain*1 function.

Generalization. In fact, handlers need not be in a chain, but they can create more complex structure. A function can process data immediatelly or it can pass them to some subhandlers. An idea of implementation id given in the figure.

```
handler f (f1,..,fn) x =
  let fx = f x in
        if handled fx then fx
         else compose_n (handler f1 x,..handler fn x)
```

The function *compose* selects relevant subhandlers and also composes its results. Using this approach in source code the handling function need not have the same type. The disadvantage is that all handlers must be given separately. This method was used in a different context for creating typed representation of XML documents [Th01].

2.2.2 Interpreter

If there is given a language, let's define a representation for its grammar along with an interpreter for it that uses the representation to interpret sentences in the language.

In a functional programming we usually interpret structured data, so the data incorporate the used rules. From this point of view the process of parsing, i.e. building structure, can be separated. The rest is an interpretation. The general function for an interpretation of data structures of a given type is the higher-order function *fold* for the type. The examples show a fold functions for lists and n-ary trees

```
fold :: a -> (a->b->b) -> [a] -> b
fold e f [] = e
fold e f (x:xs) = f x (fold e f xs)

foldNT :: (a->[b]->b) -> Tree a -> b
foldNT f (Node x ts) = f x (map (foldNT f) ts)
```

Each funcional parameter corresponds to one constructor of a type and interprets data structures with this main constructor.

The function *fold* must be implemented for each type separately in a typed language as Haskell. The ideas of polytypic programming [GHs] allow to write the *fold* function once and automatically generate instances for various types. It means that the pattern can be expressed as a code in such extended language.

2.2.3 Iterator

Iterator provides a way to access elements of an aggregate object sequentially without exposing its underlying representation. This idea can be transferred to the functional programming in two ways. They differ in understanding of the word sequentially. The first meaning is sequential data structure and the second one is sequentially in time.

In the first case we transform elements of an aggregate object to a list and then list-processing functions can be used. This is similar to the adapter pattern.

In the second case we prepare functions corresponding to an interface of an iterator. There are the functions *init*, *next*, *done* and possibly others for a given type a.

```
init    :: a -> St a
next    :: St a -> (a, St a)
done    :: St a -> Bool
```

The current state of iterator is captured in appropriate type $St\ a$ and is transferred among functions above using parameter. An implementation can use separate functions or can define type class of types equipped with an iteration.

Note that this pattern can be generalised. In both cases we are not restricted to the sequences but an element can have more following items. Such a generalised iterator can implement the method "Divide et impera".

2.2.4 State

A state pattern allows an object to alter its behavior when its internal state changes. The object will appear to change its class.

If the interface to an implementation, which should depend on a state, is a tuple of functions $(f_1, \ldots, f_n)$, then the particular funcions in the tuple have to change according to a change of the state.

A possible implementatiion of this pattern in functional programming is as follows. All parts of code, which depend on the state, take one additional parameter. The parameter is a tuple (s, t), where in s is encoded the actual implementation of functions corresponding to the current value of the state. The functions are in a form accessible for direct usage. The second part t is a list of all possible implementations, which must be accessible, when the state changes. These implementations can be in a form of tuples, so the whole tuple can be easily extracted when needed. The second part can be eliminated in such parts of code, where the state does not change.

Each change of the state cause a selection of new value of s from the list t. The actual value of the state can be one additional slot in the n-tuple. The representation of a state can be implemented using the Reader monad [Wa92].

2.2.5 Strategy and Template Method

The description of the strategy pattern is following. It defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

This pattern disappears in a functional programming as a possibility to use functions as parameters enable direct parametrization of functions with a strategy parameter.

The pattern Template Method is similar. It defines the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

In this case we use more functional parameters. Each one refers to a single step, which was deferred.

2.2.6 Visitor

The pattern Visitor represents an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

There are two types of visitor, the internal and the external. The first one performs given operation on all elements of the structure. This corresponds to the *map* function which gets the operation as a parameter. The second one needs to capture a state and its implementation is similar to the Iterator one.

2.2.7 Pipes and Filters

This architectural pattern [BMR] provides a structure for systems that process a stream of data. Each processing step is encapsulated in a filter component. Data are passed through pipes between adjacent filters.

Processing (finite) lists or (infinite) streams is a standard technique in functional programming. The binding of adjacent processing steps is realised by a function composition. The *map* function processes an input in an one-to-one style. The $filter$ function (in functional terminology) leaves out some data. Both functions have a functional parameter which describes the way of processing of an element in the first case and which data should remain in a stream in the second case. Other higher-order functions can support many-to-one or many-to-many processings.

2.3 Creational patterns

2.3.1 Builder

The Builder separates a construction of a complex object from its representation so that the same construction process can create different representations.

The data structures are built in a functional programming using constructor functions. We use the same style but instead real constructors we use virtual ones which hide the real construction process. Then we get the same effect in a functional programming.

A real implementation can use separate functions, type classes or a set of mutually recursive constructors which pass themselves to lower levels of a structure.

The described process of a construction is incremental and the real data structure can be repeatedly rebuild. So it may be more effective to give all data to the (abstract) construction process in one batch. The pattern can be also coded using the functions $fold$ and $unfold$. The first one can be used in cases when we have a structure and we want to reinterpret it. The second one enables replace constructors by given functional parameters during recursive building process.

3 Conclusion

We have shown that design patterns for many problems can be transferred from object-oriented programming to a functional programming and more generally to a declarative programming. However some problems and their published patterns are too specific for an object-oriented programming, so they were not covered in this paper. Also high-level architectural patterns and low-level patterns – programming idioms were left out.

In FP as well as in OOP it is usually possible to write a template for the core of a pattern. The template and examples are important for usefullness of a pattern library. Patterns are interconnected and rules of thumb were formulated [PPR].

There is no single universal entity in a functional programming as there is an object in OOP. The core idea of decoupling can be targetted to functions or to data structures and can be realized by various means. A comparison of various approaches is left for future work.

Some patterns correspond to well-known techniques in a functional programming. Other approach to analysis of correspondence can be taken. We can take such techniques and look for problems which they solve. A more general or more parametric pattern can be found using abstraction. Also an analysis of a relevance of published problems in a context of a functional (and logic) programming followed by a reformulation of the problems remains to be done.

4 Acknowlegment

Many thanks to Luděk Marek for discussion about patterns in object-oriented programming and for explaining some details. He and Michal Žemlička also commented a previous version of the paper.

References

[BMR] Buschmann F., Meunier R., Rohnert H., Sommerland P., Stal M., *A System of Patterns*, John Wiley, Chichester, England, 1996

[GHJ] Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, USA, 1995

[GHs] `http://www.generic-haskell.org/`

[Hs] `http://www.haskell.org/`

[HDP] *Houston Design Patterns*,
`http://rampages.onramp.net/~huston/dp/patterns.html`

[PPR] *Portland Pattern Repository*,
`http://www.c2.com/cgi/wiki?PortlandPatternRepository`

[Pr] Pree W., *Object-Oriented Design*, SOFSEM'97, LNCS 1338, Springer-Verlag, Berlin, 1997

[Th01] Thiemann P., *A Typed Representation for HTML and XML Documents in Haskell*, to be published in Journal of Functional Programming

[Wa92] Wadler P., *The Essence of Functional Programming*, In: Proc. Nineteenth Annual ACM Symposium on Principles of Programming Languages, Association for Computing Machinery, 1992

Redukční automaty, monotonie a redukovanost

Martin Procházka a Martin Plátek
katedra kybernetiky a teoretické informatiky
Matematicko-fyzikální fakulty
University Karlovy v Praze,
Česká republika
e-mail: martin.prochazka@adastra.cz, platek@ksi.ms.mff.cuni.cz

Abstrakt

Redukční automaty jsou variantou deterministických restartovacích automatů. Redukční automaty modelují redukční analýzu vstupních vět. Pomocí monotónních redukčních automatů charakterizujeme třídu jazyků DCFL a ukazujeme, že k redukčním automatům lze sestrojit redukt v podobném smyslu jako k Mooreovým automatům. Redukty zachovávají rozpoznávaný jazyk a redukční analýzu původních automatů.

1 Úvod a základní pojmy

V tomto příspěvku zavádíme redukční automaty a ukazujeme jejich základní vlastnosti. Redukční automaty jsou variantou deterministických restartovacích automatů. Restartovací automat byl poprvé představen v [3] jako zařízení vhodné pro modelování redukčních analýzy jak formálních, tak i přirozených jazyků.

Redukční automat si můžeme představit jako žáka, který provádí větný rozbor. Takový žák čte zadanou větu zleva doprava, jedno slovo po druhém. Aby se ve větě neztratil, ukazuje si v ní prstem. Jeho ukazovák přitom míří mezi dvě slova, čímž větu rozděluje na dvě části: již přečtenou a ještě nepřečtenou. O přečtené části věty si dělá poznámky na kus papíru omezené velikosti.

Žák postupuje velmi systematicky: Nejprve se podívá na svůj poznámkový papír, pak si přečte první ještě nepřečtené slovo, přesune za ně svůj ukazovák a nakonec přepíše poznámku na svém papíru. Na vhodném místě pak větu prohlásí za bezchybnou, nebo ji prohlásí za chybnou, nebo ji zkrátí a s čistým kusem papíru ji začne číst znovu od začátku. Větu může prohlásit za bezchybnou, teprve až když ji celou přečte. Zkrácení věty provede tak, že z ní odstraní několik slov nalevo před ukazovákem. Vzdálenost všech odstraněných slov od ukazováku je přitom omezená nějakou konstantou.

Představu žáka provádějícího větný rozbor nyní zformalizujeme. Místo věty tvořené slovy máme *řetězec* složený ze *symbolů vstupní abecedy*. Pravý konec řetězce explicitně vymezujeme speciálním symbolem – *omezovačem*. Omezovač se liší od všech symbolů zmíněné vstupní abecedy. Žákovu poznámkovému papíru odpovídá *řídící jednotka*, která může nabývat jednoho z konečně mnoha *stavů*. Ukazovák nahradíme *ukazatelem* aktuální pozice v řetězci, který je spojen s řídící jednotkou. Místům v řetězci, na která automat ukazuje, říkáme *pozice*. Pozice v řetězci vyznačujeme přirozenými čísly tak, že zleva doprava tvoří rostoucí (ne nutně souvislou) posloupnost. Začátku řetězce odpovídá pozice 0. Pozicí symbolu v řetězci myslíme pozici v řetězci bezprostředně za tímto symbolem. Čistému poznámkovému papíru odpovídá tzv. *počáteční stav* označovaný jako q_0. S tímto stavem v řídící jednotce a s ukazatelem na úplném začátku řetězce (na pozici 0) začíná redukční automat svou práci. Redukční automat v každém kroku přesune svůj ukazatel přes jeden symbol doprava na následující pozici. Podle stavu své řídící jednotky a podle symbolu, přes který ukazatel přesunul, nastaví svou řídící jednotku do nového stavu.

Některé stavy mají speciální význam. Takovým stavům říkáme *operace*. Redukční automat využívá operací tří typů – `ACC`, `ERR` a `RED`. Operace `ACC` formalizuje prohlášení o bezchybnosti daného řetězce (*přijetí* řetězce), operace `ERR` prohlášení o jeho chybnosti (*zamítnutí* řetězce). Operace `RED` (*redukční* operace) říkají, jak má automat zpracovávaný řetězec zkrátit. Jakmile se v řídící jednotce automatu objeví operace $\texttt{RED}(n)$, zkrátí automat zpracovávaný řetězec podle binární posloupnosti n. Končí-li n jedničkou, pak automat odstraní z řetězce pozici, na kterou ukazuje a poslední přečtený symbol vlevo za touto pozicí. Končí-li naopak posloupnost n nulou, pak automat „couvne“ zpět přes poslední přečtený symbol směrem doleva. Poté v obou případech zkrátí binární posloupnost n ve své řídící jednotce o poslední cifru. Takto postupuje až do chvíle, kdy jeho ukazatel ukazuje na pozici 0, nebo kdy je binární posloupnost v jeho řídící jednotce prázdná. Vykonávání RED-operace končí návratem na pozici 0 a nastavením řídící jednotky do počátečního stavu q_0.

RED-operace mají pro práci automatu značný význam. Díky nim rozpoznávají redukční automaty větší třídu jazyků než automaty konečné (viz věta 2.4 na straně 8).

Poté, co jsme se získali základní představu o tom, co je to redukční automat, a jakým způsobem nakládá s řetězcem symbolů, můžeme tuto představu formalizovat. Začneme jeho definicí:

Definice 1.1 (Redukční automat). *Redukční automat M je pětice*

$$(\Sigma, Q, R, q_0, \Delta),$$

kde Σ je konečná vstupní abeceda, Q je konečná množina stavů, $R \subseteq Q$ je množina operací, $q_0 \in Q$ je počáteční stav a Δ je přechodová funkce. Množina R obsahuje operace `ACC`*,* `ERR` *a dále několik operací* $\texttt{RED}(n)$*, kde $n \in \texttt{1}\{\texttt{0},\texttt{1}\}^*$. Přechodová funkce Δ přiřadí každé dvojici z $(Q \setminus R) \times (\Sigma \cup \{\bullet\})$ stav nebo operaci z Q. Přechodová funkce v libovolné dvojici (q, a) vyhovuje následujícím podmínkám:*

- *je-li $\Delta(q, a) = \texttt{ACC}$, pak $a = \bullet$,*
- *je-li $\Delta(q, a) \in (Q \setminus R)$, pak $a \neq \bullet$.*

- *je-li* $\Delta(q, \bullet) = \texttt{RED}(n)$*, pak* n *končí číslicí* `0`.

Pro automat M zavedeme tzv. *charakteristickou konstantu* k_M jako délku nejdelší binární posloupnosti obsažené v nějaké RED-operaci automatu M.

$$k_M = \max\{|n| \mid \texttt{RED}(n) \text{ je operace automatu } M\} \tag{1}$$

Stavy včetně operací (prvky množiny Q) budeme označovat písmenem s nebo s', stavy různé od operací (prvky množiny $Q \setminus R$) pak písmenem q nebo q'. V obou případech budeme často používat dolní indexy.

Zobecněná přechodová funkce. Přechodovou funkci zobecníme stejným způsobem jako přechodovou funkci konečných automatů:

$$\begin{aligned} \Delta(s, w) &= s, \quad \text{jestliže} \quad s \in Q \quad \text{a} \quad w = \lambda, \\ \Delta(q, wa) &= s, \quad \text{jestliže} \quad \Delta\left(\Delta(q, w), a\right) = s. \end{aligned}$$

Je-li $\Delta(q, w) = s$, pak říkáme, že automat přešel ze stavu q přes slovo w do stavu s. Jestliže je s operace, pak říkáme, že automat po přechodu ze stavu q přes slovo w vykonal operaci s. Stav q nazýváme *výchozím stavem* a stav s *dosaženým stavem*, resp. *vykonanou operací*. Slovo w je *přečtené slovo*. V případě, že s je operace `ACC`, resp. `ERR`, mluvíme o ACC-, resp. ERR-přechodu.

Etapa. Přechod redukčního automatu z počátečního stavu s ukazatelem na začátku řetězce až k operaci, kterou má nad seznamem vykonat, budeme nazývat *etapou*. Formálně etapu zapisujeme takto:

$$\Delta(q_0, w) = s, \quad \text{kde } s \in R.$$

Podle operace, kterou etapa končí, budeme rozlišovat ACC-, ERR- a RED-etapy.

Relace redukce. Zkrácení řetězce podle binární posloupnosti, které redukční automat provádí svými RED-operacemi, zachytíme pomocí notace definované následujícími vztahy:

$$\lfloor \tfrac{u}{\lambda} \rfloor = \underline{u} \qquad \lfloor \tfrac{\lambda}{n} \rfloor = \lambda \qquad \lfloor \tfrac{u \cdot a}{n \cdot 0} \rfloor = \lfloor \tfrac{u}{n} \rfloor \cdot \underline{a} \qquad \lfloor \tfrac{u \cdot a}{n \cdot 1} \rfloor = \lfloor \tfrac{u}{n} \rfloor$$

V uvedených vztazích je n binární posloupnost, $u \in (\Sigma \cup \{\bullet\})^*$ a $a \in \Sigma \cup \{\bullet\}$. Tuto notaci zavádíme obecně pro řetězce a binární posloupnosti, jejichž délka není omezena žádnou konstantou. Zkrácení řetězce $(_6\,\mathtt{a}_7\,)_8$ podle posloupnosti `101` zapíšeme takto:

$$\lfloor \begin{matrix} (_6 & \mathtt{a}_7 &)_8 \\ 1 & 0 & 1 \end{matrix} \rfloor = \mathtt{a}_7$$

Pomocí uvedené notace můžeme nyní popsat, jak automat svými RED-operacemi zkracuje zpracovávaný řetězec. Zavedeme pro to relaci *redukce*:

$$w_1 \Rightarrow w_2, \quad \text{jestliže} \quad w_1 \bullet = w w' \quad \text{a} \quad \Delta(q_0, w) = \texttt{RED}(n) \quad \text{a} \quad w_2 \bullet = \lfloor \tfrac{w}{n} \rfloor \cdot w'.$$

Platí-li $w_1 \Rightarrow w_2$, říkáme, že automat *redukuje* řetězec w_1 na řetězec w_2. Je-li navíc $|w_1| > |w_2|$, pak mluvíme o *zkracující* redukci. Reflexivní a tranzitivní uzávěr relace redukce budeme označovat jako $\Rightarrow^*$. Někdy bude vhodné explicitně vyznačit, který automat redukci provedl. Uděláme to tak, že dotyčný automat uvedeme jako dolní index.

Redukční analýza. *Konečnou redukční analýzou* automatu M myslíme libovolnou konečnou posloupnost redukcí

$$w_1 \Rightarrow w_2 \Rightarrow ... \Rightarrow w_n,$$

kde w_n je řetězec, nad kterým může tento automat vykonat ACC- nebo ERR-etapu. Končí-li poslední etapa ACC-operací, mluvíme o *přijímající* analýze, končí-li ERR-operací, jedná se o *zamítající* analýzu. V následujícím textu budeme často místo termínu redukční analýza používat zkrácený termín analýza.

Kromě konečných analýz může redukční automat provádět i analýzy *nekonečné*. Analýza

$$w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \ldots$$

je nekonečná, jestliže od nějakého k je $w_k = w_{k+i}$ pro každé $i \geq 0$. K tomu dojde v případě, že automat po přečtení prefixu w řetězce w_k vykoná operaci $\texttt{RED}(n)$, kde n je nějaká binární posloupnost končící $|w|$ nulami. To, že pro každé $i \geq 0$ platí $w_k = w_{k+i}$, jistě plyne z rovnosti slov w_k a w_{k+1}. Nekonečnou analýzu prohlásíme definitoricky za zamítající. Za chvíli si ukážeme, že se bez újmy na obecnosti můžeme omezit jen na redukční automaty, jejichž libovolná analýza je konečná.

Přijímaný jazyk. Jazyk *přijímaný* nebo též *rozpoznávaný* redukčním automatem M definujeme jako množinu slov $L(M)$, pro která existuje přijímající analýza automatu M:

$$L_{\texttt{ACC}}(M) = \{\, w \in \Sigma^* \mid \Delta(q_0, w\bullet) = \texttt{ACC} \,\}$$
$$L(M) = \{\, w \in \Sigma^* \mid \exists\, w' \in L_{\texttt{ACC}}(M) : w \Rightarrow^* w' \,\}$$

Ekvivalence red-automatů. Pomocí rovnosti množin přijímaných jazyků definujeme ekvivalenci na třídě všech redukčních automatů. Dva automaty M_1 a M_2 jsou *ekvivalentní*, jestliže

$$L(M_1) = L(M_2)$$

Následující lemma uvádí postačující podmínku pro ekvivalenci dvou redukčních automatů.

Lemma 1.2. *Libovolné dva red-automaty M, M' jsou ekvivalentní, jestliže zároveň*

(i) $L_{\texttt{ACC}}(M) = L_{\texttt{ACC}}(M')$ *a*

(ii) $w_1 \Rightarrow_M w_2$ *je zkracující redukce, právě když* $w_1 \Rightarrow_{M'} w_2$ *je zkracující redukce.*

Důkaz. Nejprve ukážeme, že $L(M) \subseteq L(M')$. Indukcí podle n ukážeme, že z $w \Rightarrow_M^n w' \in L_{\texttt{ACC}}(M)$ plyne $w \in L(M')$ pro každé $n \geq 0$.

1. Je-li $n = 0$, pak $w \in L_{\texttt{ACC}}(M)$ a podle (i) je $w \in L_{\texttt{ACC}}(M')$.
2. Předpokládejme, že tvrzení platí pro každé $m \leq n$. Ukážeme, že potom platí i pro $m = n + 1$.
   Nechť $w \Rightarrow_M w'' \Rightarrow_M^n w' \in L_{\texttt{ACC}}(M)$. Potom jistě $|w| > |w''|$ a podle indukčního předpokladu platí, že $w'' \in L(M')$. Z (ii) plyne, že $w \Rightarrow_{M'} w''$, takže $w \in L(M')$.

Stejným postupem můžeme dokázat i obrácenou inkluzi: $L(M) \supseteq L(M')$. □

Vlastnost zachování chybnosti a bezchybnosti. Pomocí relace redukce můžeme vyslovit základní vlastnost, kterou splňují všechny redukční automaty:

Lemma 1.3. *Jestliže $w_1 \Rightarrow_M w_2$, potom $w_1 \in L(M)$, právě když $w_2 \in L(M)$.*

Tuto vlastnost budeme nazývat *vlastností zachovávání chybnosti a bezchybnosti.* Její platnost snadno nahlédneme z následující úvahy: Je-li $w_2 \Rightarrow_M^* w$ přijímající analýza pro slovo w_2, pak $w_1 \Rightarrow_M w_2 \Rightarrow_M^* w$ je přijímající analýza pro slovo w_1. Vyjdeme-li naopak z předpokladu, že $w_1 \in L(M)$, pak máme analýza $w_1 \Rightarrow_M^* w$, kde w je nějaké slovo z $L(M)$. Tato analýza musí začínat redukcí slova w_1 na slovo w_2, jinak by nebyla přechodová funkce automatu M definována jednoznačně.

Eliminace nekonečných analýz.

Tvrzení 1.4. *K libovolnému redukčnímu automatu lze sestrojit ekvivalentní redukční automat, jehož libovolná analýza je konečná.*

Důkaz. Předpokládejme, že $M = (\Sigma, Q, R, q_0, \Delta)$ je redukční automat s charakteristickou konstantou k_M. K automatu M sestrojíme redukční automat $M' = (\Sigma, Q', R', q_0', \Delta')$ a ukážeme, že je ekvivalentní s automatem M, a že každá jeho analýza je konečná.

Konstrukce. Množinu operací a množinu stavů automatu M' vymezíme takto:

$$R' = \{\mathtt{ACC}, \mathtt{ERR}\} \cup \{\mathtt{RED}(1n) \mid \mathtt{RED}(n'1n) \in R \text{ pro nějaké } n'\},$$
$$Q' = R' \cup \big((Q \setminus R) \times \{1, \ldots, k_M\} \big).$$

Počátečním stavem automatu M' je dvojice $(q_0, 1)$.

Přechodovou funkci Δ' definujeme pro každý stav $(q, m) \in Q' \setminus R'$ a libovolný symbol a abecedy $\Sigma \cup \{\bullet\}$ takto

$$\Delta'\big((q,m),a\big) = \begin{cases} (q', m+1), & \text{je-li } \Delta(q,a) = q' \notin R \text{ a } m < k_M, \\ (q', m), & \text{je-li } \Delta(q,a) = q' \notin R \text{ a } m = k_M, \\ \mathtt{ACC}, & \text{je-li } \Delta(q,a) = \mathtt{ACC}, \\ \mathtt{RED}(n), & \text{je-li } \Delta(q,a) = \mathtt{RED}(n) \text{ a } |n| \le m, \\ \mathtt{RED}(n'), & \text{je-li } \Delta(q,a) = \mathtt{RED}(n),\ |n| > m \text{ a } n' \text{ nejdelší sufix} \\ & n, \text{ který začíná jedničkou a je kratší než } m, \\ \mathtt{ERR}, & \text{jinak.} \end{cases}$$

Automat M' tedy simuluje práci automatu M a navíc ještě v každé etapě odpočítává prvních k_M jeho přechodů. Pokud během těchto prvních k_M přechodů vykoná simulovaný automat RED-operaci, vykoná ji i automat M', ale jen tehdy, když odstraní z řetězce aspoň jeden symbol. Neodstraní-li žádný symbol, pak řetězec zamítne. Je-li etapa automatu M delší než k_M přechodů, pak je automatem M věrně simulována.

	Δ	a	+	(	)	•
⇒	q_0	q_1	ERR	q_2	ERR	ERR
	q_1	ERR	q_3	ERR	ERR	ACC
	q_2	q_4	ERR	q_2	ERR	ERR
	q_3	q_5	ERR	q_2	ERR	ERR
	q_4	ERR	q_3	ERR	RED(101)	ERR
	q_5	ERR	q_3	ERR	RED(110)	RED(110)

Obrázek 1: Reprezentace redukčního automatu.

Konečnost analýz. Konečnost analýz automatu M' okamžitě plyne z popisu jeho etapy.

Ekvivalence. Z popisu etapy automatu M' plyne, že $L_{\mathtt{ACC}}(M') = L_{\mathtt{ACC}}(M)$, a že $w_1 \Rightarrow_{M'} w_2$ je zkracující redukce, právě když i $w_1 \Rightarrow_M w_2$ je zkracující redukce. Je tedy splněna postačující podmínka pro ekvivalenci red-automatů z lemmatu 1.2. □

Právě dokázané tvrzení nám dává jistotu, že se bez újmy na obecnosti můžeme dále omezit jen na redukční automaty bez nekonečných analýz.

Reprezentace. Lepší představu o daném redukčním automatu dává jeho vhodná reprezentace. Redukční automat můžeme reprezentovat *přechodovou tabulkou* převzatou z teorie konečných automatů. Příklad takové tabulky vidíme na obrázku 1. Automat zadaný uvedenou tabulkou rozpoznává jazyk zjednodušených aritmetických výrazů. Ty se skládají ze symbolů `a`, `+`, `(` a `)`. Počátečním stavem automatu je stav q_0. Tabulku čteme takto: Je-li v řádku označeného stavem q a sloupci označeného symbolem a stav nebo operace s, pak platí $\Delta(q, a) = s$ a naopak. Řádek obsahující instrukce s počátečním vstupním stavem je v tabulce označen šipkou (⇒).

2 Monotonie

Monotonie je důležitá vlastnost, která umožňuje charakterizovat třídu DCFL deterministických bezkontextových jazyků prostřednictvím redukčních automatů.

Nechť $M = (\Sigma, Q, R, q_0, \Delta)$ je redukční automat. Řekneme, že M je *monotónní*, pokud pro každou RED-etapu tvaru $\Delta(q_0, w) = \mathtt{RED}(n)$, existuje $s \in Q \cup R$ takové, že platí $\Delta(q_0, \lfloor {}^{w}_{n} \rfloor) = s$.

Monotónní redukční automaty budeme zkráceně zapisovat jako mon-red-automaty. Třídu všech jazyků rozpoznávaných mon-red-automaty budeme označovat $\mathcal{L}(\textit{mon-red})$.

V této sekci ukážeme, že monotónní redukční automaty charakterizují deterministické bezkontextové jazyky. Za tímto účelem převezmeme pojem R-automatu z [3].

R-automaty. *R-automat* $M = (Q, \Sigma, k, I, q_0)$ je zařízení složené z řídící jednotky nacházející se vždy v jednom ze stavů z konečné množiny Q a z pracovní hlavy, ke které je připojen výhled velikosti $k \geq 0$. R-automat pracuje nad lineárním seznamem, který sestává z položek. První i poslední položka seznamu obsahuje speciální symbol (levý sentinel # resp. pravý sentinel \$). Všechny ostatní položky obsahují po jednom symbolu konečné *vstupní abecedy* Σ (#, \$ $\notin \Sigma$). Pracovní hlava automatu snímá jednu položku pracovního seznamu. Kromě položky snímané hlavou, čte M ve výhledovém okně ještě k sousedních položek napravo od hlavy (nebo konec seznamu, je-li vzdálenost k pravému sentinelu menší než k).

Práci R-automatu popisujeme pomocí *konfigurací*. Konfigurace udává obsah pracovního seznamu, stav řídící jednotky a pozici hlavy v seznamu. Konfiguraci s hlavou snímající levý sentinel a s řídící jednotkou v *počátečním stavu* q_0, nazýváme *startovací konfigurací*. Je-li řídící jednotka ve stavu ACC resp. REJ, je automat v *přijímající* resp. zamítající koncové konfiguraci.

Výpočtem R-automatu M rozumíme posloupnost konfigurací, která začíná startovací a končí koncovou konfigurací. Přechody automatu z jedné konfigurace do konfigurace následující jsou řízeny *instrukcemi* z konečné množiny I. R-automat má instrukce následujících tří typů:

$$(q, au) \to (q', \mathrm{MVR}), \qquad (q, au) \to \mathrm{ACC},$$
$$(q, au) \to (q', \mathrm{RST}(v)), \qquad (q, au) \to \mathrm{REJ},$$

kde q, $q' \in Q$, $a \in \Sigma \cup \{\#, \$\}$, u, $v \in (\Sigma)^* \cup (\Sigma)^* \cdot \{\$\}$ a v je vlastní podposloupnost slova au. Instrukce je použitelná, když je řídící jednotka automatu ve stavu q a jeho hlava spolu s výhledem snímá slovo au. Použitelnost instrukce tedy určuje její levá strana. Pravá strana popisuje akci, která má být provedena. MVR-instrukce změní stávající stav na q' a přesune hlavu o jednu položku doprava. RST-instrukce vypustí z části seznamu právě snímané hlavou a výhledovým oknem několik položek (alespoň jednu) tak, že poté tato část seznamu obsahuje slovo v, a provede restart. To znamená, že nastaví automat M do startovací konfigurace nad zkráceným slovem. Poznamenejme, že není dovoleno vypustit žádný ze sentinelů. Jak ACC-, tak i REJ-instrukce znamená konec výpočtu. V prvním případě je seznam přijat, ve druhém zamítnut.

Podobně jako u redukčních automatů jsou výpočty restartovacích automatů rozděleny na etapy. Každá etapa začíná ve startovací konfiguraci. Etapě, která končí startovací konfigurací říkáme cyklus. Etapu, která končí operací (stavem) ACC, nazýváme přijímající etapou, etapu, která končí stavem REJ, nazýváme zamítající etapou. Cykly zapisujeme pomocí jejich redukcí. Označení $u \Rightarrow_M v$ (redukce u na v podle M) znamená, že existuje cyklus automatu M začínající ve startovací konfiguraci se slovem u a končící v restartovací konfiguraci se slovem v; relace $\Rightarrow_M^*$ je reflexivním a transitivním uzávěrem $\Rightarrow_M$.

Slovo w je přijato R-automatem M pokud existuje výpočet, který začíná startovací konfigurací se slovem $w \in \Sigma^*$ a končí *přijímací konfigurací*, kde řídící jednotka je ve stavu ACC. $L(M)$ označuje jazyk sestávající ze všech slov přijímaných automatem M; říkáme, že M *rozpoznává jazyk* $L(M)$.

Říkáme, že R-automat M je *deterministický*, pokud každé dvě jeho různé instrukce mají různou levou stranu. Zde nás zajímají pouze deterministické R-automaty. U deterministických R-automatů odpovídá každé redukci $u \Rightarrow_M v$ právě

jediný cyklus. Deterministické R-automaty budeme označovat prefixem *det-*. Zavedeme také pojem monotonie výpočtů R-automatů. $Dist(u \Rightarrow_M v)$ označuje pro libovolný cyklus $u \Rightarrow_M v$ vzdálenost poslední položky, která se dostala do výhledového okna během cyklu $u \Rightarrow_M v$, od pravého sentinelu. Říkáme, že výpočet $\mathcal{C}$ automatu M je *monotonní*, pokud pro posloupnost jeho cyklů $u_1 \Rightarrow_M u_2 \Rightarrow_M \cdots \Rightarrow_M u_n$ je $Dist(u_1 \Rightarrow_M u_2)$, $Dist(u_2 \Rightarrow_M u_3)$, ..., $Dist(u_{n-1} \Rightarrow_M u_n)$ monotonní (neklesající) posloupnost. *Monotonním R-automatem* myslíme R-automat, jehož všechny výpočty jsou monotonní. Prefixem *mon-* budeme označovat monotonní verze R-automatů.

Třídu právě všech jazyků rozpoznávaných det-mon-R-automaty budeme zapisovat jako $\mathcal{L}(det\text{-}mon\text{-}R)$. Následující větu přebíráme z [3]. Proto ji nebudeme dokazovat.

Věta 2.1. $\mathcal{L}(det\text{-}mon\text{-}R) = \text{DCFL} \subset \mathcal{L}(det\text{-}R)$

Vztah R- a red-automatů. Není těžké nahlédnout, že platí následující lemma. Proto si dovolíme vynechat jeho důkaz, který potvrzuje, že mezi R-automaty a redukčními automaty je pouze technický rozdíl.

Lemma 2.2.

(i) Ke každému det-(mon-)R-automatu M lze sestrojit (mon-)red-automat M_1 tak, že $L(M) = L(M_1)$ a pro každé u,v platí, že $u \Rightarrow_M v$ právě tehdy, když $u \Rightarrow_{M_1} v$.

(ii) Ke každému (mon-)red-automatu M lze sestrojit det-(mon-)R-automat M_1 tak, že $L(M) = L(M_1)$ a pro každé u,v platí, že $u \Rightarrow_M v$ právě tehdy, když $u \Rightarrow_{M_1} v$.

Věta 2.3. $\mathcal{L}(red) = \mathcal{L}(det\text{-}R)$ *a* $\mathcal{L}(mon\text{-}red) = \mathcal{L}(det\text{-}mon\text{-}R)$

Důkaz. Věta vyplývá z předchozího lemmatu. □

Věta 2.4. $\mathcal{L}(mon\text{-}red) = \text{DCFL} \subset \mathcal{L}(red)$

Důkaz. Věta je důsledkem vět 2.1 a 2.3. □

3 Redukovanost

V této sekci si ukážeme, jak lze k libovolnému redukčnímu automatu sestrojit redukční automat, který provádí stejné analýzy a je ze všech takových automatů minimální.

Dosažitelnost. Stav $s \in Q$ automatu M je *dosažitelný*, jestliže pro nějaké slovo $w \in \Sigma^* \cup \Sigma^* \bullet$ nastává

$$\Delta(q_0, w) = s,$$

kde q_0 je počáteční stav automatu M. Stav, který není dosažitelný, nazýváme *nedosažitelný*.

Jak jsme již řekli dříve, na redukční automat se můžeme dívat jako na konečný automat s množinou stavů Q. Přechodová funkce δ takového konečného automatu

se nad množinou $Q \setminus R$ shoduje s přechodovou funkcí Δ. Nad množinou R ji můžeme dodefinovat jako identitu. Z teorie konečných automatů tak můžeme převzít konstrukci konečného automatu obsahujícího pouze dosažitelné stavy. Bez důkazu vyslovíme následující větu:

Věta 3.1. *K libovolnému redukčnímu automatu lze sestrojit ekvivalentní automat, jehož všechny stavy (a tedy i operace) jsou dosažitelné.*

Etapová ekvivalence. Nechť $M = (\Sigma, Q, R, q_0, \Delta)$ a $M' = (\Sigma, Q', R', q_0', \Delta')$ jsou dva redukční automaty. Mezi množinami stavů obou automatů definujeme relaci $\sim$ takto: $q \sim q'$, jestliže pro libovolné slovo $w \in \Sigma^* \cup \Sigma^*\bullet$ a libovolnou operaci $s \in R$ a pro libovolné $s' \in Q'$ platí následující implikace:

$$\left.\begin{aligned}\Delta(q, w) &= s\\ \Delta'(q', w) &= s'\end{aligned}\right\} \quad \Longrightarrow \quad s = s'$$

Pokud $M = M'$, pak relace $\sim$ je ekvivalencí na množině stavů Q automatu M. Automaty, jejichž počáteční stavy jsou v relaci $\sim$, budeme nazývat *etapově ekvivalentní*.

Redukovanost. Automat M je *redukovaný*, jestliže všechny jeho stavy jsou dosažitelné a žádné dva jeho různé stavy nejsou ekvivalentní. Automat M' nazveme *reduktem* automatu M, jestliže M' je etapově ekvivalentní s M a M' je redukovaný.

Věta 3.2. *K libovolnému redukčnímu automatu lze sestrojit jeho redukt.*

Důkaz. K redukčnímu automatu $M = (\Sigma, Q, R, q_0, \Delta)$ sestrojíme jeho redukt $M' = (\Sigma, Q', R', q_0', \Delta')$. Díky větě 3.1 můžeme bez újmy na obecnosti předpokládat, že každý stav automatu M je dosažitelný.

Jak jsme si již řekli dříve, na redukční automat se můžeme dívat jako na Mooreův stroj A doplněný o možnost redukovat vstupní slovo. Množinou stavů tohoto stroje je množina Q všech stavů M. Přechodová funkce redukčního automatu spolu s množinou operací vede ihned k přechodové funkci δ a značkovací funkci μ stroje A:

$$\delta(s, a) = \begin{cases}\Delta(s, a), & \text{jestliže } s \in Q \setminus R \text{ a } a \in \Sigma \cup \{\bullet\},\\ s, & \text{jestliže } s \in R \text{ a } a \in \Sigma \cup \{\bullet\},\end{cases}$$

$$\mu(s) = \begin{cases}q_0, & \text{jestliže } s \in Q \setminus R,\\ s, & \text{jestliže } s \in R.\end{cases}$$

K takto definovanému Mooreovu stroji sestrojíme jeho redukt A' s přechodovou funkcí δ' a značkovací funkcí μ' a od reduktu A' přejdeme zpět k redukčnímu automatu M'. Množina jeho stavů Q' je tvořena právě všemi stavy stroje A'. Množinu jeho operací R' tvoří hodnoty značkovací funkce μ' ve všech stavech s stroje A' až na hodnotu q_0. Přechodovou funkci Δ' definujeme pro každou dvojici $(q, a) \in (Q' \setminus R') \times (\Sigma \cup \{\bullet\})$ takto:

$$\Delta'(q, a) = \begin{cases}s, & \text{jestliže } s = \delta'(q, a) \text{ a } \mu'(s) = q_0,\\ \mu'(s), & \text{jestliže } s = \delta'(q, a) \text{ a } \mu'(s) \neq q_0.\end{cases}$$

Počátečním stavem automatu M' je stav s stroje A', který je v relaci $\sim$ se stavem q_0 stroje A. Protože A' je redukt stroje A, je takový stav právě jeden.

Fakt, že M' je reduktem automatu M plyne přímo z jeho konstrukce a z teorie Mooreových strojů. □

Isomorfismus. Říkáme, že redukční automaty $M = (\Sigma, Q, R, q_0, \Delta)$ a $M' = (\Sigma, Q', R', q_0', \Delta')$ jsou *isomorfní*, jestliže $R = R'$ a pro nějaké zobrazení $h : Q \longrightarrow Q'$ platí zároveň, že h je bijekce, h je identita na množině operací R a pro každé $q \in Q \setminus R$, $s \in Q$ a $a \in \Sigma \cup \{\bullet\}$ je $\Delta_M(q, a) = s$, právě když $\Delta_{M'}(h(q), a) = h(s)$.

Z teorie Mooreových strojů převezmeme bez důkazu následující větu:

Věta 3.3. *Libovolné dva redukty téhož redukčního automatu jsou isomorfní.*

Věta 3.4. *Žádný redukční automat nemá méně stavů než jeho redukt.*

Důkaz. Pro důkaz sporem předpokládejme, že automat $M = (\Sigma, Q, R, q_0, \Delta)$ má méně stavů než jeho redukt $M' = (\Sigma, Q', R', q_0', \Delta')$. Nechť $h : Q' \longrightarrow Q$ je definované takto: $h(q') = q$, jestliže $q' \sim q$. Potom ale pro nějaká dvě různá q_1, $q_2 \in Q'$ platí, že $h(q_1) = h(q_2) = q$, takže $q_1 \sim q_2$, což je spor s redukovaností automatu M'. □

4 Závěrečná poznámka

Předchozí sekce ukázala jisté pěkné vlastnosti redukčních automatů. O dalších vlastnostech red-automatů důležitých pro lokalizaci syntaktických chyb se čtenář bude moci dočíst v připravované dizertační práci prvého autora. Zvláště upozorňujeme na pojem stromu výpočtu, který charakterizuje výpočty mon-red-automatů. Tyto stromy se svojí formou velmi blíží závislostním stromům, které známe z matematické lingvistiky.

Poděkování

Práce na tomto tématu je podporována grantem GAČR č. 201/02/1456 a grantem GAUK č. 300/2002/A INF/MFF.

Reference

[1] Calude C. S., Calude E., Khoussainov B.: *Finite Nondeterministic Automata: Simulation and Minimality*; 1997

[2] Chytil M.: *Teorie automatů a formálních jazyků*; SPN Praha, 1978

[3] P. Jančar, F. Mráz, M. Plátek, J. Vogel: *Restarting Automata;*, in Proc. FCT'95, Dresden, Germany, August 1995, LNCS 965, Springer Verlag 1995, pp. 283 - 292

Hypertext Presentation of Relational Data Structures

Jana KOHOUTKOVÁ

Masaryk University Brno, Institute of Computer Science
Botanická 68a, 602 00 Brno, Czech Republic
kohoutkova@ics.muni.cz

Abstract. The paper overviews main features of a document description language specifically designed to present relational data structures in the form of hypertext (hypermedia) documents – typically, sets of mutually cross-linked web pages.

Keywords: modelling, integration, relational data, hypertext/media documents.

1 Motivation

Several internet/intranet information systems built on top of relational databases have been the motivation for the development of the data and document description language presented here. Already some quite common Internet presentations have raised the problem of keeping the web of presentation pages permanently consistent with the underlying – quite often changing and expanding – data structures. Nonetheless, the real need for a suitable formal description of presentation documents came with two R&D projects – the older *HyperMeData* one (CP 94-0943, [1]), and the recent *MeDiMed* one. The aim of HyperMeData was to build an environment for mutual data exchange between independent hospital information systems allowing authorised users to browse and view the interchanged data. The task of MeDiMed is to build a hypermedia atlas of annotated medical images serving to research and university-level education. In all these cases, the data in question is organised in relational databases, mostly in large volumes, and the common requirement is to transform it in some systematic way into hypertext/media presentation documents.

In the following text we shall illustrate the problem on a running example, identify the integration interfaces, and present a solution – the DDL language that was designed and implemented to support *integration of relational data structures and presentation hyperdocuments* by defining *transformational relationships* among data and document *instances* based on the *description* of the respective (data, document) *schemas*.

2 Running Example

Let us first consider an example of a very simple *relational data schema* of a *school* storing basic information about *people, university workplaces*, and *addresses* together

with a few relationships between them capturing the hierarchy of workplaces, workplace addresses, people at the workplaces, and headpersons of the workplaces. The schema consists of *entities* (bearing *content*, or *descriptive* information in their instances) and *associations* (bearing *relational*, or *binding* information in their instances), and may be expressed as a diagram consisting of specific construction elements – rectangles (representing entities) and diamonds (representing relationships) together with the respective joins between pairs of them, as shown in Fig. 1.

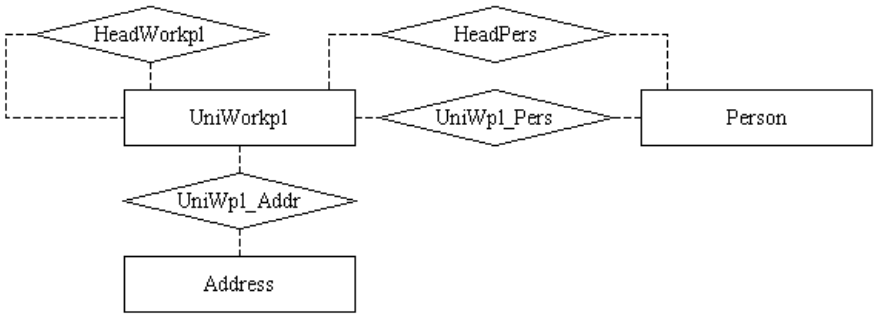


Fig. 1. Running example: The data schema.

Let's then suppose that the *hypertext* (or, more generally, *hypermedia*) *presentation document* built on top of the above data schema is expected to provide a more detailed (specialized) view at the data, namely, to differentiate between *faculties* and *departments* (both ISA *university workplaces*). The document schema consists of *sections* (content bearing, descriptive) and *references* (linking, or binding) and, again, may be expressed as a diagram – this time constructed of rectangles with headings (sections) and double-diamonds (references), as illustrated in Fig. 2.

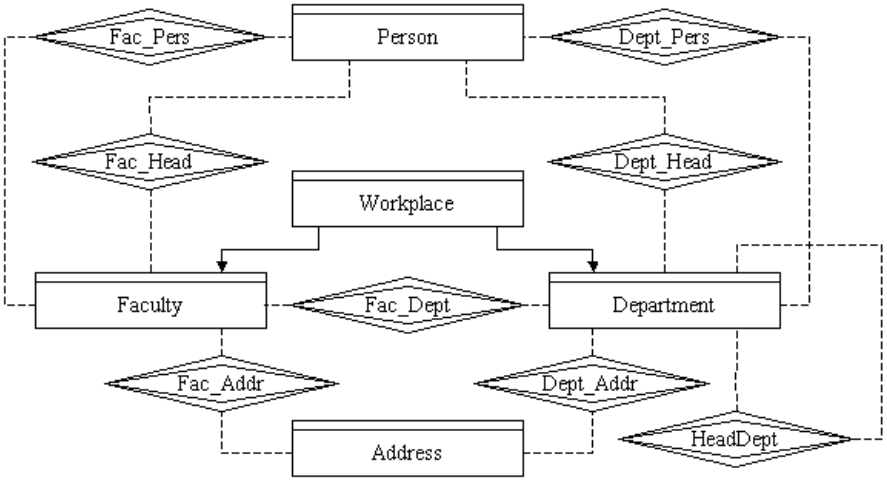


Fig. 2. Running example: The document schema.

Additionally to the running example data schema diagram, the joins in the document schema diagram are of two types, thus differentiating between ISA relationships (full lines with arrows) and referential relationships (dashed lines). Let's remark that the ISA relationships exist in data schemas as well – just the example used in this paper does not utilise them.

3 The Integration Problems

The integration task, as illustrated on the above example, covers two sets of problems:

1. the problem of transforming data organised by one schema (data schema) to data organised by another schema (document schema);
2. the problem of presenting data organised by a document schema to the user.

The former problem has been dealt with within the HyperMeData project mentioned above in the motivation part. The results of the project included design and implementation of a language called *DDL* (Data Description Language) that enables to describe formally various data schemas and, based on this description, also the transformations defining conversions of data instances between pairs of these schemas. In the next section 4, a natural extension to the language will be overviewed enabling to describe also hypertext/media documents and transformations of data instances between a data schema and a document schema. This extension builds on the analogy between relational data structures and hyperdocument structures, and fully utilises the principles of data transformations defined in DDL for pairs of data schemas.

A possible solution to the latter problem is proposed in section 5: a presentation document (for instance: a collection of Internet web pages) builds on the definitions of the two object types – section and reference – of the hyperdocument schema, aiming at the provision of a full variety of cross-linked information to the user.

4 DDL: Data, Documents, Transformations

The DDL language uses declarative descriptions for data schemas (entities, ISA entities, simple associations, complex associations) and document schemas (sections, ISA sections, simple references, complex references), and functional expressions for constraints inside the schemas and also rules defining transformations of data instances between pairs of the schemas. Functional data expressions are evaluated over a schema instance, and are based on a functional language (see e.g. [4]). Besides common functions and operators for numeric arithmetics and list processing, the language defines several special operators for accessing data in schema instances, namely, operators '+>' or '–>' providing for a particular instance of an entity figuring in a binary association the list of all instances of the associated entity, or the first element of that list, respectively.

4.1 Data Definition

The data definition part of the DDL language has been described in detail in [5] and also briefly overviewed in [2]. Here we shall only demonstrate its use on our running

example (prefixes *PK_* and *FK_* are used to denote primary and foreign keys, respectively):

```
DATA School

ENTITY Person
  HAS PK_Person: integer; Surname: char[30]; Name: char[15];
      TitleA: char[15]; TitleB: char[10]; Street:…;
      Number:…; Zip:…; City:…; Phone:…; Fax:…; Email:…;
  KEY PK_Person; UNDER PK_Person<>null; Surname<>""; END

ENTITY UniWorkpl
  HAS PK_UniWorkpl:…; FK_HeadWorkpl:…; FK_Address:…;
      FK_HeadPers:…; Name:…; Profile:…; KEY … END

ENTITY Address
  HAS PK_Address:…; Street:…; Number:…; Zip:…; City:…;
  KEY … END

ASSOC HeadWorkpl
  CONN sub: UniWpl[0,*], super: UniWpl[0,1]
  WHEN sub.FK_HeadWorkpl==super.PK_UniWorkpl; END

ASSOC UniWpl_Pers HAS FK_UniWorkpl:…; FK_Person:…; Func:…;
  CONN  UniWpl[1,*], Person[1,*]
  WHEN  FK_UniWorkpl==PK_UniWorkpl AND FK_Person==PK_Person;
  KEY   FK_UniWorkpl, FK_Person; END

ASSOC UniWpl_Addr CONN UniWpl[0,*], Address[1,1] WHEN … END

ASSOC HeadPers CONN UniWpl[0,*], Person[1,1] WHEN … END

END School;
```

4.2 Document Definition

Similarly to data definition, in its document definition part the DDL combines features of three levels of modelling: *conceptual* (description of sections, roles, references, complex data types, etc.), *logical* (a set of instances – data tuples – corresponds to every conceptual object having attributes), and *intensional* (constraints specify how conceptual objects correspond to logical sets of tuples).

The language differentiates two *document object types* – *section* and *reference*:

```
SECTION section_name
  HAS     <attributes declarations – names, labels, and types>
  ISA     ancestor_section_name WHEN <ISA condition>
  KEY     <key attributes qualifications>
  UNIQ    <unique attributes qualifications>
  UNDER   <constraining condition>
  LINE    <link page attributes qualifications>
  INATR   <internal attributes qualifications>
END

REFER reference_name
  HAS     <attributes declarations – names, labels, and types>
  CONN    <connected sections declarations – names, labels, cardinalities,
```

```
            and SELF qualifications>
  WHEN      <connection condition>
  KEY       <key attributes qualifications>
  UNIQ      <unique attributes qualifications>
  UNDER     <constraining condition>
  LINE      <link page attributes qualifications>
  INATR     <internal attributes qualifications>
END
```

Besides all *attribute* declarations, the declaration of a *section* contains *constraints* defining properties of the conforming instances of this type (UNDER), and an optional *ISA clause* defining section hierarchy relationships and the inheritance of attributes. Additional clauses (LINE, INATR) are purely related to the presentation form, as discussed further in section 5. *Reference* is an object type defining relationships among sections, the instances being tuples of instances of section types. The declaration contains *referential predicate* specifying which tuples of sections are the elements of the reference (WHEN), *constraints* defining the properties of the conforming instances (UNDER), *cardinality constraints*, and a *list of attributes* in case of a *complex reference.* Additional clauses (LINE, INATR, SELF) again purely concern the presentation form of the document, and are discussed in section 5.

Using DDL, the running example is described as:

```
DOCUMENT School$"School XYZ"$"School"

SECTION Person$"People at the School"$"People"
  HAS PersonCode: integer; PersonName: char[75];
      StreetNumber$"Address": char[31]; ZipCity:…;
      Phone$"Telephone":…; Fax$"Fax":…; Email$"E-mail":…;
  KEY PersonCode; INATR PersonCode; LINE PersonName; END

SECTION Workplace
  HAS WplCode:…; Descr:…; KEY WplCode; INATR WplCode; END

SECTION Faculty$"Faculties at the School"$"Faculties"
  HAS FacCode:…; FacName:…; FK_FacAddr:…;
  ISA Workplace WHEN FacCode==WplCode;
  KEY FacCode; INATR FacCode, FK_FacAddr; LINE FacName; END

SECTION Department$"List of Departments"$"Departments"
  HAS DeptCode:…; DeptName:…; FK_DeptAddr:…;
      FK_Fac:…; FK_HeadDept:…; FK_Head:…;
  ISA Workplace WHEN DeptCode==WplCode;
  INATR DeptCode, FK_DeptAddr, FK_Fac, FK_HeadDept, FK_Head;
  KEY DeptCode; LINE DeptName; UNDER DeptCode<>null; END

SECTION Address$"List of Addresses"$"Addresses"
  HAS AddrCode:…; StreetNumber:…; ZipCity:…; KEY … END

REFER Fac_Addr CONN … WHEN … END

REFER Dept_Addr CONN … WHEN … END

REFER Fac_Head HAS FacId:…; PersonId:…;
  CONN Faculty$"Academic functions"[0,*],
       Person$"Faculty management"[1,1]
  WHEN FacId==FacCode AND PersonID==PersonCode;
```

```
  KEY  FacId, PersonId; INATR FacId, PersonId; END
REFER Fac_Dept CONN … WHEN … END

REFER Head_Dept CONN … WHEN … END

REFER Dept_Head
  CONN Department$"Heading departments"[0,*],
       Person$"Department head"[1,1] WHEN … END

REFER Fac_Pers HAS FacId:…; PersonId:…;
  CONN Faculty$"Member of faculties"[1,*],
       SELF Person$"People at the faculty"[1,*] WHEN … END

REFER Dept_Pers
  HAS DeptId:…; PersonId:…; Position$"Position":…;
  CONN Department$"Working at departments"[1,*],
       Person$"Employees"[1,*] WHEN … END

END School;
```

4.3 Data-Document Transformations

In general, transformation is a process of creating an instance of a *target* (*data* or *document*) *schema* from an instance of a *source* (*data*) *schema*, i.e., of creating lists of instances of *target schema objects* (entities and associations, or sections and references, respectively) from lists of instances of *source schema objects* (entities and associations). The transformation process is driven by *transformation rules* that decompose the transformation into blocks describing the way instances of one *target object* are constructed from instances of one or more *source object*(*s*) – see [5] (or the overview in [2]) for more detail.

In data-document transformations, the target objects are document *sections* or *complex references*, and the source objects are data *entities* or *complex associations*. In the running example, the transformation from *School* data schema to *SchoolDoc* document schema is defined as follows ('++' is a string concatenation operator):

```
FUNC getStrLink (str1, str2: string): string
  { if trim(str1)=="" and trim(str2)=="" then "" else ", "; }
FUNC getSupWpl (wpl: TYPEOF UniWorkpl): TYPEOF UniWorkpl
  { (super JOIN HeadWorkpl | sub := wpl); }
FUNC getTopWpl (wpl: TYPEOF UniWorkpl): TYPEOF UniWorkpl
  { if getSupWpl(wpl)==null then wpl
    else getTopWpl(getSupWpl(wpl)); }

TRANSF  SchoolDoc <- School

BUILD Person <- Person
  ASGN PersonCode:= gen_id(); PersonName:= trim(trim(Surname)
       ++" "++trim(Name))++getStrLink(TitleA,TitleB)++
       trim(trim(TitleA)++" "++trim(TitleB)); StreetNumber:=
       trim(trim(Street)++" "++trim(Number)); ZipCity:=…;
       Phone:=…; Fax:=…; Email:=…; END
BUILD Workplace <- UniWorkpl
  ASGN WplCode:= gen_id(); Descr:=…; END
```

```
BUILD Faculty <- UniWorkpl
  WHEN getSupWpl(UniWorkpl)==null;
  ASGN FacCode:= gen_FK(Workplace,UniWorkpl); FacName:=…;
       FK_FacAddr:= gen_fk(->UniWpl_Addr); END
BUILD Department <- UniWorkpl
  LET  supWpl := getSupWpl(UniWorkpl) WHEN supWpl<>null;
  ASGN DeptCode:= gen_FK(Workplace,UniWorkpl); DeptName:=…;
       FK_DeptAddr:= gen_fk(->UniWpl_Addr); FK_Fac:=
       gen_FK(Faculty,getTopWpl(UniWorkpl)); FK_HeadDept:=
       gen_FK(Department,supWpl); FK_Head:=
       gen_fk(->HeadPers); END
BUILD Address <- Address
  ASGN AddrCode:= gen_id(); StreetNumber:=…; ZipCity:=…; END
BUILD Fac_Head <- UniWorkpl
  ASGN FacId:= gen_FK(Faculty,UniWorkpl); PersonId:=
       gen_fk(->HeadPers); END
BUILD Fac_Pers <- UniWpl_Pers
  WHEN getSupWpl(UniWorkpl)==null;
  ASGN FacId:= gen_FK(Faculty,UniWorkpl); PersonId:=
       gen_fk(Person); END
BUILD Dept_Pers <- UniWpl_Pers
  WHEN getSupWpl(UniWorkpl)<>null;
  ASGN DeptId:= gen_FK(Department,UniWorkpl); PersonId:=
       gen_fk(Person); Position:=…; END
END SchoolDoc;
```

5 Presentation Hyperdocument

The user presentation hyperdocument based on the DDL definition allows browsing and viewing instances of the document schema – both the structure of the schema and the instances of individual schema objects (sections and references). The DDL document description therefore contains the minimum information needed for the presentation purposes: labels of schemas, objects, and attributes, the former two both in full and shortened versions.

The resulting hypertext or hypermedia presentation (for instance, a set of cross-linked WWW pages) then consists of:

- the *index* (or *map*) *page*;
- the *information pages*, either related to section objects or to section object instances; and
- the *link* (or *reference*) *pages* related to section objects.

The *index page* is unique within the document, showing the document structure as a collection of all section objects – or, more exactly, collection of *hyperlinks to link* or *information pages* of individual section objects. The collection may either be in form of a *list* (as shown in Fig. 3) or in form of some other, more sophisticated structure (e.g., a *graph* with nodes representing the sections and the references, and edges representing the interconnections). If presented in the list form the information *ordering*

on the index page may build on the proper DDL definition (*implicit* ordering), or on some more sophisticated (*explicit*) ordering specifically defined in a separate formatting description enhancing the DDL definition. In the former case, section object *full labels* are used, either ordered alphabetically or by the succession of object definitions in DDL, possibly nested level by level according to the partial ordering that can be automatically derived from the document schema structure. In the running example the simple text form of the index page would be that of Fig. 3 (successively ordered by DDL definitions).

Note: The following notation is used in Fig. 3–Fig. 5: [*PersonName*] stands for the *PersonName* attribute value, <>*School XYZ*<*idxPg*> means a hyperlink to the index page labelled *School XYZ*, <*Department infPg by Dept_Pers*> means a hyperlink to the *Department* instance information page as defined by the reference *Dept_Pers*, etc.

School XYZ

Page Index

- <>*People at the School*<Person lnkPg>
- <>*Faculties at the School*<Faculty lnkPg>
- <>*List of Departments*<Department lnkPg>
- <>*List of Addresses*<Address infPg>

...

Fig. 3. A simple document index page.

From the index page list, a reduced version – *reduced index page list* – is derived to be included in all information and link pages (see further) to provide for easier orientation and hypertext navigation. Object *short labels* are used here to label the hyperlinks to the respective link or information pages. By default, the reduced index page list includes hyperlinks to all document section objects. In a more sophisticated case – building on partial ordering of the document objects – only hyperlinks to the highest level objects are included: each of them, when activated by a mouse-click, besides navigating to the respective link or information page, also out-rolls the sublist of next level objects related by DDL references.

The *information pages* exist for all section objects – either for any individual *instance* of a particular section object (in the 1 : 1 sense), or for the section *object* as a whole (in the 1 : *m* sense uniting all instances in one presentation page) depending on whether the LINE clause is present or not in the respective DDL definition: if the LINE section is omitted only one information page is generated into the resulting document for that section object containing the complete collection of object instances chained into one list. The information page shows the series of all *attributes* (labels and values) of the respective section object instance or instances – together with all relevant hyperlinks, namely:

– hyperlink to the document *index page*;
– hyperlink to the respective *link page* (if there is any);
– hyperlinks to all *information pages* related by DDL reference definitions.

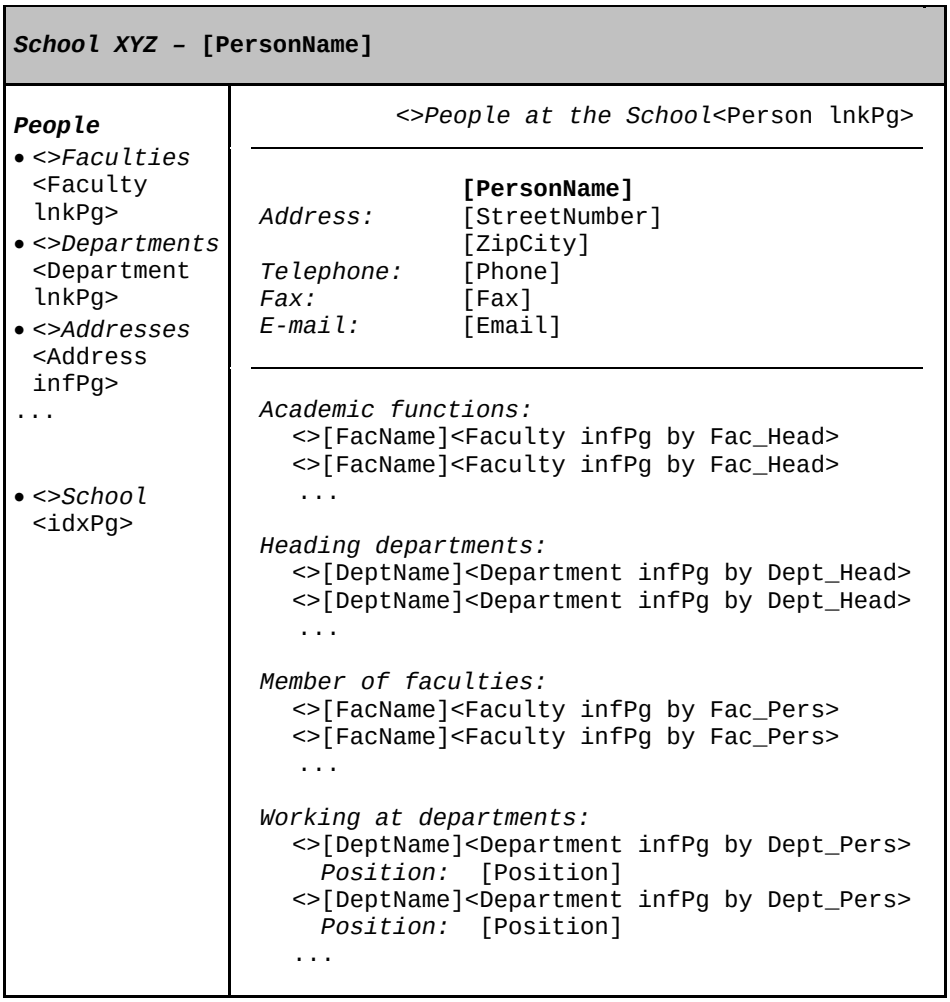


Fig. 4. Instance information page.

Similarly to the index page case, the ordering in the attribute section of an information page may be either implicit or explicit one, i.e. either may build on the DDL definition (a simple alphabetical list of attribute labels or a list ordered in accordance with the succession of attribute definitions in DDL), or may follow some other ordering rules defined in an attached formatting description. The *attribute labels* are specified in DDL section object definitions: if being empty no label for the respective value is output on the presentation page, if the attribute is included in the *INATR* clause neither the value is output (the *INATR* attributes are purely internal ones, exclusively serving to identification & hyperlink navigation). The *Person* instance information page of the running example is illustrated in Fig. 4.

The index page hyperlink is uniformly placed on all information pages as well as a hyperlink to the respective link page provided the link page exists: if the *LINE* clause is not included in the DDL definition of this object it means that the object link page

is identical with the object information page hence the self-referential hyperlink (leading from the information page to the link one) is omitted; a similar unification of object link page and object information page is done if section object cardinality equals 1. In both these cases also the hyperlink from index page goes directly to the object information page rather than to the link one.

The hyperlinks to other information pages related by DDL reference definitions are attached to the values of attributes realizing the references. In case of *m* : 1 references the hyperlinks are attached to the section instance own attributes while for each *m* : *n* reference, a separate *reference block* is generated containing multiple hyperlinks as defined by the respective LINE clause. If the LINE clause is missing the attribute labels and values of the referred section instance(s) are *directly included* into the referring information page instead of the hyperlinks. Should the reference block be placed on a separate page a *SELF* clause is used in the DDL definition of the reference. In case of complex references the reference own attributes are attached to the hyperlinks (see Fig. 4) or to the directly included attributes.

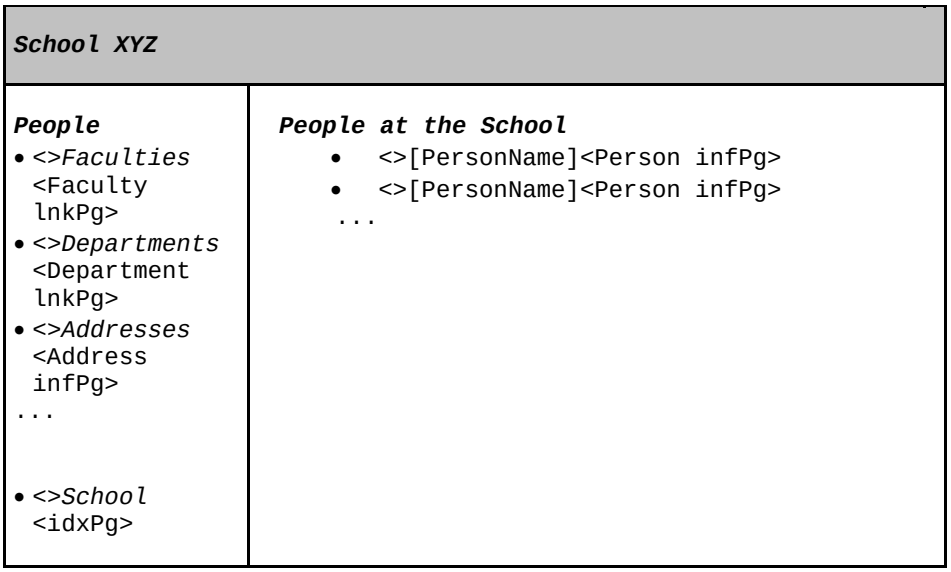


Fig. 5. Object link page.

The *link pages* only exist for document section objects having the LINE clauses included in their DDL definitions – one link page per object. The link page is named by the *long label* of the respective object (or by its proper name if the long label is missing), and shows the list of all *instances* of the object – or, more exactly, list of *hyperlinks to* the relevant *information pages* – together with the hyperlink to the document *index page*. If the object cardinality exceeds a given system limit, an instance search mechanism is output on the link page rather than the complete instance list.

Unlike the previous two cases, the ordering on the link page is derived from attribute values (*intensional* data) rather than attribute labels (*extensional* DDL definition), and the attributes in question are those specified in the LINE clause. The ordering may either be the implicit one, i.e. a simple alphabeti-

cal/alphanumerical/numerical ordering depending on attribute type, or be explicitly stated in an attached formatting description.

As mentioned earlier, the link page is omitted (a possible LINE clause in the DDL definition being ignored) if the respective section object cardinality (number of instances) equals 1 – in this case the index page directly refers to the only information page available. In our running example, the link page for *Person* (and similarly for *Department*) looks as in Fig. 5.

6 Directions for Future Work

The basic characteristics of a data & document description language called DDL have been overviewed in the previous sections. The document description and presentation aspects of the language have been focused on, and demonstrated on a simple illustrative example.

The language has been implemented and used to support transformations and hypermedia presentations of relational data structures in the medical domain. From a semi-routine use of the language it turned up that producing transformation descriptions (of either data-data or data-document transformations) is a most laborious task requiring (to be at all manageable) some supportive means. This is where current considerations on the given topic are being orientated raising a series of interesting questions that had to remain outside the scope of this paper. For all, let's mention the proposal of a series of meta-transformations covering typical schematic heterogeneities in multidatabases, as discussed recently in [3].

The author wishes to thank the CZ Ministry of Education for providing funding – within the CEZ:J07/98:143300004 research plan *Digital Libraries* – that supports work on the topics discussed or mentioned here, both at theoretical and application levels.

References

1. *CP 940943 HyperMeData* (internal project documentation). HyperMeData Consortium, 1998.
2. Kohoutková, J.: *Orientované grafy jako nástroj systémové integrace*. In: ITAT 2001 (Zborník príspevkov), Košice : UPJŠ Košice, 2001.
3. Kohoutková, J.: *Meta-Level Transformations in Systems Integration*. In: Proc. of ADBIS 2002. Bratislava : Slovak University of Technology in Bratislava, 2002.
4. Paton, N., Cooper, R., Williams, H., Trinder, P.: *Database Programming Languages*. Prentice-Hall, 1996. Chapter 4: *Functional Data Languages.*
5. Skoupý, K., Kohoutková, J., Benešovský, M., Jeffery, K.G.: *HYPERMEDATA Approach: A Way to Systems Integration*. In: Proc. of ADBIS'99, Maribor, 1999.

Organizing Image Documents with Self-Organizing Feature Maps *

Ján Antolík, Iveta Mrázová †

Department of Computer Science, Charles University,
Malostranské nám. 25, 118 00 Praha 1, Czech Republic

e-mail: antolikjan@hotmail.com, mrazova@ksi.ms.mff.cuni.cz

Abstract

A rapid development in recent technologies for information systems enables us to store, retrieve and process data in a relatively convenient manner. Anyway, especially when dealing with huge amounts of high-dimensional data (e.g. pictures or spatial maps stored in large image databases) we are facing a lot of - sometimes mutually contradicting - requirements. In particular, these requirements refer to a quick and reliable but robust information storage and retrieval. Also, tools for an easy but dynamical knowledge extraction and management should be provided for the system. Let us consider e.g. a large image database with an efficient search-engine. Furthermore, we would sure appreciate such a kind of search-engine that would be able to "follow our kind of querying" and that could retrieve adaptively the correct data also for previously vague, incorrect or incomplete queries.

From this point of view, we will discuss in this paper the abilities of various known models based on self-organization. These models comprise the standard Kohonen model of Self-Organizing feature Maps - SOMs - and its modification which defines the current network topology dynamically as the minimum spanning tree over the neurons. The other two examined models - namely the so-called Tree-Structured Self-Organizing feature Map (TS-SOM) and the Multi-Layer Self-Organizing Feature Map (MLSOFM) - employ a hierarchical structure. The first model represents in the clustering process a quicker top-down approach, whereas the second one corresponds to a bottom-up strategy yielding in general more reliable results. Result of supporting experiments performed so far with large sets of images will be discussed here in more detail, too.

*This research was supported by the grant No. 300/2002/A INF/MFF of the GA UK and by the grant No. 201/02/1456 of the GA ČR.

†Currently Fulbright Visiting Scholar in Smart Engineering Systems Laboratory, Engineering Management Department, University of Missouri-Rolla, Rolla, MO 65409-0370, USA.

1 Introduction

The emerging progress in the development of new technologies allows to process efficiently huge amounts of data. Anyway, in order to develop an "intelligent" search-engine over a large database of bitmap pictures we would like to process not only very specific and precisely defined queries, but also requests for groups of pictures with some common properties, that could be stated pretty vague. Using for this purpose e.g. the main idea of full-text search-engines, we would associate every picture in our database with a short textual description of its contents. Then, the search-engine can then perform a search over these descriptions in a full-text mode, using the keywords presented by the user.

However, such a system cannot build the database automatically from raw data – before storing it, the data has to be preprocessed. Moreover, for large databases of arbitrary images it is often difficult to transform natural queries into a very specific form which could be processed by a computer. One way how to get along this problem is to develop a suitable structure for organizing the documents in the database, and find the desired documents in an iterative search/query process. Here arises a new problem: How to organize the database of documents? A possible technique for solving this task could be to use vector quantization (self organizing maps) or one of its numerous modifications.

For this purpose, each image (document) should be described by some means that can be automatically computed from the given picture and correspond with the graphical nature of the documents. An example for such means are colour histograms or mathematical descriptions of textures. For each object to be stored in the database, an array of the considered descriptions forms a feature vector. Certainly, we should be able to measure the similarity between any two such feature vectors. According to the degree their mutual similarity, these feature vectors can be arranged automatically in the database. Now we can build the search-engine such that the searching process will be iterative and the user could successively locate the requested picture – or group of pictures – according to similarity between the already exploited pictures (the degree of their similarity is implicit contained in the structure of the database).

One of the most promising classes of algorithms, that have the ability to automatically build a structure in which the presented vectors are arranged according to a given similarity measure are the SOM algorithms and their hierarchical versions. In our experiments, we have tested three different SOM-models and two types of topology - one static and one dynamic. The tested models comprise in particular the basic SOM , the TS-SOM model and the ML-SOFM model combined with the static rectangular topology and with the dynamical spanning tree topology.

The main aim of the basic SOM-algorithm is to spread the neurons in the input space such that they approximate the input pattern density as closely as possible. The other tested models incorporate a hierarchical structure into the basic model. In both cases the network consists of multiple layers. Each layer is a basic SOM-network. The TS-SOM model represents a top-down approach whereas the ML-SOFM represents a bottom-down approach. In the TS-SOM model there exists a mapping defined between layers. This mapping restricts the set of neurons among which the winner neuron is

looked for. This way the output of the network - the winner neuron in the bottom layer can be found faster. The basic idea of the ML-SOFM model is even simpler. The input is presented to the bottom layer and the winner neuron is computed. As the input of the other layers the position vector of the winner neuron of the previous layer is used.

2 The Basic Self-Organizing Map

In the standard model of Self-Organizing Maps (SOM) [2], the set of all N neurons is arranged in a two-dimensional rectangular lattice. Each neuron $i(1 \leq i \leq N)$ is associated with a weight vector $\vec{m}_i$ of the same dimension as the input space $\vec{m}_i = [m_{i1}, m_{i2}, ..., m_{in}] \in R^n$. Further, let we have the training set $X = \{\vec{x}_p : \vec{x}_p = [\xi_{p1}, \xi_{p2}, ..., \xi_{pn}] \in R^n, 0 < p < \infty\}$. Assumed that we have a metric ρ defined over R^n we say, that the neuron c is the winning neuron if $c = argmin_{i \in N}\{\rho(\vec{m}_i, \vec{x})\}$, i.e. if the weight vector of the neuron c has the smallest distance from $\vec{x}$ according to the metric ρ – often chosen as the Euclidean distance. For any presented input only a single neuron - the best representant of the presented input pattern will be active. During training, the SOM-algorithm adjusts iteratively the weights of the winning neuron and its neighbours towards the presented input patterns from the training set.

The learning algorithm for the basic SOM :

1. Initialize the parameters of the SOM learning algorithm, by setting the size of the network, the number of iterations, learning rates and neighbourhood function.
2. Initialize the weight vectors of the neurons in the network randomly
3. Present a pattern $\vec{x} \in X$ from the training set.
4. For every neuron $n \in N$ compute the distance of its weight vector $\vec{m}_n$ to the pattern $\vec{x}$.
5. Select the winning neuron c as the neuron $j \in N$ with the minimum distance $\rho(\vec{m}_j, \vec{x})$
$$c = argmin_{i \in N}\{\rho(\vec{m}_i, \vec{x})\}$$
6. Adjust the weight vectors of all neurons $i \in N$ according to the formula
$$\vec{m}_i(t+1) = \vec{m}_i(t) + h_c(i,t)[\vec{x}(t) - \vec{m}_i(t)]$$
where $t = 0, 1, 2, ...$ is the discrete time coordinate and $h_c(i,t)$ is the neighbourhood function.
7. If the maximum number of iterations is not achieved go to Step 3.

The function h_c is a function of distance of neuron i from the winning neuron c and time:
$$h_c(i,t) = \begin{cases} \alpha(t), & i \in N_c \\ 0, & i \notin N_c \end{cases}$$

where α corresponds to the learning-rate ($0 < \alpha(t) < 1$). N_c defines rectangular neighbourhood (usually decreasing in time) over the set of neurons centered at neuron c. Usually training proceeds in two phases. During the first phase, the size of the neighbourhood and the elasticity of the network (i.e. its learning rates) are relatively high. With their decrease in time, an initial arrangement of neurons is formed. The second phase, which is usually longer than the first one, attains the fine tuning of approximation. After the ordering phase the learning rate and size of the neighbourhood should stay small. Setting of a sufficient number of iterations is important for the convergence of the network.

The above discussed properties of the standard SOM comprise their ability to reduce the dimensionality of input data by mapping them onto a less-dimensional lattice of neurons. This mapping often preserves the topology of the original data, which ensures that the structure of the data and inter-relationships between them are not lost. The next advantage is the ability to approximate the probability distribution of the input data, by allocating neurons such, that the density of them in the given area corresponds to the density of the input data. This property can be further enhanced by using dynamical types of topology which can be defined e.g. as minimum spanning tree [2]. Numerous application areas of SOMs include e.g. computer visualization, data mining, computer vision, image processing, databases or speech recognition. Their main limitations refer to their relatively high computational costs (necessary to find the winning neuron).

3 Hierarchical SOM-models

In many problems, the data embody a hierarchical structure. In order to incorporate a kind of hierarchy into the basic SOM model we could use not only a single lattice of neurons but arrange multiple SOMs in several layers forming a hierarchy. Except the ability to represent data at multiple levels of abstraction, other improvements of the basic SOM algorithm can be achieved, such as reduction of the size of the set in which the winning neuron is looked for. In this paper, we will examine two variants of hierarchical SOMs – the so-called Tree-Structured SOM and Multi-Layer Self-Organizing Feature Map.

3.1 Tree-Structures Self-Organizing Maps (TS-SOM)

In TS-SOM, the neurons are arranged in a finite number of layers. Each layer is a basic two-dimensional SOM. The first layer consists of a single neuron. From each but the deepest layer i a mapping Z_i to the following layer is defined - each neuron from the previous layer is mapped on a set of neurons in the next layer. There is one common definition of Z_i which we will describe here. Let us assume to have a constant d and each layer has a simple rectangular topology. When layer j is a $(k \otimes k)$ lattice, the following layer will be a $(dk \otimes dk)$ lattice and the neuron n from the layer j with the coordinates $[x, y], (x \in \{0, 1, .., k-1\}, y \in \{0, 1, .., k-1\})$ will be mapped on a "rectangle" of neurons

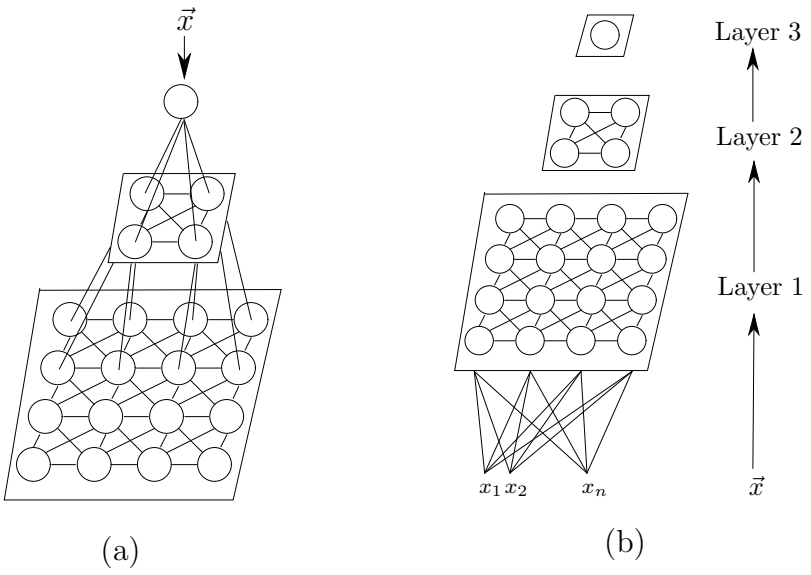


Figure 1: (a) A TS-SOM with the 2D-rectangular topology (b) A MLSOFM with the 2D-rectangular topology

from the next layer which will have the following coordinates:

$$Z_j(n_{[x,y]}) = \{m_{[dx+o,dy+p]} : o, p \in \{0, 1, ..., d-1\}\}$$

where $n_{[x,y]}$ is a neuron with the coordinates $[x, y], x \in \{0, 1, .., k-1\}, y \in \{0, 1, .., k-1\}$ in the layer j and $m_{[a,b]}$ is a neuron with the coordinates $[a, b], a \in \{0, 1, .., dk-1\}, b \in \{0, 1, .., dk-1\}$ in the layer $j+1$.

The TS-SOM learning algorithm uses a modification of the basic SOM learning algorithm. Assume we have the training set $X \in R^n$, the metric ρ defined over the input space $I \in R^n$ and z is the number of layers in the TS-SOM. Let us define $c_i, i \in 1, 2, ..., z$ as the winning neuron of layer i. For each input vector $x(t)$ the TS-SOM learning algorithm adjusts neurons in all layers. It starts at the top layer and proceeds deeper. In each layer it uses the basic SOM learning algorithm with a new algorithm for selecting the winning neuron. Winning neuron of each but the top layer is selected according to the winning neuron from previous layer. At first a set of neurons is computed, such that it is the union of images (according to the mapping Z) of all neurons from the neighbourhood of the winning neuron of the preceding layer. The winning neuron of the current layer is then selected from this set according to the same rules as in the basic SOM algorithm.

The learning algorithm for TS-SOM :

1. Initialize the parameters of the TS-SOM learning algorithm - set the number of layers, size of each layer, the mapping between layers, number of iterations, learning rates and the neighbourhood function.
2. Initialize the weight vectors of the neurons in all layers randomly
3. Present a pattern $\vec{x} \in X$ from the training set.
4. For each layer i of the network ($i = 0, 1, 2, ..., z$) do :

   (a) if $i = 0$ then S contains a single neuron of the top layer, else let us define a set Y_u of neurons as the image of neuron u from layer $i-1$ according to the mapping $Z_{i-1}(u)$. Then, the set of neurons S corresponds to $S = \{\bigcup_{o \in N_{c_{i-1}}} Y_o\}$, where $N_{c_{i-1}}$ is the neighbourhood of the winning neuron c_{i-1} from the layer $i-1$.
   (b) For every neuron $n \in S$ compute the distance of its weight vector $\vec{m}_n$ from the vector $\vec{x}$.
   (c) Set the winning neuron of the layer i, c_i, as the neuron with the minimum distance:

   $$c_i = argmin_n\{\rho(\vec{x}, \vec{m}_n) : n \in S\}$$

   where $\vec{m}_n$ is the weight vector of neuron n
   (d) Adjust the weights of all neurons in the layer i according to the following formula

   $$\vec{m}_j(t+1) = \vec{m}_j(t) + h_c(i,t)[\vec{x}(t) - \vec{m}_j(t)]$$

   where t = 0,1,2,... is the discrete time coordinate and $h_c(j,t)$ is the neighbourhood function.

5. If the maximum number of iterations is not achieved go to Step 3.

Successfully, the TS-SOM-model was applied in the image retrieval system **PicSOM** developed by *Laaksonen, et al.* [5], [4]. The PicSOM provides a kind of iterative search engine over a database of 4350 pictures (aircrafts, building and human faces). For each stored picture, a set of feature vectors is computed [6] – average colour components and texture characteristics. For each query, a set of "similar" images (from the image collection) is presented to the user. From them, the user selects those – from his point of view – most similar images. Then, the value of the neurons in all layers which correspond to the feature vector of the selected image are increased. Similarly when the user rejects an image the value of the corresponding neurons is decreased. The mutual relationships of positively evaluated neurons that are located nearby is further enhanced by convolving each layer of neurons by a low-pass filter after each query. This way, similar images should be located close to this image in the SOM and they should also be presented to the user. A similar strategy was also adopted in the WEBSOM-system [1], [2] designed for storing text documents.

3.2 Multi-Layer Self-Organizing Feature Maps (MLSOFM)

A Multi-Layer Self-Organizing Feature Map (MLSOFM) [3] consists of multiple layers, each of one corresponds to the basic SOM-model. The bottom layer is the largest one and as we proceed up to higher layers their number of neurons decreases. The top layer usually consists of a single neuron. Unlike the TS-SOM learning algorithm, which follows the top-down approach, the MLSOFM learning algorithm starts the adaptation process in the bottom layer and proceeds successively to higher layers. Only the bottom layer of MLSOFM is connected directly to the input vector via weighted connections. The inputs of higher layers are the outputs of their preceding layer. In this way a kind of hierarchical clustering is formed, where each but the bottom layer computes a "clustering of clusters" of the lower layer. The input for the bottom layer is the vector $\vec{x}$ and the input for other layers is the weight vector of the winning neuron from the lower layer. Therefore, it is necessary for a successful MLSOFM-training first to adjust the weight vectors at the bottom layer, and then – after their stabilizing –to proceed to higher layers.

The learning algorithm of the MLSOFM :

1. Initialize the parameters of the MLSOFM learning algorithm, by setting the number of layers, size of each layer, the number of iterations, learning rates and the neighbourhood function.
2. Initialize the weight vectors of the neurons in all layers randomly
3. Set the presented pattern $\vec{x}$ from the training set to $\vec{y}_1$.
4. For each layer $i = 1, 2, ..., z$ do (note that layer number one is the bottom layer - the largest layer unlike in the TS-SOM algorithm):

   (a) Use the vector y_i as the input of layer i.
   (b) For every neuron n in layer i compute the distance of its weight vector $\vec{m}_n$ from the vector $\vec{y}_i$.
   (c) Set the winning neuron of the layer i, c_i, as the neuron with the minimum distance:

   $$c_i = argmin_n\{\rho(\vec{y}_i, \vec{m}_n) : n \in S\}$$

   where $\vec{m}_n$ is the weight vector of neuron n
   (d) Assign the value of the weight vector $\vec{m}_{c_i}$ of the winning neuron c_i of layer i to the vector $\vec{y}_{i+1}$.
   (e) Adjust weight vectors of all neurons in the layer i according to the following formula

   $$\vec{m}_j(t+1) = \vec{m}_j(t) + h_c(i,t)[\vec{y}_i(t) - \vec{m}_j(t)]$$

   where t = 0,1,2,... is the discrete time coordinate and $h_c(j,t)$ is the neighbourhood function.

5. If the maximum number of iterations is not achieved go to Step 3.

Koh et al [3] applied the MLSOFM-model to range image segmentation. A range image is usually formatted as an array of pixels (pixel grey values encode the depths or the distances of points on a visible scene surface from the range sensor). In this way, a hierarchy of clusters of range image pixels can be formed. At the bottom layer small but very homogeneous regions are found. At higher layers, the SOMs "glued" smaller regions from preceding layers according to their mutual similarity. Moreover, the MLSOFM-model proved to overcome some disadvantages of standard vector quantization techniques. In particular, the regions identified by standard vector quantization methods are not guaranteed to be spatially connected in terms of image coordinates and the number of neurons has to be known a priori.

4 Experimental Results

In our experiments, we tested the basic SOM-model and its hierarchical modifications – TS-SOM and MLSOFM (both of them having five layers). A symmetric square neighbourhood was used in the experiments with the rectangular topology (SOM,TS-SOM and MLSOFM). In the case of the spanning tree topology (SOM and MLSOFM), symmetric neighbourhood (comprising all the neurons within the given distance from the central neuron) was used as well. For the test, two different databases of bitmap pictures were involved. The first database was created by decomposing a large gray-scale aerial photograph into 1721 60x60 pixel bitmaps. The second database consisted of 364 colour bitmaps of three different types - photographs of airplanes, undersea photographs and manga pictures of varying size. Both databases were used for testing all of the above specified configurations except the TS-SOM with spanning tree topology (this architecture was tested only for the aerial photograph database).

For each bitmap in both databases, a feature vector was computed in following way: each bitmap was divided into 9 equally sized areas (3x3 chess-board). For each of these nine areas two different features were computed, namely an average colour and a texture descriptor. The average colour of gray-scale bitmaps was described by an single byte, for the colour bitmaps three bytes were needed - one for each of the red, blue and green colour components. As a texture descriptor a pixel neighbourhood was used. For each of the 8 possible neighbouring pixels a probability that the colour intensity of the central pixel is higher than the colour intensity of the given neighbouring pixel was computed. This probability was then scaled to the $[0 - 255]$ interval. In this way two training sets of vectors were created. The size of vectors in the first training set was 81 (for each of the 9 areas 1 byte for average colour and 8 bytes for texture descriptor - $9x9 = 81$) and in the second 99 (3 bytes for average colour and 8 bytes for texture descriptor in each area - $11 * 9 = 99$).

Four different tests were applied to some of the combinations of configurations and training sets. Their main purpose was as it follows:

1. evaluate and compare the ability of tested models to develop well structured picture

databases

2. test the robustness – in particular noise resistance – of considered models
3. examine the reliability of the "focused" winner search in TS-SOM (consider various size of searched neighbourhood)
4. visualize the mapping (relationship) between neurons in the last two layers.

In all test following parameters were used: the nummber of learning cycles was set to 10000, the learning rate factor was defined as $0.05 * (1 - t/10000)$ where t is the current learning cycle and the size of the neighbourhood was defined as $15 - (t/10000) * 15$. In the case of the first training set the size of the input vector was 81, in the case of the second 99 bytes. The static square topologyused the standard Euklidian metric. In spanning tree topology, metric was defined as the length of the path between the given two neurons in the tree defined by the topology.

The first test was aimed on visualization of the distribution of neurons of the given SOM in the input space. For each layer of the given SOM (we consider that a basic SOM has a single layer) a image was created. Because two different topologies were involved in the experiments two different visualization algorithms were used. In both cases all layers were calibrated with the training set after the adaptation phase finished. Each neuron was then represented by the bitmap belonging to the vector which the calibration process assigned to it. Down-scaled versions (100x100 pixels) of the pictures from the second database were used. In the case of square topology the final image was created such, that at the position [i*60,j*60] (or [i*100,j*100] when the second training set was used), the bitmap assigned to the neuron placed at position [i,j] in the lattice of neurons was pasted. In the experiments with spanning tree topology, the tree of the neurons was rooted at the first neuron and drawn onto the bitmap such, that each neuron in the tree was in the final image represented by the bitmap assigned to it.

In the TS-SOM algorithm the neighbourhood is also involved in the process of selection of the winner neuron. Hence the properties of the neighbourhood influence the performance of the whole algorithm. In our experiments we have used a square neighbourhood. The influence of size of this neighbourhood on the optimality of the winner neuron in the last(deepest) layer was examined in the second test. For all examples in the training set following process was performed: At first optimal winner neuron (over whole layer) was computed for the last layer. Then for each but the last layer a winner over whole layer was computed (an optimal winner for this layer). Then successively from each layer the winner selection algorithm of the TS-SOM model, starting with the optimal winner as the winner in this layer, was executed producing a new winner in the last layer. 4 values computed in this test indicate the result. The i-th number is percentile of cases (over the training set) when the winner in the last layer selected by the execution of the winner algorithm from the i-th layer is the same as the optimal winner neuron in the last layer.

The visualization of topological relationships (in the input space) between the last two layers in the TS-SOM model were the aim of the third test. To each neuron in the 4-th layer a colour is assigned so that similar colours correspond at least partially

with topologically similar neurons. Also neurons in last layer have an colour assigned in following way: a position vector of the given neuron is forwarded to the 4-th layer an the winner neuron for this vector is computed. The colour of the winner neuron is then assigned to the current neuron. As the output two colour images were created. The first describes the 4-th and the second the 5-th layer. The final images are created such, that at the position $[i * 60, j * 60]$ a rectangle of the same colour as the colour of the neuron at position $[i, j]$ in the lattice of neurons is drawn.

The last test examines the ability of the different models to resist noise. This test was performed only with first database of bitmaps and the basic SOM and TS-SOM model. For each level of noise a new training set was created such, that the feature vectors of the new training set were computed from original bitmaps which were altered by a noise filter of given level. The p percentile noise was .produced. by repeating following operation $[(number\ of\ pixels\ in\ bitmap) * (p/100)]$ times: swap two random pixels in the bitmap. After the adaptation phase of the network was finished, all vectors from both the original and new training set were presented to the network. The output of the test for the given level of noise was the percentile of matches, when the original feature vector and new feature vector computed from the bitmap altered by noise activated the same winner neuron.

5 Conclusions

In general, the tests affirmed the relevance SOM-based algorithms for the development of large image databases even though the static topology and a relatively low noise resistance (test 2) represent – at least in our opinion – their main limitation. Anyway, visualized distribution of neurons in the input space revealed that the tested configurations are able to group the pictures according their mutual similarity (test 1). In the case of the rectangular topology and the second training set almost homogeneous areas with single type of pictures were formed in the output images. When the first training set was used the SOM-models clustered together the map pieces of the sea, land or sand areas. Also some indication for grouping of settled areas can be found in the output pictures. On the other hand, although the output pictures for tests with the spanning tree topology were less descriptive, apparently similar pictures were assigned to the same branches of the tree.

The last two tests confirmed our assumption that larger neighbourhoods rise the frequency of finding optimal winner neurons, but we failed in finding a reliable explicit relationship between the starting layer and the success of the search. Also the visualized topological relationships between neurons in last two layers (test 4) of the tested TS-SOM models reveals that it is in general very difficult to achieve configurations where the trained SOM-networks in different layers would correspond to the same areas of the input space. Although most homogeneous areas of input space identified in one layer can be found also in the second layer, frequently different parts of the lattice are occupying the same area.

References

[1] T. Kohonen: *Self-Organization of Very Large Document Collections: State of the Art*, in: I. Niklasson, M. Boden and T. Ziemke (Eds.): Proc. of ICANN'98, Vol. 1, pp. 65-74, Springer-Verlag, (1998)

[2] T. Kohonen: *Self-Organizing Maps*, Springer-Verlag Heidelberg, (2001)

[3] J. Koh , M. Suk, S. M. Bhandarkar: *A Multilayer Self-Organinzing Feature Map for Range Image Segementation*, in: Neural Networks, Vol. 8, No. 1, pp 67-86, (1995)

[4] M. Koskela, J. Laaksonen, S. Laakso, E. Oja: *The PicSOM Retrieval System: Description and Evaluations*, in: Proceedings of CIR'2000, Brighton, UK, (May 2000)

[5] J. Laaksonen,M. Koskela, E. Oja: *Application of Tree Structured Self-Organizing Maps in Content-Based Image Retrieval*, in: Proceedings of ICANN'99, Edingburgh, UK, (September 1999)

[6] J. Laaksonen, M. Koskela, S. Brandt: *Analyzing Low-lLevel Visual Features Using Content-Based Image Retrieval*, in: Proc. of ICONIP'2000, Taejon, Korea, 6pp., (November 2000)

A Test outputs

Model	5% noise	10% noise	20% noise
Basic SOM	0.415	0.173	0.073
TSSOM	0.490	0.291	0.197
Basic SOM with STT	0.184	0.225	0.065

Table 1: Results of the robustness test

Training set	Size of neighbourhood	Layer 1	Layer 2	Layer 3	Layer 4
1st	0	0.05812	0.05812	0.75581	1.39535
1st	1	2.32558	2.32558	2.32558	7.44184
1st	2	11.2204	11.2204	13.8953	10.4651
2nd	0	1.37363	1.37363	1.37363	6.31868
2nd	1	16.2088	16.2088	16.2088	27.1978
2nd	2	35.1698	35.1698	32.963	22.2527

Table 2: Results of the second test (see the section 'Experiment results')

Figure 2: The original areal photograph

Figure 3: Visualisation of the Basic SOM after the training phase

Figure 4: Visualisation of the first layer of the MLSOFM model

Figure 5: Visualisation of the last (5th) layer of the TSSOM model

DDDS - The Replicated Database System

David Bednárek, David Obdržálek, Jakub Yaghob and Filip Zavoral
Department of Software Engineering
Faculty of Mathematics and Physics, Charles University Prague, CZ

Abstract

The object of our project is to develop effective client-side caching and data replication scheme so that clients access data as much locally as possible. To achieve such goals, we aim our effort at a specific class of applications, whose data queries are much more frequent than inserts and updates and whose structure of queries is known in the phase of the application development. For that purpose, we additionally suggest a client-side data access language capable to utilize the power of our data replication scheme.

1 Project goals

Contemporary data storage architectures are strongly server-oriented: the server handles almost all activities; clients care about constructing their requests and retrieving data supplied by the server. In the last years, the number of enterprise-wide and mainly internet-based data-oriented applications used by huge number (tens of thousands to millions) of users rapidly grows [2]. It is commonly expected that this trend will continue; looking ahead, the most successful companies are beginning to develop, implement and use e-business applications, they capture the advantages the internet brings, without abandoning their existing investments in systems and data.

In the fast-moving internet economy, the database servers require the most powerful hardware available and, in some cases, even this is not enough. One approach to overcome this problem is to distribute the computing load of a database server among several application servers and to adopt the three-layer architecture (fig. 1). Nevertheless, there are serious problems with data consistency in most implementations [3], usually solved by the mechanisms of distributed databases. Although the concept of database distribution has been adopted by many currently used database engines, the problem of keeping full data consistency and availability in a distributed environment in association with full SQL functionality required of such engines frequently leads to a significantly worse performance in comparison to centralized solutions.

This research is supported by Intel Corp. PO#4507012020
http://ulita.ms.mff.cuni.cz e-mail: {bednarek, obdrzalek, yaghob, zavoral}@ksi.ms.mff.cuni.cz

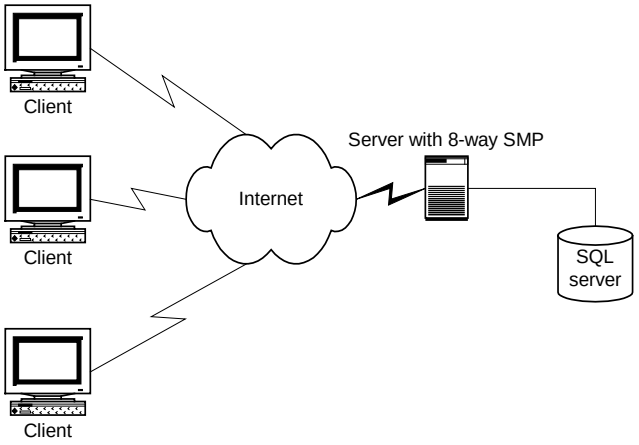


Fig 1 - Usual current state

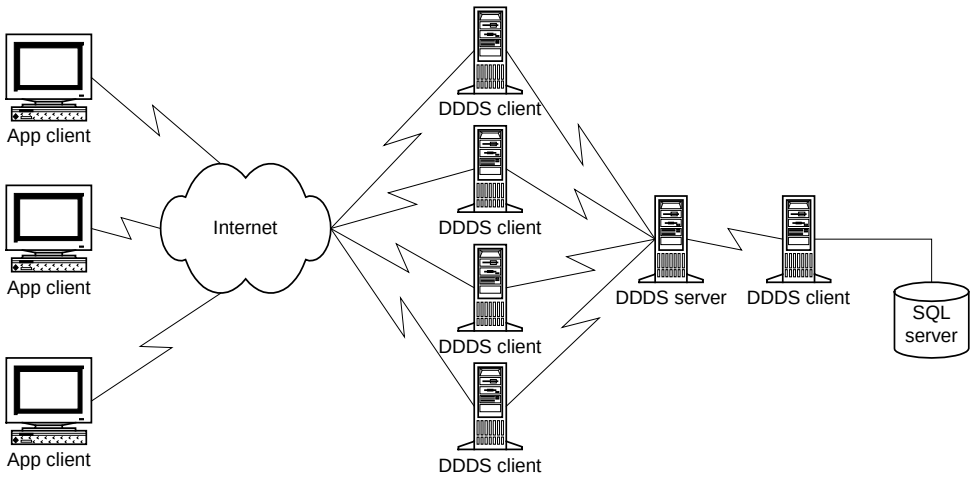


Fig 2 – Application structure using dedicated DDDS nodes

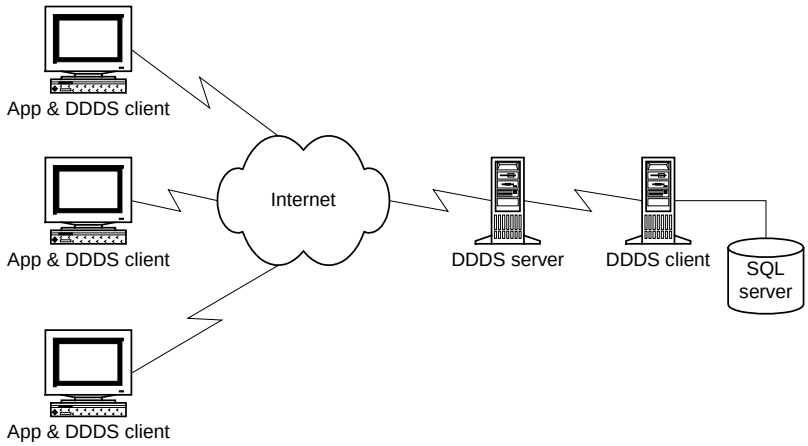


Fig 3 – Application structure using client-side DDDS computation

Unlike most research projects in the area of distributed databases ([1], [4], [6], [8]), the DDDS project does not deal with splitting data (rows, clusters or tables) among several sites and trying to optimize queries. The object of our project is to develop effective client-side caching and data replication scheme ([5], [7]) so that clients access data as much locally as possible. To achieve such goals, we aim our effort to a specific class of applications, whose data queries are much more frequent than database changes and whose structure of queries is known in the phase of the application development. Exactly this class of applications is often used in the e-business; application are specially tailored to provide pre-defined services with dominance of data retrieval.

For that purpose, we additionally suggest client-side data access language capable to utilize the power of our data replication scheme. Human readable form of this language allows notation disambiguity and lucidity of query code. Moreover, this language allows to exploit explicit parallelisms for better employment of client resources.

Since the DDDS project strongly relies on memory caching, it is especially applicable on 64-bit architectures and their large virtual address space.

Client-side caching and computing allow to distribute server workload among DDDS clients with various options, like fig. 2, 3.

This project is focused mainly in data accessibility and coherence; the higher levels of application logic such as service brokers, fault tolerance, and application-level transaction management are subject of a linked research project.

2 Database architecture

The DDDS system is a hierarchically organized replicated database system. The server keeps the whole database while the clients maintain partial mirrors. Data retrieval and manipulation are handled by clients; the server collects and redistributes the updates among the clients.

Compared to conventional clustered systems, the update load is finally concentrated in one node (the server), while the read load is distributed among the clients. This asymetry corresponds to the expected class of applications where the read load is significantly higher than the update load. This architecture allows to achieve higher read throughput without the performance overhead of fully distributed write transactions.

The system asserts transaction isolation among clients at ISO level 3 - sequential equivalence. Clients may decide to lower their degree of isolation for individual transactions. Anyway, the server distributes only committed updates; uncommitted updates are visible only to their issuer. The isolation mechanism is based on optimistic commit-time conflict detection; clients may additionally apply conflict prediction throughout their transactions. Isolation conflicts are resolved through

client-initiated rollbacks. Since there is no locking, there are no deadlocks. There is a potential risk of starvation by repeated rollbacks; proper commit-priority scheme is required.

3 System structure

The DDDS is composed of one *server* and several *clients*. There are independent peer-to-peer *connections* between the server and the clients, the communication is provided by an UGNP protocol. The client-to-server channel is called *uplink*; the server-to-client channel is called *downlink*.

The server maintains the *primary* structure; the clients maintain partial *mirrors* and place *requests* for changes. The structure consists of *persistent* and *temporal objects*; persistent objects form the database while temporal objects reflect the connections, distribution of the data, pending changes to the data, and transactions.

Persistent objects are visible for the server and all the clients; each temporal object is attached to a connection and is invisible to the other clients. Each object is primarily controlled by the server; the clients may only place *requests* to change the state of the objects. Changes in the object state are called *events*; the object state and the *event ordering* are controlled solely by the server. The server keeps lists of objects replicated at clients and distributes all events of a particular object to all the clients that have a replica of the object. In this way, each client is directed by a stream of applicable events, the event ordering is determined by the message order in the downlink channel.

3.1 Table

Table is a set of records indexed by a primary key. Tables allow interval queries on their primary keys and exact matching by their primary keys. The primary key must be unique, non-updateable, and of well-ordered domain.

3.2 Reader

Reader is a downlink subchannel that carries data information from the server to the client. Each reader is associated with a table and with a value or a range of its primary key. When a reader is opened, the server sends to the client all table records with the value or within the range of the reader. Later on, the server forwards every committed update (including insertions and deletions) within the range of the reader to the client.

Each reader is associated with a group; the groups allow referencing more readers at once.

3.3 Writer

Writer is an uplink subchannel that carries update (including insert and delete) requests from the client to the server. The update requests are collected at the server until a commit or a rollback is requested on the writer.

Each writer is associated with a group; before a commit, all the readers in the group (and its subtree) are checked for transaction isolation conflicts.

4 The Client

The main part of the DDDS client is the DQI interpreter, which executes compiled queries. The client maintains a mirror of data specified by its readers. Updates are stored locally to the mirror BOBS (specially tailored b-tree) data structures and simultaneously sent through the corresponding writer. The server subsequently propagates the committed updates to the conflicting readers of the other clients.

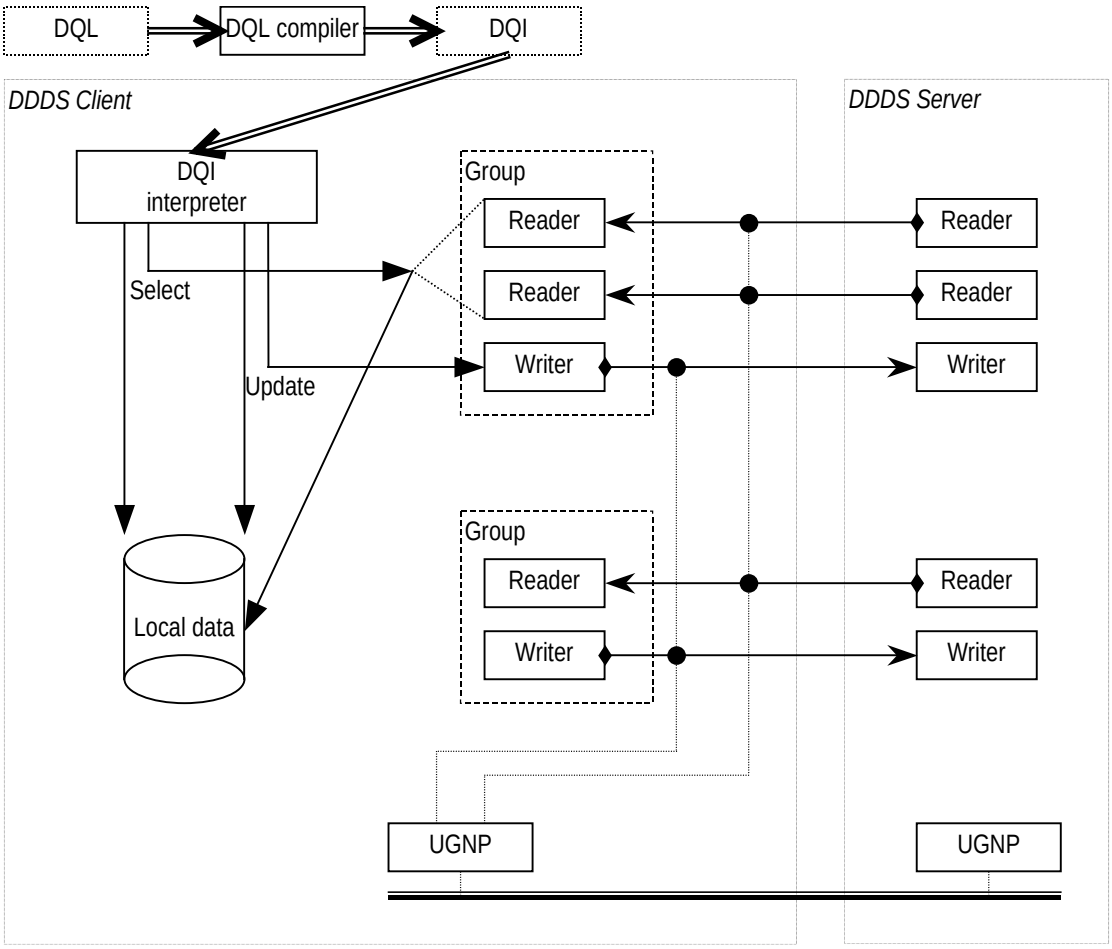


Fig 4 – Client side data flow

4.1 UGNP

UGNP is a proprietary high-performance message-oriented datagram-based network protocol. It offers standard features like optional reliable and serializable subchannels for one peer-to-peer connection, accurate packet-roundtrip measuring for early packet-loss prediction and a unique feature named scatter-gather channel.

5 The Server

The server collects updates sent by clients and received by writer channels. Data of these updates are held by update gates in their buffers. After the successfull commit, the data are stored into database and propagated to readers connected to changed data area.

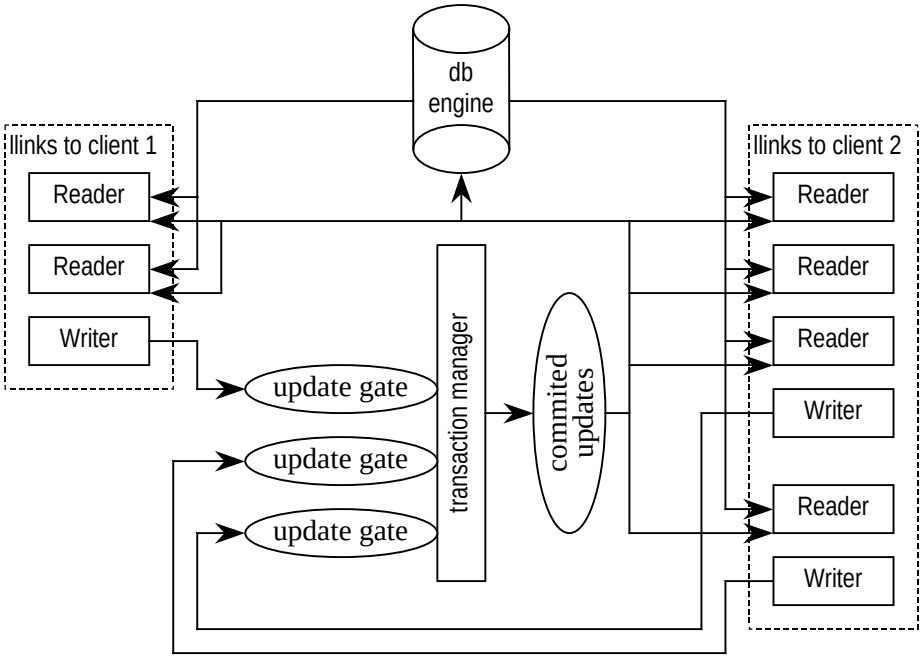


Fig 5 – Server side data flow

6 DQL/DQI

SQL is a widely accepted data query language. It has some advantages but there is a bunch of disadvantages. As advantages we may count:

It is widely accepted and implemented by current commercial available database servers.

It allows dynamically constructed queries.

Main disadvantages appear to be:

Its roots are 30 years old and this fact has clearly negative impact even on current version of SQL.

SQL server, namely its parts SQL parser and query optimizer, decides, how queries should be computed using general heuristic algorithms and table statistics.

There are no standardized language constructs for exploiting explicit parallelism.

The first disadvantage is well known from popular programming languages (e.g. C). Compatibility, dependency on the system, where the language has been born, and changes of program runtime environment cause problems to all programming languages (even to currently most popular ones, e.g. Java).

The second point goes by the universality of SQL. SQL parser and query optimizer do not know precise meaning of the given query in the context of the application. Therefore, for query executions they use general heuristic algorithms and table statistics in the better case, or the brute force in the worse case.

Different commercial products have different approaches to exploit explicit parallelism in their SQL clones. There is no standardized language construct for this. It disables better CPU utilization on large multiprocessor servers, because implicit parallelisms are usually undetected or badly detected.

We have addressed these three disadvantages of SQL and we propose novel data query languages - DQL (Direct Query Language) and DQI (Direct Query Instructions), which solve these disadvantages at the cost of sacrificing above-mentioned advantages of SQL. This decision has been made with respect to the intended class of applications.

The DQI is a binary code, which describes data and control operations of a client on an elementary level. It also offers interfacing to high-level languages. There is also human readable assembly-like language form DQA, which directly corresponds to DQI.

The DQL is a high level programming language designed for data queries with explicit parallelism and lucid query notation in the mind. It allows making queries in the most controlled (and thus the most efficient) fashion, and it hides index and join implementation in DQI from an application programmer for easy of use. A compiler from DQL to DQI will be constructed with optional DQA output.

The first advantage of the SQL lost in the DDDS project can be easily regained. A compiler from SQL to DQL could be constructed to achieve compatibility with SQL. The second one we do not consider important. The area of supposed applications doesn't require dynamically constructed queries; the set of applicable queries is known in the phase of application development.

7 Conclusions

The current state of the project is as follows:

The database architecture and data flows are specified

The caching scheme, data access and data changes propagation are specified and partially implemented

The draft of DQL is proposed

The communication layers are specified and implemented

We assume that distributed or internet-oriented applications based on DDDS would access data in more natural way implying their performance boost in comparison to contemporary systems. In addition, the application development would be easier and more straightforward and the final product would be more maintainable.

8 References

[1] Bouganim L., Florescu D., Valduriez P.: "*Dynamic Load Balancing in Hierarchical Parallel Database Systems*", INRIA, 1996.

[2] Florescu D., Levy A., Mendelzon A.: "*Database Technique for the World-Wide Web: A Survey*", INRIA, 1999.

[3] Florescu D., Levy A., Mendelzon A.: "*Run-time Management of Data Intensive Web-sites*", INRIA, 1999.

[4] Karlsson J.S., Kersten M.L.: "*Scalable Storage for a DBMS using transparent Distribution*", Centrum voor Wiskunde en Informatica, INS-R9710, 1999.

[5] Little M.C., McCue D.L.: "*The Replica Management System: a Scheme for Flexible and Dynamic Replication*", 2nd Workshop on Configurable Distributed Systems, 1994.

[6] Little M.C., Shrivastava S.K.: "*Using Application Specific Knowledge for Configuring Object Replicas*", IEEE 3rd International Conference on Configurable Distributed Systems, 1996.

[7] Little M.C., Shrivastava S.K.: "*A method for combining replication with cacheing*", International Workshop on Reliable Middleware Systems, 1999.

[8] Stonebraker M., Aoki P.M., Devine R., Litwin W., Olson M.: "*Mariposa: A New Architecture for Distributed data*", EECS, University of California, 1998.

Elastic Self-Organizing Feature Maps *

Martin Beran, Iveta Mrázová †

Department of Computer Science, Charles University,
Malostranské nám. 25, 118 00 Praha 1, Czech Republic

e-mail: beran@ss1000.ms.mff.cuni.cz, mrazova@ksi.ms.mff.cuni.cz

Abstract

In this paper, we will discuss various models suitable for (self-)organization of large data sets. In particular, these models will comprise the Oja learning algorithm and the Kohonen Self-Organizing Feature Maps. Making use of some of the ideas present in the Kohonen and Oja model, we will propose a new learning rule applicable – at least to a certain extent – to a PCA-like analysis (Principal Component Analysis) of data sets consisting of various sub-clusters. We will call the new-proposed model Elastic Self-Organizing Feature Map – ESOM. In addition to the classical Kohonen-like grid, ESOM-networks incorporate another "more elastic" neighbourhood compounding those neurons representing principal components of each respective cluster.

The aim of the new ESOM-learning rule consists in approaching the centers of the respective sub-clusters (possibly even hierarchically arranged). In the ideal case, for each found sub-cluster, mutually independent components with maximum possible pattern occurrence (e.g. density or variance) should be found. Results of preliminary supporting experiments done so far will be briefly discussed in this article, too.

We see the main application area for this model in pre-processing large mutually correlated patterns which should be stored later on in hierarchical Hopfield-like networks (e.g. in the so-called Cascade Associative Memories introduced by Hirahara et al., or in the Hierarchical Associative Memory Model). The trained weight vectors of the ESOM-model should correspond (hierarchically) to (mutually nearly-orthogonal) patterns which could be stored more reliably in the respective associative memory. We expect that similar principal ideas could be applied also for the dynamical adjustment of the topology in hierarchical, e.g. Tree-Structured SOM-networks.

*This research was supported by the grant No. 300/2002/A INF/MFF of the GA UK and by the grant No. 201/02/1456 of the GA ČR.

†Currently Fulbright Visiting Professor in Smart Engineering Systems Laboratory, Engineering Management Department, University of Missouri-Rolla, Rolla, MO 65409-0370, USA.

1 Introduction

Both the rapid development in the area of information systems and a wide availability of efficient computers support applications of previously computationally too expensive neural techniques. For few examples, let us mention the usage of associative memories for robust pattern recognition in spatial maps and self-organizing feature maps for localizing and organizing large files of text documents or images by means of hierarchical two-dimensional Kohonen grids. The application of traditional associative memory models – mainly Hopfield-like networks – is limited mainly by their relatively low storage capacity but also due to their restricted recognition abilities – very low or even no shift-, rotation- or scale-invariance, etc.

Therefore, it seems to be advantageous to pre-process the incoming data with the aim to increase the robustness of the whole system with regard to degraded input data and their deviations. Such a kind of pre-processing could increase the capacity of applied associative memories remarkably. The methods proposed for this purpose are based on the principle of the so-called cascade associative memories with a hierarchical structure.

At the same time, these methods incorporate several ideas of Kohonen self-organizing feature maps. Within the framework of e.g. information systems, mutually similar patterns (having the form of feature vectors) can be organized into groups, which are not defined previously but emerge during the data clustering process. In this way, information retrieval can be restricted to those groups relevant to the particular query. Moreover, clustering can reduce the dimensionality of the search space.

At the same time, it can contribute remarkably to a more transparent representation (and visualization) of the retrieved data relevant to the respective query. In such a case, it could be sufficient to output only e.g. few representatives of the respective clusters and it is not necessary to retrieve all of the relevant documents. The documents from the same cluster should be mutually as similar as possible. On the other hand, documents from different clusters should be as different as possible. Then, the task would be to find such cluster representatives which could be stored in the system reliably and recalled efficiently.

An example for such models represents the so-called WEBSOM architecture [5] using the SOM-training algorithm to locate and organize large files of text documents onto hierarchical two-dimensional Kohonen grids. In such maps, closely located areas contain documents with a mutually similar contents. To reduce the relatively high computational costs of searching for the "winning neurons", the so-called Tree-Structured Self-Organizing feature Maps (TS-SOMs) can be used [6].

These techniques can be further enhanced by incorporating methods for automatic creation of keywords (for detected clusters of text data). Good keywords characterize in this context an outstanding – in the sense of characteristic – property of the documents from the respective cluster and compared to those documents not contained in this cluster. A similar approach was adopted in the so-called PicSOM-system as well [8], [7].

An important ability of neural networks is to organize representations of the external world through learning. On the other hand, the representations of neural networks are

too complicated to estimate the capability of a neural system in practical use. A geometrical method to analyse the representation of an associative memory was introduced by [4], who presented a practical application of associative memory models – information categorization. In this application, the concept formation ability of associative memory is important.

From this point of view, an interesting idea would be to (re)generate automatically (e.g. by means of associative memories with incorporated feed-back) further queries e.g. in tree-structured SOM-networks. The generated queries would be based on several previously incomplete queries of the user. Such a process represents in principle learning the right query using the response from the user.

In this paper, we propose a new model for preprocessing the data which should be stored in an associative memory (having a hierarchical structure). The model – called ESOM (Elastic Self-Organizing Feature Maps) – is based on the idea of SOM-networks. In addition to (possibly hierarchical) clustering of the input data, it improves mutual orthogonality of the found cluster representatives and hence it improves both the capacity and reliability of the used associative memory. In the following Section 2, we will describe the existing models which our work will be based on. The definition of the ESOM model is presented in Section 3 and the results of supporting experiments for the ESOM-model will be given in Section 4. The final Section 5 contains some concluding remarks and outlines a plan for our further research.

2 Previous models

In this section, we will discuss briefly various existing neural network models which our new-proposed model will be based on. In particular, these models comprise associative memories (both the standard Hopfield model and the so-called Cascade ASsociative Memories – CASM) and neural network models based on self-organization (the Oja-algorithm and Kohonen Self-Organizing Feature Maps).

Hopfield networks and hierarchical associative memories. The *Hopfield networks* [2] represent an auto-associative memory model which consists of a mutually fully inter-connected set of neurons with symmetric weights ($w_{i,j} = w_{j,i}$) trained by means of the Hebbian learning rule

$$w_{i,j} = \sum_k x_i^{(k)} x_j^{(k)} \quad ; \quad i \neq j \tag{1}$$

x_i^k stands for the i-th element of the k-th training pattern, i and j index the neurons of the network. During recall, presented input patterns are retrieved iteratively. A serious disadvantage of the basic Hopfield network model refers to its low capacity (about 0.14 times the number of neurons). Moreover, this capacity can be achieved only for (nearly) orthogonal input vectors.

A possible means to overcome these limitations incorporate the so-called *cascade associative memories* [1]. The basic idea is to have a hierarchy (two levels in the simplest

case) of Hopfield networks. The first-level network stores some representative patterns from the input space. On the second level, only the differences between the presented patterns and their first-level representatives are stored. This strategy poses a question which we attempt to solve: *how to choose the first level representatives?*

These representatives should correspond to the centers of the input data clusters and at the same time, they should be as much orthogonal one to each other as possible. Such representatives would be in general more suitable for being stored in the first-level network than the randomly chosen ones. At the same time, the difference patterns will be sparser and thus could be stored more efficiently in the second-level network. In the next section, we will propose a method for finding first level representative patterns using ESOM – a type of Kohonen self-organizing feature map with a modified learning rule. Its performance will be compared with the standard SOM-network.

PCA and the Oja's learning rule. The *principal component analysis* (PCA) is a common tool widely used e.g. to reduce the dimensionality of input data and to determine an orthogonal base (rotated coordinate system) such that projections of the data to new coordinates retain as much information as possible (the most important of them even more than the original coordinates). Let us have a set $\{\vec{x}_1, \ldots, \vec{x}_m\}$ of input data (n-dimensional vectors). The first principal component of this set is a vector $\vec{w_1}$ maximizing the expression:

$$\frac{1}{m}\sum_{i=1}^{m} \|\vec{w}_1 \cdot \vec{x}_i\|^2 \ , \tag{2}$$

Thus the first principal component runs in the direction of maximum variance of the input vectors. The second principal component $\vec{w}_2$ is then the first principal component of the set of residues. The residues are determined as the rest of the original data vectors after subtracting their projections to the first principal component. The third principal component is computed after subtracting projections onto the first and the second principal component, and so on up to n. All principal components are mutually orthogonal.

One possible solution for the PCA is the Karhunen-Loeve Transform, introduced by Karhunen [3] and Loeve [9]. Another, iterative learning algorithm for computing the first principal component was proposed by Oja [10]. The vector $\vec{w}_1$ is initialized randomly. In each step, a vector $\vec{x}_i$ is selected randomly and new vector $\vec{w}_1' = \vec{w}_1 + \gamma\phi(\vec{x}_i - \phi\vec{w}_1)$ is computed, where $\phi = \vec{x}_i \cdot \vec{w}_1$ and $0 < \gamma \leq 1$ is a gradually decreasing learning parameter.

Self-organization and Kohonen maps. Self-organizing feature maps (SOMs) were introduced by Kohonen [5]. A SOM-network consists of a set of neurons, organized in a neighbourhood grid, e.g. a 2-dimensional mesh. The position of a neuron in the d-dimensional input space is defined by the neuron's d-dimensional weight vector $\vec{w}$. The goal of a SOM is to adjust the weight vectors of neurons ("move the neurons") such that the neurons would approximate the spatial distribution of input data. For example, if there are several clusters of input patterns present in the feature space, the

neurons should move to the centers of those clusters. At the same time, the neurons should become topologically ordered, i.e., neurons which are close one to each other in the neighbourhood grid represent similar input vectors.

The learning algorithm of SOM starts with weight vectors initialized to random values. In each time-step, the weights are adapted according to a single input vector $\vec{x}$ from the training set. First, the best-matching (nearest) neuron c is selected, such that $\|\vec{x} - \vec{w}_c\| = \min_i\{\|\vec{x} - \vec{w}_i\|\}$. Then, the weights of c but also of other neurons (indexed by i) lying in the neighbourhood of c are adapted according to the following formula:

$$\vec{w}_i(t+1) = \vec{w}_i(t) + \alpha(t)h_{ci}(t)(\vec{x} - \vec{w}_i(t)) \ . \tag{3}$$

The value of the learning rate $\alpha(t)$ gradually decreases from values close to 1 to 0. The neighbourhood function $h_{ci}(t)$ defines the size of the neighbourhood neurons of c which will be adapted together with c and how strongly the neurons from the neighbourhood will be adapted (compared to the neuron c). Initially, large neighbourhood is used (up to the whole network). During time, it shrinks down to the single neuron c. An example of a widely used neighbourhood function is the Gaussian function

$$h_{ci}(t) = exp\left(-\frac{\|r_c - r_i\|^2}{2\sigma^2(t)}\right) \ , \tag{4}$$

where $\sigma^2(t)$ is a monotonically decreasing function and r_i are the coordinates of the neuron i in the neighbourhood grid.

3 The Elastic Self-Organizing Feature Maps – ESOM

In this section, we will describe a new model – the so-called Elastic Self-Organizing feature Map (ESOM). It is based on the principle of Kohonen networks described in the preceding section. As we mentioned earlier, we intend to employ ESOM as a preprocessing stage for storing patterns in hierarchical associative memories, which work best if the patterns to be stored are mutually orthogonal. Thus, the modified learning rule for ESOM should promote orthogonality between the found weight vectors.

The simple version of ESOM uses a standard Kohonen neighbourhood grid. Anyway, the learning rule consists of two phases which run as it follows: in the first phase of each step (Kohonen learning phase), the "winning" weight vector $\vec{w}_i(t)$ is adapted according to the presented input vector $\vec{x}$ and the Kohonen learning rule (3) yielding an intermediate vector $\vec{w}'_i(t)$. The second phase then tries to improve orthogonality of weight vectors. For all vectors $\vec{w}'_i(t)$, the following algorithm is run in parallel. It computes a "more orthogonal" vector $\vec{w}_i(t+1)$. This second phase of each learning step proceeds in the following three sub-steps:

1. For each neuron $j \neq i$, compute vector $\vec{y}_{ij}(t)$ perpendicular to $\vec{w}'_j(t)$ in the plane defined by $\vec{w}'_i(t)$ and $\vec{w}'_j(t)$.

$$\vec{y}_{ij}(t) = \vec{w}'_i(t) - \frac{\vec{w}'_j(t) \cdot \vec{w}'_i(t)}{\|\vec{w}'_j(t)\|^2} \cdot \vec{w}'_j(t) \ . \tag{5}$$

2. For each neuron $j \neq i$, find the normalized vector $\vec{u}_{ij}(t)$ determining the direction "more orthogonal" to $\vec{w}'_j(t)$.

$$\vec{u}'_{ij}(t) = \omega(t)\vec{y}_{ij}(t) + (1-\omega(t))\vec{w}'_i(t) \ , \tag{6}$$

$$\vec{u}_{ij}(t) = \frac{\vec{u}_{ij'}(t)}{\|\vec{u}_{ij'}\|} \ . \tag{7}$$

   The function $\omega(t)$ plays a similar role like the learning rate $\alpha(t)$ in the Kohonen rule – it controls the amount of adaptation actually performed. Its initial value should be close to 1 and it should monotonically decrease towards 0.

3. The direction of the final vector $\vec{w}_i(t+1)$ is an average of all the vectors $\vec{u}_{ij}(t)$ (denoted as $\vec{v}_i(t)$) multiplied by the norm $\|\vec{w}'_i(t)\|$.

$$\vec{v}_i(t) = \frac{1}{n-1}\sum_{j\neq i}\vec{u}_{ij}(t) \ , \tag{8}$$

$$\vec{w}_i(t+1) = \vec{v}_i(t)\|\vec{w}'_i(t)\| \ , \tag{9}$$

   where n is the number of neurons in the network.

The algorithm is run for all neurons in parallel. In order to be able to achieve orthogonal weight vectors, the number of neurons n should be at most equal to the dimensionality d of the input space.

Further, we are working also on more elaborate ESOMs. Their main idea consists in defining an additional neighbourhood function ν_{ij} superposed to the traditional Kohonen-like neighbourhood grid. Then, the orthogonalization of $\vec{w}'_i(t)$ will be performed using not all $\vec{w}'_j(t)$, but only those ones with $\nu_{ij} \neq 0$. For example, if the network has more neurons n than is the dimension d of the input space, the function ν may define partitioning of neurons into n/d groups of size d. Each group of neurons should then be stored in a separate associative memory, because the weight vectors would be orthogonal only within the respective groups.

Additionally, there might be detected a hierarchical structure for the evolved ESOMs, such that the first level defines d top level mutually orthogonal clusters and the second level defines orthogonal sub-clusters in each cluster. Thus, the first level ESOM would produce the representatives to be stored in the first level of the hierarchical associative memory (see page 3), while the second level ESOM generates difference patterns stored in the second level of the hierarchical associative memory, etc.

4 Supporting Experiments

Supporting experiments done so far test the behaviour of the simple ESOM. The combined Kohonen and orthogonalization learning algorithm (as described in the previous section) was run on randomly generated data. We used two error measures: the learning error, i.e. the squared Euclidean distance of the input vector from the nearest neuron

$\|\vec{x} - \vec{w}_c\|^2$ and the orthogonalization error, that is the average angle in degrees (over all pairs of weight vectors) by which an angle between two weight vectors differs from 90 degrees.

We observed how the two error measures depend on the number of learning steps performed, i.e., the number of input patterns presented to the network. The network had the same number of neurons as was the dimension of the input space, which varied from 2 to 10. The neurons were arranged in a linear cyclic Kohonen grid (its topology was a single cycle). The training set had 200 members – it was formed by a union of several clusters of vectors with the Gaussian distribution. This set was randomly permuted 10 times (in order to obtain input patterns for 2000 steps). After every 10 steps, the orthogonalization errors of all neurons and the sum of learning errors from the last 10 steps were recorded. All experiments were run both for a Kohonen network without orthogonalization (dash-dotted line in graphs), as well as for a net with orthogonalization applied (ESOM, solid line). Results displayed in the graphs represent average values from 10 repetitions of each test.

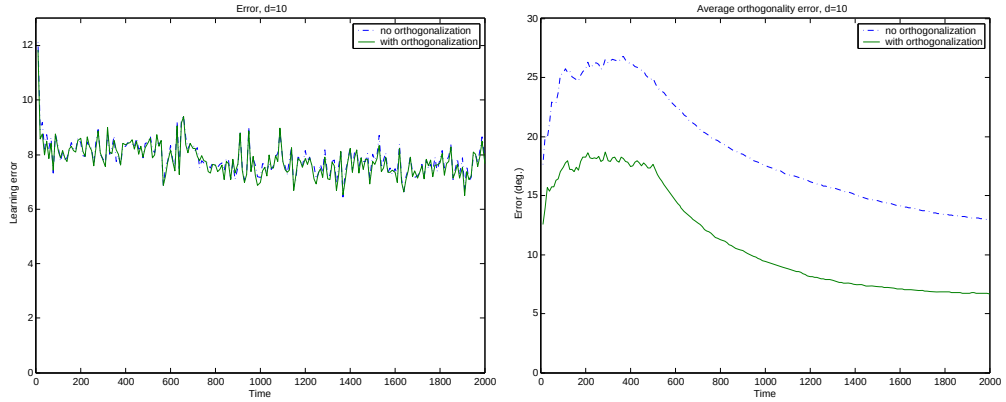


Figure 1: Single 10-dimensional cluster with mean 0 and variance 1

The first set of experiments used the data consisting of only one single cluster with the mean 0 and variance 1. Figure 1 shows that the learning error remains high and approximately the same in both networks during learning, because a small number of neurons cannot fill the cluster well. On the other hand, after an initial phase, the orthogonalization error decreases and is much smaller in the ESOM case (about 50%).

The inputs for the second set of experiments consisted of d clusters for a d-dimensional case. We tested data sets having clusters with several values of variance and with different angles among the vectors from the origin to centers of the respective clusters. These angles were defined by the parameter rot such that the center of a cluster is $rot \cdot (1, \ldots, 1) + (1 - rot) \cdot (0, \ldots, 0, 1, 0, \ldots, 0)$, i.e. for $rot = 0$, the centers of clusters lie on the axes of the coordinate system.

Figure 2 displays 3 example inputs for dimension 2. In Figure 3 we can see the

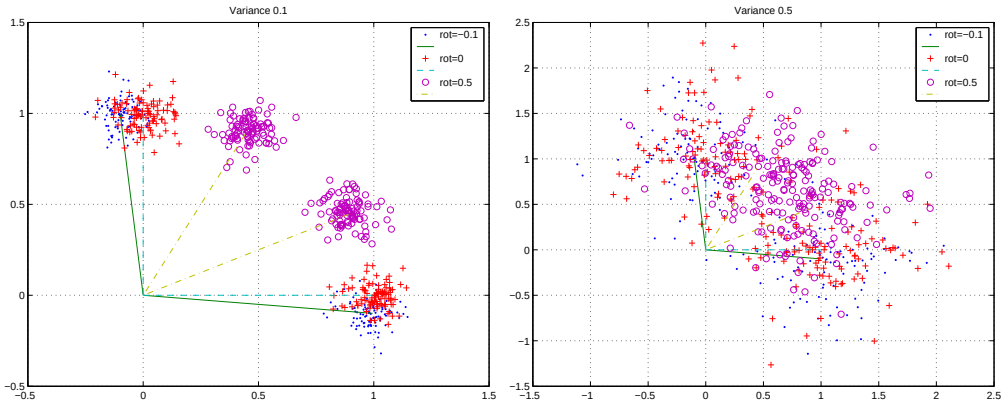


Figure 2: Example of input data having different angles and variances.

tradeoff between learning and orthogonality error. As there was in this case only a small

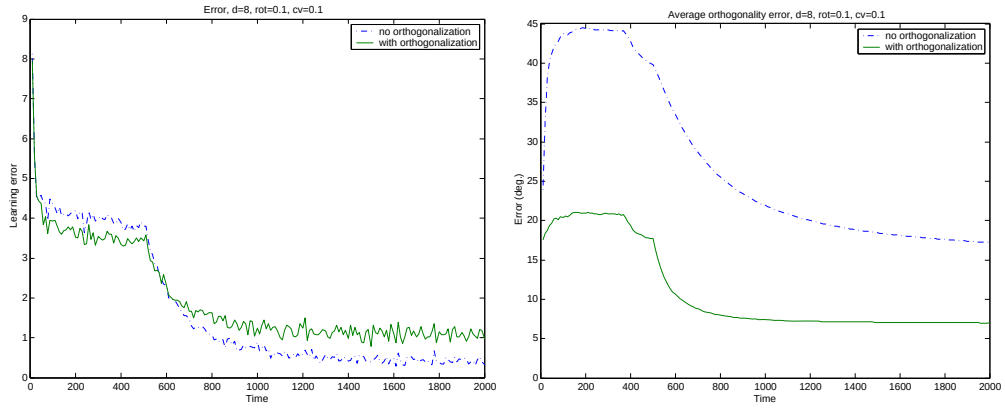


Figure 3: Nearly orthogonal 8 clusters in 8 dimensions with variance 0.1

variance in the clusters, orthogonalization moves the neurons out of the cluster centers and increases the learning error while decreasing orthogonality error. If the clusters are more rotated, this effect is even stronger, as can be seen in Figure 4.

On the other hand, larger variance of inputs makes displacements of neurons caused by orthogonalization less important, see Figure 5. The experiments from the third set considered various numbers of randomly positioned clusters as inputs. Figure 6 shows an example of results for 12 clusters with variance 0.1 in 6 dimensions. The trade-off between learning and orthogonality errors is obvious again.

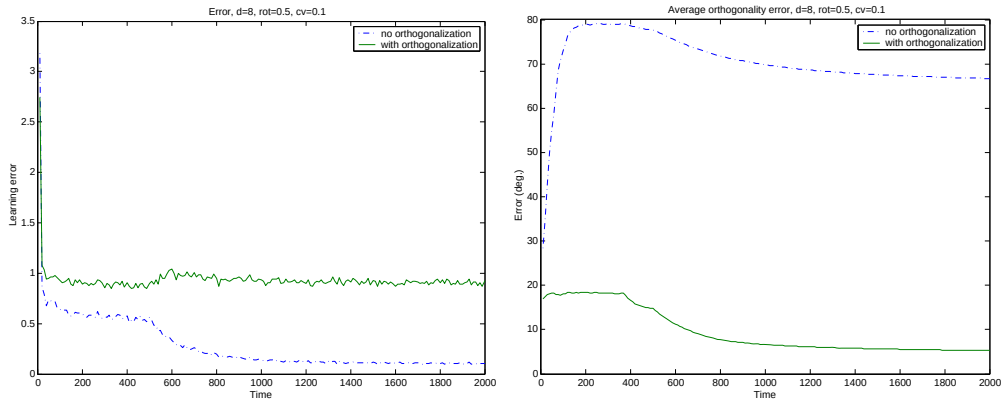


Figure 4: More rotated 8 clusters in 8 dimensions with variance 0.1

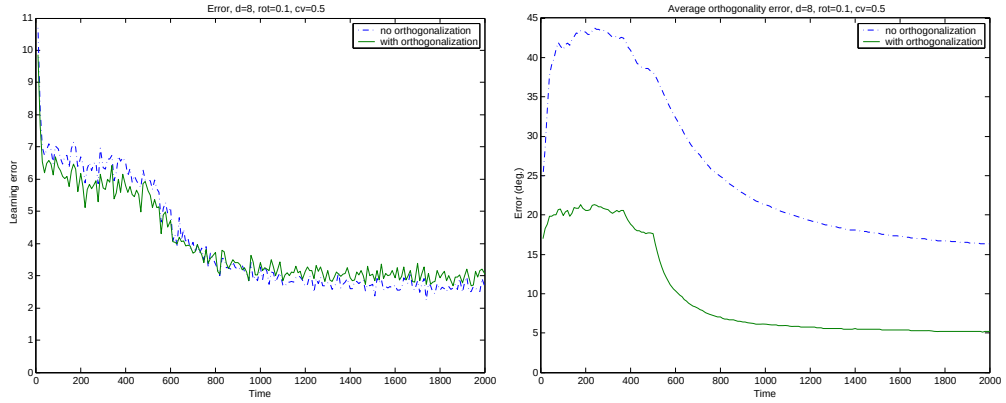


Figure 5: 8 clusters in 8 dimensions with variance 0.5 and nearly orthogonally positioned

5 Conclusions

The results presented in this paper represent an initial part of our ongoing research. Its main goal is to apply ESOMs for finding "parent patterns" to be stored in Hopfield-like hierarchical associative memories. Especially the orthogonalization property of ESOMs could improve the main weakness of associative memories referring especially to their capacity limits. In this paper, we have presented the motivations and we have specified and tested the basic ESOM model. Experimental results done so far have shown that in comparison with the standard Kohonen model, ESOM is able to improve orthogonality of the set of weight vectors without enlarging the learning error too much.

Within the framework of our further research, we are aimed at specifying more elaborated variants of ESOM (able to process hierarchical data organized arbitrarily in

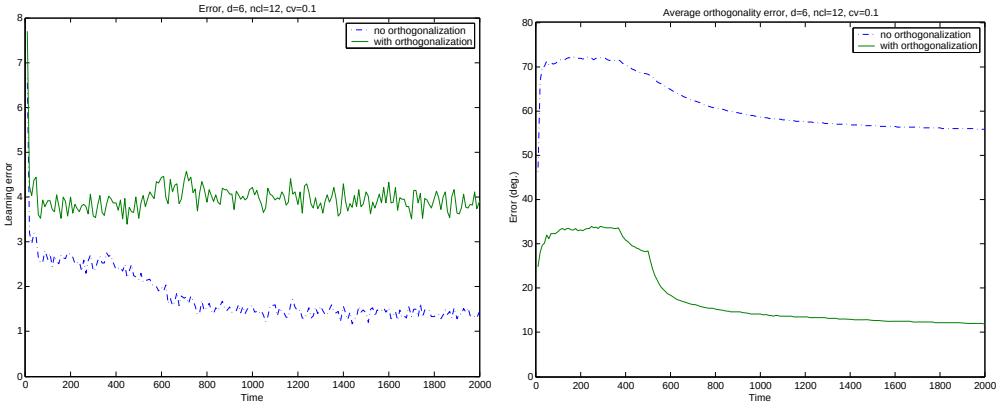


Figure 6: 12 random clusters in 6 dimensions with variance 0.1

multiple levels), test them by experiments similar to those described in this paper, and perform experiments processing high-dimensional real-world data, e.g. digital images of various kinds of scenes.

References

[1] Hirahara, M., Oka, N., and Kindo, T. (2000) A cascade associative memory model with a hierarchical memory structure, *Neural Networks*, **13**

[2] Hopfield, J. J. (1982) Neural networks and physical systems with emergent collective computational properties, *Proc. Nat. Ac. Sci. USA* **79**, pp. 2554–2588.

[3] Karhunen, K. (1946) Zur Spektraltheorie Stochastischer Prozesse, *Ann. Acad. Sci. Fennicae*, **37**

[4] Kindo, T., Yoshida, H., and Shida, T. (2001) Automatic information categorization through concept formation of associative memory model, *Advances in Neural Networks and Applications*, N. Mastorakis (ed), (WSES Press), pp 134–139.

[5] Kohonen, T. (2001) *Self-Organizing Maps* (Springer-Verlag).

[6] Koikkalainen, P., and Oja, E. (1990) in: *Proc. of IJCNN'90*, (IEEE Service Center), Piscataway, NJ, p. 279.

[7] Koskela, M., Laaksonen, J., Laakso, S., and Oja, E. (2000) The PicSOM Retrieval System: Description and Evaluations, *Proc. of CIR2000*, Brighton, UK.

[8] Laaksonen, J., Koskela, M., and Oja, E. (1999) Application of Tree Structured Self-Organizing Maps in Content-Based Image Retrieval, *Proc. of ICANN'99*, Edinburgh, UK.

[9] Loeve, M. M. (1955) *Probability Theory* (Van Nostrand, Princeton).

[10] Oja, E. (1982) A simplified neuron model as a principal component analyzer, *Journal of Mathematical Biology*, **15**.

Random access compression methods

Jiří Dvorský and Václav Snášel

Department of Computer Science, Technical University of Ostrava,
17. listopadu 15, Ostrava - Poruba, Czech Republic
{jiri.dvorsky,vaclav.snasel}@vsb.cz

Abstract. Compression methods based on finite automatons are presented in this paper. Simple algorithm for construction finite automaton for given regular expression is shown. The best advantage of this algorithms is the possibility of random access to a compressed text. The compression ratio achieved is fairly good. The methods are independent on source alphabet i.e. algorithms can be character or word based.

Keywords: word-based compression, text databases, information retrieval, HuffWord, WLZW

1 Introduction

Data compression is an important part of the implementation of full text retrieval systems. The compression is used to reduce space occupied by indexes and text of documents. There are many popular algorithms to compress a text, but none of them can perform direct access to the compressed text. This article presents an algorithm, based on finite automaton, which allows such type of access. The definition of finite automata is given in the first section. Compression algorithm itself is described in the second section and the third section shows some experimental results. At the end the conclusion is given.

2 Finite automata

Definition 1. *A deterministic finite automaton (DFA) [6] is a quintuple* (Q, A, δ, q_0, F)*, where* Q *is a finite set of states,* A *is a finite set of input symbols (input alphabet),* δ *is a state transition function* $Q \times A \to Q$*,* q_0 *is the initial state,* $F \subseteq Q$ *is the set of final states.*

Definition 2. *Regular expression* U *on alphabet* A *is defined as follows:*

1. $\emptyset$*,* ε *and* a *are regular expression for all* $a \in A$

2. *If U, V are regular expression on A then $(U+V), (U \cdot V)$ and $(U)^*$ are regular expression on A.*

Definition 3. *Value $h(U)$ of regular expression U is defined as:*

$$\begin{aligned} h(\emptyset) &= \emptyset \\ h(\varepsilon) &= \{\varepsilon\} \\ h(a) &= \{a\} \\ h(U+V) &= h(U) \cup h(V) \\ h(U \cdot V) &= h(U) \cdot h(V) \\ h(U^*) &= (h(U))^* \end{aligned}$$

Definition 4. *Derivation $\frac{dU}{dx}$ of regular expression U by $x \in A^*$ is defined as:*

1.
$$\frac{dU}{d\varepsilon} = U$$

2. *$\forall a \in A$ it holds:*

$$\begin{aligned} \frac{d\varepsilon}{da} &= \emptyset \\ \frac{d\emptyset}{da} &= \emptyset \\ \frac{db}{da} &= \begin{cases} \emptyset & \text{if } a \neq b \\ \varepsilon & \text{otherwise} \end{cases} \\ \frac{d(U+V)}{da} &= \frac{dU}{da} + \frac{dV}{da} \\ \frac{d(U \cdot V)}{da} &= \frac{dU}{da} \cdot V + \left\{ \frac{dV}{da} : \varepsilon \in h(U) \right\} \\ \frac{d(V^*)}{da} &= \frac{dV}{da} \cdot V^* \end{aligned}$$

3. *For $x = a_1 a_2 \dots a_n$, where $a_i \in A$ it holds:*

$$\frac{dV}{dx} = \frac{dV}{da_n}\left(\frac{dV}{da_{n-1}}\left(\cdots \frac{dV}{da_2}\left(\frac{dV}{da_1}\right)\cdots\right)\right)$$

Derivation of regular expression V by string x is an equivalent

$$\frac{dV}{dx} = \{y : xy \in h(V)\}$$

In other words, derivation of V by x is expression U such $h(U)$ contains strings which arise from strings in $h(V)$ by cutting prefix x.

Example 1. Let be $h(V) = \{abccabb, abbacb, babbcab\}$. Then $h(\frac{dV}{da}) = \{bccabb, bbacb\}$.

2.1 Construction of DFA for regular expression V

One possibility how to construct DFA for given regular expression is based on following theorem:

Theorem 1. *When DFA accepts, in state q, language defined by V then accepts in state $\delta(q,a)$ language defined by $\frac{dV}{da}$, for all $a \in A$ (see [6]).*

For given regular expression V we construct $DFA(V) = (Q, A, \delta, q_0, F)$, where

- Q is a set of regular expressions (states),
- A is given alphabet,
- $\delta(q,a) = \frac{dq}{da}, \forall a \in A$,
- $q_0 = V$,
- $F = \{q \in Q | \ \varepsilon \in q\}$

Example 2. Let's construct automaton for $V = (0+1)^* \cdot 01$ – words ending with 01. Sequence of derivations is given in following table:

	$dV/d0$	$dV/d1$
$(0+1)^* \cdot 01$	$(0+1)^* \cdot 01 + 1$	$(0+1)^* \cdot 01$
$(0+1)^* \cdot 01 + 1$	$(0+1)^* \cdot 01 + 1$	$(0+1)^* \cdot 01 + \varepsilon$
$(0+1)^* \cdot 01 + \varepsilon$	$(0+1)^* \cdot 01 + 1$	$(0+1)^* \cdot 01$

Particular derivations can be marked as states in this manner: $(0+1)^* \cdot 01$ as q_0, $(0+1)^* \cdot 01 + 1$ as q_1, $(0+1)^* \cdot 01 + \varepsilon$ as q_2. Then state transition function δ can be written in this form:

q	$\delta(q,0)$	$\delta(q,1)$
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_0

Remark that final state is q_2 only because it contains empty string. Final automaton is drawn in figure 1.

3 Random access compression

Let $A = \{a_1, a_2, \ldots, a_n\}$ be an alphabet. Document D of length m can be written as sequence $D = d_0, d_1, \ldots, d_{m-1}$, where $d_i \in A$. For each position i we are able to find out which symbol is at position d_i. We must save this property to create compressed document with random access.

A set of position $\{i; 0 \leq i < m\}$ can be written as a set of binary words $\{b_i\}$ of fixed length. This set can be considered as language $L(D)$ on alphabet

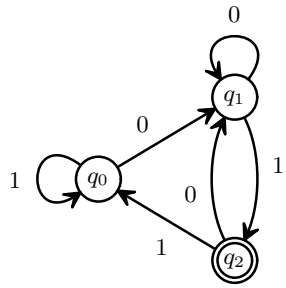


Fig. 1. DFA for regular expression $V = (0 + 1)^* \cdot 01$

$\{0, 1\}$. It can be easy shown that the language $L(D)$ is regular and it is possible to construct DFA which accepts the language $L(D)$. This DFA can be created, for example, by algorithm given in section 2. Regular expression is formed as $b_0 + b_1 + \cdots + b_{m-1}$.

Compression of the document D consists in creating a corresponding DFA. But decompression is impossible. The DFA for the document D can only decide, whether binary word b_i belongs to the language $L(D)$ or not. The DFA does not say anything about a symbol which appears in position i. In order to do this, the definition of DFA must be extended or more than one automaton should be used.

4 Multiple DFAs

The first way how to achieve random access compression by finite automaton is using several, independent automata. Each of the automata can compute part of character lying on particular position.

Let $D = d_0, d_1, \ldots, d_{m-1}$ is the document over alphabet A. Let $c_{i,0} \ldots c_{i,k-1}$ $(k = \lceil \log_2 n \rceil)$ is a binary representation of symbol $d_i \in D$. Then document D can be written as matrix C:

$$C = \begin{pmatrix} c_{0,0} & \cdots & c_{0,k-1} \\ \vdots & \ddots & \vdots \\ c_{m-1,0} & \cdots & c_{m-1,k-1} \end{pmatrix}$$

For each column of matrix C lets define set of positions $P_i(D)$ $(0 \leq i < k)$ as

$$P_i(D) = \{bin(j) \mid c_{j,i} = 1\},$$

where $bin(j)$ is a binary representation of number j.

The set $P_i(D)$ contains only positions of symbols from A which have i^{th} bit set to 1.

It is obvious that sets $P_i(D)$ form regular languages and finite automata $DFA_i(D)$ can be constructed for each of $P_i(D)$, for example, by algorithm given in section 2.

Definition 5. *Let $DFA_i(D), 0 \leq i < k$ be automatons. Function $Decomp_D : N \to \{0|1\}^k$ defined as*

$$Decomp_D(x) = \begin{cases} 1 \text{ if } DFA_i(D) \text{ accepts } bin(x) \\ 0 \text{ otherwise} \end{cases}$$

is called decompression function of document D.

Decompression is then trivial. If the decompression function is given, each symbol $d_i \in D$ can be computed as $d_i = bin^{-1}(Decomp_D(i))$, for $0 \leq i \leq m-1$.

Example 3. Let be for example document $D = abracadabra$, $m = 11$. Then $A = \{a, b, c, d, r\}$, $k = 3$. The alphabet A has following representation:

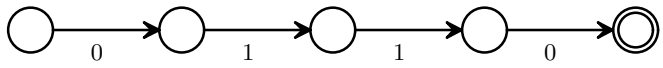
Symbol from alphabet	Frequency	Code	Binary code
a	5	0	000
b	2	1	001
c	1	3	011
d	1	4	100
r	2	2	010

Encoding of symbol of the alphabet can be taken at random, for example ASCII. $k = \lceil \log_2 n \rceil$ bits are necessary. We suggest to encoding symbols of the alphabet from 0 to n according to decreasing frequency of occurrence of particular symbol. In this way the most frequent symbol will have code with all bits set to zeroes i.e. it will not be included in any set $P_i(D)$.

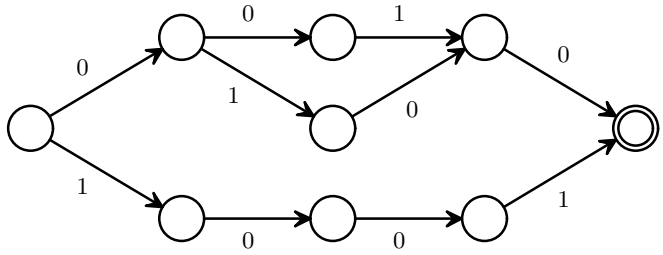
Then matrix C for our document D is formed as:

$$C = \begin{pmatrix} 0\,0\,0 \\ 0\,0\,1 \\ 0\,1\,0 \\ 0\,0\,0 \\ 0\,1\,1 \\ 0\,0\,0 \\ 1\,0\,0 \\ 0\,0\,0 \\ 0\,0\,1 \\ 0\,1\,0 \\ 0\,0\,0 \end{pmatrix}$$

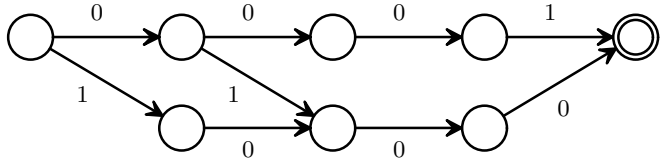
Now we can construct three sets $P_0(D) = \{0110\}, P_1(D) = \{0010, 0100, 1001\}$, $P_2(D) = \{0001, 0100, 1000\}$. Four bit representation to store row of matrix C. Longer representation can be used but sets $P_3(D), P_4(D), \ldots$ would be empty sets. Particular automata can be seen in figure 2.



(a) Automaton for set $P_0(D)$



(b) Automaton for set $P_1(D)$



(c) Automaton for set $P_2(D)$

Fig. 2. Automata constructed in example 3

Decompression - for example position 4 is given. Binary representation of 4 is 0100 (we use four bit representation). This binary number is put as input to automata $P_0(D)$, $P_1(D)$, and $P_2(D)$. The automaton $P0$ doesn't accept this input, so the first bit of decompressed symbol is zero. Two other automata accept given input, the second and the third bit are equal to one. We obtain 011 binary as decompressed symbol at given position. 011 is codeword for symbol 'c' in our encoding schema. Result of decompression - symbol 'c' can be found at position 4.

4.1 Extension of DFA

Definition 6. *A deterministic finite automaton with output (DFAO) is a 7-tuple* $(Q, A, B, \delta, \sigma, q_0, F)$*, where* Q *is a finite set of states,* A *is a finite set of input symbols (input alphabet),* B *is a finite set of output symbols (output alphabet),* δ *is a state transition function* $Q \times A \to Q$*,* q_0 *is the initial state,* σ *is an output function* $F \to B$*,* $F \subseteq Q$ *is the set of final states.*

This type of automaton is able to determine for each of the accepted words b_i which symbol lies on position i. To create an automaton of such a type the algorithm mentioned in section 2 must be extended too. Regular expression V, which is input into the algorithm, consists of words b_i. Each b_i must carry its output symbol d_i. Regular expression is now formed as $b_0d_0 + b_1d_1 + \cdots + b_{m-1}d_{m-1}$,

Example 4. Let be for example document $D = abracadabra$, $m = 11$. Regular expression V will be

$$\begin{aligned} V = {} & 0000a + 0001b + 0010r + 0011a \\ & 0100c + 0101a + 0110d + 0111a \\ & 1000b + 1001r + 1010a \end{aligned}$$

$DFAO(V) = (Q, A, B, \delta, \sigma, q_0, F)$ will be constructed, where $Q = \{q_0, \ldots q_{16}\}$, $A = \{0, 1\}$, $B = \{a, b, c, d, r\}$, $F = \{q_{12}, q_{13}, q_{14}, q_{15}, q_{16}\}$. Final automaton from our example is drawn in figure 3(a).

Such constructed automaton have following properties:

1. there are no transitions from final states,
2. let be $|q|$ for $q \in Q$ the length of words in appropriate regular expression. If $\delta(q_i, a) = q_j$, where $q_i, q_j \in Q$, $a \in A$, then $|q_i| > |q_j|$. In other words, the state transition function contain only forward transitions. There are no cycles.

The set of states Q of the automaton $DFAO(V)$ is divided into disjunct subsets (so called *layers*). Transitions are done only between two adjacent layers. Thus states can be numbered locally in those layer. In out example layer 0 consists of state q_0, layer 1 of states q_1, q_2, layer 2 of states q_3, q_4, q_5, layer 3 of states $q_6, \ldots, q_{11}$ and layer 4 of states $q_{12}, \ldots, q_{16}$.

Final automaton is stored on disk after construction. Particular layers are stored sequentially. Three methods of storing layers are available now:

Raw – the layer is stored as a sequence of integer numbers (4 bytes each). Appropriate for short layers.

Bitwise – maximum number max in layer is found. The layer is stored as a sequence of $\lceil \log_2 max \rceil$ binary words.

Linear – linear prediction of transitions is made. Parameters of the founded line and a correction table are stored.

Mechanism of victim Horspool & Cormack [5] propose mechanism of the spaceless words mechanism to eliminate space immediately following any word. They used strict alternation of words and non-words. In some texts strict alternation cannot be achieved for some reasoms (e.g. limited length of word or non-word). We propose mechanism of eliminating of the most frequent non-word (so called victim [3, 4]). Mechanism of victim was adopted in several standard compression algorithms.

We would like to try to join mechanism of victim and random access compression. It is easy for both of methods. In the first case, the victim should be encoded as zero, so it doesn't participate in any constructed automatons. For the second case positions of victim is not included in regular expression V that expresses document D. In decompression phase particular position of victim isn't accepted by DFAO(V). Consequently, there is only one explanation - victim should be there. It holds only for positions between 0 and $m-1$, i.e. in document.

Example 5. Let's construct DFAO for document mentioned in example 4. Victim is symbol *a*. Regular expression V will be

$$\begin{aligned} V = {} & 0001b + 0010r + \\ & 0100c + 0110d + \\ & 1000b + 1001r \end{aligned}$$

Final automaton can be seen in figure 3(b).

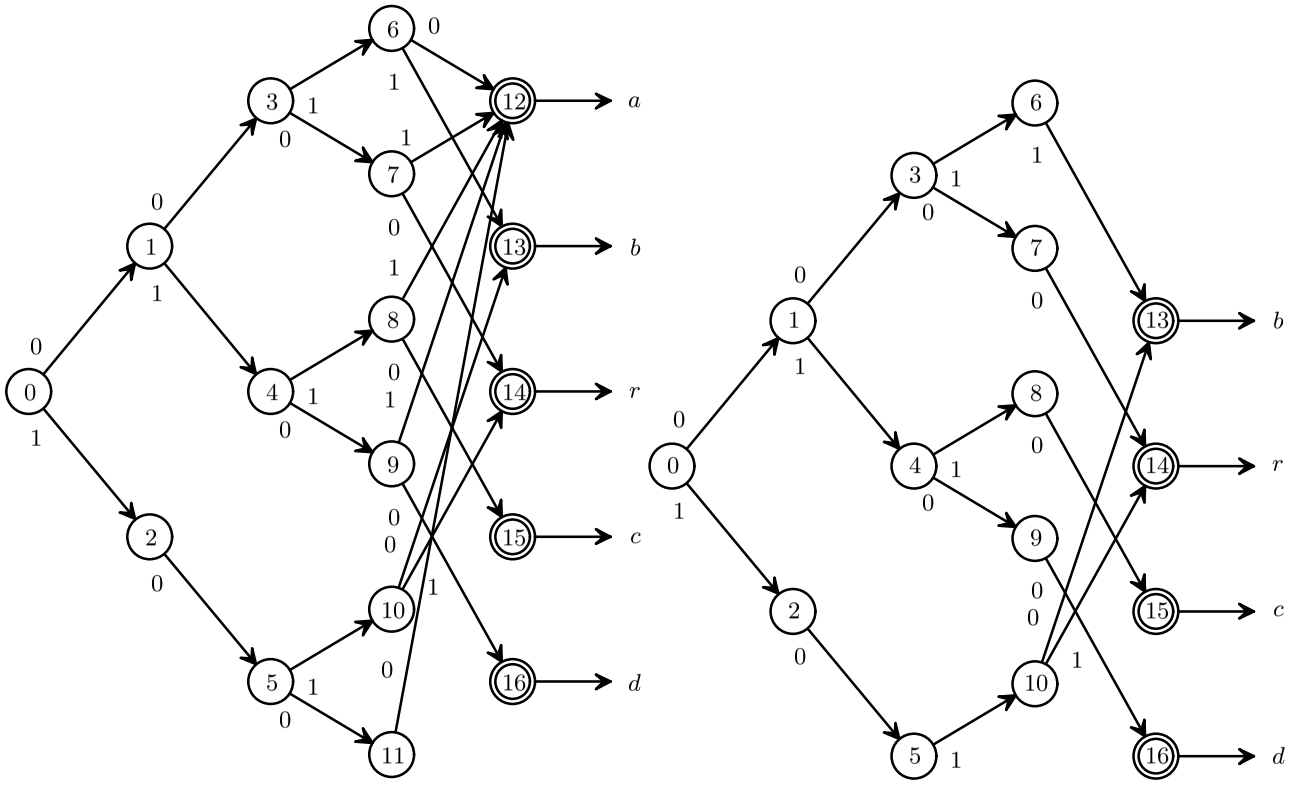
5 Experimental results

To allow practical comparison of algorithm, experiments have been performed on some compression corpus.

Let's remark, that algorithm of construction of automaton is independent with respect to its output alphabet. There are two possibilities. The first is a classic character based version. Algorithm is one-pass and output alphabet is a standard ASCII. For the text retrieval systems word-based version (the second possibility) is more advantageous because of the character of natural languages.

For test has been used Canterbury Compression Corpus (large files) [1], especially King's James Bible (bible.txt) file which is 4047392 bytes long. There are 1535710 tokens and 13508 of them are distinct.

A word-based version of algorithm has been used for a test. The size of the compressed file and the compression ratio have been observed. Results are given in table 1. Tests were done on Pentium II/400Mhz with 256MB of RAM. Program was compiled by MS Visual C++ 6.0 as 32-bit console application under MS Windows 2000.



(a) Automaton for expression V from example 4

(b) Automaton for expression V from example 5

Fig. 3. Sample automata

Table 1. Experimental results for file bible.txt

Used method	Compressed size [bytes]
Multiple DFA's	1761878
DFAO	2088389
DFAO (elimination of victim)	2072418

6 Conclusion and future works

Compression ratio is worse than other algorithm can achieve, but none of then can directly access compressed text. It is interesting that elimination of victim hasn't any significant impact to size of compressed document. It means that most of states and transitions in automaton was preserved.

It is important to realise that this method does not actually depend on text encoding. This means that it performs successfully for a text encoded in UNICODE as well.

Basic algorithm for random access compression was published in [2]. Several word-based compression algorithms were developed for the text retrieval systems. There is well-known Huffword [7], and WLZW [3, 4] (our version of word-based, two-phase LZW).

References

1. R. Arnold and T. Bell. A corpus for evaluation of lossless compression algorithms. In *Proceedings Data Compression Conference 1997*, 1997. `http://corpus.canterbury.ac.nz`.
2. J. Dvorský and V. Snášel. Word-random access compression. In *Lecture Notes on Computer Science*. Springer-Verlag Berlin, 2000.
3. J. Dvorský, V. Snášel, and J. Pokorný. Word-based compression methods and indexing for text retrieval systems. In *Lecture Notes on Computer Science 1691*. Springer-Verlag Berlin, 1999.
4. J. Dvorský, V. Snášel, and J. Pokorný. Word-based compression methods for large text documents. In *Proc. of Data Compression Conference (DCC99)*, Snowbird, Utah, USA, 1999.
5. R. Horspool and G. Cormack. Constructing word-based text compression algorithms. In *Proc. 2nd IEEE Data Compression Conference*, Snowbird, Utah, USA, 1992.
6. G. Rozenberg and E. A. Salomaa. *Handbook of Formal Language*. Springer-Verlag Berlin, 1997.
7. I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, 1994.

Random access storage system for sparse matrices

Jiří Dvorský[†], Václav Snášel[†], and Vít Vondrák[‡]

[†]Department of Computer Science, Technical University of Ostrava, 17. listopadu 15, Ostrava - Poruba, Czech Republic
{jiri.dvorsky,vaclav.snasel}@vsb.cz
[‡]Dept. of Applied Mathematics, Technical University of Ostrava, 17. listopadu 15, Ostrava - Poruba, Czech Republic
vit.vondrak@vsb.cz

Abstract. New system for storage is presented. This system allows direct access to matrix. Complexity of each access is proportional to $\log_2 p$, where $p = max(m, n)$ for matrix of order $m \times n$. Space complexity is similar to other storage systems.

1 Introduction

Using finite element method for solving any practical problem we obtain a stiffness matrix. This matrix plays important role in the process of solving these problems. Usually, this matrix is very large and very sparse. The term sparse means, that the matrix contains many zero members but only a few nonzero in comparison with all members of this matrix. So, it is very good idea to store only nonzeros and reduce amount of memory for stiffness matrix.

For the finite solvers like LU factorization is impossible to use the system which store only nonzero members. This restriction arises from basic feature of all factorization algorithms, that a zero member in original matrix can become (and usually become) to nonzero in factorized matrix.

However, a storage of only nonzero member fits very good to iterative solvers such as conjugate gradient method. Sparse storage decreases a number of operation rapidly and saves amount of memory necessary for storing stiffness matrix.

In the following, system of storing stiffness matrices based on finite automata will be explained. This system of storage allows direct access into matrix i.e. any element of the matrix can be read or written with constant time complexity.

2 Properties of stiffness matrices

Our storage system is based on the following observation. In finite element method the global stiffness matrix is assembled from large number of local stiffness matrices. All these local stiffness matrices have the same structure.

Let us assume a local stiffness matrix of triangle element with 3 nodes (see figure 1) and let $nDOF$ is number of degrees of freedom in each node. Let the following storage scheme of degrees of freedom is used

$$x = (\ \underbrace{u_1, v_1, w_1, ...;}_{DOF\ of\ 1^{st}\ node}\ \underbrace{u_2, v_2, w_2, ...;}_{DOF\ of\ 2^{nd}\ node}\ \underbrace{u_3, v_3, w_3, ...}_{DOF\ of\ 3^{rd}\ node}\ )^T$$

Then local stiffness matrix can be written as block matrix

$$K_e = \begin{array}{c} \begin{pmatrix} K^e_{11}\ K^e_{12}\ K^e_{13} \\ K^e_{21}\ K^e_{22}\ K^e_{23} \\ K^e_{31}\ K^e_{32}\ K^e_{33} \end{pmatrix} \begin{array}{l} node\ 1 \\ node\ 2 \\ node\ 3 \end{array} \\ node\ 1\ node\ 2\ node\ 3 \end{array}$$

Each block of this matrix is submatrix of dimension $nDOF \times nDOF$. Due to symmetry of K^e the equality $K^e_{ji} = {K^e_{ij}}^T$ is valid for all $i \neq j$. This symmetry implies that also diagonal blocks K^e_{ii} are symmetric matrices. Hence, it is sufficient to store only diagonal and upper triangular members of these diagonal blocks.

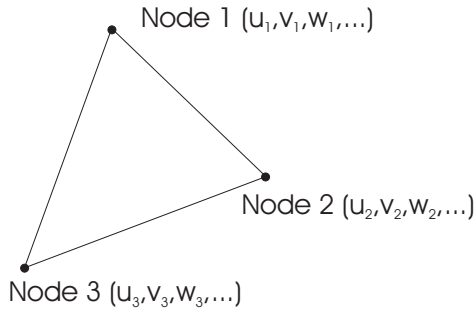


Fig. 1. Example of element

For the other types of elements, only the number of blocks changes. So we can generalize this block scheme for all types of elements. For example, in the case of tetrahedral element with 20 nodes, the local stiffness matrix with 20×20 blocks is obtained.

Let m denotes total number of nodes in the finite element model. Then global stiffness matrix has form

$$K = \begin{pmatrix} K_{11} & K_{12} & \cdots & K_{1m} \\ & K_{22} & & \vdots \\ & & \ddots & \vdots \\ & & & K_{mm} \end{pmatrix},$$

where all K_{ij} are submatrices of dimension $nDOF \times nDOF$. These matrices are assembled from blocks of local element stiffness matrices. The process of assembling is identical to assembling of global stiffness matrix with 1 degree of freedom per node. Only difference is, that the members of stiffness matrix are replaced by so called *nodal* submatrices. Hence, we shall obtain submatrix at i, j position using following formula

$$K_{ij} = \sum_e K^e_{i_e, j_e},$$

where summation is considered over all elements containing nodes i and j in global numbering of nodes. Indices i_e, j_e here denotes local numbers of nodes i, j in element e.

The sparsity of this storage is provided by inserting only nonzero submatrices into global stiffness matrix. Also only diagonal and upper triangular submatrices are stored due to symmetry of stiffness matrix. Therefore, the lower triangular part of matrix K is blank on the picture. Drawback of this system of storage is, that some zeros what appear in the block submatrices are stored, too. But usually, amount of memory saved by storing whole blocks and only indices for these blocks is larger than memory saved by storing only nonzero members and all indices for these members in global stiffness matrix, even when some zeros are stored in blocks. Especially, for the problems with large number of degrees of freedom (3 and more) per node the saving of memory is dominant.

3 Sparse matrices and finite automata

Culik and Valenta [1] introduce using of finite automata for compression of bi-level and simple color images. A digitized image of the finite resolution $m \times n$ consists of $m \times n$ pixels each of which takes a Boolean value (1 for black, 0 for white) for bilevel image, or a real value (practically digitized to an integer between 0 and 256) for a grayscale image.

Sparse matrix can be viewed, in some manner, as simple color image too. Zero element of matrix corresponds to white pixel in bi-level image and nonzero element to black or gray-scale pixel.

Here we will consider square matrix M of order $2^n \times 2^n$ (typically $13 \leq n \leq 24$). In order to facilitate the application of finite automata to matrix description we will assign each element at $2^n \times 2^n$ resolution a word of length n over the alphabet $\Sigma = \{0, 1, 2, 3\}$ as its address. A element of the matrix corresponds to a subsquare of size 2^{-n} of the unit square. We choose ε as the address of the whole square matrix.

Its submatrices (quadrants) are addressed by single digits as shown in Fig. 2(a). The four submatries of the matrix with address ω are addressed $\omega 0$, $\omega 1$, $\omega 2$ and $\omega 3$, recursively. Addresses of all the submatrices of dimension 4×4 are shown in Fig. 2(b). The submatrix (element) with address 3203 is shown on the right of Fig. 2(c).

In order to specify a values of matrix of dimension $2^n \times 2^n$, we need to specify a function $\Sigma^n \to R$, or alternately we can specify just the set of nonzero values, i.e. a language $L \subseteq \Sigma^n$ and function $f_M : L \to R$.

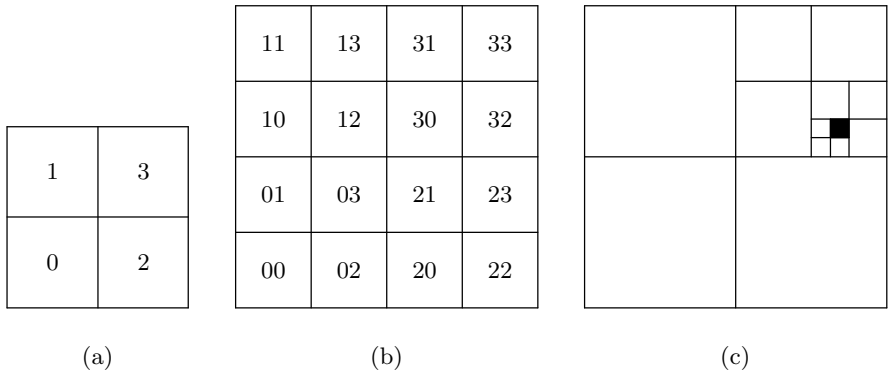


Fig. 2. The addresses of the submatrices (quadrants), of the subsmatrices of dimension 4×4, and the submatrix specified by the string 3203

Example 1. Let M be a matrix of order 8×8.

$$M = \begin{pmatrix} 2&0&0&0&0&0&0&0 \\ 0&4&0&0&1&0&0&0 \\ 0&0&3&0&0&6&0&9 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&5&0&0 \\ 0&0&0&0&0&0&9&0 \\ 0&0&0&0&0&0&0&7 \end{pmatrix}$$

The language $L \subseteq \Sigma^3$ is now

$$L = \{111, 112, 121, 122, 211, 212, 221, 222, 303, 310, 323\}.$$

Then function f_M will have following values (see table 1).

Table 1. Positions in matrix M and corresponding values – function f_M

$x \in L$	$f_M(x)$	$x \in L$	$f_M(x)$
111	2	221	9
112	4	222	7
121	3	303	6
122	1	310	1
211	1	323	9
212	5		

Now automaton that computes function f_M cn be constructed (see Fig. 3). The automaton is four order tree, where values are stored only at leaves.

If matrix M is considered as read-only the automaton can be reduced into compact form (see Fig. 4).

The global stiffness matrix is assembled from large number of local stiffnes matrices that have the same structure as it was mentioned in section 2. From the point of view of finite automaton dividing of whole matrix can be terminated at the level of local matrices. We need to specify function $L \to R_{nDOF,nDOF}$

This kind of storage system allows direct access to stored matrix. Each of elements can be accessed independetly to previous accesses and access to each element has same, constant time complexity. Let A be a matrix of order $2^n \times 2^n$. Then time complexity of access is boudned by $O(\log_2 n)$.

Fig. 3. Automaton for matrix M

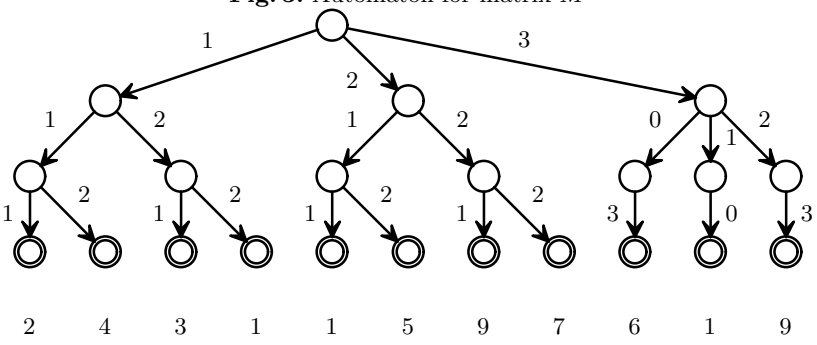
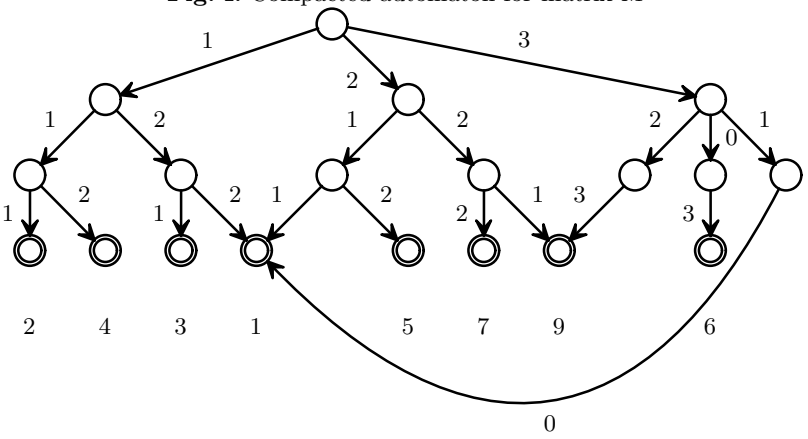


Fig. 4. Compacted automaton for matrix M



3.1 Implementation notes

To allow practical comparison of algorithm experimental implementation was written in MS Visual C++, but it can be compiled on any other platform with standard C++ compiler.

The crucial point of our algorithm is transformation of element's row and column to quadrant coordinates – word over alphabet $\{0, 1, 2, 3\}$. The transformation consists in bit rotating and masking over themselves. Transformation can be performed in time $O(n)$ for matrix of order $2^n \times 2^n$.

```
unsigned int cPosition::Convert(unsigned int x, unsigned int y) const
{
    unsigned int iRet = 0;
    x = x / m_NDOF;
    y = y / m_NDOF;
    for(unsigned int i = 0; i < m_Bites; i++)
    {
        iRet |= ((1 & x) + ((1 & y) ≪ 1)) ≪ (i ≪ 1);
        x ≫= 1;
        y ≫= 1;
    }
    return iRet;
}
```

Where `m_Bits` denotes how many bits are used to store row and column of element. `m_Bits` is typically from 16 to 32 bits which depends on platform or size of matrices.

Other important feature of our implementation is iterator [6] on the automaton. The iterator allows sequential scanning of nonzero elements through whole matrix. Then there is no need to read all elements in matrix, for example in matrix multiplication. The most of them are zero, so it is better go through nonzero only. For example, when we have square matrix of order 10^4 only q queries to values of nonzero elements must be done but no 10^8.

```
class cAutomatIterator
{public:
    cAutomatIterator(cAutomaton* Automaton);
    virtual void Next(void);
    virtual void Reset(void);
    virtual t_Item& Data(void);
    bool isEnd(void) const;
};
```

Method Reset resets iterator to initial state (same as constructor). Method isEnd becomes true if iteration process is over. The Next method moves iterator to the next nonzero element of matrix. Method Data provides its result row, column and value of current nonzero element of the matrix.

4 Conclusion

Storage system of sparse matrices is presented. The storage system allows random access to elements of the matrix. This property makes it different to other systems that usually used linked lists to accommodate matrix's elements. Our experiments shows that presented system is about 40 % faster than K3 system implemented at Dept. of Applied Mathematics in Ostrava [5]. Space complexity is similar to other storage systems. Other systems for storage of sparse matrices can be founded in [4]. Random access compression can be used to compress textual data also, see [2][3].

References

1. K. Culik and V.Valenta. Finite automata based compression of bi-level and simple color images. In *Computer and Graphics*, volume 21, pages 61–68, 1997.
2. J. Dvorský. Text compression with random access. In *Proceedings of ISM 2000*, 2000.
3. J. Dvorský and V. Snášel. Word-random access compression. In *Proceedings of CIAA2000*. University of Tours.
4. R.Barret, M.Berry, T.F.Chan, J.Demmel, J.M.Donato, J.Dongarra, and C. . H. V. d. V. V.Eijkhout, R.Pozo. *Templates for the Solution of Linear Systems: Bulding Blocks for Iterative Methods*. SIAM, Philadelphia, PA, 1994. http://www.netlib.org/templates/templates.ps.
5. V. Vondrák. Description of k3 sparse matrix storage system (preprint). 2002.
6. W.Ford and W.Topp. *Data Structures with C++*. Prentice Hall, 1996.

Information contained in an observed value

Zdeněk Fabián

Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod vodárenskou věží 2, 18200 Prague,
zdenek@cs.cas.cz

Abstract

The square of the recently proposed core function $T_f(x)$ of continuous probability distribution F is shown to express relative information contained at x. The mean value of this function can be viewed as the mean information of F and the average information contained in single observation taken from F.

1 Problem

We have an apparently simple question: What amount of information carries an observed value x, a realization of random variable X with distribution F?

Information has to be additive. Before some inference mechanism is applied, all items from an observed sample $(x_1, ..., x_n)$ should carry the same amount of information, equal noticeably to the mean information of the sample, which is obviously the mean information of distribution F, generating the sample. The question can be reformulated:

What is the mean information of a probability distribution?

Surprisingly, no generally accepted answer is at disposal.

2 Shannon information

Denote by f the density and by S the support of distribution F. It is well known that the Shannon answer

$$H(f) = \sum_{i=1}^{n} \ln \frac{1}{f(x_i)} f(x_i)$$

fails in cases of continuous distributions with sharply peaked densities, since in case it holds that $f(x) > 1$ for $x \in (x_1, x_2)$, the expression

$$H(f) = \int_S -\ln f(x) f(x)\, dx \equiv E_f(-\ln f)$$

may be negative. The sharply peaked distributions should have, on the contrary, higher mean information as distributions with densities with broad peaks. Mean information of the distribution is apparently inversely proportional to its variance. This fact cannot be used for a definition of information, however, as a variance of many so called heavy-tailed distributions is infinite.

3 Maximum likelihood

We begin with the standard solution of the parametric point estimation problem: Let us have a sample $(x_1, ..., x_n)$, a realization of independent identically distributed according to F_θ random variables $X_1, ..., X_n$. F_θ is a distribution with density, whose mathematical form $f(x|\theta)$ is known or assumed, apart from an unknown parameter $\theta \in \Theta \subseteq R_m$. The task is to estimate the true value θ_0 of θ (which gives the 'true' distribution F_{θ_0}).

The relative probability of x_1 (the probability of an occurrence of x_1 in a small interval around x_1) is the density $f(x_1|\theta)$ taken as a function of θ. The simultaneous relative probability of $(x_1, ..., x_n)$ is a function of θ,

$$L(\theta) = \Pi_{i=1}^n f(x_i|\theta),$$

called *likelihood*. Maximizing the likelihood or its logarithm

$$\log L(\theta) = \sum_{i=1}^n \log f(x_i|\theta) = \max.$$

one obtains $\hat{\theta}$ with the largest possible probability of θ as the solution of equation

$$\sum_{i=1}^n \frac{\partial \log f(x_i|\theta)}{\partial \theta} = 0. \tag{1}$$

This $\hat{\theta}$ is so called maximum likelihood (ML) estimate. By the law of large numbers, it converges to the true value θ_0 and, moreover, it has the minimal possible asymptotic variance.

4 Likelihood score and Fisher information

Rewrite (1) into

$$\sum_{i=1}^{n} \psi(x_i|\theta) = 0 \tag{2}$$

where

$$\psi(x|\theta) = \frac{\partial \ln f(x|\theta)}{\partial \theta}. \tag{3}$$

At a fixed x, the function ψ of θ is known as the *likelihood score*. Consider now ψ as a function of x. The mean value of its square in point θ

$$J(\theta) = E_f(\psi^2(x|\theta)) \tag{4}$$

is the Fisher information of distribution F about parameter θ. The estimate of $J(\theta_0)$ is obviously the value $J(\hat{\theta})$ where $\hat{\theta}$ is the ML estimate of θ_0. The famous Cramér-Rao theorem says that this value is inversely proportional to the asymptotic variance of $\hat{\theta}$.

5 Outline of the solution

After the inference mechanism has been applied and we know $\hat{\theta}$, the value $\psi^2(x_i|\hat{\theta})$ is considered as the information about $\hat{\theta}$ contained in data item x_i. This 'observed' information has a property that the information of a more or less expected event is low and, on the other hand, an unexpected observation carries high information. Its mean value, the Fisher information about the true θ_0, is the solution to our problem.

There is a serious obstacle of this project, however. Parameter θ can be a vector parameter and the Fisher information a matrix. How to obtain the mean information in these cases ? And what about a distribution F which has no parameter?

Our idea was a simple one: To determine the 'central point' x^* of distribution F, to define it as a parameter $\tau : \tau_0 = x^*$ and to consider the parametric distribution with density $f(x|\tau)$. In some cases, this distribution matches a known distribution, in other cases it does not. Sometimes, it

matches some distribution after its reparametrization. Finally, other parameters can be added to obtain a general family of distributions $f(x|\theta)$ where $\theta = (\tau, \sigma, \theta_3, ..., \theta_m)$ and where σ is the scale parameter (see [1]).

Naturally, the information function of $f(x|\theta)$ should be the squared likelihood score for parameter τ in its 'true' value τ_0.

6 The central point of continuous distributions

Continuous distributions can be divided into two large groups.

The 'central point' of the distributions of the first group, which are distributions with density $g(y)$ positive for any $y \in R$ (i.e. with whole support) is, naturally, the maximum of the density. Taking it as a *location parameter*, $\mu = y^\star$, we obtain density in the form $g(y - \mu)$ or, in a general case, $g(y - \mu, \sigma, \theta_3..., \theta_n)$.

In the other group there are the distributions which have densities positive only on some interval $S \neq R$ (with partial support). They may not have the maximum in S, they may not have the mean. What is their 'central point' was not clear.

We noticed that the densities are in these cases usually in a mathematical form

$$f(x) = g(\varphi(x)) \cdot \varphi'(x) \tag{5}$$

where g is the density of some distribution from the first group, $\varphi : S \to R$ some one-to-one differentiable mapping and $\varphi'(x) = d\varphi(x)/dx$. Let us call the 'central point' of a distribution the *centre of gravity* (in [1], we called it the Johnson location). We construct it as follows: from (5) we determine g, consider it as a 'source' distribution of f, find the maximum $y^\star$ of $g(y)$, set $\mu = y^\star$, find

$$x^\star = \varphi^{-1}(y^\star),$$

introduce the parameter $\tau = x^\star$ and generalize or reparametrize $f(x)$ into $f(x|\tau)$ or into a general parametric form

$$f(x|\theta) = f(x|\tau, \sigma, \theta_3..., \theta_n). \tag{6}$$

Using the described procedure, any distribution can be written in a form with parameter τ, expressing the centre point of the distribution.

7 Core function

Core function T_f introduced in [1] can be now defined as the *inner part of the likelihood score for the center of gravity* in the form (formula (12) in [1])

$$\frac{\varphi'(\tau)}{\sigma} T_f(x|\theta) = \frac{\partial \log f(x|\theta)}{\partial \tau} \equiv \psi_\tau(x|\theta). \tag{7}$$

Let us give some examples illustrating the introduced concepts and some technical problems of the procedure described above.

Example 1. Exponential distribution has density $f(x) = e^{-x}$. It can be rewritten into form (5) by

$$e^{-x} = xe^{-x}\frac{1}{x},$$

which has form (5) with $\varphi(x) = \ln x$. The 'source distribution' and the core function of it are given in Example in [1].

It should be said that other φ might be considered (as for example $\varphi(x) = \ln^3 x$). This leads, however, to a more complicated g in (5) and to more complicated core functions of both distributions G and F. The principle of parsimony says that the model should be as simple as possible. In the course of time, densities of model distributions have been selected according to this principle and we expect that the forms of core functions should obey the same principle.

Example 2. Gamma distribution has density

$$f_{\gamma,\alpha}(x) = \frac{\gamma^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\gamma x} = \frac{\gamma^\alpha}{\Gamma(\alpha)} x^\alpha e^{-\gamma x} \frac{1}{x}. \tag{8}$$

Obviously, $\varphi(x) = \ln x$ as well. The procedure described above leads to a reparametrized form of gamma distribution

$$f_{\tau,\alpha}(x) = \frac{\alpha^\alpha}{\Gamma(\alpha)} \left(\frac{x}{\tau}\right)^\alpha e^{-\alpha x/\tau} \cdot \frac{1}{x}$$

with the centre of gravity $\tau = \alpha/\gamma$.

Example 3. Uniform distribution. The simplest decomposition of the density $f(x) = 1$ on $S = (0, 1)$ is $1 = x(1-x) \cdot \frac{1}{x(1-x)}$, giving $\varphi(x) = \ln \frac{x}{1-x}$ and core function $T_f(x) = 2x-1$. The uniqueness of this result is questionable even when considering the principle of parsimony. We conjecture, however, that few distributions with no unique centre of gravity are better than many distributions without mean.

8 Information function

Returning to the problem of the mean information of distribution F or F_θ, we suppose that function $T_f^2(x)$ or $T_f^2(x|\theta)$ is the information function of distribution F or F_θ. It means that a point $x \in S$ of F or an observed data item x_i from F_θ (after the inference mechanism was applied) carries the relative information $T_f^2(x)$ or $T_f^2(x_i|\hat{\theta})$. In the latter case, $\hat{\theta}$ is the ML estimate of the true value of θ.

Arguments supporting this conviction are as follows.

(i) *Proposition.* The center of gravity is the least informative point of the distribution. (*Proof:* Let X be distributed by $F = G\varphi$. For a given φ there is a large class $F : \{F_\alpha : F_\alpha = G_\alpha\varphi\}$ of composite distributions with densities $f_\alpha(x) = g_\alpha(\varphi(x))\ \varphi'(x)$. The term $\varphi'(x)$ is common to all these distributions and, therefore, does not carry any information about X, and all information contained in X is condensed in term $g_\alpha(\varphi(x))$. This is minimal at the point $\tilde{x} : \frac{d}{dx} g_\alpha(\varphi(x)) = 0$. By (5) one obtains $\tilde{x} : \frac{d}{dx}(f_\alpha(x)/\varphi'(x)) = 0$, which reduces by Theorem 1 in [1] into $\tilde{x} : T_{f_\alpha}(x) = 0$. The solution of the last equation is the centre of gravity of F_α, $\tilde{x} = x^*$.)

(ii) Function

$$i_f(x) = T_f^2(x)$$

is a non-negative function, attaining its minimum $i_f(x^*) = 0$ in the least informative point of the distribution. Its increase in both directions is quick if the distributions have unbounded core functions, which imply sharply to zero tending density, for which some observed outlier values (values far from the 'bulk' of the data) have immense informative values: their occurrence indicates a necessity of a change of the model. This increase is slow if the distribution has a bounded core function which implies heavy-tailed density, for which an observation of outlier value is more or less expected.

Density $f(x|\tau,\alpha) = \frac{\alpha^\alpha}{\Gamma(\alpha)}(x/\alpha)^{-\alpha} e^{-\alpha\tau/x}$, core function $T_f(x|\tau,\alpha) = \alpha(1 - \tau/x)$ and information function $i_f(x|\tau,\alpha) = \alpha^2(1-\tau/x)^2$ of the generalized extreme value II distribution with support $S = (0,\infty)$ and values $\tau = 2, \alpha = 3$ are given on Fig.1. The zero of the core function, point $x = 2$, is the centre of gravity of the distribution (different from the mode, mean and median). The core and information function are 'semibounded'.

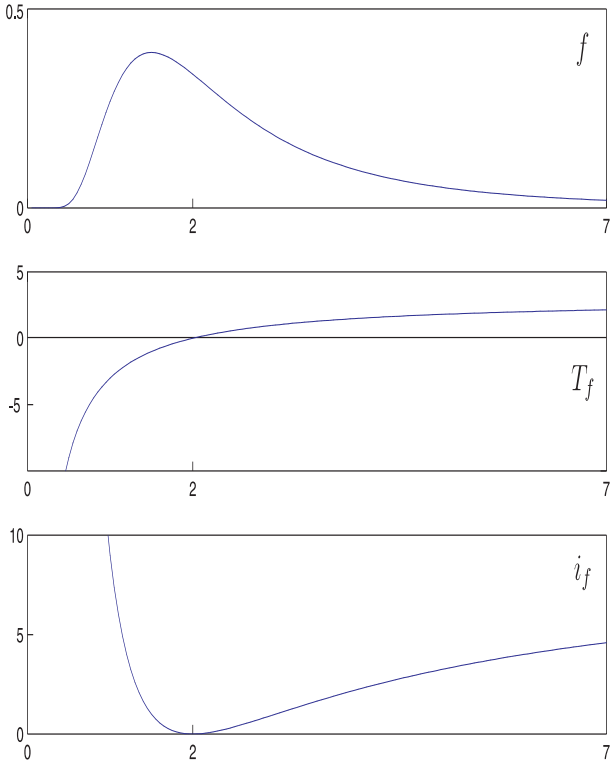


Fig.1

9 Mean information of a distribution

The mean value of the information function,

$$I_f = E_f(i_f(x))$$

is an 'inner part' of the Fisher information of distribution F for the most important point of the distribution, its centre of gravity. Indeed, taking the mean value of the square of equation (7) one obtains

$$J_f = E_f(\psi_\tau^2(x|\theta)) = \frac{1}{\sigma^2}\varphi'(\tau)^2 E_f(T_f^2(x|\theta)) = \frac{1}{\sigma^2}\varphi'(\tau)^2 I_f. \qquad (9)$$

We conjecture that J_f represents the *mean information of* distribution F_θ.

Table 1 gives information functions $i_f(x)$ and their mean values I_f of distributions given in Table 1 in [1].

TABLE 1. Information functions and mean information of some F's

Name	$f(x)$	$i_f(x)$	I_f
Normal	$\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$	$\left(\frac{x-\mu}{\sigma}\right)^2$	1
Lognormal	$\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}\ln^2(x/\tau)^\beta}$	$\beta\ln^2(x/\tau)$	1
Gumbel	$e^{\frac{x-\mu}{\sigma}}e^{-e^{\frac{x-\mu}{\sigma}}}$	$(e^{\frac{x-\mu}{\sigma}}-1)^2$	1
Weibull	$\frac{\beta}{x}((x/\tau)^\beta e^{-(x/\tau)^\beta}$	$\left((x/\tau)^\beta-1\right)^2$	1
Extreme val. II	$x^{-2}e^{-1/x}$	$(1-1/x)^2$	1
Logistic	$\frac{e^x}{(1+e^x)^2}$	$\tanh^2(x/2)$	1/3
Log-logistic	$1/(1+x)^2$	$\left(\frac{x-1}{x+1}\right)^2$	1/3
Lomax	$\alpha/(1+x)^{\alpha+1}$	$\alpha^2\left(\frac{x-1}{x+1}\right)^2$	$\alpha^2/3$
Gamma	$\frac{\tau\alpha^\alpha}{\Gamma(\alpha)}(x/\tau)^{\alpha-1}e^{-\alpha x/\tau}$	$\alpha^2(x/\tau-1)^2$	α
Beta	$\frac{1}{B(p,q)}x^{p-1}(1-x)^{q-1}$	$[(p+q)x-p]^2$	$\frac{pq}{p+q+1}$
Cauchy	$\frac{1}{\pi(1+x^2)}$	$\frac{4x^2}{(1+x^2)^2}$	1/2

Example 4. By (9) and Table 1, the information in a single observed value taken from the normal distribution is $J_f = 1/\sigma^2$, from the lognormal distribution $J_f = \beta^2/\tau^2$, from the log-logistic $J_f = \beta^2/\tau^2$ and so on. Using the usual parametrization of the gamma distribution (see formula (8)), we obtain the mean information of the gamma distribution expressed in usual parameters by $J_f = \alpha/\tau^2 = \gamma^2/\alpha$.

Aknowledgement. The work was supported by grant GA AV A1075101.

References

[1] Fabián, Z. (2001). Relationship between probability measure and metric in the sample space. *ITAT '2001*, PF Košice.

[2] Cover, T.M. and Thomas, J.A. (1991). *Elements of Information Theory,* Wiley.

[3] Fabián, Z. (2001). Induced cores and their use in robust parametric estimation. *Commun. Stat., Theory Methods*, 30, 3, 537-556.

[4] Lehmann E.L. (2001). *Elements of Large Sample Theory*, Springer.

Neural Networks in Speech Recognition

Ľudovít Hvizdoš

Institute of Computer Science

Faculty of Science

University of P. J. Šafárik,

Jesenná 5, 041 54, Košice, Slovakia

hvizdos@cs.science.upjs.sk

Abstract

Better recognition of speech signals require for new methods. This paper is devoted to implementation of neural network technique in speech recognition systems. We present an application of time delay neural networks (TDNN), to processing a Slovak language. We study classification capabilities. Simulation results are shown at the end of the paper.

1 Introduction

Speech recognition is a modern technology. It creates a new interface to machines and allows us to create natural aids for disabled people. Robust speech recognition could be the answer to many technical problems. Speech recognition and language processing is a modern expanding research field. It uses knowledge of linguistics, signal processing and informatics. The task of speech recognition system is to identify sound signals.

We are trying to develop algorithms that are universal and can be implemented in a simple chip. These are preliminary results. We used an artificial neural network (ANN) capability to learn patterns [1]. We choose Time Delay Neural Network (TDNN) architecture which was introduced in [7]. It is a feed forward neural network (FFNN). It is designed for large time dependent data. The ANN capability of universal aproximation make them a universal method for speech recogmition.

2 Speech recognition

To make make speech recognition (SR) possible we must use a Speech Recognition System (SRS) becouse of complexity of the task.

We have: Record signal, Digitize, Compute spectral features, classify time frames, match category scores, measure confidence, output result.

In fact , we digitize the speech that we want to recognize (for telephone speech the sampling rate is 8000 samples per second). Second, we compute features that represent the spectral-domain content of the speech (regions of strong energy at particular frequencies). These features are computed every 10 msec, with one 10- msec section called a frame. Third, a neural network is used to classify a set of these features into phonetic- based categories at each frame. Fourth, some search is used to match the neural-network output scores to the target words (the words that are assumed to be in the input speech), in order to determine the word that was most likely uttered (for example Viterbi algorithm). We are traning network to recognize the phonemes of Slovak language. That is our standard SRS. To create such a system we use the SNNS.

3 SNNS

SNNS (Stuttgart Neural Network Simulator) is a simulator for neural networks developed at the Institute for Parallel and Distributed High Performance Systems (Institut für Parallele und Verteilte Höchstleistungsrechner, IPVR) at the University of Stuttgart since 1989. The goal of the project is to create an efficient and flexible simulation environment for research on and application of neural nets.

The SNNS simulator consists of four main components : Simulator kernel, graphical user interface, batch simullator version snnsbat, and network compiler snns2c. The simulator kernel operates on the internal network data structures of the neural nets and performs all operations on them. The graphical user interface XGUI, built on top of the kernel, gives a graphical representation of the neural networks and controls the kernel during the simulation run. In addition, the user interface can be used to directly create, manipulate and visualize neural nets in various ways. Complex networks can be created quickly and easily. The free code of program makes possible take a part at developing the system. We plan re-design system for our needs.

4 TDNN

Time delay networks (or TDNN for short), introduced by Alex Waibel [7], are a group of neural networks that have a special topology. They are used for position independent recognition of features within a larger pattern. A special convention for naming different parts of the network is used here (see figure 1).

Feature: A component of the pattern to be learned. Feature Unit: The unit connected with the feature to be learned. There are as many feature units in

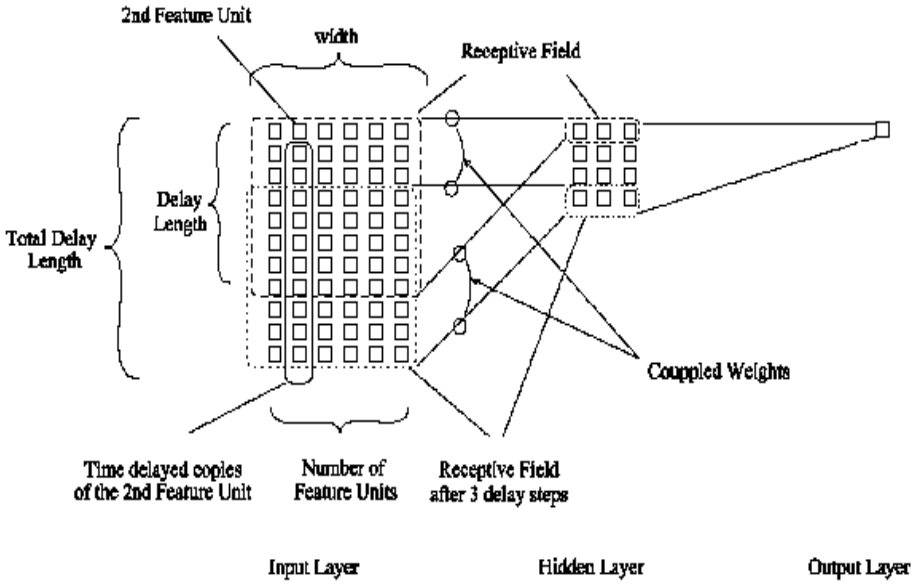


Fig. 1. The naming conventions of TDNNs

the input layer of a TDNN as there are features. Delay: In order to be able to recognize patterns place or time-invariant, older activation and connection values of the feature units have to be stored. This is performed by making a copy of the feature units with all their outgoing connections in each time step, before updating the original units. The total number of time steps saved by this procedure is called delay.

Receptive Field: The feature units and their delays are fully connected to the original units of the subsequent layer. These units are called the receptive field. The receptive field is usually, but not necessarily, as wide as the number of feature units; the feature units might also be split up between several receptive fields. Receptive fields may overlap in the source plane, but do have to cover all feature units.

Total Delay Length: The length of the layer. It equals the sum of the length of all delays of the network layers topological following the current one minus the number of these subsequent layers.

Coupled Links: Each link in a receptive field is reduplicated for every subsequent step of time up to the total delay length. During the learning phase, these links are treated as a single one and are changed according to the average of the changes they would experience if treated separately. Also the units' bias which realizes a special sort of link weight is duplicated over all delay steps of a current feature unit. In figure only two pairs of coupled links are depicted (out of 54 quadruples) for simplicity reasons.

4.1 The algorithm

The activation of a unit is normally computed by passing the weighted sum of its inputs to an activation function, usually a threshold or sigmoid function. For TDNNs this behavior is modified through the introduction of delays. Now all the inputs of a unit are each multiplied by the N delay steps defined for this layer. So a hidden unit in figure would get 6 undelayed input links from the six feature units, and 7x6 = 48 input links from the seven delay steps of the 6 feature units for a total of 54 input connections. Note, that all units in the hidden layer have 54 input links, but only those hidden units activated at time 0 (at the top most row of the layer) have connections to the actual feature units. All other hidden units have the same connection pattern, but shifted to the bottom (i.e. to a later point in time) according to their position in the layer (i.e. delay position in time). By building a whole network of time delay layers, the TDNN can relate inputs in different points in time or input space.

Training in this kind of network is performed by a procedure similar to backpropagation, that takes the special semantics of coupled links into account. To enable the network to achieve the desired behavior, a sequence of patterns has to be presented to the input layer with the feature shifted within the patterns. Remember that since each of the feature units is duplicated for each frame shift in time, the whole history of activations is available at once. But since the shifted copies of the units are mere duplicates looking for the same event, weights of the corresponding connections between the time shifted copies have to be treated as one. First, a regular forward pass of backpropagation is performed, and the error in the output layer is computed. Then the error derivatives are computed and propagated backward. This yields different correction values for corresponding connections. Now all correction values for corresponding links are averaged and the weights are updated with this value.

This update algorithm forces the network to train for time/position independent detection of sub-patterns. This important feature of TDNNs makes them independent from error-prone preprocessing algorithms for time alignment. The drawback is, of course, a rather long, computationally intensive, learning phase.

The original time delay algorithm was slightly modified for implementation in SNNS, since it requires either variable network sizes or fixed length input patterns. Time delay networks in SNNS are allowed no delay in the output layer. This has the following consequences:

The input layer has fixed size.

Not the whole pattern is present at the input layer at once. Therefore one pass through the network is not enough to compute all necessary weight changes. This makes learning more computationally intensive.

The coupled links are implemented as one physical (i.e. normal) link and a set of logical links associated with it. Only the physical links are displayed in the graphical user interface. The bias of all delay units has no effect. Instead, the bias of the corresponding feature unit is used during propagation and backpropagation.

4.2 Activation Function

For time delay networks the new activation function Act_TD_Logistic has been implemented. It is similar to the regular logistic activation function Act_Logistic but takes care of the special coupled links. The mathematical notation is again

$$a_j(t+1) = \frac{1}{1 + e^{-(\sum_i w_{ij} a_i(t) - \theta_j)}} \tag{1}$$

where o_i includes now also the predecessor units along logical links.

4.3 Update Function

The update function TimeDelay_Order is used to propagate patterns through a time delay network. It's behavior is analogous to the Topological_Order function with recognition of logical links.

4.4 Learning Function

The learning function TimeDelayBackprop implements the modified backpropagation algorithm discussed above. It uses the same learning parameters as standard backpropagation.

5 Analysis and implementation of the SRS

When using a standard based system for SR (for example HMM), the probability of a speech unit w(i) given the observation O is given according to Bayes's rule. In our phoneme recognition system each phoneme of the database is represented by a model consisting of one or more states. Each state of the model consist of one neural network, which is used to predict the recurrent observation vector given one past observation vector. The neural networks of eache model are trained with the backpropagation / TimeDelayBack propalgorithm in order to minimize the prediction error the neural nets.

6 Experiments

Creating a test system to prove TDNN capability towe based on SNNS that enables eficient work. We created many test networks to obtain the network with best results of learning time.

6.1 Data

As a training and testing data we used our small database which was inspired TUKE2 sound library of human voices. We trained Slovak phonemes .

Phoneme	PAS I	PAS II	Phoneme	PAS I	PAS II
a	a	a	m	M	mg
á	á	aa	n	n	n
ā	ä	ae	ň		n
b	b	b	n	N	ng
č	č	ch	ň	ň	nj
d	d	d	o	o	o
ď	ď	dj	ó	ó	oo
e	e	e	r	r	r
é	é	ee	ŕ	R	rx
f	f	f	ŕ	ŕ	rr
g	g	g	s	s	s
h	h	h	ś	ś	śh
x	X	x	t	t	t
i	i	i	u	u	u
i	i	ii	ŭ		
i̯a	A	ia	úa	'ua	uu
i̯e	E	iu	w	w	w
j	j	j	z	z	z
i̯u	U	iu	w	w	w
j	j	j	z	z	z
...					

Table 1. PAS phonetic alphabet for Slovak language

6.2 Results

	Test patterns	Training patterns	Number of input neurons	Number of hidden neurons	Number of output neurons
NN	92%	99%	100	50	26
TDNN	95%	98%	150	50	26

Table 2. Results of training of neural networks

7 Conclusions

We proved the posibility of creating a speech recognition system based on artificial neural network. We trained networks with Standard Feed Forward and Time Delay architecture. TDNN provided good recognition capabilities of large time dependent data. The results show a large time requirements to train networks. Advantage comparing to other systems is accuracy and scalability. At present time the neural network algorithms have disadvantage of large consumption of time and memory. So there are algorithms that try to compress amount of input data with little of noise to reduce the data flow. We would like to test other algorithms of speech recognition. Our next experiments will be training a hybrid systems, containing a neural network and Hidden Markov model (HMM).

References

1. J. Fritsch: *A Modular Neural Networks for Speech Recognition, Tech. Report CMU-CS-96-203, Carnegie Mellon University, Pittsburgh, PA., 1996*
2. Hertz, J., Krogh, A., Palmer, R.: *AIntroduction to the Theory of Neural Computation. Addison-Wesley Pub., 1991.*
3. Steve Young, Dan Kershaw, Julian Odell, Dave Ollason, Valtcho Valtchev, Phil Woodland *A The HTK Book,*
4. Jozef Juhar: *A Signal processing in system of automatic speech recognition,, Košice, 1999*
5. Psutka , J.:*A Komunikace s počítačem mluvenou řečí. Academia, Praha, 1995.*
6. Tebelskis, J.: *A Speech Recognition using Neural Networks Ph. D. Thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh PA., 1995*
7. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang:*A Phoneme Recognition Using Time Delay Neural Networks. IEEE Transactions on Accoustics, Speech and Signal Processing, 37:328–339, 1989.*
8. ISIP - Institute for Signal and Information Processing, http://www.isip.msstate.edu
9. CSLU - Center for Spoken Language Understanding, http://cslu.cse.ogi.edu/
10. ICASSP - International Conference on Acoustics, Speech, and Signal Processing, http://www.icassp2002.com/
11. SNNS - Stuttgart Neural Network Simulator, http://www-ra.informatik.uni-tuebingen.de/SNNS/

Vektorová min/max metrika

Stanislav Krajči
Ústav informatiky, PF UPJŠ Košice
krajci@science.upjs.sk

8. listopadu 2002

1 Motivácia

Súčasná doba je zasiahnutá informačnou explóziou. Množstvo informácií sa zvyšuje a je čoraz ťažšie nájsť relevantné informácie. Jednou z metód sémantického webu je hľadanie podobnosti medzi objektmi, a to v najrôznejšom slova zmysle (spomeňme napríklad editačnú vzdialenosť, synonymitu slov alebo kosínusovú metriku vo vektorovom modeli). Pomocou takejto podobnosti sa tak vytvoria zhluky (tzv. clustre), ktoré výrazne pomáhajú zorientovať sa v spleti na prvý pohľad rovnocenných objektov.

Prirodzený jazyk, ktorý používame na vyjadrenie otázok využívajúcich takéto podobnosti, je vágny. Na modelovanie tejto neurčitosti môžeme použiť fuzzy prístup (viď [1]). Pri ňom už neplatí, že objekt atribút buď má alebo nemá, ale má ho do určitej miery, v určitom stupni vyjadrenom číslom z intervalu $[0, 1]$. Takto môžeme aj nečíselné atribúty objektov vyjadriť číslami a na takéto hodnoty aplikovať prepracované metódy vyhľadávania.

Jedným zo spôsobov určovania podobnosti je využitie metriky, t.j. vzdialenosti medzi objektmi. (Takáto binárna funkcia musí spĺňať nezápornosť ($d(x, y) \geq 0$), reflexiu ($d(x, y) = 0$ práve vtedy, keď $x = y$), symetriu ($d(x, y) = d(y, x)$) a trojuholníkovú nerovnosť ($d(x, z) \leq d(x, y) + d(x, z)$) ([2]). Ak má naviac metrika d svoje funkčné hodnoty v intervale $[0, 1]$, tak funkcia s definovaná $s(x, y) = 1 - d(x, y)$ dobre vyjadruje podobnosť.

Jednou z metód hľadania podobnosti na množine objektov majúcich isté atribúty je použitie konceptuálnych zväzov ([3]). V článku [4] sme ukázali metódu fuzzifikácie konceptuálnych zväzov, prostredníctvom ktorej sme definovali istú metriku (a tým aj podobnosť) na množine objektov.

Konceptuálnymi zväzmi sa zaoberali aj Rice a Siff v článku [5]: Predstavme si tabuľku, ktorej riadky sú objekty z množiny O a stĺpce sú atribúty z množiny A. V každom políčku je číslo 1 alebo 0, ktoré hovorí, či daný objekt príslušný atribút má alebo nemá. Objekt tak možno pokladať za množinu jeho atribútov (i keď takto ekvivalentné objekty splynú). Rice a Siff na množine $\mathcal{P}(A)$ zaviedli dištančnú funkciu dvoch objektov (čiže množín) $\rho(P,Q) = \frac{|P \triangle Q|}{|P \cup Q|} = 1 - \frac{|P \cap Q|}{|P \cup Q|}$ a ukázali, že to je metrika. Túto funkciu potom využili pri vytváraní zhlukov objektov.

Ak v políčkach tabuľky nie sú len nuly a jednotky, ale čísla z intervalu $[0,1]$, ku každému atribútu tak vyjadríme stupeň, v ktorom ho príslušný objekt má. Každý objekt je tak charakterizovaný fuzzy množinou, čo je vlastne funkcia z množiny A do intervalu $[0,1]$, resp. (v prípade konečného počtu n atribútov) vektor z $[0,1]^n$. V tomto článku zovšeobecníme Riceho a Siffovu metriku na takéto vektory, ba dokonca sa ukáže, že táto metrika funguje pre ľubovoľné vektory s nezápornými zložkami. Táto metrika sa vzhľadom na jednoduchosť svojej definície ľahko vyčísľuje a možno ju použiť (napr. na indexovanie) v ľubovoľnom modeli typu objekt-atribút (alebo nejakej jeho časti) s ľubovoľnými nezápornými hodnotami.

2 Metrika

Definujme funkciu $d : (\mathcal{R}^n_{+,0})^2 \to \mathcal{R}_{+,0}$ takto:

$$d(\langle a_1, \ldots, a_n \rangle, \langle b_1, \ldots, b_n \rangle) = 1 - \frac{\sum_{i \in \{1,\ldots,n\}} \min\{a_i, b_i\}}{\sum_{i \in \{1,\ldots,n\}} \max\{a_i, b_i\}}$$

$$d(\langle 0, \ldots, 0 \rangle, \langle 0, \ldots, 0 \rangle) = 0$$

Všimnime si, že táto funkcia je v špeciálnom prípade, keď majú vektory len hodnoty 0 alebo 1, totožná s metrikou Riceho a Siffa.

Veta 1 *Funkcia d je metrika.*

Dôkaz:

Nezápornosť a reflexia sú zrejmé a symetria d vyplýva zo symetrie funkcií min a max. Takisto je tvrdenie zrejmé v prípade, že je niektorý z vektorov nulový. Dokážeme, že pre nezáporné a nenulové vektory $a = \langle a_1, \ldots, a_n \rangle$, $b = \langle b_1, \ldots, b_n \rangle$ a $c = \langle c_1, \ldots, c_n \rangle$ platí $d(a,b) + d(b,c) \geq d(a,c)$.

Predpokladajme najprv, že pre každé $i \in \{1, \ldots, n\}$ platí $0 \leq a_i \leq b_i \leq c_i$ alebo $a_i \geq b_i \geq c_i \geq 0$. Označme $P \subseteq \{1, \ldots, n\}$ množinu tých indexov i, pre ktoré platí $a_i \leq b_i \leq c_i$, pre ostatné indexy i z $Q = \{1, \ldots, n\} \setminus P$ musí platiť $a_i \geq b_i \geq c_i$, nie však $a_i = b_i = c_i$.

Označme ďalej $A_P = \sum_{i \in P} a_i$, $A_Q = \sum_{i \in Q} a_i$, $B_P = \sum_{i \in P} b_i$, $B_Q = \sum_{i \in Q} b_i$, $C_P = \sum_{i \in P} c_i$, $C_Q = \sum_{i \in Q} c_i$, zrejme $A_P \leq B_P \leq C_P$ a $A_Q \geq B_Q \geq C_Q$.

Potom

$$d(a,b) = 1 - \frac{\sum_{i \in \{1,\ldots,n\}} \min\{a_i, b_i\}}{\sum_{i \in \{1,\ldots,n\}} \max\{a_i, b_i\}} =$$

$$= 1 - \frac{\sum_{i \in P} \min\{a_i, b_i\} + \sum_{i \in Q} \min\{a_i, b_i\}}{\sum_{i \in P} \max\{a_i, b_i\} + \sum_{i \in Q} \max\{a_i, b_i\}} =$$

$$= 1 - \frac{\sum_{i \in P} a_i + \sum_{i \in Q} b_i}{\sum_{i \in P} b_i + \sum_{i \in Q} a_i} = 1 - \frac{A_P + B_Q}{B_P + A_Q},$$

analogicky

$$d(b,c) = 1 - \frac{B_P + C_Q}{C_P + B_Q} \quad \text{a} \quad d(a,c) = 1 - \frac{A_P + C_Q}{C_P + A_Q}.$$

Chceme teda ukázať, že

$$\left(1 - \frac{A_P + B_Q}{B_P + A_Q}\right) + \left(1 - \frac{B_P + C_Q}{C_P + B_Q}\right) \geq \left(1 - \frac{A_P + C_Q}{C_P + A_Q}\right).$$

Označme ďalej $t = A_P \geq 0$, $u = B_P - A_P \geq 0$, $v = C_P - B_P \geq 0$ a $x = C_Q \geq 0$, $y = B_Q - C_Q \geq 0$, $z = A_Q - B_Q \geq 0$, potom chceme

$$\left(1 - \frac{t + (x+y)}{(t+u) + (x+y+z)}\right) + \left(1 - \frac{(t+u) + x}{(t+u+v) + (x+y)}\right) \geq$$

$$\geq \left(1 - \frac{t+x}{(t+u+v) + (x+y+z)}\right).$$

Sublema 1 *Ak* $0 \leq a \leq b$, $b \neq 0$ *a* $x \geq 0$, *tak platí* $\frac{a}{b} \leq \frac{a+x}{b+x}$.

Dôkaz: Ak by $\frac{a}{b} > \frac{a+x}{b+x}$, tak $ab + ax > ba + bx$, z čoho $0 > (b-a)x$ – spor.

Použitím tejto lemy dostávame

$$\left(1 - \frac{t + (x+y)}{(t+u) + (x+y+z)}\right) + \left(1 - \frac{(t+u) + x}{(t+u+v) + (x+y)}\right) \geq$$

$$\geq \left(1 - \frac{(t + (x+y)) + v}{((t+u) + (x+y+z)) + v}\right) + \left(1 - \frac{((t+u) + x) + z}{((t+u+v) + (x+y)) + z}\right) =$$

$$= 2 - \frac{(t+x+y+v) + (t+u+x+z)}{(t+u+v) + (x+y+z)} =$$

$$= 2 - \frac{t+u+v+x+y+z}{t+u+v+x+y+z} - \frac{t+x}{(t+u+v) + (x+y+z)} =$$

$$= 1 - \frac{t+x}{(t+u+v) + (x+y+z)},$$

čo sme chceli dokázať.

Teraz rozoberme všeobecný prípad, t.j. že $a, b, c \in \mathcal{R}_{+,0} \setminus \{\langle 0, \ldots, 0\rangle\}$. Označme $P \subseteq \{1, \ldots, n\}$ množinu tých indexov, pre ktoré platí $a_i \leq c_i$, pre indexy i z $Q = \{1, \ldots, n\} \setminus P$ potom musí byť $a_i > c_i$. Množinu P rozdeľme na tri disjunktné časti podľa umiestnenia b_i:

- $i \in P_{BAC}$ práve vtedy, keď $b_i < a_i \leq c_i$,
- $i \in P_{ABC}$ práve vtedy, keď $a_i \leq b_i \leq c_i$,
- $i \in P_{ACB}$ práve vtedy, keď $a_i \leq c_i < b_i$,

analogicky rozdeľme Q na tri disjunktné časti:

- $i \in Q_{BCA}$ práve vtedy, keď $b_i < c_i < a_i$,
- $i \in Q_{CBA}$ práve vtedy, keď $c_i \leq b_i \leq a_i$, ale nie $c_i = b_i = a_i$,
- $i \in Q_{CAB}$ práve vtedy, keď $c_i < a_i < b_i$.

Definujme $f = \langle f_1, \ldots, f_n \rangle$, $g = \langle g_1, \ldots, g_n \rangle$ a $h = \langle h_1, \ldots, h_n \rangle$ takto:

$$f_i = \begin{cases} a_i, & \text{ak } i \in P_{ABC} \cup P_{ACB} \cup Q_{BCA} \cup Q_{CBA}, \\ b_i, & \text{ak } i \in P_{BAC} \cup Q_{CAB}, \end{cases}$$

$$g_i = \begin{cases} a_i, & \text{ak } i \in P_{BAC} \cup Q_{CAB}, \\ b_i, & \text{ak } i \in P_{ABC} \cup Q_{CBA}, \\ c_i, & \text{ak } i \in P_{ACB} \cup Q_{BCA}, \end{cases}$$

$$h_i = \begin{cases} b_i, & \text{ak } i \in P_{ACB} \cup Q_{BCA}, \\ c_i, & \text{ak } i \in P_{BAC} \cup P_{ABC} \cup Q_{CBA} \cup Q_{CAB}, \end{cases}$$

Všimnime si, že pre každé i je $\langle f_i, g_i, h_i \rangle$ permutáciou $\langle a_i, b_i, c_i \rangle$. Naviac pre $i \in P_{BAC} \cup P_{ABC} \cup P_{ACB} = P$ platí $0 \leq f_i \leq g_i \leq h_i$ a pre zvyšné $i \in Q_{BCA} \cup Q_{CBA} \cup Q_{CAB} = Q$ zas $f_i \geq g_i \geq h_i \geq 0$. Podľa predchádzajúcej časti teda pre vektory f, g a h platí $d(f,g) + d(g,h) \geq d(f,h)$.

Ukážeme, že $d(f,h) \geq d(a,c)$, $d(f,g) \leq d(a,b)$ a $d(g,h) \leq d(b,c)$, z čoho už vyplynie žiadané $d(a,b) + d(b,c) \geq d(a,c)$.

$$d(f,h) = 1 - \frac{\sum_{i \in \{1,\ldots,n\}} \min\{f_i, h_i\}}{\sum_{i \in \{1,\ldots,n\}} \max\{f_i, h_i\}} =$$

$$= 1 - \frac{\sum_{i \in P_{BAC}} \min\{f_i, h_i\} + \sum_{i \in P_{ABC}} \min\{f_i, h_i\} + \sum_{i \in P_{ACB}} \min\{f_i, h_i\} +}{\sum_{i \in P_{BAC}} \max\{f_i, h_i\} + \sum_{i \in P_{ABC}} \max\{f_i, h_i\} + \sum_{i \in P_{ACB}} \max\{f_i, h_i\} +}$$

$$\frac{+ \sum_{i \in Q_{BCA}} \min\{f_i, h_i\} + \sum_{i \in Q_{CBA}} \min\{f_i, h_i\} + \sum_{i \in Q_{CAB}} \min\{f_i, h_i\}}{+ \sum_{i \in Q_{BCA}} \max\{f_i, h_i\} + \sum_{i \in Q_{CBA}} \max\{f_i, h_i\} + \sum_{i \in Q_{CAB}} \max\{f_i, h_i\}} =$$

$$= 1 - \frac{\sum_{i \in P_{BAC}} \min\{b_i, c_i\} + \sum_{i \in P_{ABC}} \min\{a_i, c_i\} + \sum_{i \in P_{ACB}} \min\{a_i, b_i\} +}{\sum_{i \in P_{BAC}} \max\{b_i, c_i\} + \sum_{i \in P_{ABC}} \max\{a_i, c_i\} + \sum_{i \in P_{ACB}} \max\{a_i, b_i\} +}$$

$$\frac{+ \sum_{i \in Q_{BCA}} \min\{a_i, b_i\} + \sum_{i \in Q_{CBA}} \min\{a_i, c_i\} + \sum_{i \in Q_{CAB}} \min\{b_i, c_i\}}{+ \sum_{i \in Q_{BCA}} \max\{a_i, b_i\} + \sum_{i \in Q_{CBA}} \max\{a_i, c_i\} + \sum_{i \in Q_{CAB}} \max\{b_i, c_i\}} =$$

$$= 1 - \frac{\sum_{i \in P_{BAC}} b_i + \sum_{i \in P_{ABC}} a_i + \sum_{i \in P_{ACB}} a_i +}{\sum_{i \in P_{BAC}} c_i + \sum_{i \in P_{ABC}} c_i + \sum_{i \in P_{ACB}} b_i +}$$

$$\frac{+ \sum_{+i \in Q_{BCA}} b_i + \sum_{i \in Q_{CBA}} c_i + \sum_{i \in Q_{CAB}} c_i}{+ \sum_{+i \in Q_{BCA}} a_i + \sum_{i \in Q_{CBA}} a_i + \sum_{i \in Q_{CAB}} b_i} \geq$$

$$\geq 1 - \frac{\sum_{i\in P_{BAC}} a_i + \sum_{i\in P_{ABC}} a_i + \sum_{i\in P_{ACB}} a_i + }{\sum_{i\in P_{BAC}} c_i + \sum_{i\in P_{ABC}} c_i + \sum_{i\in P_{ACB}} c_i + }$$
$$\frac{+ \sum_{i\in Q_{BCA}} c_i + \sum_{i\in Q_{CBA}} c_i + \sum_{i\in Q_{CAB}} c_i}{+ \sum_{i\in Q_{BCA}} a_i + \sum_{i\in Q_{CBA}} a_i + \sum_{i\in Q_{CAB}} a_i}$$

(pretože $\sum_{i\in P_{BAC}} b_i \leq \sum_{i\in P_{BAC}} a_i$, $\sum_{i\in P_{ACB}} b_i \geq \sum_{i\in P_{ACB}} c_i$, $\sum_{i\in Q_{BCA}} b_i \leq \sum_{i\in Q_{BCA}} c_i$ a $\sum_{i\in Q_{CAB}} b_i \geq \sum_{i\in Q_{CAB}} a_i$), a ďalej

$$= 1 - \frac{\sum_{i\in P} a_i + \sum_{i\in Q} c_i}{\sum_{i\in P} c_i + \sum_{i\in Q} a_i} = 1 - \frac{\sum_{i\in P} \min\{a_i, c_i\} + \sum_{i\in Q} \min\{a_i, c_i\}}{\sum_{i\in P} \max\{a_i, c_i\} + \sum_{i\in Q} \max\{a_i, c_i\}} =$$

$$= 1 - \frac{\sum_{i\in\{1,\ldots,n\}} \min\{a_i, c_i\}}{\sum_{i\in\{1,\ldots,n\}} \max\{a_i, c_i\}} = d(a, c).$$

Ďalej

$$d(f, g) = 1 - \frac{\sum_{i\in\{1,\ldots,n\}} \min\{f_i, g_i\}}{\sum_{i\in\{1,\ldots,n\}} \max\{f_i, g_i\}} =$$

$$= 1 - \frac{\sum_{i\in P_{BAC}} \min\{f_i, g_i\} + \sum_{i\in P_{ABC}} \min\{f_i, g_i\} + \sum_{i\in P_{ACB}} \min\{f_i, g_i\} + }{\sum_{i\in P_{BAC}} \max\{f_i, g_i\} + \sum_{i\in P_{ABC}} \max\{f_i, g_i\} + \sum_{i\in P_{ACB}} \max\{f_i, g_i\} + }$$
$$\frac{+ \sum_{i\in Q_{BCA}} \min\{f_i, g_i\} + \sum_{i\in Q_{CBA}} \min\{f_i, g_i\} + \sum_{i\in Q_{CAB}} \min\{f_i, g_i\}}{+ \sum_{i\in Q_{BCA}} \max\{f_i, g_i\} + \sum_{i\in Q_{CBA}} \max\{f_i, g_i\} + \sum_{i\in Q_{CAB}} \max\{f_i, g_i\}} =$$

$$= 1 - \frac{\sum_{i\in P_{BAC}} \min\{b_i, a_i\} + \sum_{i\in P_{ABC}} \min\{a_i, b_i\} + \sum_{i\in P_{ACB}} \min\{a_i, c_i\} + }{\sum_{i\in P_{BAC}} \max\{b_i, a_i\} + \sum_{i\in P_{ABC}} \max\{a_i, b_i\} + \sum_{i\in P_{ACB}} \max\{a_i, c_i\} + }$$
$$\frac{+ \sum_{i\in Q_{BCA}} \min\{a_i, c_i\} + \sum_{i\in Q_{CBA}} \min\{a_i, b_i\} + \sum_{i\in Q_{CAB}} \min\{b_i, a_i\}}{+ \sum_{i\in Q_{BCA}} \max\{a_i, c_i\} + \sum_{i\in Q_{CBA}} \max\{a_i, b_i\} + \sum_{i\in Q_{CAB}} \max\{b_i, a_i\}} =$$

$$= 1 - \frac{\sum_{i\in P_{BAC}} b_i + \sum_{i\in P_{ABC}} a_i + \sum_{i\in P_{ACB}} a_i + }{\sum_{i\in P_{BAC}} a_i + \sum_{i\in P_{ABC}} b_i + \sum_{i\in P_{ACB}} c_i + }$$
$$\frac{+ \sum_{i\in Q_{BCA}} c_i + \sum_{i\in Q_{CBA}} b_i + \sum_{i\in Q_{CAB}} a_i}{+ \sum_{i\in Q_{BCA}} a_i + \sum_{i\in Q_{CBA}} a_i + \sum_{i\in Q_{CAB}} b_i} \leq$$

$$\leq 1 - \frac{\sum_{i\in P_{BAC}} b_i + \sum_{i\in P_{ABC}} a_i + \sum_{i\in P_{ACB}} a_i + }{\sum_{i\in P_{BAC}} a_i + \sum_{i\in P_{ABC}} b_i + \sum_{i\in P_{ACB}} b_i + }$$

$$\frac{+\ \sum_{i\in Q_{BCA}} b_i + \sum_{i\in Q_{CBA}} b_i + \sum_{i\in Q_{CAB}} a_i}{+\ \sum_{i\in Q_{BCA}} a_i + \sum_{i\in Q_{CBA}} a_i + \sum_{i\in Q_{CAB}} b_i}$$

(pretože $\sum_{i\in P_{ACB}} c_i \leq \sum_{i\in P_{ACB}} b_i$ a $\sum_{i\in Q_{BCA}} c_i \geq \sum_{i\in Q_{BCA}} b_i$), a ďalej

$$= 1 - \frac{\sum_{i\in P_{BAC}} \min\{a_i, b_i\} + \sum_{i\in P_{ABC}} \min\{a_i, b_i\} + \sum_{i\in P_{ACB}} \min\{a_i, b_i\} +}{\sum_{i\in P_{BAC}} \max\{a_i, b_i\} + \sum_{i\in P_{ABC}} \max\{a_i, b_i\} + \sum_{i\in P_{ACB}} \max\{a_i, b_i\} +}$$

$$\frac{+\ \sum_{i\in Q_{BCA}} \min\{a_i, b_i\} + \sum_{i\in Q_{CBA}} \min\{a_i, b_i\} + \sum_{i\in Q_{CAB}} \min\{a_i, b_i\}}{+\ \sum_{i\in Q_{BCA}} \max\{a_i, b_i\} + \sum_{i\in Q_{CBA}} \max\{a_i, b_i\} + \sum_{i\in Q_{CAB}} \max\{a_i, b_i\}} =$$

$$= 1 - \frac{\sum_{i\in\{1,\ldots,n\}} \min\{a_i, b_i\}}{\sum_{i\in\{1,\ldots,n\}} \max\{a_i, b_i\}} = d(a, b).$$

Vzťah $d(g, h) \leq d(b, c)$ dokážeme analogicky.

Literatúra

1 Krajči, S., Lencses, R., Medina, J., Ojeda-Aciego, M., Vojtáš, P.: A Similarity-Based Unification Model for Flexible Querying, Flexible Query Answering Systems, 5th International Conference, FQAS 2002, Copenhagen, Denmark, October 27-29, 2002, Proceedings, Springer-Verlag, 2002

2 Šalát, T.: Metrické priestory, Alfa, 1981

3 Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations, Springer-Verlag, 1999

4 Snášel, V., Ďuráková, D., Krajči, S., Vojtáš: Fuzzy Concept Order, proceedings of Advances in Formal Concept Analysis for Knowledge Discovery in Databases, Workshop at the 15th European Conference on Artificial Intelligence, Lyon, France, July 22-26, 2002, p. 94-98

5 Rice, M. D., Siff, M.: Clusters, Concepts, and Pseudometrics, preprint

Formálna analýza bezpečnosti protokolu IKE

Rastislav Krivoš-Belluš

Institute Of Computer Science
Faculty Of Science
Pavol Jozef Šafárik University,
Jesenná 5, 041 54 Košice, Slovakia

rkb@science.upjs.sk

Abstrakt

Táto práca sa zaoberá popísaním možností analyzovania a dokazovania bezpečnosti kryptografických protokolov. Analyzuje ISAKMP/IKE protokol použitím BAN logiky a analyzátora SPEAR2.

1 Úvod

Rozširujúce používanie Internetu prináša bezpečnosť komunikácie do pozornosti verejnosti. Zachovanie bezpečnosti informácií posielaných prostredníctvom počítačových sietí však nie je len ochranou údajov elektronického bankovníctva, ale existujú aj iné chúlostivé informácie, ktoré si vymieňajú firmy navzájom medzi sebou. Ich získanie by mohlo mať veľký význam pre konkurenciu. Z týchto dôvodov sa jednotlivé správy šifrujú pomocou kryptografických algoritmov. Všetky algoritmy sú založené na použití nejakého kľúča, na ktorom sa odosielateľ a prijímateľ dohodli. Práve na dohodnutie kľúčov sa používajú kryptografické protokoly, o bezpečnosti ktorých sa zaoberáme v tejto práci.

Úlohou tejto práce je popísanie možností dokazovania bezpečnosti kryptografických protokolov a ich uplatnenie pri posúdení bezpečnosti kryptografických protokolov.

2 Protokoly

Protokol je konečná postupnosť posielaných správ.

Počas vykonávania protokolu (spojenia), účastníci komunikujú správami v poradí ako je definované v protokole. Avšak v skutočnosti účastníci môžu niektoré správy poslať simultánne a prenos týchto správ sa môže prekrývať v čase (správy môžu absolvovať rôzne cesty, aj keď sa týkajú tých istých účastníkov).

V počítačových sieťach komunikujúce strany nezdieľajú len médium (prenosový kanál), ale aj množinu pravidiel na komunikáciu. Tieto pravidlá, protokoly,

sa v dnešnej dobe stávajú stále dôležitejšími v komunikačných sieťach. Lenže zvyšovanie množstva informácií o komunikačných protokoloch vyvoláva zvýšený záujem o otázky ako zabezpečiť komunikáciu pred ,,votrelcami“, útočníkmi.

Kryptografické protokoly sú špeciálnym druhom protokolov. Základnými úlohami [10] kryptografických protokolov sú výmena kľúčov (všeobecnejšie manažment kľúčov) a autentifikácia. Cieľom autentifikácie je zaručenie identity účastníka. Cieľom výmeny kľúča je dohodnúť medzi účastníkmi kľúč, ktorý bude používaný v ďalšej komunikácii medzi nimi, na šifrovanie posielaných údajov.

Kryptografické protokoly boli vyvinuté na boj proti útokom votrelcov v počítačových sieťach. Dnešné chápanie bezpečnosti je, že bezpečnosť údajov sa má spoliehať na kryptografickú technológiu, a že protokoly by mali byť otvorené a dostupné. Pôvodná predstava o kryptografických protokoloch bola založená na tom, že ich bezpečnosť záleží na spôsobe použitého šifrovania (a jeho sile, dĺžke použitého kľúča). Avšak, veľké množstvo protokolov sa ukázalo napadnuteľných útokom nie získaním kľúča prelomením kryptografického algoritmu, ale manipuláciou posielaných správ v protokole k získaniu nejakých výhod.

Možné útoky [5], presnejšie útočníkov, môžeme podľa spôsobu pripojenia rozdeliť na: odpočúvateľ (eavesdrop) - zachytí a pošle nezmenenú správu, prerušiteľ (intercept) - zachytí správu a nepošle nič, napodobiteľ (fake) - pošle vlastnú novú správu, modifikátor (modificate) - zachytí správu a pošle zmenenú.
Konkrétne útoky na protokol môžu byť zložené aj z viacerých typov útočníkov, napríklad najprv útočník môže odpočúvať, a ak sa mu podarí získať dosť informácii, pošle novú (alebo modifikovanú) správu. Ďalším typickým útokom je odpočutie správy, a následné zopakovanie tej istej správy.

3 Formálna analýza

Chyby nachádzané v rôznych kryptografických protokoloch viedli k úvahám o možnosti formálne dokazovať vlastnosti-bezpečnosť protokolov.
Základné otázky - problémy, na ktoré hľadáme odpovede, sú:

1. Čo protokolom dosiahneme? Robí to, čo chceme?
2. Aké predpoklady vyžaduje protokol? Potrebuje ich viac ako iné protokoly?
3. Robí protokol niečo redundantné (nadbytočné kroky, resp. správy)?

Na pomoc v súčasnosti existuje viacero dokázaných a použiteľných formálnych techník. Rozdeliť by sme ich mohli do 3 skupín [9]:

- *metódy konštruujúce odvodenie (Inference-construction methods)* - využívajú modálnu logiku. Najznámejšie sú BAN [1] a GNY [4].
- *metódy konštruujúce útok (Attack-construction methods)* - vytvárajú množinu možných útokov založenú na algebraických vlastnostiach algoritmov použitých v protokole.

– *metódy konštrujúce dôkaz (Proof-construction methods)* - pokúšajú sa odstrániť exponencionálne vyhľadávanie metód konštruujúcich útok formálnym modelovaním výpočtov spracovaných v protokoloch a dokazovaním hypotéz o týchto výpočtoch. Najznámejšie je AAPA, zatiaľ vo verzii 2 [2].

Kryptografické logiky môžu byť použité tiež na explicitné popísanie vývoja viер počas jedného spojenia protokolu, týkajúcich sa obsahu správ, počtu správ, pomáhajúc odhaliť minimálny počet správ potrebných na dosiahnutie stanovených cieľov.
BAN logika vyvolala zrod množstva príbuzných logík, každá sa snažila niečo vylepšiť alebo pridať podstatné predpoklady. Každá má svoje výhody, nevýhody a vlastné záujmy.
My sa budeme podrobne zaoberať metódami konštruujúcimi odvodenie.

3.1 Ohraničenia formálnych metód

Žiadny systém nebude nikdy 100% spoľahlivý. Systém nikdy nie je spustený v izolácii, pracuje v nejakom prostredí. Formálna špecifikácia systému musí vždy obsahovať predpoklady kladené na toto prostredie. Dôkaz korektnosti je platný len vtedy, keď tieto predpoklady sú dodržané. Takže, akonáhle nie je nejaký predpoklad splnený, všetky dôkazy sú neplatné. V skutočnosti, na zdolanie systému šikovný útočník zistí ako sa tieto predpoklady dajú porušiť.
Metódy formálnej analýzy kryptografických protokolov, nemôžu zaručiť, že protokol je naozaj bezpečný. To, čo zaručujú je, že protokol je korektný, v súlade s formálnym aparátom, ktorý logika poskytuje. V každom prípade však úspešná analýza zvyšuje dôveru k protokolu.

3.2 SPEAR 2

Security Protocol Engineering and Analysis Resource, SPEAR bolo vyvinuté roku 1997 pánmi Paul de Goede, J.P. Bekmann a Andrew Hutchison. Na samotnú analýzu využíva logiku GNY a volania prologu - protokol pretransformuje do súboru v jazyku prolog (v tomto jazyku sú popísané všetky odvodenia), a následne spustí interpreter jazyka prolog (odporúčaný je voľne šíriteľný (licencia GPL2[1]) program SWI-Prolog, momentálne vo verzii 3.4.1, vytvorený v roku 2000 na Amsterdamskej univerzite).
Samotná analýza spočíva v zadefinovaní protokolu pomocou nástroja GYPSIE, počiatočných vier ako aj cieľov vo Visual GNY (čo je súčasť GYPSIE), ďalej spustenie analyzátora GYNGER, ktorý protokol zapíše v syntaxi prologu a spustí analýzu. Na analýzu je síce štandardne použitá logika GNY (zapísana v súbore `gny.pl`), ale možno použiť aj iné logiky (prípadne pridať či upraviť niektoré pravidlá). Výsledok je potom zase graficky znázornený (čo sa podarilo dokázať - uvedené takisto odvodenie, a čo nie).

[1] General Public License

4 IKE

4.1 IKE protokol

Internet Key Exchange (IKE) je akousi protokolov SKEME a Oakley, ktoré pracujú v rámci protokolu ISAKMP(Internet Security Association and Key Management Protokol). ISAKMP predstavuje systém na vytvorenie bezpečných asociácií (konkrétne bezpečné spojenie) a kryptografických kľúčov, ale nepredpisuje nijaký konkrétny autentifikačný mechanizmus, teda je použiteľný s veľkým množstvom bezpečnostných protokolov. Oakley, naproti tomu, je sprievodcom, protokolom dohovoru kľúča, ktorý generujú obidve strany spolu. IKE dokument popisuje ako môže byť Oakley protokol použitý ako konkretizácia ISAKMP protokolu.
Cieľom tohto protokolu je zabezpečiť, že dohodnutá bezpečnostná asociácia (Security Association, ďalej len SA) je dôverná, a známa iba dvom účastníkom výmeny, a že títo účastníci sú naozaj tými, za ktorých sa vydávajú.
Oakley môže byť použitý v štyroch režimoch: základný (main), agresívny (aggresive), rýchly (quick) a najnovšie aj skupinový (group). Jednotlivé režimy sa líšia v spôsobe uplatnenia bezpečnosti jednotlivých fáz, resp. zlúčenia prvej a druhej fázy (v rýchlom režime). IKE je kombináciou množstva subprotokolov. V prvej fáze máme na výber z 8 protokolov, 2 režimov a 4 autentifikačných metód. Druhá fáza nám ponúka 4 protokoly. V tejto práci sa zameriame len na prvú fázu protokolu, keďže tá je podstatná, pri nej sa autentifikujú komunikujúci účastníci, dohodne sa spôsob a kľúče používané v druhej fáze. Druhá fáza nám len zabezpečuje čerstvosť kľúčov, a jej bezpečnosť je silne závislá (vôbec nemôžeme hovoriť o bezpečnosti druhej fázy, ak používa kľúč, ktorý nepovažujeme za bezpečný, teda dohodnutý bezpečnou prvou fázou) od bezpečnosti prvej fázy.
Dopredu dohodnutý master-kľúč, pomocou ktorého sa dohaduje spojenie medzi účastníkmi môže byť symetrický, vtedy hovoríme o autentifikačnej metóde PSK (pre-shared key), dohodnutom symetrickom kľúči.

PSK - pre-shared key, dohodnutý symetrický kľúč
Dohodnutý symetrický kľúč je označený KIR. Potom hash funkcie používané na kontrolu $HASH_I = HASH(KIR, N_I, N_R)$, $HASH_R = HASH(KIR, N_R, N_I)$.
Základný režim popisuje nasledujúci zápis protokolu:

1. $I \rightarrow R : HDR, SA_I$
2. $R \rightarrow I : HDR, SA_R$
3. $I \rightarrow R : HDR, KE_I, N_I$
4. $R \rightarrow I : HDR, KE_R, N_R$
5. $I \rightarrow R : HDR*, ID_{II}, HASH_I$
6. $R \rightarrow I : HDR*, ID_{IR}, HASH_R$

Agresívny režim:

1. $I \rightarrow R : HDR, SA_I, KE_I, N_I, ID_{II}$
2. $R \rightarrow I : HDR, SA_R, KE_R, N_R, ID_{IR}, HASH_R$
3. $I \rightarrow R : HDR, HASH_I$

4.2 Analýza agresívneho režimu IKE protokolu BAN logikou

V 1 metóde autentifikácie PSK IKE protokolu dokážeme, že agresívny režim prvej fázy je bezpečný. Samotná analýza je založená na myšlienkách a ideách Kai Martiusa[6]. Dodefinujeme, rozšírme základnú BAN logiku, o vlastnosti:
`has` - mať nejakú formulu (ide o rozšírenie `sees`, kde mať môžeme aj nejaké kľúče na začiatku ako predpoklady, neskôr všetko čo ,,vidíme“, už máme).
`says` - ,,hovorí“, teda povedal niečo čerstvé.
Chceme dokázať, že účastníci boli autentifikovaní, dohodli si kľúč, ktorý je pre dané spojenie čerstvý.
Vlastníctvo kľúča:

$$I \,\texttt{has}\, K \tag{4.1a}$$
$$R \,\texttt{has}\, K \tag{4.1b}$$

Viera, že druhý účastník má kľúč:

$$I \models R \,\texttt{says}\, K \tag{4.2a}$$
$$R \models I \,\texttt{says}\, K \tag{4.2b}$$

Vhodnosť kľúča (to že je dohodnutý na komunikáciu medzi účastníkmi I a R):

$$I \models I \overset{K}{\longleftrightarrow} R \tag{4.3a}$$
$$R \models I \overset{K}{\longleftrightarrow} R \tag{4.3b}$$

Posledným cieľom je čerstvosť kľúča:

$$I \models \sharp(K) \tag{4.4a}$$
$$R \models \sharp(K) \tag{4.4b}$$

V tejto metóde je KIR dohodnutý symetrický kľúč. Predpoklady zahŕňajú vlastníctvo dohodnutého kľúča, vieru, že je vhodný, a vieru vo vlastníctvo a v čerstvosť vlastných parametrov:

$$I \,\texttt{has}\, KIR \tag{4.5}$$
$$I \models I \overset{KIR}{\longleftrightarrow} R \tag{4.6}$$
$$I \models \sharp(KE_I, N_I, SA_I) \tag{4.7}$$
$$I \,\texttt{has}\, \{KE_I, N_I, SA_I\} \tag{4.8}$$
$$R \,\texttt{has}\, KIR \tag{4.9}$$
$$R \models I \overset{KIR}{\longleftrightarrow} R \tag{4.10}$$
$$R \models \sharp(KE_R, N_R, SA_R) \tag{4.11}$$
$$R \,\texttt{has}\, \{KE_R, N_R, SA_R\} \tag{4.12}$$

Výsledným, dohadovaným kľúčom je kľúč $K = h(KIR, N_I, N_R)$ (def.), ktorý je vytvorený jednosmernou funkciou (označenou ako h) z formúl KIR, N_I, N_R. Na výpočet kľúča potrebujeme dodefinovať ešte pravidlo, ktoré hovorí, že z dopredu dohodnutého kľúča KIR, použitím funkcie, ktorej jedným z parametrov je práve vhodný kľúč KIR, dostaneme zase vhodný kľúč medzi účastníkmi:

$$\frac{P \models P \overset{KIR}{\longleftrightarrow} Q,\ P\ \texttt{has}\ F(KIR, X)}{P \models P \overset{F(KIR,X)}{\longleftrightarrow} Q} \tag{4.13}$$

Samotná analýza pozostáva z postupného odvodzovania posielaných správ:

1. $I \to R : \{SA_I, KE_I, N_I, ID_{II}\}$

$$\Longrightarrow R \triangleleft (SA_I, KE_I, N_I, ID_{II}) \Longrightarrow$$
$$\Longrightarrow R\ \texttt{has}\ (SA_I, KE_I, N_I, ID_{II}) \tag{4.14}$$
$$(4.14), (4.5) \Longrightarrow R\ \texttt{has}\ HASH(KIR, N_I, N_R) \overset{def.}{\Longrightarrow} \tag{4.15}$$
$$\textbf{4.1b}: \quad \overset{def.}{\Longrightarrow} R\ \texttt{has}\ K$$
$$\textbf{4.3b}: \quad (4.10), (4.15) \overset{(4.13)}{\Longrightarrow} R \models I \overset{K}{\longleftrightarrow} R$$
$$\textbf{4.4b}: \quad (4.11), def. \Longrightarrow R \models \sharp(K)$$

Postupne sú takto odvodené všetky ciele.

4.3 Analýza IKE protokolu v prostredí SPEAR

Na analýzu protokolu ďalej potrebujeme definovať predpoklady - počiatočné viery a vlastníctvo, ciele a rozšírenia formúl. Počiatočné viery a ciele sú obdobné ako pri analýze pomocou BAN logiky. Počiatočné viery a ciele pre účastníka I sú zapísané v GNY logike to znamená:
$I \ni S\_KIR, N_I, SA_I, KE_I, ID_{II}$ ... verí, že vlastní vlastné komponenty a dopredu dohodnutý kľúč
$I \models I \overset{S\_KIR}{\longleftrightarrow} R$ ... verí, že S_KIR je vhodným tajomstvom medzi účastníkmi I a R
$I \models \sharp(N_I)$ ... verí v čerstvosť nonce
$I \models R \Rightarrow R \models *$ ... verí, že účastník R je čestný a kompetentný
Ciele môžeme vyjadriť ako (takisto uvádzame len ciele pre účastníka I):
$I \ni K1$ ... verí že vlastní výsledný kľúč $K1$
$I \models R \ni K1$ ... verí, že aj druhý účastník má výsledný kľúč
$I \models \sharp(K1)$ ... verí, že výsledný kľúč je čerstvý
$I \models I \overset{K1}{\longleftrightarrow} R$ ... verí, že výsledný kľúč je vhodný na použitie medzi účastníkmi I a R
$I \models R \models I \overset{K1}{\longleftrightarrow} R$ ... verí, že aj účastník R verí vo vhodnosť kľúča
Pre účastníka R sú predpoklady aj ciele zapísané analogicky.

Takto sme definovali celý protokol. Nasleduje samotná analýza protokolu. tá sa spúšťa len stlačením tlačítka `Analyze`. Protokol je automaticky pretransformovaný do výsledného súboru v prologu. Výsledky zapíše do Visual GNY, kde

pre každého účastníka uvedie, ktoré viery a vlastníctva boli dokázané a ktoré nie.
Rozanalyzovaním takto definovaného protokolu však ešte nedokážeme všetky ciele. Presnejšie, dokázané je len jedno tvrdenie pre každého účastníka. Príčinou nedokázania ostatných tvrdení je to, že doteraz v prostredí SPEAR2 nebolo definované ako sa vlastne vytvára nový kľúč. Treba nám teda dodefinovať ešte odvodzovacie pravidlá:
$\frac{P \ni N_I,\ P \ni N_R,\ P \ni S\_KIR}{P \ni K1}$ ... výsledný kľúč je vytvorený z formúl N_I, N_R, S_KIR
$\frac{P \ni N_I,\ P \ni N_R,\ P \ni S\_KIR,\ P \ni ID_{II},\ P \ni ID_{IR}}{P \models I \overset{K1}{\longleftrightarrow} R}$ ... výsledný kľúč je vhodný na použitie medzi účastníkmi identifikovanými pomocou ID
$\frac{P \ni K1,\ P \models \sharp(N_I)}{P \sharp(K1)}$, $\frac{P \ni K1,\ P \models \sharp(N_R)}{P \sharp(K1)}$ ... kľúč vytvorený z čerstvej formuly je čerstvý

Analýzou s takto dodefinovanými pravidlami už dokážeme všetky ciele okrem $I \models R \ni K1$ (a obdobne pre účastníka R). Po doplnení posledného(štvrtého) pravidla už dokážeme všetky ciele. Odvodenie cieľa $R \in K1$ je popísané postupnosťou:

```
[1] Proof for R possesses K1:
1. R was told (HDR, SAi, KEi, Ni, IDii).  {Step}
2. R possesses Nr.  {Assumption}
3. R possesses S_KIR.  {Assumption}
4. R was told (SAi, KEi, Ni, IDii).  {1, T4}
5. R possesses (SAi, KEi, Ni, IDii).  {4, P1}
6. R possesses (KEi, Ni, IDii).  {5, P4}
7. R possesses (Ni, IDii).  {6, P4}
8. R possesses Ni.  {7, P4}
9. R possesses K1.  {8, 2, 3, IKE}
```

Táto postupnosť nám presne krok po kroku popisuje jednotlivé odvodenia. Prvé odvodenie je krok protokolu, druhé a tretie sú predpoklady, teda počiatočné vlastníctvo. Štvrté odvodenie dostaneme použitím pravidla **T4**, ktorého predpokladom je tvrdenie uvedené v prvom odvodení. Posledné, deviate, odvodenie je použitie pravidla **IKE**, ktoré sme dodefinovali k odvodzovacím pravidlám. Predpokladmi tohto tvrdenia sú odvodenia v poradí ôsme, druhé a tretie.
PSK metóda agresívneho režimu Protokol IKE má teda hľadané vlastnosti (ciele) aj z pohľadu GNY logiky (a nástroja SPEAR2). Na rozdiel od BAN logiky sme naviac dokázali, že obaja účastníci veria, že aj ten druhý verí vo vhodnosť kľúča (v BAN logike to nie je potrebné dokazovať, keďže verí všetkému povedanému. Práca s týmto programom je intuitívna, protokol sa špecifikuje veľmi jednoducho. Zo skúsenosti získaných pri práci s týmto programom pri analýze IKE protokolu, možno tomuto program vytknúť nutnosť zásahu do definovanej logike, ak chceme doplniť pravidlá len pre náš konkrétny protokol. Bolo by asi vhodnejšie, keby aj tieto pravidlá sa dali dodefinovať priamo v užívateľskom prostredí, a nemuseli zapisovať do textového súboru. Ináč je ale plusom tohto programu, že vôbec môžeme špecifikovať aj vlastnú logiku, nie len autormi použitú GNY.

5 Záver

Popísali sme používané formálne metódy a analytické analyzátory slúžiace na analýzu a kontrolu kryptografických protokolov. Je ťažké povedať ktorá metóda je lepšia. Asi najlepším riešením (aj keď nepraktickým) by bolo každý protokol dokázať pomocou čo najviac rôznych metód. Vždy tie novšie nástroje dopĺňajú nejaké nové pravidlá, zvyšujú bezpečnosť. Z nami popísaných analyzátorov, asi za najlepší možno označiť AAPA2, ktorý aj pri nedokázaní nejakého cieľa, dokáže pokračovať v dokazovaní ďalších cieľov tak, že daný cieľ považuje za dokázaný (dodefinuje axiomaticky). Dokáže tak odhaliť viacero chýb naraz. Nájde chyby, ktoré sú zvyčajne nájdené až po opravení predpokladov alebo pravidiel, a dokázaní pôvodného cieľa. Na druhej strane nástroj SPEAR2 je lepší z pohľadu používateľa, aj z pohľadu jednoznačného zápisu logiky, ktorú možno doplniť, prípadne napísať kompletne novú (bohužiaľ v tomto prípade už treba logiku popísať v textovom súbore).
Existuje ešte stále veľa otázok, ktoré ostávajú otvorené. Používané analýzy popisujú situácie, keď útočník sa snaží získať nejaké informácie len zo spojení účastníkov protokolu. Existuje aj možnosť, že viacerí účastníci komunikujú so serverom pomocou toho istého protokolu, a útočník môže na útok použiť aj správy, ktoré získal od servera. Tieto spojenia so serverom sa môžu prekrývať v čase, treba zistiť, či môže útočník využiť (prerušiť) nedokončené spojenie medzi iným účastníkom a serverom. Tento spôsob posielania správ zohraje dôležitú úlohu pri spracovaní úloh použitím GRIDov (t.j. zdieľania nielen údajov ale aj výpočtového času medzi počítačmi).

Automatické analyzátory sa snažia dokázať jednotlivé ciele, ale nie vždy nájdu najkratšie odvodenia ako sa k danému cieľu dá dopracovať. Ak by existovali nástroje hľadajúce najkratšie odvodenia, tak by pri analýze bezpečnosti protokolov sa mohlo zistiť, že niektoré informácie sú posielané zbytočne (čo nie vždy musí viesť ku chybe protokolu). Takisto by daný dôkaz bol oveľa čitateľnejší ako dlhšie odvodenia.

Referencie

1. M.Burrows, M.Abadi, R.Needham, *A logic of authentication*, ACM Transaction on Computer Systems 8, February 1990
2. Stephen H. Bracking, *Automatic Analysis of Cryptographical Protocols: Final Report*, Arca Systems / Exodus Communications, www.arca.com
3. S.D.Dexter, *An Adversary-Centric Logic of Security and Authenticity, Dissertation*, University of Michigan, 1998
4. L. Gong, *Cryptographical Protocols for Distributed Systems*, PhD thesis, University of Cambridge, April 1990
5. LU MA and JEFFEREY J.P. TSAI, *Formal Verification Techniques For Computer Communications Security Protocols*, Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, 2000
6. Kai Martius, *IKE Protocol Analysis*, Department of Medical Computer Sciences and Biometrics, Dresden University of technology, October 1998

7. Catherine Meadows, *Formal Analysis of Cryptographic Protocols: A Survey*, Code 5543, Center for High Assurance Computer Systems, Washington, meadows@itd.nrl.navy.mil
8. A.Rubin and P. Honeyman, *Formal methods for the analysis of authentication protocols*, Technical Report 93-97, CITI, November 1993
9. Elton Saul, *Facilitating the Modelling and automated Analysis of cryptografic Protcols*, Data Network Architectures Laboratory, Department of Computer Science, University of Cape Town, South Africa, 2001, `www.cs.uct.ac.za/Research/DNA/SPEAR2`
10. W.Stallings, *Cryptography and Network Security*, Prentice Hall, 1999
11. Martin Abadi, Mark R. Tuttle, *A Semantics for a Logic of Authentication*, ACM Symposium on Principles of Distributed Computing 10, Montreal, Canada, August 1991
12. Jeanette M. Wing, *A Symbiotic Relationship Between Formal Methods and Security*, Carnegie Mellon University, Pittsburg, December 1998

Neural Networks in Documents Classification

Miroslav Levický

Institute of Computer Science
Faculty of Science
Pavol Jozef Šafárik University
Jesenná 5, 040 01 Košice, Slovakia

email: levicky@science.upjs.sk

Abstract

In this paper, we present a feed-forward neural network to which the Bayesian theory is applied. The Metropolis algorithm and Hybrid Monte Carlo Method, used to train concrete neural network models, are described. We have defined the document classification problem, as well as the method of their representation – vector space representation. Described neural networks models are then applied to the document classification problem. Simulation results are shown at the end of the paper.

Introduction

Amount of information in form of documents that are nowadays present on www-pages is extensive. So for a user, the information retrieval is time-consuming. Classification of the documents could help to categorize related documents and to decrease information retrieval time for users.

Documents classification is an expanding research field that presents an interface of information retrieval and machine learning. The task of a system for documents classification is to assign categories to electronic documents automatically. The categories may indicate content of the document, concepts that are considered in the documents, or the document's relevance to the user (his requests).

Training set of data with assigned class is given in the classification task. By using these data we train the model that may be used for classification of new data into single existing classes.

In this paper, we examine the use of Bayesian training networks in classification problem. We are using feed-forward neural network. Followingly, the Bayesian theory is applied to it. We describe learning process of such neural network in the paper, and present reached results at the end.

Neural Networks and the Bayesian Learning

A neural network is in general a non-linear parametrized mapping from input $\mathbf{x}$ to output $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$. The output is a continuous function of input $\mathbf{x}$ and

parameters $\mathbf{w}$. Function f denotes network architecture, or in other words, a functional form of mapping from input to output. The network can be trained to solve regression or classification problems.

The neural network is trained using the training set D by adapting $\mathbf{w}$ so that it minimizes some error function. Because in this paper we deal with multiclassification problem, we chose following error function:

$$E_D(\mathbf{w}) = -\sum_{s=1}^{N}\sum_{i=1}^{m} y_i^{(s)} \ln f_i(\mathbf{x}_s, \mathbf{w}), \tag{1}$$

where f_i, y_i is i-th element of the neural network output vector or target vector $\mathbf{y}$.

This minimization is based upon repeated evaluation of the gradient of E_D by using backpropagation algorithm. Often, regularization (also known as 'weight decay') is included in the objective function. Thus, the objective function is modified into the form of

$$E(\mathbf{w}) = E_D(\mathbf{w}) + \alpha E_W(\mathbf{w}), \tag{2}$$

where for example $E_W(\mathbf{w}) = \frac{1}{2}|\mathbf{w}|^2$, and α is a regularization constant. The term favours small values of $\mathbf{w}$, and reduces the tendency of the model to overfit the noise in the training data.

The neural network learning process can be expressed by use of probability. The error function is interpreted as minus log likelihood for a noise model:

$$P(D\,|\,\mathbf{w}) = e^{-E_D(\mathbf{w})}. \tag{3}$$

So, the use of the sum-squared error E_D (3) corresponds to an assumed Gaussian noise on the target variables.

Similarly, the regularization is denoted as a log prior probability distribution over parameters $\mathbf{w}$:

$$P(\mathbf{w}\,|\,\alpha) = \frac{1}{Z_W(\alpha)} e^{-\alpha E_W(\mathbf{w})}. \tag{4}$$

If E_W is quadratic then corresponding prior probability distribution is Gaussian with variance $\sigma_W^2 = \frac{1}{\alpha}$.

The objective function E afterwards corresponds to the inference of the parameters $\mathbf{w}$:

$$P(\mathbf{w}\,|\,D, \alpha) = \frac{P(D\,|\,\mathbf{w}, \alpha)P(\mathbf{w}\,|\,\alpha)}{P(D\,|\,\alpha)} \tag{5}$$

$$= \frac{1}{Z_E} e^{-E(\mathbf{w})}. \tag{6}$$

It might be interesting to point out, that the formula (5) was derived by using the Bayes theorem. That's why this type of learning is also called the Bayesian learning.

In the terms mentioned above, $Z_W(\alpha)$ and Z_E are constants, while $Z_W(\alpha)$ depends on parameter α.

The Bayesian inference for data modeling problems can be realized by analytical methods, Monte Carlo methods, or deterministic methods using the Gaussian approximation. Unfortunately, only few specific analytical methods for neural networks are known. That's why we deal only with Monte Carlo methods. Detailed description of Gaussian approximation of posterior distribution (5) can be found in [4], [5], [6].

Suppose the training set $D = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$ where $\mathbf{x}_i$ is input vector and $\mathbf{y}_i$ is respective output vector, and the vector $\mathbf{x}_{N+1}$. The task is to predict the respective output vector $\mathbf{y}_{N+1}$.

As mentioned above, in traditional training of neural network (e.g. by gradient descent method) we search for vector $\widehat{\mathbf{w}}$ that maximizes degree of fit to the data. In the Bayesian approach, one does not use a single set of network weights, but rather integrates the predictions from all possible weight vectors over the posterior weight distribution. The best prediction for input from test pattern (assuming squared-error loss) is given by

$$\widehat{\mathbf{y}}_{N+1} = \int_{\mathbb{R}^A} f(\mathbf{x}_{N+1}, \mathbf{w}) P(\mathbf{w} \,|\, D) \, \mathrm{d}\mathbf{w}, \tag{7}$$

where A is the dimension of the weight vector $\mathbf{w}$.

The prediction of equation (7) contains no indication of uncertainty. It does not sufficiently represent the situations when the output can be in several disjoint regions. The complete expression concerning output is given by its predictive distribution:

$$P(\mathbf{y}_{N+1} \,|\, \mathbf{x}_{N+1}, D) = \int_{\mathbb{R}^A} P(\mathbf{y}_{N+1} \,|\, \mathbf{x}_{N+1}, D, \mathbf{w}) P(\mathbf{w} \,|\, D) \, \mathrm{d}\mathbf{w}. \tag{8}$$

The distribution includes not only the uncertainty due to inherent noise, but also uncertainty in the weight vector $\mathbf{w}$. Posterior probabilities for weight vectors are given by:

$$\begin{aligned} P(\mathbf{w} \,|\, D) &= \frac{P(\mathbf{w}) P(D \,|\, \mathbf{w})}{P(D)} \\ &= \frac{P(\mathbf{w}) P(\mathbf{y}_1, \ldots, \mathbf{y}_N \,|\, \mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{w})}{P(\mathbf{y}_1, \ldots, \mathbf{y}_N \,|\, \mathbf{x}_1, \ldots, \mathbf{x}_N)} \qquad (9) \\ &= \frac{P(\mathbf{w}) \prod_{i=1}^{N} P(\mathbf{y}_i \,|\, \mathbf{x}_i, \mathbf{w})}{P(\mathbf{y}_1, \ldots, \mathbf{y}_N \,|\, \mathbf{x}_1, \ldots, \mathbf{x}_N)}. \qquad (10) \end{aligned}$$

The prior weight distribution can be following:

$$P(\mathbf{w}) = \left(2\pi\omega^2\right)^{-\frac{A}{2}} e^{-\frac{|\mathbf{w}|^2}{2\omega^2}}, \tag{11}$$

where ω is the expected weight scale; it should be set by hand.

High-dimensional integrals necessary for prediction are in general analytically unsolvable and numerically difficult to compute. This leads to problem of Bayesian learning. While in traditional training we deal with optimization problem, in Bayesian training we deal with evaluation of high-dimension integrals. Metropolis algorithm presents a method for evaluation. However, this algorithm is slow for our problem, it forms the basis for Hybrid Monte Carlo method, which should be more effective to evaluate the integrals. Now we will describe those methods simply.

Suppose that we wish to evaluate

$$\langle g \rangle = \int_{\mathbb{R}^A} g(\mathbf{q}) P(\mathbf{q}) \mathrm{d}\mathbf{q}. \tag{12}$$

The Metropolis algorithm generates a sequence of vectors $\mathbf{q}_0, \mathbf{q}_1, \ldots$, which forms Markov chain with stationary distribution $P(\mathbf{q})$. The integral in equation (12) is then approximated as

$$\langle g \rangle \approx \frac{1}{M} \sum_{t=I}^{I+M-1} g(\mathbf{q}_t), \tag{13}$$

where I stands for number of initial values which will be not used in evaluation and M stands for number of functional values g. Averaging those values one gets approximate value of $\langle g \rangle$. In limit case, as M increases, approximation converges to the real value $\langle g \rangle$. But, it is hard to determine how long it takes to reach the stationary distribution (and hence to determine the value of I), or determine how are the values of $\mathbf{q}_t$ correlated at following iterations (and hence how large M should be). One cannot avoid such difficulties in applications that utilize Metropolis algorithm.

Generation of the Markov chain is described by energy function, defined as $E(\mathbf{q}) = -\ln(P(\mathbf{q})) - \ln(Z_E)$, where Z_E is a positive constant chosen for convenience. Algorithm starts by random sampling of $\mathbf{q}_0$. In every t-th iteration, a random candidate $\widetilde{\mathbf{q}}_{t+1}$ for the next state is sampled from distribution $P(\widetilde{\mathbf{q}}_{t+1} \mid \mathbf{q}_t)$. The candidate is accepted if its energy is lower than previous state; if its energy is higher it is accepted with probability $\exp(-\Delta E)$, where $\Delta E = E(\widetilde{\mathbf{q}}_{t+1}) - E(\mathbf{q}_t)$. In other words,

$$\mathbf{q}_{t+1} = \begin{cases} \widetilde{\mathbf{q}}_{t+1} & \text{if } U < \exp(-\Delta E) \\ \mathbf{q}_t & \text{otherwise} \end{cases} \tag{14}$$

and U is a random number from uniform distribution from interval $\langle 0, 1)$.

The Metropolis algorithm can be used for evaluation of integrals in equation (7), where $\mathbf{w}$ represents $\mathbf{q}$, and energy function E is derived from equations (10) and (11):

$$E(\mathbf{w}) = -\ln P(\mathbf{w} \mid D) - \ln Z_E. \tag{15}$$

In our case, provided that the prior weight distribution (11) is Gaussian, the energy function is respective objective function for given network architecture.

The Hybrid Monte Carlo method as an improvement of Metropolis algorithm eliminates much of the random walk in weight space, and further accelerates exploration of the weight space. The method uses a gradient information provided by backgropagation algorithm.

Unlike the Metropolis algorithm, the Hybrid Monte Carlo method generates sequence of vector couples $(\mathbf{q}_0, \mathbf{p}_0), (\mathbf{q}_1, \mathbf{p}_1), \ldots$, where vectors $\mathbf{q}$ are called position vectors and vectors $\mathbf{p}$ are called momentum vectors. Both these vectors are of the same dimension. Potencial energy function $E(\mathbf{q})$ used in Metropolis algorithm is extended to Hamiltonian function $H(\mathbf{q}, \mathbf{p})$ that combines potencial and kinetic energy:

$$H(\mathbf{q}, \mathbf{p}) = E(\mathbf{q}) + \frac{1}{2} |\mathbf{p}|^2 . \tag{16}$$

$P(\mathbf{q}, \mathbf{p}) = P(\mathbf{q})P(\mathbf{p})$ is fact for the stationary distribution of generated Markov chain. Marginal distribution $\mathbf{q}$ is the same as the one for Metropolis algorithm. Thus, the value of $\langle g \rangle$ can be again approximated by use of equation (13). Momentum characteristics have Gaussian distributions, and they are independent of $\mathbf{q}$ and of each other.

The Markov chain is generated by two types of transitions - dynamic and stochastic moves. In stochastic moves, it is common to unconditionally replace actual momentum vector $\mathbf{p}$ by vector sampled from stationary distribution $P(\mathbf{p}) = (2\pi)^{-\frac{A}{2}} \exp\left(-|\mathbf{p}|^2 /2\right)$.

Dynamic moves are determined in terms of Hamilton's equations that specify the derivatives of $\mathbf{q}$ and $\mathbf{p}$ with respect to fictitious time variable τ as follows:

$$\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}\tau} = +\frac{\partial H}{\partial \mathbf{p}} = \mathbf{p} \tag{17}$$

$$\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}\tau} = -\frac{\partial H}{\partial \mathbf{q}} = -\nabla E(\mathbf{q}) \tag{18}$$

A candidate state $(\widetilde{\mathbf{q}}_{t+1}, \widetilde{\mathbf{p}}_{t+1})$ is generated in three steps. At first, the momentum vector is negated with one-half probability, then the dynamics (17) and (18) are performed for predefined period of time, and finally, the momentum vector is again negated with one-half probability. The result is accepted or rejected as it is in equation (14) but H is used instead of E.

The equations (17) and (18) are usually transformed into discrete form with some non-zero step ε. This method is called "leapfrog" method.

$$\mathbf{p}\left(\tau + \tfrac{\varepsilon}{2}\right) = \mathbf{p}(\tau) - \tfrac{\varepsilon}{2} \nabla E(\mathbf{q}(\tau)) \tag{19}$$

$$\mathbf{q}(\tau + \varepsilon) = \mathbf{q}(\tau) + \varepsilon \mathbf{p}\left(\tau + \tfrac{\varepsilon}{2}\right) \tag{20}$$

$$\mathbf{p}(\tau + \varepsilon) = \mathbf{p}\left(\tau + \tfrac{\varepsilon}{2}\right) - \tfrac{\varepsilon}{2} \nabla E(\mathbf{q}(\tau + \varepsilon)) \tag{21}$$

Results

Collection Reuters-21578 is used as documents collection to which the classification problem is used. The collection is widely accessible on Internet (please

see [10]). Every document from the collection is characterized by certain characters, e.g. date, identification number, name. The category under which the document comes is stated in every document too. Considering the facts that more categories can be assigned to the document and that we do not use "multicategories" in our tests, a pattern that assigns only one category to the documents had to be set up. For this reason, new categories which contained original categories were created. For example, new category BANKING originated by merging original categories MONEY-FX, INTEREST, RESERVES in one. Using this way we created 7 new categories named BANKING, COMMODITY, CORPORATE, CURRENCY, ECONOMIC, ENERGY and SHIPPING. We excluded the documents that did not fit to any pattern for assigning new category.

Consequently, the documents were sorted into training and testing set. The document attribute that determines to which the document belongs was used for this purpose. Seeing that the sets were too large (thousands of documents) and that algorithms were time-consuming, the sets were reduced. The training set contained 304 documents, and the testing set contained 294 documents.

It was necesary to present single documents appropriately before classification. In our tests, we used vector documents representation as the most used way of document representation. In the representation, each document is characterised either by boolean or numeric vector. The vectors are set in space whose dimensions correspond with documents corpus terms. Such vector has numeric value assigned by a function f in each element. The value mainly depends on the fact how often the term corresponding with certain dimension occurs in the document. By changing f function we may reach different ways of assigning weights to terms. For closer information about the representation please see [9].

In order to reduce the vector space we used Porter stemming algorithm, excluded stop words (using the database of 600 words), and applied Zipf's law. When using Zipf's law, we excluded words with occurence lesser than 10 times. The value was taken after few initial tests. The final vector representation was derived by using TFIDF function (also see [9]).

Following multiclassification neural network calculating the function $\mathbf{y} = f(\mathbf{x}, \mathbf{w})$ was used in the tests:

$$\text{hidden layer: } a_j^{(1)} = \sum_{l=1}^{n} w_{jl}^{(1)} x_l + \theta_j^{(1)};\ h_j = \tanh(a_j^{(1)}), \quad j = 1, \ldots, p$$
$$\text{output layer: } a_i^{(2)} = \sum_{j=1}^{p} w_{ij}^{(2)} h_j + \theta_i^{(2)};\ y_i = \frac{\exp(a_i^{(2)})}{\sum_{r=1}^{m} \exp(a_r^{(2)})},\ i = 1, \ldots, m$$

Vectors $\mathbf{x} = (x_1, \ldots, x_n)$, where n is dimension of documents vector space, represent neural network inputs. Every vector $\mathbf{w}$ stands for coordinates of one document. Vectors $\mathbf{y} = (y_1, \ldots, y_m)$, where m equals count of all categories are outputs of the neural network. Network's architecture provides that values y_i are positive and $\sum_{i=1}^{m} y_i$ converges to 1. Saying it more simply, value of y_i denotes the probability ot the fact that document with coordinates $\mathbf{x}$ belongs to the class i.

The objective function is in the form $E(\mathbf{w}) = \frac{\alpha}{2} |\mathbf{w}|^2 + E_D(\mathbf{w})$, where $E_D(\mathbf{w})$ is given by equation (1). We made experiments with different values of parameter α in our tests. The best results were achieved in the case of $\alpha = 1$.

Single models training consists of generating the Markov chain of neural network weight vectors $\mathbf{w}_0, \mathbf{w}_1, \ldots$. We used Metropolis algorithm and Hybrid Monte Carlo method in order to generate the Markov chain.

Using Metropolis algorithm we generated Markov chain $\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_{M+I}$. The meaning of M, I was described above.

Final predictions were determined by using the equation (13), where function $g(\mathbf{q}_t)$ was replaced by function $f(\mathbf{x}, \mathbf{w}_t)$. In our case $\mathbf{w}$ equals $\mathbf{q}$.

Success of each single prediction was evaluated followingly. Let $\widehat{\mathbf{y}} = (\widehat{y}_1, \ldots, \widehat{y}_m)$ be prediction determined by neural network for input vector $\mathbf{w}$, and let $\mathbf{y} = (y_1, \ldots, y_m)$ be the expected output vector for input $\mathbf{x}$. The prediction $\widehat{\mathbf{y}}$ was considered to be correct if $k = l$, where k, l were such indexes that $\widehat{y}_k = \max\{\widehat{y}_i \,|\, i = 1, \ldots, m\}$ and $y_l = \max\{y_i \,|\, i = 1, \ldots, m\}$.

Step by step we generated several Markov chains of weight vectors for our model. We made several experiments with number of generated weights. The longest chain was compound of 300 000 weights. At chain with 200 000 weights, we started experiments with setting the number of discarded iterations in prediction. By changing the amount of discarded iterations we wanted to find out their influence on final predictions success. The final result was positively influenced by the setting up to certain limit; results worsened after that certain limit was exceeded. The limit was approximately first 50 000 unused weights. The best prediction was gained at 200 000 generated weights; first 30 000 were not used for prediction. In this case, the success was 41.84% (123 successfuly classified documents).

When comparing results of our tests to results of other documents classification methods, we may conclude that our methods are not very appropriate for solving this problem. The main disadvantage - time-consuming - results from large amount of input data. We may also mention the fact that amounts of memory are required when large Markov chain is generated and memorized as an other disadvantage.

Even if the above mentioned methods were shown as inadequate for solving the documents classification problem, more ways of their use exist. For example, a part of our software was used for solving the prediction problem of geomagnetic storms (please see [1]). In this case, we trained regression neural network. By increasing the number of iterations, the predictions' successfulness noticeably increased, and total results were more than good.

Conclusions

In the paper, we dealt with feed-forward neural networks and their training. We described two Monte Carlo methods. Monte Carlo method is nowadays applied to majority of Bayesian neural networks because it provides good approximation of evaluating high-dimensional integral which is needed for training the neural networks, and because achieved results do not depend generally on data. Its greatest disadvantage is the fact that it is extremely time-consuming. But the use of Monte Carlo methods together with Gaussian approximation could

determine whether it is possible to apply Gaussian approximation to a specific problem. Comparing to Monte Carlo methods, Gaussian approximation is faster, but can not be applied to an arbitrary problem. Simulated annealing (see [2]) that represents further expansion of above mentioned methods was not covered in our work.

References

1. Andrejková, G. (2002). *Bayesian Neural Networks in Prediction of Geomagnetic Storms*. In Proceedings on EISCI International Conference.
2. Neal, R.M. (1992). *Bayesian Training of Backpropagation Networks by the Hybrid Monte Carlo Method*. Technical Report CRG-TR-92-1, Dept. of Computer Science, University of Toronto.
3. Neal, R.M. (1993). *Probabilistic Inference Using Markov Chain Monte Carlo Methods*. Technical Report CRG-TR-93-1, Dept. of Computer Science, University of Toronto.
4. MacKay, D.J.C. (1991). *Bayesian Methods for Adaptive Models*. PhD thesis, California Institute of Technology.
5. MacKay, D.J.C. (1992). *A practical Bayesian framework for backpropagation networks*. Neural Computation **4**(3): 448-472.
6. MacKay, D.J.C. (1995). *Bayesian Methods for Neural Networks: Theory and Applications*. Course notes for Neural Networks Summer School.
7. MacKay, D.J.C. (1997). *Introduction to Monte Carlo methods*. A review paper in the proceedings of an Erice summer school, ed. M.Jordan.
8. Porter, M.F. (1980). *An algorithm for suffix stripping*. Program 14(3), pp.130–137.
9. Sahami, M. (1998). *Using Machine Learning to Improve Information Access*. PhD thesis, Stanford University, Computer Science Department.
10. Reuters. *Reuters-21578, Distribution 1.0 documents collection*. Distribution for research purposes has been granted by Reuters and Carnegie Group. Arrangements for access were made by David Lewis. This data set is publicly available from David Lewis at http://www.research.att.com/∼lewis, 1997.

Hierarchical Associative Memories for Storing Spatial Patterns *

Iveta Mrázová†, Jana Tesková

Department of Software Engineering, Charles University,
Malostranské nám. 25, 118 00 Praha 1, Czech Republic

e-mail: mrazova@ksi.ms.mff.cuni.cz, teskova@ksi.ms.mff.cuni.cz

Abstract

The emerging progress in information technologies enables applications of artificial neural networks also in areas like navigation and geographical information systems, telecommunications, etc. This kind of problems requires processing of high-dimensional spatial data. Spatial patterns correspond in this context to geographic maps, room plans, etc. Techniques for an associative recall of spatial maps can be based on ideas of Fukushima [2] and they should enable a reliably quick and robust recall of presented patterns, often unknown in some parts or affected by noise. Anyway, the performance of traditional associative memories (usually Hopfield-like networks) is very sensitive to the number of stored patterns and their mutual correlations.

To avoid these limitations, we will introduce here the so-called Hierarchical Associative Memory (HAM) model. This model represents a generalization of Cascade ASsociative Memories (CASM-model) proposed by Hirahara et al. [3]. CASM-models comprise in principle two Hopfield-like networks and can be used for storing mutually highly correlated patterns. The HAM-model consists of an arbitrary number of associative memories (of the Hopfield type) grouped hierarchically in several layers. A suitable strategy applied during training the respective networks can lead to a more appropriate processing of huge amounts of real spatial data often containing mutually correlated patterns. Experimental results will be briefly discussed, too.

1 Introduction

The principle of applying associative memories in path prediction is based on the following idea. Let us imagine that we are walking through a place we have been before. Often, we have an idea of the scenery that we do not see yet but shall see soon. Triggered by the newly recalled image, we can also recall another scenery further ahead of us. As a result, we can recall the scenery of a wide area by a chain of recall processes. This brings us to the idea that the human brain stores fragmentary maps without any further information representing the exact location

*This research was supported by the grant No. 300/2002/A INF/MFF of the GA UK and by the grant No. 201/02/1456 of the GA ČR.

†Currently Fulbright Visiting Scholar in Smart Engineering Systems Laboratory, Engineering Management Department, University of Missouri-Rolla, MO 65409-0370, USA

of the maps [2]. In this way, the iterative recall of fragmentary maps helps to ensure a quick and safe movement through the environment – especially when recalling the sceneries ahead from an unknown position and "knowing only a portion of the scene" (the rest of the scene can be "damaged" or "unknown").

On the other hand, standard associative memories require higher memory costs for storing spatial patterns compared with memorizing the whole map of the scenery. The capacity of associative memories corresponds to $0.15n$, where n is denotes the dimension of the pattern space (moreover, the stored patterns should be almost orthogonal one to each other). Memory costs required to store $0.15n$ (in general real-valued) pattern vectors of the dimension n would be $0.15n^2$. The memory costs for a Hopfield network with n neurons represent $n(n-1)/2$. Thus, the ratio between the memory costs required by a conventional and an associative memory approaches 0.3 for large-dimensional patterns.

It would be possible to increase the capacity of standard associative memories by increasing the dimensionality of the patterns to be stored in them (having now proportionally more neurons). But at the same time, the number of network's weights would raise quadratically with the number of network's neurons. For this reason, the recall process would require higher computational costs as well. Moreover, real-world patterns (e.g. spatial maps) tend to be correlated rather than nearly orthogonal. This also limits the capacity of the networks. Therefore, we decided to focus our research on possibilities of a parallel implementation of this model. In this paper, we will introduce the so-called Hierarchical Associative Memories (HAM). This model is inspired by the Cascade associative memory model (CASM) proposed by Hirahara et al. [3] but it was developed with a stressed necessity to process huge amounts of data in parallel. In comparison to the CASM-model, we expect that the HAM-model will allow a reliable and quick storage and recall of larger amounts of spatial patterns.

2 The Standard Hopfield Model

First, let us specify some basic notions. A neuron with the weight vector $\vec{w} = (w_1, \ldots, w_n)$, $w_1, \ldots, w_n \in \mathbb{R}$ and the transfer function $f : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is an abstract device capable of computing the output $y \in \mathbb{R}$ as the value of the transfer function $f[\vec{w}](\vec{z})$ for any input $\vec{z} \in \mathbb{R}^n$; $\mathbb{R}$ denotes the set of all real numbers. In this paper, we will consider the hard-limiting transfer function with values equal to -1 or 1. The output of a neuron will be determined according to:

$$y(t+1) = f[\vec{w}](\vec{z}) = f(\xi) = \begin{cases} 1 & \text{if } \xi > 0 \\ y(t) & \text{if } \xi = 0 \\ -1 & \text{if } \xi < 0 \end{cases} \tag{1}$$

In the above relationship, ξ denotes the so-called neuron potential value, which can be determined as $\xi = \sum_{i=1}^{n} z_i w_i$. t refers to the current time-step. A neural network is a 5-tuple $M = (N, C, I, O, w)$, where N is a finite non-empty set of neurons, $C \subseteq N \times N$ is a set of oriented interconnections among neurons, $I \subseteq N$ is a set of input neurons, $O \subseteq N$ is a set of output neurons and $w : C \to \mathbb{R}$ is a weight function. The pair (N, C) forms an oriented graph that is called the topology of M.

A Hopfield network $H = (N, C, I, O, w)$ is a neural network, for which all its neurons are input and output neurons simultaneously ($N = I = O$) and oriented interconnections are defined among all the neurons ($C = N \times N$). All its weights are symmetric ($w_{ij} = w_{ji} \ \ \forall i, j \in N$) and each neuron is connected to all other neurons except itself ($w_{ii} = 0 \ \ \forall i \in N$). For the Hopfield

network H, an output $\vec{y}$ is the vector of the outputs of all the neurons of H. A weight matrix W of H having n ($n > 0$) neurons is an $n \times n$ matrix $W = (w_{ij})$ where w_{ij} denotes the weight between the neuron i and the neuron j. A training set T of H is a finite non-empty set of m training patterns $\vec{x}^1, \ldots, \vec{x}^m$: $T = \{\vec{x}^1, \ldots, \vec{x}^m | \ \vec{x}^k \in \mathbb{R}^n \ k = 1, \ldots, m\}$.

For training Hopfield networks, the Hebbian rule can be applied. According to this rule, the presented training pattern $\vec{x}^k = (x_1^k, \ldots, x_n^k)$ can be stored in the Hopfield network with the weights w_{ij} $(i, j = 1, \ldots, n)$ by adjusting the respective weight values w_{ij} as it follows:

$$w_{ij} \leftarrow w_{ij} + x_i^k x_j^k \quad i, j = 1, \ldots, n \quad i \neq j \tag{2}$$

Initial weight values w_{ij} $(i, j = 1, \ldots, n)$ are assumed to be zero. During the iterative recall, individual neurons preserve their output until they are selected for a new update. It can be shown [4] that the Hopfield model with an asynchronous dynamics - each neuron is selected to update (according to the sign of its potential value ξ to $+1$ or -1) randomly and independently - converges to a local minimum of the energy function.

Hopfield networks represent a model of associative memories applicable to image processing and pattern recognition. They can recall reliably even "damaged" patterns but their storage capacity is relatively small (approximately $0.15n$ where n is the dimension of the stored patterns [4]). Moreover, the stored patterns should be orthogonal or close to orthogonal one to each other. Storing correlated patterns can cause serious problems and previously stored training patterns can even become lost. This is because, when recalling a stored pattern, the cross-talk does not average to zero [1]).

2.1 The Fukushima Model

In the Fukushima model [2], the chain process of recalling the scenery far ahead of a given place is simulated by means of the correlation matrix memory similar to the Hopfield model. In the Fukushima model, a "geographic map" is divided into m fragmentary patterns overlapping each other. These fragmentary patterns are memorized in the correlation matrix memory. The actual scenery is represented in the form of a spatial pattern with an egocentric coordinate system. When we should "move", the actual fragmentary pattern should "become shifted" in order to keep our body always in the center of the "scenery" pattern to be recalled. If the "scenery image" shifts following the movement of the body, a vacant region appears in the "still not seen scenery" pattern. During recall, a pattern with a vacant "not seen" region is presented to the correlation matrix and the recalled pattern should fill its missing part (see Figure 1).

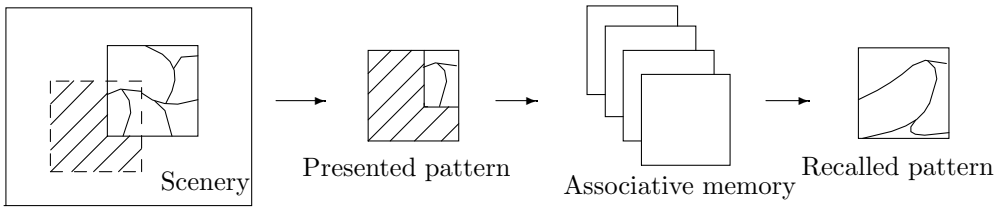


Figure 1: The recall process for incomplete patterns

Unfortunately, it is necessary to "place" the pattern presented to the correlation matrix exactly at the same location as one of the memorized patterns. The pattern to be recalled is

shifted in such a way that the non-vacant region coincides with one of the memorized patterns. In order to speed-up the evaluation of the region-matching criteria, the Fukushima model incorporates the concept of the piled pattern. The point yielding the maximum correlation between the "seen scenery" and the corresponding part of the piled pattern should become the center of the next region.

Then, the vacant part of the shifted pattern is filled, i.e. recalled by the auto-associative matrix memory. Although the recall process sometimes fails, it usually does not harm too much because the model contains the so-called monitoring circuit that detects the failure. If a failure is detected, the recalled pattern is simply discarded and recall is repeated after some time when the "body" was moved to another location.

3 The Cascade Associative Memory - CASM

Traditional associative memories cannot store reliably correlated patterns, since the cross-talk generated by other stored patterns during recall does not average to zero [1]. To overcome this problem, Hirahara et al. proposed the Cascade ASsociative Memory model (CASM) which enables to store hierarchically correlated patterns [3]. In the case of a two-level hierarchy, the CASM-model consists of two associative memories. The first associative memory (AAM) is an auto-associative memory storing the first-level patterns called ancestors. The second associative memory (DAM) is a recurrent hetero-associative memory associating the second-level patterns called descendants with their corresponding ancestors.

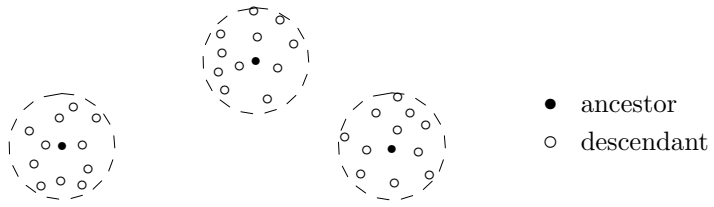


Figure 2: Hierarchically correlated patterns – a two-level hierarchy

In CASM, the ancestors are randomly generated. For each ancestor, a set of descendants – correlated with their ancestor – is generated. Hence, the descendants belonging to the same ancestor represent a cluster and the ancestor represents the center of the cluster (Figure 2). During training, the ancestors are stored in the AAM. Then, the training algorithm divides the descendants into groups according to their ancestors. The descendant associative memory (DAM) stores the respective groups of descendant patterns separately. In principle, the weight matrix of DAM has a form of a pile of covariance matrices, each of which is responsible for recalling only the descendants of each ancestor – the model represents a multiplex associative memory.

During recall, the pattern is first presented to the ancestor associative memory (AAM) to recall its ancestor. Then, the descendant associative memory (DAM) combines the recalled ancestor with its corresponding covariance matrix "responsible to recall its descendants". This mechanism enhances the recall abilities of the model by suppressing the cross-talk noise generated by descendants of other ancestors [3].

4 The Hierarchical Associative Memory

Standard associative memories are able to recall reliably "damaged" or "incomplete" images if the number of stored patterns is relatively small and the patterns are almost orthogonal to each other. But real patterns (and spatial maps in particular) rather tend to be correlated. This greatly reduces the possibility to apply standard associative memories in practice. To avoid (at least to a certain extent) these limitations, we designed the model of the so-called Hierarchical Associative Memory. This model is based on the ideas of a Cascade associative memory (CASM) of Hirahara et al. [3] which allows to deal with a special type of correlated patterns. But our goal is to use the basic CASM-model more efficiently by allowing an arbitrary number of layers with more networks grouped in each layer.

A Hierarchical associative memory $\mathcal{H}$ with L layers ($L > 0$) is an ordered $L-$tuple $\mathcal{H} = (M_1, \ldots, M_L)$ where $M_1, \ldots, M_L$ are finite non-empty sets of Hopfield networks; each of the networks having the same number of neurons n ($n > 0$). The sets $M_1, \ldots, M_L$ are called layers of the memory $\mathcal{H}$. For l ($1 \leq l < L$) the layer M_l will be further called the ancestor layer of the layer M_{l+1} and the layer M_{l+1} will be called the descendant layer of the layer M_l. $\mid M_l \mid$ denotes the number of Hopfield networks in the layer M_l ($l = 1, \ldots, L$). A training tuple $\mathcal{T}$ of $\mathcal{H}$ is an ordered $L-$tuple $\mathcal{T} = (T_1, \ldots, T_L)$ where T_l ($l = 1, \ldots, L$) is a finite non-empty set of training patterns ${}^l\vec{x}^1, \ldots, {}^l\vec{x}^{m_l}$, $m_l \in \mathbb{N}$ for the layer M_l

$$T_l = \{{}^l\vec{x}^1, \ldots, {}^l\vec{x}^{m_l} \mid {}^l\vec{x}^p \in \{+1, -1\}^n,\ p = 1, \ldots, m_l\}, \tag{3}$$

m_l denotes the number of training patterns for the layer M_l and $\mathbb{N}$ denotes the set of all natural numbers.

The training patterns (from the training tuple $\mathcal{T}$) are to be stored in the model separately for different layers. In this way, the training patterns ${}^l\vec{x}^1, \ldots, {}^l\vec{x}^{m_l}$ from the set T_l will be stored in the Hopfield networks $H_1, \ldots, H_{|M_l|}$ of the corresponding layer M_l of $\mathcal{H}$. Training of the whole network $\mathcal{H}$ consists in training each layer M_l ($l = 1, \ldots, L$) separately (we will call it layer training) and can be done for all layers in parallel.

During training the layer M_l, training patterns ${}^l\vec{x}^1, \ldots, {}^l\vec{x}^{m_l}$ from the set T_l are presented to the layer M_l sequentially. For each presented training pattern, "the most suitable" Hopfield network in the layer M_l is found in which the processed training pattern is stored. If there is no "suitable" Hopfield network for storing presented pattern, the new Hopfield network is created and added to the layer M_l. Then, the pattern is stored in the newly created Hopfield network. Now, we describe the so-called DPAT-algorithm (dynamical parallel layer training algorithm) for training each layer in the model.

The DPAT-algorithm (for the layer M_l):

Step 1: The weight matrices of all Hopfield networks in the layer M_l are set to zero.

Step 2: A training pattern $\vec{x}$ is selected from T_l.

Step 3: The pattern $\vec{x}$ is stored in all Hopfield networks in the layer M_l (according to the Hebbian training rule).

Step 4: The pattern $\vec{x}$ is recalled by all Hopfield networks from M_l. We get recalled outputs $\vec{y}^1, \ldots, \vec{y}^{|M_l|}$, where $\vec{y}^i$ is the recalled output of the $i-$th Hopfield network in the layer M_l.

Step 5: The Hamming distance d_i of the training pattern $\vec{x}$ and the recalled output $\vec{y}^i$ is computed

$$d_i = \text{HammingDistance}(\vec{x}, \vec{y}^i)$$

for $i = 1, \ldots, \mid M_l \mid$. The Hamming distance of $\vec{z}$ and $\vec{z}'$ is function given by the formula:

$$\text{HammingDistance}(\vec{z}, \vec{z}') = \sum_{i=1}^{n} \text{sgn}(\mid z_i - z'_i \mid).$$

Step 6: The minimum Hamming distance d_{min} is found

$$d_{min} = \min_{i=1,\ldots,|M_l|} d_i.$$

min is set to the index of the network with satisfying d_{min}. If there exist more Hopfield networks in M_l with the same minimum Hamming distance d_{min}, min will be set to the lowest index of the network satisfying d_{min}.

Step 7: The pattern $\vec{x}$ is unlearnt from all Hopfield networks i ($i = 1, \ldots, \mid M_l \mid$) in the layer M_l where $d_i \neq 0$ or $i \neq min$.

Step 8: If the pattern $\vec{x}$ is unlearnt from all Hopfield networks in M_l, a new Hopfield network is created in the layer M_l. The pattern $\vec{x}$ is stored in the newly created Hopfield network.

Step 9: If there is any other training pattern in T_l, **Step 2**.

During recall, an input pattern $\vec{x} = (x_1, \ldots, x_n)$ is presented to the memory $\mathcal{H}$. The input pattern ${}^1\vec{x}$ of the layer M_1 is identical to the presented input pattern $\vec{x}$. For the following time steps l ($1 \leq l < L$), each layer M_l produces a set of outputs (one output for each particular network in the layer). Then, the "best" output ${}^l\vec{y}$ from this set is chosen (according to the layer recall algorithm). This output ${}^l\vec{y}$ represents then the input pattern for the "next" layer M_{l+1} (i.e. ${}^{l+1}\vec{x} = {}^l\vec{y},\ 1 \leq l < L$). The output $\vec{y}$ of the HAM $\mathcal{H}$ is defined as the output ${}^L\vec{y}$ of the "last" layer M_L. The recall process of the HAM-model is illustrated in Fig. 3.

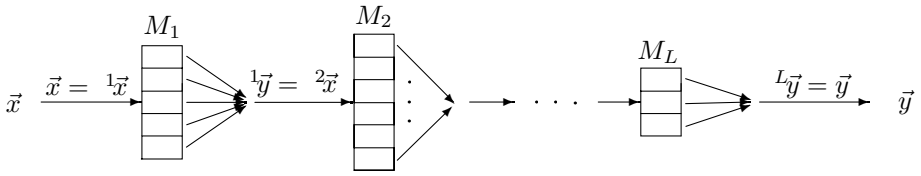


Figure 3: The recall process in the Hierarchical Associative Memory with L layers

Here, we focus on the recall process of one layer in the HAM-model in more details. During recall of the layer M_l, input pattern ${}^l\vec{x}$ is presented to the layer M_l. For the presented input pattern ${}^l\vec{x}$, each Hopfield network in the layer M_l produces the corresponding recalled output ${}^l\vec{y}^i$ ($i = 1, \ldots, \mid M_l \mid$). Then, the output ${}^l\vec{y}$ of the layer M_l is such a recalled output which is "the most similar" to the presented input pattern ${}^l\vec{x}$. Now, we describe the so-called PAR-algorithm (parallel layer recall algorithm) for recall in each layer in the model in formal way.

The PAR-algorithm (for the layer M_l):

Step 1: The input pattern ${}^l\vec{x} = ({}^lx_1, \ldots, {}^l x_n)$ is presented to all Hopfield networks in the layer M_l.

Step 2: The pattern ${}^l\vec{x}$ is recalled by all Hopfield networks in the layer M_l. Hence, we get outputs ${}^l\vec{y}^1, \ldots, {}^l\vec{y}^{|M_l|}$, where ${}^l\vec{y}^i$ is the recalled output of the $i-$th Hopfield network in the layer M_l ($i = 1, \ldots, \mid M_l \mid$).

Step 3: The Hamming distance ${}^l d_i$ of the presented input pattern ${}^l\vec{x}$ and the recalled output ${}^l\vec{y}^i$ is computed

$$ {}^l d_i = \text{HammingDistance}({}^l\vec{x}, {}^l\vec{y}^i) $$

for $i = 1, \ldots, \mid M_l \mid$.

Step 4: The minimum Hamming distance ${}^l d_{min}$ is found

$$ {}^l d_{min} = \min_{i=1,\ldots,|M_l|} {}^l d_i. $$

min is set to the index of the network satisfying ${}^l d_{min}$. If there exist more Hopfield networks with the same minimum Hamming distance ${}^l d_{min}$, min will be set to the lowest index of the network from M_l satisfying ${}^l d_{min}$.

Step 5: The output ${}^l\vec{y}$ of the layer M_l is the output ${}^l\vec{y}^{min}$ (i.e. ${}^l\vec{y} = {}^l\vec{y}^{min}$).

Anyway, we should keep in mind that the above-sketched heuristic for storing presented patterns in the dynamically trained HAM is quick, simple and easy to implement but it is not optimal. Considering the DPAT-algorithm, a pattern remains stored in such a Hopfield network where the pattern is correctly recalled (Step 7 of the DPAT-algorithm). If there is no such Hopfield network, a new Hopfield network is created for storing the pattern. Anyway, using this method for choosing the "most suitable" Hopfield network, we cannot predict anything of recalling previously stored patterns. Some previously stored patterns can be recalled incorrectly (after storing some other patterns) or can even become lost.

5 Simulations

The following experiments are restricted to a two-level hierarchy of Hierarchical Associative Memories (i.e. $\mathcal{H} = (M_1, M_2)$). Therefore, we will call the first- and second-level patterns to be ancestors and descendants, respectively. For this kind of tasks (processing of spatial patterns), we can assume that the descendants will be with a high probability correlated with their ancestors. Therefore, we did not store the descendants themselves in the second-level memories but the so-called difference patterns. The difference patterns contain only the information of the differences between the respective descendant and its ancestor and have the form of bipolar vectors as well. In this way, the difference patterns become sparser with increasing correlation between the descendants and their ancestors, which is in general expected to allow a higher storage capacity of the constructed Hierarchical Associative Memory [3].

The test patterns to be stored in the Hierarchical Associative Memory have been chosen as fragmentary patterns of a fictive map, which consists of 81×54 pixels with bipolar values corresponding to 1 and -1. The fictive scenery was divided into 21 fragmentary patterns mutually overlapping one another as shown in Fig. 4. From these fragmentary patterns, 8 patterns with the smallest cumulative correlation between the respective pattern and all other patterns were chosen to be the ancestors. The remaining 13 fragmentary patterns were used to form the descendant patterns.

During training the HAM, the ancestors were stored in the layer M_1 according to the DPAT-algorithm. Storing the remaining fragmentary patterns as descendants in the second layer M_2 proceeds as it follows. After presenting the respective fragmentary pattern to the HAM, its corresponding ancestor should be recalled in the layer M_1. Then, the difference pattern is

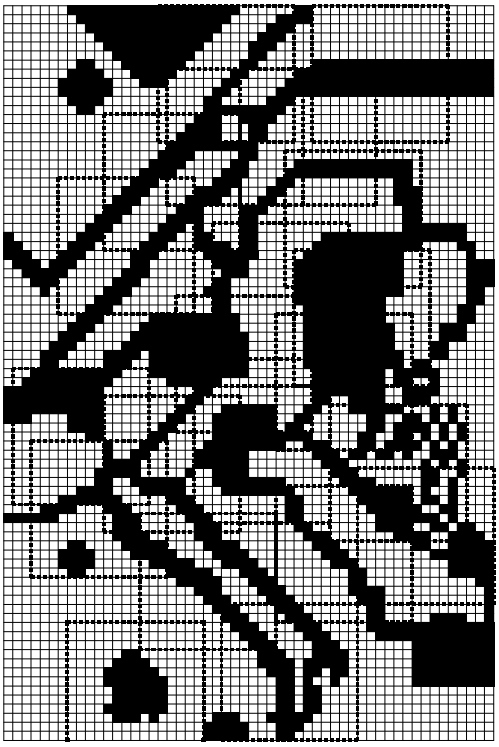


Figure 4: The scenery with marked areas stored in the Fukushima- and in the HAM-model

created according to the recalled ancestor and the presented fragmentary pattern. Afterwards, the difference pattern is stored as a descendant in the second layer M_2 according to the DPAT-algorithm as well.

A similar idea was adopted also for the recall process. During recall, a "noisy" or "unknown" input pattern is presented to the top layer of the HAM-model. First, the layer M_1 recalls the corresponding ancestor according to the PAR-algorithm. From the recalled ancestor and the presented input pattern, the difference pattern is created. This difference pattern is then recalled by the layer M_2, again according to the PAR-algorithm. From the recalled difference pattern and ancestor, the final descendant pattern is created. This descendant represents the output of the whole HAM-model. The recall process of the presented "incomplete" descendant is shown in Fig. 5.

An application of the HAM-model for solving the path prediction problem requires a robust recall of presented patterns, often unknown in some parts. Mutual overlapping of the stored patterns covers in our case approximately 1/3 of their surface (the respective cases may vary in size) and thus a presented pattern has to be recalled reliably even from a portion of it. Furthermore, the fragmentary patterns cannot be assumed mutually orthogonal in a general case and we can expect their correlations to be rather high. Due to these requirements, we have applied in our simulations the following two restrictions to the HAM-model.

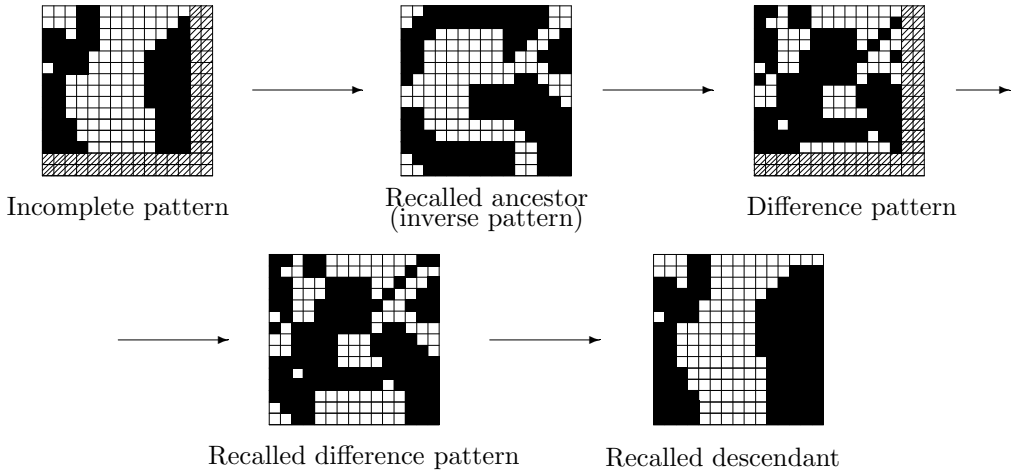


Figure 5: The recall process of the HAM-model

- The first restriction consists in limiting the number of patterns which can be stored in each Hopfield network of the HAM-model to $0.05n$, where n is the dimension of patterns to be stored in it. This corresponds approximately to 1/3 of the capacity for standard Hopfield networks.
- The essence of the second modification consists in slightly changing Step 7 of the DPAT-algorithm. Now, a pattern remains stored in that Hopfield network where it was recalled correctly only from a portion of it. If there is no such network, a new one is created to store this pattern.

Anyway, the above two modifications lead in general to an increased number of Hopfield networks created. On the other hand, they might improve significantly the overall performance of the HAM-model in path prediction. The results yielded in path prediction by the Fukushima-model and those obtained by the HAM-model are shown in Fig. 6. Fig. 6(a) shows an incorrect

Figure 6: Patterns recalled by the Fukushima model (a) and by the HAM-model (b)

lower segment of the scenery recalled by the Fukushima-model. On the other hand, the HAM-

model recalled the lower segment without any major error - see Fig. 6(b).

6 Conclusions

Our current research in the area of associative memories is focused on applications of associative memories for storing spatial patterns (Hopfield-like networks) and for path prediction in spatial maps (the Fukushima model). Unfortunately, the performance of standard associative memories (of the Hopfield type) depends on the number of training patterns stored in the memory, and is very sensitive to mutual correlations of the stored patterns. In order to overcome these limitations, we have proposed in this paper the HAM-model – the Hierarchical Associative Memory. This model was inspired by the Cascade ASsociative Memories (CASM) and a stressed necessity to treat reliably huge amounts of data.

In comparison with the Fukushima model, the experiments performed so far with the HAM-model yield promising results for the path prediction problem. Especially dynamical adding of Hopfield-like networks to existing layers (cf. the DPAT-algorithm) enables the HAM-model to store even larger amounts of mutually correlated data reliably. The experiments carried out have further confirmed that the right choice of the ancestor patterns represents a crucial point of a successful application of the HAM-model. Appropriately chosen ancestor patterns should represent "well enough" the corresponding clusters of patterns. Simultaneously, they should be mutually orthogonal (or at least nearly orthogonal). Therefore, we would like to propose means for improving the storage capacity of the HAM-model within the framework of our further research.

These will probably require the development of more sophisticated methods – based on self-organization – for choosing "the most suitable" ancestor patterns. Further improvements can be expected when recalling the patterns in the "right" ancestor network. In our opinion, strategies for achieving this could comprise "keys" (corresponding in principle to further pattern elements artificially added to the feature vectors). The keys introduced in such a way could then better control the recall process. The time- and space-complexity of the above-sketched models should be analysed thoroughly.

References

[1] D. J. Amit, H. Gutfreund, H. Sompolinsky: *Information storage in neural networks with low levels of activity*, in: Physical Review A, 35, 2293-2303,1987.

[2] K. Fukushima, Y. Yamaguchi, M. Okada: *Neural Network Model of Spatial Memory: Associative Recall of Maps*, in: Neural Network, Vol. 10, No. 6, pp. 971-979, 1997

[3] M. Hirahara, N. Oka, T. Kindo: *A cascade associative memory model with a hierarchical memory structure*, in: Neural Networks, Vol. 13, No. 1, pp. 41-50, 2000

[4] J. J. Hopfield: *Neural Networks and physical system with emergent collective computational abilities*, Proc. Natl. Acad. Sci. USA, 79: 2554-2558, 1982

[5] I. Mrázová, J. Tesková: *Path Prediction in Geographic Maps*, Proceedings of Nostradamus, pp. 190-195, 2000.

Properties Of Space Filling Curves And Usage With UB-trees

Tomáš Skopal, Michal Krátký, Václav Snášel

Department of Computer Science, Technical University Ostrava, Czech Republic
tomas.skopal@vsb.cz, michal.kratky@vsb.cz, vaclav.snasel@vsb.cz

Abstract. In this paper we want to investigate certain properties of space filling curves and their usage with UB-trees. We will examine several curve classes according to range queries in UB-trees. Our motivation is to propose a new curve for the UB-tree which will improve the range queries efficiency. In particular, the address of this new curve is constructed using so-called proportional bit interleaving.

Keywords: space filling curve, UB-trees, space partitioning, Z-curve

1 Introduction

In Information Retrieval there is often requirement on data clustering or ordering and their consequential indexing. This requirement arises when we need to search for some objects in large amounts of data. The process of data clustering→indexing gathers similar data together, thereby allows effective searching in large data volumes.
Specific approaches of data clustering were established on the vector space basis where each data object is represented as a vector of its significant attributes. Such object vectors can be stored as points/vectors within a multidimensional vector space. In this article we are going to examine indexing method for discrete multidimensional vector spaces based on *space filling curves*. A space filling curve allows linearize a multidimensional space in such way that to each point in space is assigned single unique value – i.e. *address* on curve. Thus, the space could be considered as partitioned and even ordered due to the curve ordering.
There exists one indexing structure which combines the principles of space filling curves and techniques of data indexing. This structure is called UB-tree.

1.1 Vector Spaces

Definition 1. *Let* $\Omega = D_1 \times D_2 \times \ldots \times D_n$ *is a* n-dimensional vector space. *The set* D_i *is called the* domain *of dimension i. The* vector *(point, tuple)* $x \in \Omega$ *is a n-tuple* $\langle a_1, a_2 \ldots, a_n \rangle \in \Omega$. *In other words x can be interpreted as a point at coordinates* $(a_1, a_2 \ldots, a_n)$ *within n-dimensional space. The coordinate* a_i *is also called* attribute value *and can be represented as a binary number (string) of length* l_i. *Thus, to each domain* D_i *is assigned a bit-size* l_i *and its* cardinality $c_i = 2^{l_i}$. *Then holds* $0 \leq a_i \leq c_i - 1$.

In discrete vector space Ω we use integer coordinates. Usually, the space domains are also finite due to limited capacity of integer attribute.
In figure 1 we see two-dimensional space "animal" where the first dimension represents "animal class " and the second "limb count". Animal "ant" is represented with vector [0,2] ([insect, 6] respectively).

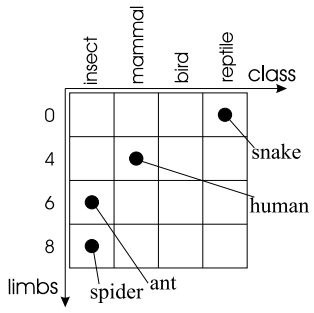


Fig. 1. Two-dimensional vector space 4×4

Representing objects as vectors in space brings following possibilities:
- Formal apparatus of vector spaces allows objects to be treated uniformly. Object is vector and there are defined certain operations on vectors in vector space.
- In metric vector spaces we can measure *similarity* between two objects (distance of vectors respectively)
- In vector space we can easily construct *range queries*

1.2 Space Partitioning

Because of the requirements of data indexing we need to transform the objects within vector space Ω into some hierarchical index structure. This space transformation means some kind of space partition which in turn produces space subregions.

Definition 2. *(space filling function)*
We call a function $f : S \subset \mathcal{N}_0 \rightarrow \Omega$ a space filling function, if $f(S) = \Omega$ is a bijective function. Ordinal number a, where $f(a) = o_i, o_i \in \Omega$ is called address of vector o_i on curve $f(S)$.

Lemma 1. *A space filling function defines one-dimensional ordering for a multidimensional space.*
Space filling curves order points in space and may so define *regions* in the space. A region is spatially determined with an interval $[\alpha : \beta], \alpha \leq \beta$ on the curve (α, β are addresses). In figure 2 we can see our example space (extended to 8×8) filled with three types of curves. Suppose we want to find the smallest region covering objects "snake" and "ant", we would obtain [3:16] for C-curve, [5:8] for Z-curve and [4:15] for H-curve.

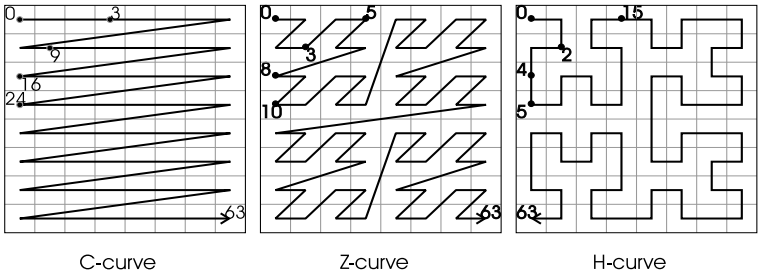


Fig. 2. Compound (C) curve, Lebesgue (Z) curve, Hilbert (H) curve and addresses of points within our "animal" space

1.3 Universal B-trees

The idea of UB-tree incorporates B^+-tree and the space filling curves. Inner nodes of B^+-tree contain hierarchy of curve regions. Region of inner node always spatially covers all regions of its children. Regions on the same depth level in tree do not overlap. In leaves are stored data objects (their vectors respectively). The structure of UB-tree (with Z-curve) for our example is shown in figure 3. We can see that the objects on the leaf level are ordered left-to-right – the same way as on Z-curve. Further informations about UB-trees can be found in [Ba97,Ma99].

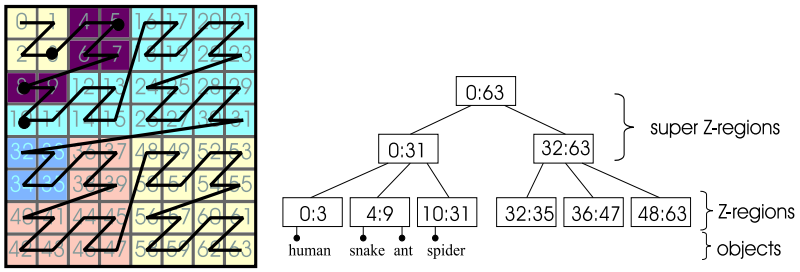


Fig. 3. Structure of UB-tree

So far, the published papers about UB-tree mentioned only the possibility of Z-curve. In following analysis we are going to examine also another curve orderings.

2 Properties of Space Filling Curves

In [Ma99] we can find a curve quality criterion based on curve symmetry measure. This symmetry is somehow connected with *selfsimilarity* concept taken from fractal geometry.
We will introduce another aspects of curve quality classification. Our goal is to find ideal curve for range queries in vector space or actually in UB-tree.

2.1 Measure of Utilization

Range query processing in vector space or in UB-tree must examine each region which intersects query area. We can assume that large region overlaps will produce unnecessary space searching and thus they cause high access efficiency costs.
To reduce these costs we've developed *measure of utilization* which allows to rate each curve and choose the best one. In figure 4 we'll see greyed query area and the smallest possible region that entirely covers given query area.

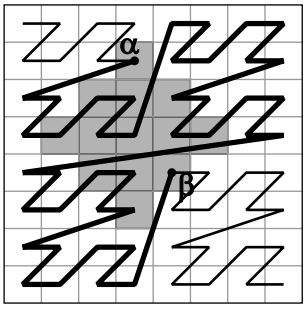


Fig. 4. Smallest possible region $[\alpha : \beta]$ covering entire query area

Definition 3. *Let $A(o_i)$ is a query area centered on coordinates of $o_i \in \Omega$. Let $R(A(o_i))$ is the smallest possible region covering $A(o_i)$. Then* utilization *of $A(o_i)$ is defined as*

$$u(A(o_i)) = \frac{volume(A(o_i))}{\beta_{R(A(o_i))} - \alpha_{R(A(o_i))}}$$

and average utilization *of space Ω with query area A and space filling curve f is defined as*

$$avgutil_{(\Omega,A,f)} = \frac{\sum_{o_i=f(0)}^{f(volume(\Omega))} u(A(o_i))}{volume(\Omega)}$$

Notes Average utilization factor helps us to find better curve from the range query point of view. Curve with *avgutil* close to 1 means that searching the vector space by range query processing is effective – no unnecessary space is searched. Interesting consequence of high rated curves is that points that are near in space (according to some metric) are also near on the curve (according to order).

Choosing query area The shape of range query area must comply with the nature of range queries. In metric vector spaces we search for similar objects given by some threshold distance value. In non-metric vector spaces we search for objects within coordinate ranges. The former aspect represents n-dimensional sphere and the latter n-dimensional

block. As we can see in figure 4 we've choosen n-dimensional sphere for further analysis.

Volume of sphere in multidimensional discrete space is defined as follows: Let $K(r)$ denote number of points with integer coordinates contained within circle, i.e. circle volume in discrete space.

Theorem 1. *(Gauss) in [Cha69]*
For $K(r)$ the following asymptotic formula holds:

$$K(r) = \pi r^2 + \Delta(r)$$

$$\Delta(r) = O(r)$$

The circle consists of all the points satisfying $x_1^2 + x_2^2 + \cdots + x_n^2 \leq r^2$. The volume of a sphere in R^n is a function of the radius r and will be denoted as $V_n(r)$. We know that $V_1(r) = 2r$, $V_2(r) = \pi r^2$ and $V_3(r) = \frac{4}{3}\pi r^3$.
To calculate $V_n(r)$ (changing to polar coordinates) we get

$$V_n(r) = \int_0^r \Big(\int_0^{2n} V_{n-2}(\sqrt{r^2 - s^2}) s d\theta \Big) ds$$

By using induction, closed forms for calculation of higher dimensional spheres can be derived

$$V_{2n}(r) = \frac{\pi^n \cdot r^{2n}}{n!}$$

$$V_{2n+1}(r) = \frac{2^{2n+1} \cdot n! \cdot \pi^n \cdot r^{2n+1}}{(2n+1)!}$$

From theorem 1 we get the volume of n-dimensional sphere

$$K_n(r) = V_n(r) + O(V_{n-1}(r))$$

2.2 Curve Dependency Classification

We have find out that dependency on certain vector spaces characteristics has important influence on the average utilization factor. We'll venture to proclaim that the higher dependency on space characteristics, the higher average utilization. We have classified this dependency into three space filling curve classes:

Dimension dependent curves First class of curves takes in account only the dimension of vector space, e.g. classical Z-curve. This type treats the space as a multidimensional cube and is not suitable for spaces with different domain cardinalities. In figure 5 we can see that the curve overlap the real space. It is obvious that the average utilization of that curve will be very low due to large smallest covering regions.

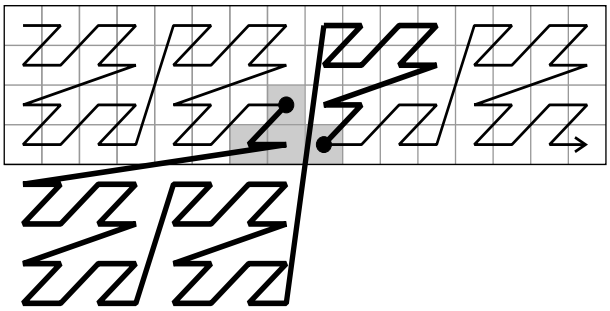


Fig. 5. Dimension dependent curve, e.g. Z-curve and one of the smallest covering regions

Address computation is also only dimension dependent. On the input stand n attributes, bit length is uniform – l. In figure 6 is shown one type of address construction, i.e. bit interleaving which is simply the permutation of bit vector $\boldsymbol{a}$ of all attributes (concatenated to single vector). To every bit in every attribute is assigned a position in the resultant address. Address contruction based on permutation can be easily realized with permutation matrix A.

$$addr = \boldsymbol{a} \cdot A$$

Permutation matrix can be created from unitary square matrix by multiple column (or row) swap. Reconstruction of original attributes from address is also simple – using the inverse matrix A^{-1} which is equal (if A is orthonormal – and permutation matrix is always orthonormal) to the transposed matrix, i.e. $A^T = A^{-1}$. Then

$$\boldsymbol{a} = A^T \cdot addr$$

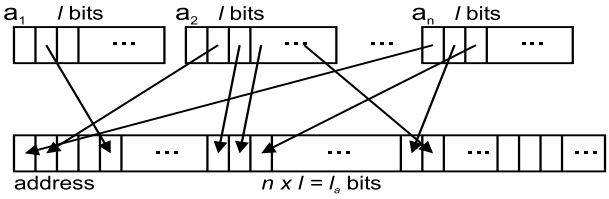


Fig. 6. Bit interleaving for dimension dependent address

Domain dependent curves The second class is not only dimension dependent but also domain dependent. Curves are constructed with considerations to domain cardinalities. Curves of that type do not overlap given vector space (e.g. C-curve). Address construction for domain dependent curves can be based on permutation matrix as well.
The curve for UB-tree we have announced in the beginning is one of this type and is called PZ-curve. PZ-curve advances the original Z-curve

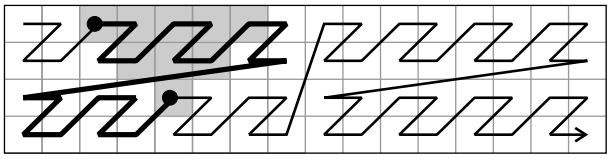


Fig. 7. Domain dependent curve, e.g. PZ-curve and one of the smallest covering regions

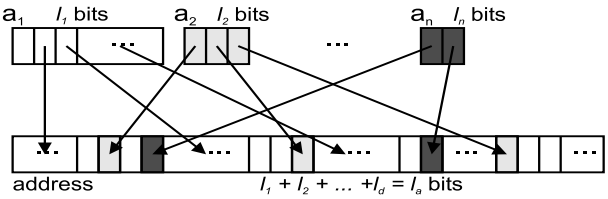


Fig. 8. Bit interleaving for domain dependent address

where the new quality we call *proportionality* of PZ-address, i.e. attribute bits are interleaved proportionally to each domain (see figures 7, 8). The PZ-address is intimately described in the next section.

Data dependent curves Third class of curves is dependent on everything known in the space even on the data stored within. However, this type is rather theoretical and we present it here only for notion and future motivation.

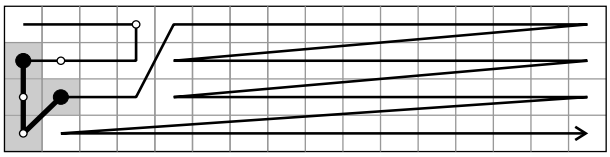


Fig. 9. Data dependent curve

Construction of address for data dependent curve is very complex. Generally, every bit of the output address is evaluated as a function of all the coordinates in input. Even if we'd accomplish address computation there is another complication. The curve is data dependent and therefore it must be completely recomputed after adding new data.

2.3 PZ-address

Construction of PZ-address is based on proportional bit interleaving. We can describe the interleaving with following simplified algorithm:

1. Vector of PZ-address is empty (all bit positions are empty). Order of attributes a_i is chosen as a parameter, $i \in I$.
2. Bits of attribute a_i are uniformly dispersed over the empty bit positions within the PZ-address vector.
3. Newly occupied bit positions are no longer empty – the vector of PZ-address is beeing filled step by step.
4. Step 2 is repeated until index set I is exhausted.

Note: The order of attribute processing is important. Attributes that are processed at first are more accurately dispersed into PZ-address.

For synoptic idea of the algorithm see figure 10.

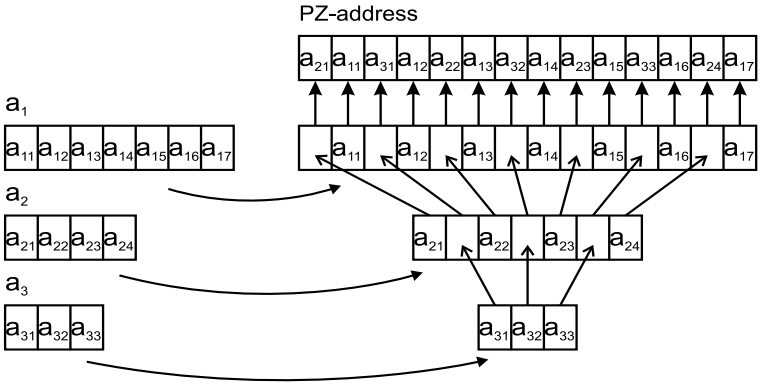


Fig. 10. PZ-address construction. First, bits of a_1 are dispersed into the empty PZ-address vector. Second, a_2 is dispersed over the rest empty positions. Last attribute (here a_3) is actually not dispersed but coppied.

Formal description Let's have n attributes (coordinates in n-dimensional space). Attribute a_i is represented as a bit vector of length l_i. PZ-address (vector $PZaddr$ with length $l_a = \sum_{i=1}^{n} l_i$) is computed using following permutation matrix:

$$PZaddr = \left(a_{11}\ a_{12}\ \ldots\ a_{21}\ a_{22}\ \ldots\ a_{ij} \right) \cdot \begin{pmatrix} \underbrace{0\,0\,\ldots\,0}_{ord(a_{11})-1}\,1\,0\,0\,0\,0\,\ldots \\ \underbrace{0\,0\,0\,0\,0\,\ldots\,0}_{ord(a_{12})-1}\,1\,0\,\ldots \\ \vdots \\ \underbrace{0\,0\,0\,\ldots\,0}_{ord(a_{21})-1}\,1\,0\,0\,0\,\ldots \\ \vdots \\ \underbrace{0\,0\,0\,0\,\ldots\,0}_{ord(a_{ij})-1}\,1\,0\,0\,\ldots \end{pmatrix}$$

where $ord(a_{ij})$ is the position of j-th bit of attribute a_i in the resultant PZ-address.

$$ord(a_{ij}) = emptypos(i, j \cdot disperse(a_i))$$

where $disperse(a_i)$ is the uniform bit dispersion of a_i over the remaining empty positions in PZ-address.

$$disperse(a_i) = integer\left(\frac{\sum_{k=i}^{n} l_k}{l_i}\right)$$

and where $emptypos(i, k)$ is the k-th empty bit position in PZ-address after the attribute a_{i-1} is processed.

$$emptypos(i, k) = min(F(i), k)$$

$F(i)$ is the set function ($F(i) \subset \mathcal{N}$) of all empty positions in PZ-address before attribute a_i is processed.

$$F(1) = \{1, 2, \ldots, l_a\} \qquad F(i+1) = F(i) - \bigcup_{j=1}^{l_i} \{ord(a_{ij})\}$$

$min(A, k)$ is the k-th minimum of an ordered set A

$$min(A, 1) = min(A) \qquad min(A, k) = min(A - \bigcup_{q=1}^{k-1} \{min(A, q)\})$$

2.4 Test results

Figure 11 shows us the influence of space dependency on particular curves and also their types. Proposed PZ-curve has relatively high average utilization rate, thus seems to be suitable for usage with UB-trees.

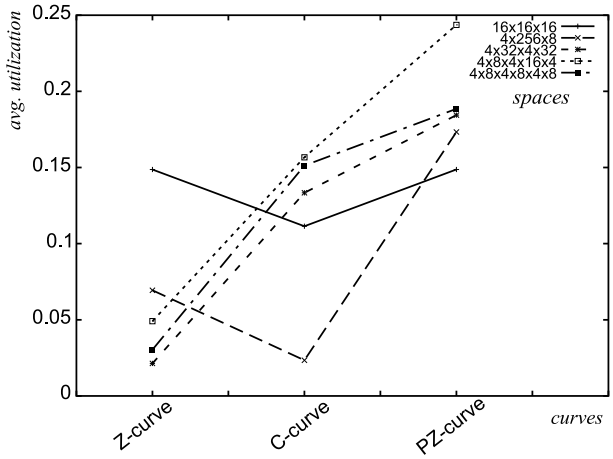


Fig. 11. Test results – with growing dependency grows also average utilization

3 Testing of the UB-tree range queries

One from the kind of the spatial queries is the *range query*. The algorithm of UB-tree range query is described in [Ba97] and [Ma99]. Range query processing finds all the tuples (objects) lying inside given n-dimensional query block.

All the regions overlapped by n-dimensional block are retrieved and searched during the processing of range query. The goal of our tests is to show that PZ-regions (created using PZ-address) part the n-dimensional space better than by using of Z-address. The goal is to show the query block overlaps less regions by usage of the PZ-address than by usage of the Z-address. If the query block overlaps less Z-regions, the less B-tree pages are retrieved and also less disk accesses are done and less CPU time is consumed.

We measure the rate of number of regions overlapped by n-dimensional block by usage of tested address (for example PZ-address) to number of regions overlapped by usage of Z-address. We will note the value as eff:

$$eff = \left(1.0 - \frac{numregsTestedA}{numregsZA}\right) \cdot 100.0 \quad [\%]$$

The eff value for m query blocks is then calculated as:

$$eff_m = \left(1.0 - \frac{\sum_{i=1}^{m} numregsTestedA_i}{\sum_{i=1}^{m} numregsZA_i}\right) \cdot 100.0 \quad [\%]$$

where

$numregsTestedA$ is the number of regions overlapped by query block by usage of tested address (for example PZ-address)

$numregsZA$ is the number of regions overlapped by query block by usage of Z-address

$numregsTestedA_i$ is $numregsTestedA$ value for query block i

$numregsZA_i$ is $numregsZA$ value for query block i

Positive eff value means that number of accessed regions by usage of tested address is less than numbers of accessed regions by usage of Z-address.

We will execute three tests which consist of three subtests (the calculation of ${eff_3}^1$, ${eff_3}^2$ and ${eff_3}^3$ values) and compare PZ-address and Z-address. The first subtest computes ${eff_3}^1$ value for three n-dimensional blocks (see Figure 12a). The second subtest computes ${eff_3}^2$ value for three n-dimensional cubes (see Figure 12b). The third subtest computes ${eff_3}^3$ value for three n-dimensional blocks (see Figure 12c). Thus, each of the test consists of three subtests, the nine tests were executed at the whole. The eff_{avg} value was the average for the nine tests.

Test 1:

We see dependency of eff at the arity of the UB-tree in Figure 13. The 16000 tuples were inserted into the 4-dimensional space 16x64x16x64. We see the usage of PZ-address gives better results by computation ${eff_3}^1$ value, the usage of PZ-address gives the worse results by computation ${eff_3}^2$ value. In spite of this – the eff_{avg} value is bigger than zero so

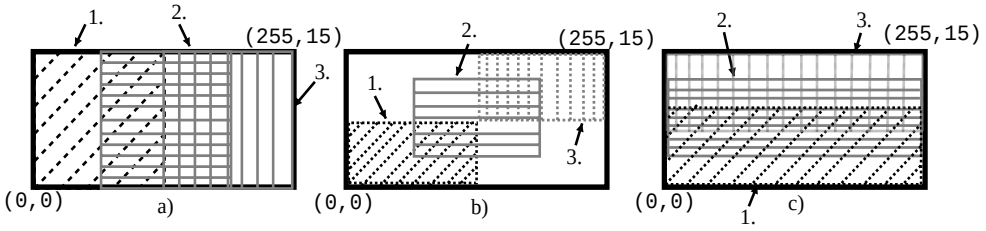


Fig. 12. The example of testing query blocks for 2-dimensional space 256x16. The testing query blocks for calculation of a) $eff_3{}^1$ b) $eff_3{}^2$ and c) $eff_3{}^3$.

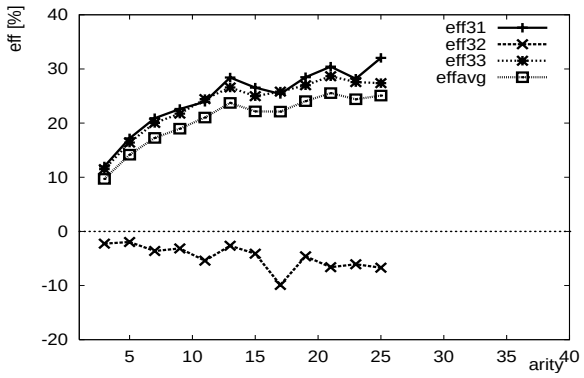


Fig. 13. The results of the test 1.

query block overlaps less number of regions by usage of PZ-address than by usage of Z-address. Thus, less B-tree pages are retrieved. We see eff_{avg} value grows with growing arity.

Test 2:
We see dependency of eff at the dimension n of space in Figure 14. The number of inserted tuples grows with the dimension n. The tuples were inserted into spaces with $n = 2$ (2D space 16x64), $n = 3$ (16x64x16), $n = 4$ (16x64x16x64) and $n = 5$ (16x64x16x64). We see the PZ-address gives a better results with growing dimension.

4 Conclusions

In this paper we have presented some properties of space filling curves according to the usage with UB-tree. The original design of UB-tree takes into account only the possibility of Z-curve. We have shown that

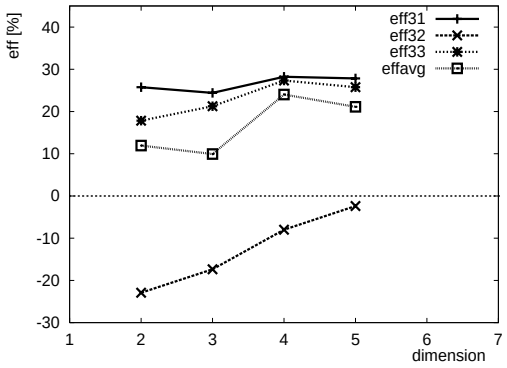


Fig. 14. The results of the test 2.

there exist several aspects giving a reason to propose another alternative curves. This reason is especially based on maximization of the range queries efficiency.
From this point of view we have designed such an alternative curve, i.e. PZ-curve, which tries to take advantage of some space knowledge. PZ-curve is domain dependent, i.e. is suitable for indexing specific vector spaces with differently ranged domains. Example of data modelled within this spaces could be the XML data. Modelling and indexing XML data we closely discuss in [KPS02a,KPS02b].

References

[Ba97] Bayer R.: *The Universal B-Tree for multidimensional indexing: General Concepts.* In: Proc. Of World-Wide Computing and its Applicazions 97 (WWCA 97). Tsukuba, Japan, 1997.

[Cha69] Chandrasekharan, K.: *Introduction to Analytic Number Theory.* Springer-Verlag New York, 1969. ISBN 0387041419.

[Ka83] Karacuba A.A.: *Introduction to Analytic Number Theory.* Nauka 1983. in russian.

[KPS02a] Krátký M., Pokorný J., Snášel V.: *Indexing XML data with UB-trees.* Accepted at ADBIS 2002, Bratislava, Slovakia

[KPS02b] Krátký M., Pokorný J., Skopal T., Snášel V.: *Geometric framework for indexing and querying XML documents.* Accepted at EurAsia ICT 2002, Tehran, Iran

[Ma99] Markl, V.: *Mistral: Processing Relational Queries using a Multidimensional Access Technique*, `http://mistral.in.tum.de/results/publications/Mar99.pdf`, 1999

Technologie eTrium: Znalosti, agenti a informační systémy

Zdenko Staníček, Filip Procházka
Fakulta informatiky MU, Botanická 68a, 602 00 Brno
e-mail: stanicek@fi.muni.cz, xprocha3@fi.muni.cz
SHINE studio, s.r.o., Minská 52, 616 00 Brno
e-mail: sta@shine.cz, pro@shine.cz, http://www.shine.cz

11. listopadu 2002

Abstrakt

V příspěvku je prezentována technologie eTrium, která umožňuje vytvářet informační systémy tak, že tyto systémy jsou schopny podporovat vyvíjející se business požadavky bez nutnosti neustálého doprogramovávání a přeprogramovávání již existujícího řešení. Základní myšlenkou je deklarativně reprezentovat znalosti, které jsou při klasickém řešení přítomny v desintegrované formě v programovém kódu, struktuře databáze a interních směrnicích firmy. Centrální roli v systému hraje znalostní agent, který řídí aktualizační operace nad databází – z požadovaných aktualizací databáze odvozuje další aktualizace, které je nutné v souladu s aktuální bází znalostí provést. Technologie navíc umožňuje automaticky dokazovat nad platnou bází znalostí, že z daného stavu dat v databázi nelze dospět do specifikovaných stavů např. chybových nebo jinak významných. Technologie byla aplikována na reálném komerčním projektu – implementaci redakčního systému pro poradenskou firmu. Způsob, jakým se redakční systém chová je definován v bázi znalostí. Modifikacemi báze znalostí je možno chování systému okamžitě změnit.

1 Dělat věci správně versus dělat správné věci

Každý informační systém (IS) má v sobě zakódovány dva druhy znalostí: jednak znalosti toho, jak provozovaný business probíhá - tedy business znalosti,

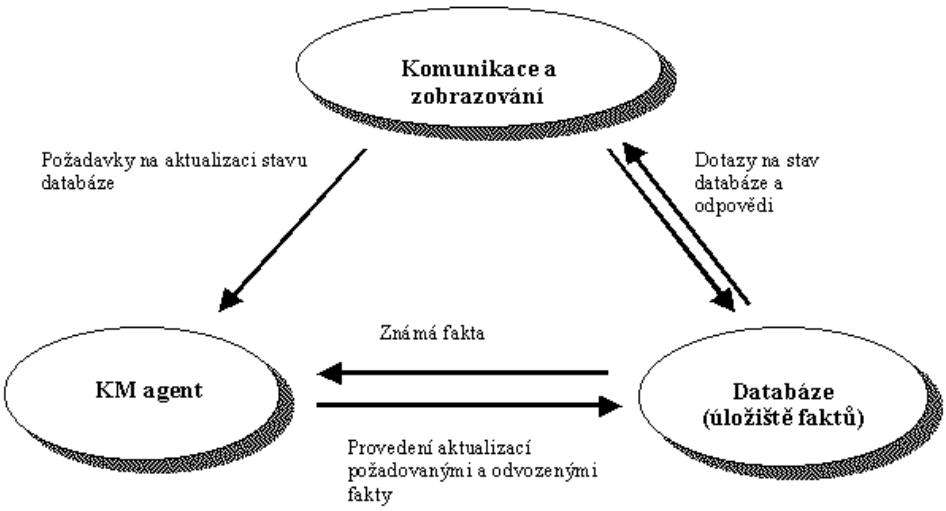
a jednak znalosti toho, jak využít aktuální potenciál informačních technologií pro podporu businessu - tedy IT znalosti. Klasický přístup k tvorbě a udržování informačních systémů je založen na tom, že se tyto znalosti převádějí do programového kódu. Dnešní metodiky a technologie vývoje SW se zajímají o to, jak toto převádění znalostí a jejich údržbu dělat správně. A jak způsobit, aby IT lidi správně porozuměli business požadavkům a implementovali v IS to, co uživatelé skutečně potřebují. Otázka však zní: **Je vůbec správné tyto věci dělat?** Co kdyby se potřebné znalosti v IS uložily centralizovaně a lidsky (nejenom programátorsky) srozumitelně v deklarativní formě tak, aby mohly snadno být předmětem analýzy? A co kdybychom po té, až se domluvíme jak má systém fungovat, pouze aktualizovali "znalosti systému" a tento systém by rovnou začal fungovat novým požadovaným způsobem?

To lze udělat! Znalosti, dnes roztroušené v programových kódech systému, je možné centralizovat do báze znalostí. Bázi znalostí pak obsluhuje program - znalostní agent, který znalosti v ní interpretuje a spolupracuje na jedné straně s firemním intranetem (pro komunikaci informací) a na druhé straně s databázovým strojem (pro rozumné ukládání informací).

2 Architektura informačního systému v technologii eTrium

Architektura libovolného IS vytvořeného v technologii eTrium je vyjádřena na obr. 1. Význam tří jejích klíčových komponent: Komunikace a zobrazování informací – Databáze – Znalostní agent je následující:

- **Komunikace a zobrazování informací** znamená webovské řešení, neboli firemní intranet; zde je naprogramována logika zobrazování a komunikace informací ve směrech systém – člověk a systém – systém.
- **Databáze** je prostým úložištěm faktů připravených tak, aby je bylo snadné podle libovolných požadavků prezentovat.
- **KM agent** (**knowledge management agent** – **znalostní agent**) zajišuje s pomocí centralizované báze znalostí celou business logiku a její podporu informačním systémem. Znalostní agent umožňuje spolupráci i s různě strukturovanými databázemi, poněvadž součástí jeho báze znalostí je i databázové schéma příslušného úložiště dat.



Obrázek 1: Schéma informačního systému v architektuře eTrium

Nyní popíšeme způsob spolupráce těchto komponent. Případ A popisuje běžné dotazování na informace při podpoře operativy v podniku/organizaci. Případ B popisuje použití téhož systému v případě, kdy je potřeba aktualizovat stav databáze. Případ C popisuje ladění businessu a jeho IT podpory Případ D popisuje rozsáhlejší změny v business logice a způsobu její IT podpory.

A. Uživatel resp. jiný systém požaduje odpově na nějaký dotaz Dvojice hran *Dotazy na stav databáze a odpovědi* představuje vnitřní komunikaci komponent při zodpovídání dotazů. Komponenta *Komunikace a zobrazování informací* zajišuje, aby okolí systému mohlo zformulovat dotaz a pošle jej komponentě *Databáze*. Komponenta *Databáze* vyhodnotí dotaz a odpově na něj (v podobě seznamu faktů) vrátí komponentě *Komunikace a zobrazování informací*. Ta zajistí zobrazení informací v požadované formě pro uživatele nebo pro jiný systém. Korektnost dotazovací akce (nepřerušení nějakou aktualizací apod.) zajistí komponenta *Databáze* svým vnitřním mechanismem – běžná funkce každého rozumně použitelného databázového stroje. Při čtení faktů z úložiště dat tedy KM agent do hry vůbec nevstupuje.

B. Uživatel resp. jiný systém chce aktualizovat obsah databáze Když uživatel resp. jiný systém vysloví *Požadavek na aktualizaci databáze* (viz příslušná hrana na obr. 1.), pak:

1. Komponenta *Komunikace a zobrazování informací* umožní tento požadavek formulovat (poskytne potřebný formulář) a aktualizační požadavek předá KM agentovi.

2. KM agent zahájí aktualizační transakci – uzavře komponentě *Komunikace a zobrazování informací* přístup do komponenty *Databáze*.

3. KM agent si k příslušnému aktualizačnímu požadavku vyžádá z databáze známá fakta (relevantní kontext), týkající se aktualizovaného záznamu.

4. KM agent, na základě báze znalostí a známých fakt o aktualizovaném záznamu a jeho souvislostech, odvodí další aktualizační akce, které je potřeba s tímto záznamem a jeho souvislostmi provést.

5. KM agent provede jak požadované, tak odvozené aktualizační akce nad databází v komponentě *Databáze*.

6. KM agent ukončí aktualizační transakci – otevře komponentě *Komunikace a zobrazování informací* přístup do komponenty *Databáze*.

C. Uživatelé zjistí, že je potřeba upravit business logiku či způsob její IT podpory Existuje okruh faktů a pravidel KM agenta, které může uživatel měnit pomocí formulářů nabízených komponentou *Komunikace a zobrazování informací*. Tento okruh faktů a pravidel tvoří "laditelnou" část modelu. Může jít třeba o změnu pravidel, zda například dokumenty kategorie *harmonogram* podléhají schvalovacímu procesu či nikoliv. Změna KM agenta probíhá tak, že se nejprve aktualizuje báze znalostí a pak se provede zkonzistentnění faktů uložených v databázi vůči nové bázi znalostí. To je realizováno následovně: z faktů v databázi se "zapomenou" všechna, která byla odvozena předchozí verzí KM agenta a znovu se nechají odvodit novou verzí KM agenta. Celý postup probíhá analogicky případu B a děje se při běžném provozu IS.

D. Uživatelé zjistí, že je třeba revidovat celou business logiku či způsob její IT podpory Po odsouhlasení změn a nové (pozměněné) znalostní báze, proběhne změna v systému těmito kroky:

1. Je zaarchivován KM agent ve stavu před změnou. Je provedena archivace datové základny *Databáze*.

2. Je vytvořena nová verze KM agenta.

3. Je provedeno otestování nových pravidel businessu ve všech důsledcích pomocí podpůrných programů. K tomu slouží ladící pravidla, testy úplnosti atd. (viz kapitola 5).

4. Je provedena oponentura výsledků testování nové business logiky s klientem.

5. KM agent provede zkonzistentnění dat v *Databáze*

3 Reprezentace znalostí KM agenta

V této části popíšeme, jakým způsobem jsou znalosti reprezentovány v KM agentovi.

Klíčovým konstruktem, který při reprezentaci znalostí využíváme je **kategorie**. Nejprve si potřebujeme objekty (ve smyslu jednotliviny), se kterými IS pracuje, nějak primárně rozdělit – otypovat. To znamená vytvořit si soubor kategorií s tou vlastností, že každý objekt patří do právě jedné kategorie z tohoto souboru (soubor kategorií namodelujeme jako kategorii jejíž prvky jsou kategorie). Například KM agent v konkrétní aplikaci technologie eTrium, která se nazývá eDialog [3], má deklarováno toto typování *dokument*, *publikační projekt*, *redaktor* apod.

Objekty je dále potřeba jemněji třídit podle jejich vlastností. Na toto třídění využijeme opět kategorie. Obecně platí, že nás zajímají takové kategorie, k jejichž instancím se v business oblasti či oblasti IT podpory potřebujeme chovat jinak než k jiným objektům. V případě dokumentů budeme například potřebovat kategorie *schvalované dokumenty*, *neschvalované dokumenty*, *dokumenty vizualizované v pravém horním okně*, *dokumenty vizualizováné v okně aktualit*. Konkrétní znalost pak může vypadat například takto: objekt *ceník kursů-2002* patří do kategorie *schvalované dokumenty*. Obecně má tento druh znalostí formu **objekt je v kategorii**.

Abychom byli schopni se v množství kategorií, které reálné aplikace potřebují (viz kapitola 6), potřebujeme se soustředit i na pořádání kategorií. Objekty našeho zájmu se tedy te stanou kategorie, znalosti budeme vyjadřovat ve formě: **kategorie je v kategorii** (ve smyslu: je prvkem kategorie). Zavedeme si tedy

například kategorie *režim schvalování* či *vizualizační kategorie dokumentů*. Konkrétní znalost pak může vypadat takto: kategorie *schvalované dokumenty* patří do kategorie *režim schvalování*.

Další druh znalostí vyjadřujeme pomocí vztahů mezi kategoriemi. Nejčastěji potřebujeme vyjádřit znalost typu: pokud je objekt v kategorii X pak je třeba ho zařadit do kategorie Y – např.: pokud je objekt v kategorii *harmonogram* pak je třeba ho zařadit do kategorie *dokumenty vizualizované v pravém horním okně*. Tyto vztahy mezi kategoriemi vyjadřujeme pomocí produkčních pravidel. Ty jsou obecně tvaru **IF podmínky THEN akce** . Pokud platí v pravidle uvedené podmínky jsou vykonány uvedené akce. Díky vyjadřování znalostí pomocí kategorií nepotřebujeme pracovat se zcela obecnou formou produkčních pravidel. Lze se omezit na produkční pravidla, ve kterých se podmínky vždy dotazují na příslušnost objektu do kategorie a akce je jedna z následujících: *zařa objekt do kategorie* nebo *vyřa objekt z kategorie*.

4 Dva pojmové systémy KM agenta

Specifikací jednotlivých kategorií a vztahů mezi nimi vlastně definujeme pojmový systém KM agenta. Tímto pojmovým systémem pak KM agent "nahlíží" na objekty v informačním systému.

Jednou z klíčových myšlenek technologie eTrium je ta, že KM agent disponuje **dvěma** pojmovými systémy: 1) pojmovým systémem dané business oblasti 2) pojmovým systémem IT podpory. Klíčovou činností agenta je realizace mapování z pojmového systému business oblasti do pojmového systému IT podpory. Konkrétně to znamená:

- provést analýzu zkoumaného záznamu a jeho relevantního okolí pojmovým aparátem dané business oblasti (viz bod 3, případ B, kapitola 2)
- na základě této analýzy popsat zkoumaný záznam a jeho relevantní okolí pomocí pojmového systému IT podpory (viz bod 4, případ B, kapitole 2)

Příkladem může být prozkoumání business vlastností dokumentu a na základě toho, že je o *ceník* (pojem z business oblasti) tento dokument zařadit mezi *dokumenty vizualizované vpravo nahoře* (pojem z IT podpory).

Hlavním důvodem pro existenci dvou pojmových systémů KM agenta je snaha zjednodušit a zefektivnit proces IT podpory. Důvodem pro existenci nějaké kategorie v pojmovém systému IT podpory je požadavek, aby se IT podpora chovala k prvkům (objektům) této kategorie jinak než k ostatním objektům.

Identifikace objektů, ke kterým se má IT podpora chovat jednotně musí být přímá a nezatížená business vlastnostmi. Příkladem IT kategorie může být kategorie *dokumenty zobrazované vpravo nahoře*. To aby do této kategorie patřily právě ty dokumenty, které se mají zobrazovat vpravo nahoře zajišuje KM agent (konkrétně nějaké produkční pravidlo). Požadavek zákazníka na změnu vizualizace ceníků lze pak uspokojit pouhou změnou příslušného produkčního pravidla v bázi znalostí – tato změna se programátorů vůbec nedotkne. Programátoři se tedy nemusí (a nesmí) zajímat o business vlastnosti zkoumaného objektu (např. zda jde o ceník nebo upoutávku na společenskou akci). Vpravo nahoře zobrazují přesně ty dokumenty, které patří do kategorie *dokumenty zobrazované vpravo nahoře*. Proces IT podpory se tak vyhne jedné z nejproblematičtějších fází - komunikace programátorů s business lidmi a snaze o vzájemné porozumění.

Princip dvou pojmových systémů dále výrazně zjednodušuje ladění celého informačního systému. U klasicky vytvářených systémů je ladění vždy komplikováno skutečností, že neočekávaná funkce programu nad danými daty může být způsobena bu:

- chybou programátora při kódování pravidel a faktů do programu
- špatným porozuměním programátora tomu, co má vlastně naprogramovat
- chybou v business logice

Informační systémy vytvořené pomocí technologie eTrium lze ladit po částech:

- odladit business logiku (to lze dělat už i ve fázi návrhu bez existence komponenty *Databáze* a *Komunikace a zobrazování*)
- laděním korektnosti IT podpory (tj. že to co se má zobrazovat vpravo nahoře se opravdu zobrazuje vpravo nahoře)
- laděním mapování business světa do IT světa

5 Podpora tvorby a údržby KM agenta

Je zřejmé, že pojmové systémy agentů řídících reálné informační systémy budou vcelku rozsáhlé (viz kapitola 6). Proto je potřeba nějakým způsobem návrh a udržování pojmových systémů podpořit. Popíšeme zde dva doplňující se způsoby.

První způsob spočívá ve využití stejného způsobu práce s kategoriemi, jako jsme již popsali. Objektem zájmu jsou nyní kategorie a pravidla použitá pro

specifikaci KM agenta. Zavádíme nové "ladící" kategorie, jako např.: *zatím neověřená pravidla*, *business kategorie, které nejsou mapovány na IT kategorie*, *zatím neodsouhlasené business kategorie* apod. Tedy stejným způsobem jakým "stavíme" KM agenta si tak můžeme stavět "konceptuální lešení" pomocí kterého KM agenta stavíme.

Druhý způsob spočívá v použití metody automatického dokazování. Chceme si nechat dokázat, že se něco nemůže stát (například, že na web nebude vystaven neschválený dokument), nebo že pokud se něco stane, tak se to stane právě daným způsobem (například, že ve sloupci aktualit budou zobrazovány právě a jenom aktuality). Zde lze využít metod, používaných pro automatickou verifikaci znalostních systémů. Nejjednodušší přístup je tento: kategorií a produkčních pravidel je konečně mnoho. Objekt daného typu může tedy být pouze v konečně mnoha stavech, když stav objektu je úplně zadán množinou kategorií, do kterých objekt náleží. Možné přechody mezi stavy objektů jsou zadány právě množinou produkčních pravidel. Pokud tedy zadáme počáteční stav a koncový stav, lze úplným prohledáním stavového prostoru zjistit, že

- Žádnou sekvencí použití aktuálně platných produkčních pravidel se z daného počátečního stavu nelze dostat do stavu koncového. Např. počáteční stav je dokument nepatřící do žádné kategorie a koncový je dokument patřící do kategorií *neschválené dokumenty* a *dokumenty vystavené na webu*.
- Z počátečního stavu do koncového stavu se lze dostat právě danými sekvencemi použití pravidel. Např. počáteční stav je dokument nepatřící do žádné kategorie, koncový je dokument patřící do kategorie *dokumenty vizualizované ve sloupci pro aktuality* a sekvence pravidel je: 1. pokud je dokument *oznámení* pak je *aktualitou*, 2. pokud je dokument *aktualitou* pak je *vizualizován ve sloupci pro aktuality*.

Lze též podpořit porovnávání chování dvou různých verzí KM agentů, a tak zabránit vzniku nežádoucích vedlejších efektů aktualizace KM agenta.

6 Informační systém eDialog

Technologie eTrium byla aplikována při implementaci redakčního systému eDialog [3] pro poradenskou firmu, který umožňuje publikovat informace o činnostech a projektech firmy na webu, vydávat newsletery, organizovat diskusní fóra, plánovat a sledovat redakční činnosti apod. Systém byl implementován v prostředí

operačního systému Linux, jako databázový stroj byl použit PostgreSQL, komunikační rozhraní bylo implementováno v PHP a KM agent byl implementován v SWI Prologu.

Definice KM agenta, který řídí celý redakční systém, obsahuje 350 kategorií – z toho je 188 business kategorií, 119 IT kategorií a 43 ladících kategorií. Pravidel je použito 140 – z toho je 90 pravidel týkajících se konkrétních objektů a 50 pravidel týkajících se kategorií. Skoro všechna pravidla jsou jednoduchého tvaru – X je v *kategorie A* pak plati X je v *kategorie B*. Redakční systém byl do rutinního provozu nasazen v březnu 2002, dynamický web, který je pomocí něj spravován lze nalézt na adrese www.expertis.cz.

7 Závěr

Logika každého businessu vychází ze stejných formálních principů (teorie pojmů a pojmových systémů [1] [7]). Na základě znalosti formálních principů je logika každého businessu [5] strukturovatelná na sadu elementárních faktů a pravidel. Nad těmito fakty a pravidly pracuje KM agent schopný z dodaných fakt vyvozovat fakta nová, kterými je definováno chování systému.

Důsledky pro návrh informačního systému

- Doba potřebná pro implementaci systému se prakticky redukuje na dobu potřebnou ke konceptuálnímu zvládnutí businessu, přípravě příslušných obrazovkových formulářů a doprogramování "plug-inových operací" pro podporu procesu tvorby a rušení některých faktů.
- Možnost simulovat nad KM agentem chod businessu bez nutnosti existence dvou zbylých komponent.
- Lze se soustředit výhradně na odhalování chyb v logice popisu businessu, nemíchají se do toho potenciální chyby způsobené programátory.
- Možnost poloautomatické verifikace namodelované logiky. Je možné definovat a nechat si automaticky vyhledávat "podezřelé" skupiny odvozených faktů a okamžitě identifikovat pravidla, která podezřelý výsledek vyrobila.
- Zcela automaticky je možné nechat dokazovat, že pomocí definovaných faktů a pravidel nelze dosáhnout specifikovaných (chybových) stavů. To nám umožňuje získat stoprocentní jistotu, že IT podpora zajišuje nedosažitelnost identifikovaných problémů. Neboli: Víme-li, které stavy v business procesu nesmí nastat (a to lze identifikovat [6]), máme jistotu, že s IT

podporou dle této technologie nenastanou. Toto je výrazný rozdíl oproti klasicky vytvářeným informačním systémům, kde se vždy jedná o otázku víry.

Důsledky pro údržbu systému

- Existence neustále aktuálního popisu businessu a jeho IS (soubor kategorií a pravidel) – toto má výrazný přínos pro získání a udržování ISO certifikace. Popis businessu neleží ve skříni, ale běhá podle něj IS. Popis informačního systému při tom není zaklet v programovém kódu či potenciálně neaktuální dokumentaci.
- Snadná identifikace dopadů změny s využitím simulace a poloautomatické verifikace.
- Možnost pustit starou a novou verzi KM agenta paralelně a automaticky vyhledat rozdíly v odvozených souborech faktů. Velká část změn se odehrává pouze v business pravidlech a není tedy nutné nic přeprogramovávat – provedení změny je rychlé a bezpečné.

Reference

[1] Materna, P.: Concepts and Objects. Acta Philosophica Fennica, Helsinki, 1998

[2] Project STRADIWARE, contract No.: COPERNICUS 977132, web site: www.itd.clrc.ac.uk/activity/stradiware/

[3] Projekt eDIALOG, EXPERTIS s.r.o. www.expertis.cz

[4] Staníček, Z.: Universální modelování a jeho vliv na tvorbu IS, Proc. of Conf. DATAKON'2001, Masaryk university, Brno 2001 (zvaná přednáška

[5] Stanicek, Z., Motal, M.: Business Process Modeling, Proc. of conf. SYSTEMS INTEGRATION '98, Vorisek, Pour eds. KIT, VSE Prague, CSSI Prague, 1998

[6] Staníček, Z.: Chaos, Strategické plánování a řízení projektů, Sborník DATASEM '97, CS-COMPEX a.s. Brno, 1997

[7] Tichy, P.: The Foundations of Frege's Logic. De Gruyter, Berlin-New York, 1988

Použití myšlenky neuronových sítí při kreslení planárních grafů

Arnošt Svoboda

Masarykova universita v Brně, Ekonomicko-správní fakulta

Lipová 41a 659 79 Brno, Česká Republika

arnost@econ.muni.cz

Abstrakt

Příspěvek presentuje využití jednoho z modelů neuronových sítí, tzv. samoorganizačních map, k vykreslování grafů, a to speciálně úrovňových (level graph) grafů, jednoho z typu planárních grafů. Samoorganizační mapy jsou založené na soutěžním modelu učení, jsou to metody, které využívají strategie učení bez učitele. Společným principem těchto modelů je, že výstupní neurony spolu soutěží o to, kdo bude vítězný, tedy aktivní. Asi nejdůležitější neuronovou architekturou, vycházející ze soutěžního učení, je Kohonenova samoorganizační mapa. Zde presentovaný příspěvek ukazuje použití myšlenky Kohonenovy samoorganizační mapy (SOM) [1] a pomocí jejího rozšíření ukazuje další možnost její aplikace na kreslení grafů. Po vysvětlení principu práce neuronových sítí a popisu používané terminologie je popsán postup, použitý při rozšíření SOM a prezentována praktická ukázka výstupu pro zadaný graf. Jedná se pravděpodobně (podle dostupné literatury) o jedno z prvních uplatnění tohoto přístupu při vykreslování planárních grafů. Rozšíření SOM pro kreslení grafů je prezentováno v [1].

1 Motivace

Příspěvek je navázáním a pokračováním příspěvku Jany Kohoutkové [2], který shrnoval hlavní rysy projektu Hypermedata (CP 94-0943, [2]), jehož cílem bylo vybudovat prostředí a nástroje pro vzájemnou výměnu dat mezi nezávislými nemocničními informačními systémy (NIS).

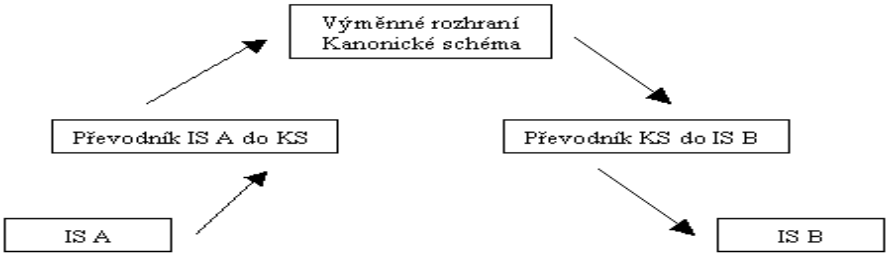
[1] SOM Self-Organizing Map

[2] ITAT 2001, Orientované grafy jako nástroj systémové integrace

Projekt Hypermedata pokrývá dvě související oblasti:

- integraci datových zdrojů,
- jejich jednotnou prezentaci uživatelům.

Cílem je integrovat data ze zcela nezávislých informačních zdrojů, jejichž existence a budoucí vývoj nesmějí být omezeny způsobem, jakým budou společně integrovány do uživatelsky jednotného celku. Proto je integrace řešena cestou vzájemné výměny dat konverzemi přes standardní datové rozhraní. Podmínkou je jednotné uživatelské rozhraní v rámci celého konverzního toku dat. Systém navržený a implementovaný v rámci projektu tedy sestává ze dvou základních komponent: datového převodníku a prohlížeče/zobrazovače.



Obr. 1 Transormace datových instancí z IS A do kanonického schématu (KS) a dále do IS B

Základní architektura je jednoduchá. Pro zvolenou aplikační oblast každý zúčastněný informační systém poskytne specifikaci svého exportního schématu, které je použito pro převod datových schémat a jejich vazeb do kanonického schématu. Z kanonického schématu je možné datové schéma a jejich vazby transformovat do formátu jiných informačních systémů. Schémata i specifikace transformačních pravidel jsou interně popsána a reprezentována pomocí orientovaných grafů s rozlišenými typy uzlů a hran. Typem uzlu je rozlišena entita od asociace, typem hrany je rozlišen asociační vztah od ISA vztahů.

Dále se soustředíme na možnosti prohlížeče, který prezentuje transformovaná data uživateli.

Vstupní data pro prohlížeč jsou:

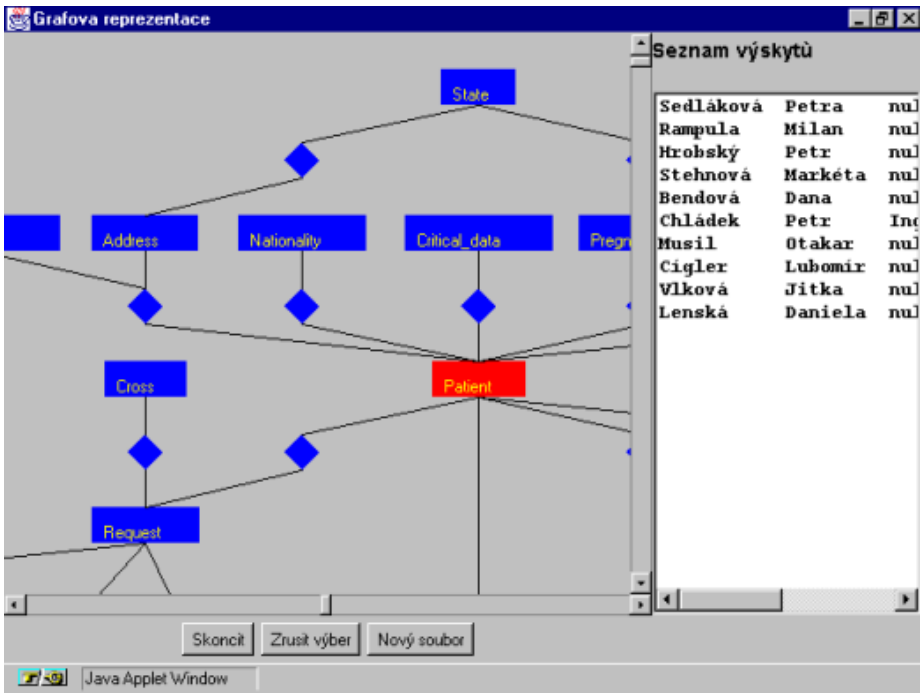
- popis dokumentu,
- data dokumentu.

Popis dokumentu je popisem grafové struktury, která bude zobrazena uživateli, tj. množiny uzlů, hran a omezujících podmínek odpovídajících jak objektům datového modelu (entitám, atributům, relacím), tak objektům dokumentového modelu (stránkám, atributům a hypertextovým odkazům).

Data pro dokumenty, tj. skutečné hodnoty pro entity, jejich atributy a asociace mezi nimi, jsou poskytována na vyžádání z datového serveru, naplněného daty v procesu konverze.

Prohlížeč pracuje jak s grafovou strukturou dokumentu, tak s jeho daty. Uzlem grafu dokumentu může být buď entita nebo relace, spojené jednoduchými hranami. Každá hrana spojuje entitu s relací.

Prohledávání se realizuje v prohledávacím okně, kde je jednak zobrazena struktura dokumentu jako graf (intenze), jednak je zobrazen seznam výskytů (extenze) zvoleného uzlu grafu. Zobrazování atributů zvoleného výskytu se realizuje v okně WWW prohlížeče. Na ukázce na obrázku 2 vidíme v levé části okna prohlížeče strukturu grafu, v pravé části je obsah vybrané entity. Pro účely prezentace struktury grafu byla použita heuristika, která více méně splňovala nároky prezentace. Jednotlivé uzly grafu byly vykresleny ve svých úrovních a spojeny hranami dle zadání.



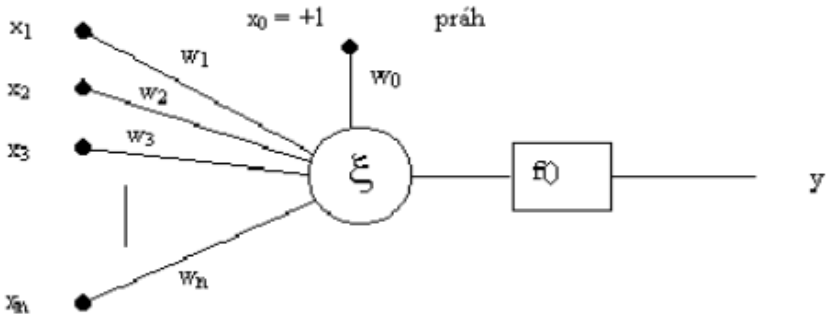
Obr. 2 Struktura dokumentu s vybranou entitou

Potřeba zobrazit strukturu dokumentu ve tvaru, jak je vidět v levé části okna, tj. vykreslení úrovňového (v literatuře level graph) grafu stála na začátku myšlenky použít možnosti neuronových sítí.

2 Matematický model neuronové sítě

2.1 Fomální neuron

Formální neuron (dále bude používán pouze pojem neuron) je základem matematického modelu neuronové sítě. Jeho struktura je schematicky znázorněna na obrázku 3.



Obr. 3 Formální neuron

Je vidět, že formální neuron má n obecně reálných vstupů $x_1, \ldots, x_n$. Vstupy jsou ohodnoceny obecně reálnými váhami $w_1, \ldots, w_n$. Vážená suma vstupních hodnot představuje vnitřní potenciál neuronu:

$$\xi = \sum_{i=1}^{n} w_i x_i$$

Hodnota vnitřního potenciálu ξ po dosažení prahové hodnoty h indukuje výstup neuronu y. Výstupní hodnota $y = f(\xi)$ při dosažení prahové hodnoty potenciálu h je dána přenosovou (aktivační) funkcí f. Přenosová funkce tedy převádí vnitřní potenciál neuronu do definovaného oboru výstupních hodnot. Tato aproximace biologických funkcí neuronu (v literatuře nazývané Linear Treshold Gate, LTG) byla popsána v [4].

Po formální úpravě dosáhneme toho, že funkce f bude mít nulový práh (takže nebude už rovný h) a práh neuronu se záporným znaménkem budeme chápat jako váhu $w_0 = -h$ dalšího formálního vstupu $x_0 = 1$ jak je nakresleno v obrázku 3. Matematická formulace funkce jednoho formálního neuronu je po těchto úpravách dána vztahem:

$$y = f(\xi) = \begin{cases} 1 & \text{pokud} \quad \xi \geq 0 \\ 0 & \text{pokud} \quad \xi < 0 \end{cases}, \quad \text{kde} \quad \xi = \sum_{i=0}^{n} w_i x_i$$

Tato diskrétní přenosová funkce bývá aproximována spojitou (případně diferencovatelnou) funkcí, používané funkce jsou, mimo ostré nelinearity, saturovaná lineární funkce, standardní (logistická) sigmoida), hyperbolický tangens a jiné.

Pro řešení složitějších problémů je třeba neurony propojit nějakým způsobem dohromady, tj. vytváříme neuronovou síť.

2.2 Neuronová síť

Vzájemné propojení neuronů v síti a jejich počet určuje architekturu (topologii) neuronové sítě. Stavy všech neuronů určují stav neuronové sítě a váhy všech spojů představují konfiguraci neuronové sítě. V neuronové síti dochází v čase ke změnám. Mění se propojení a stav neuronů, adaptují se váhy. Pro naše účely si specifikujeme blíže adaptivní dynamiku.

2.2.1 Adaptivní dynamika

Adaptivní dynamika specifikuje *počáteční konfiguraci* sítě a změny jejich vah v čase. Všechny možné konfigurace sítě tvoří *váhový prostor* neuronové sítě. Na začátku práce v adaptivním režimu se nastaví váhy všech spojů na počáteční konfiguraci (často se používá náhodné nastavení vah). Pak následuje proces vlastní adaptace. Cílem adaptace je nalezení takové konfigurace sítě ve váhovém prostoru, která při používání sítě bude realizovat předepsanou funkci. Adaptivní režim slouží k *učení* sítě pro realizaci zadané funkce, tj. aby realizovala zobrazení ϕ z množiny vstupních vektorů $X \subset \Re^n$ do množiny výstupních vektorů $Y \subset \Re^m$. Neuronová síť aproximuje požadované zobrazení funkcí $y = f(x, w, \theta)$, kde y je výstupní vektor, x je vstupní vektor, w je vektor všech vah sítě a θ je vektor prahů. Funkce f je určena typem neuronů a topologií sítě. Během učení se mění parametry w a θ. Učící algoritmus pro požadované zobrazení $\varphi : X \rightarrow Y$ nalezne takové w a θ, že funkce f je právě aproximací tohoto zobrazení.

Adaptivní režim můžeme rozdělit v zásadě na dva typy: *učení s učitelem (learning with the teacher, supervised learning)* a *bez učitele (unsupervised learning)*. Stručně si popíšeme učení bez učitele.

Při učení bez učitele je na vstupu trénovací množina vstupů, trénovací vzory. Síť v adaptivním režimu sama musí odhalit optimalizační kriteria. To ale na druhou stranu znamená, že jednotlivý typ sítě je vhodný k řešení jednotlivých problémů. Typické aplikační oblasti jsou např. shluková analýza, asociace, kódování, komprese a jiné. Nejznámější modely neuronových sítí, založené na učení bez učitele, jsou modely využívající učení založené na Hebbovském zákoně (tzv. Hebbian learning), soutěžní učení (competitive learning) a samoorganizační mapy (self-organizing feature map learning). Principem metod, založených na strategii soutěžního učení je, že výstupní neurony sítě spolu soutěží o to, který z nich bude

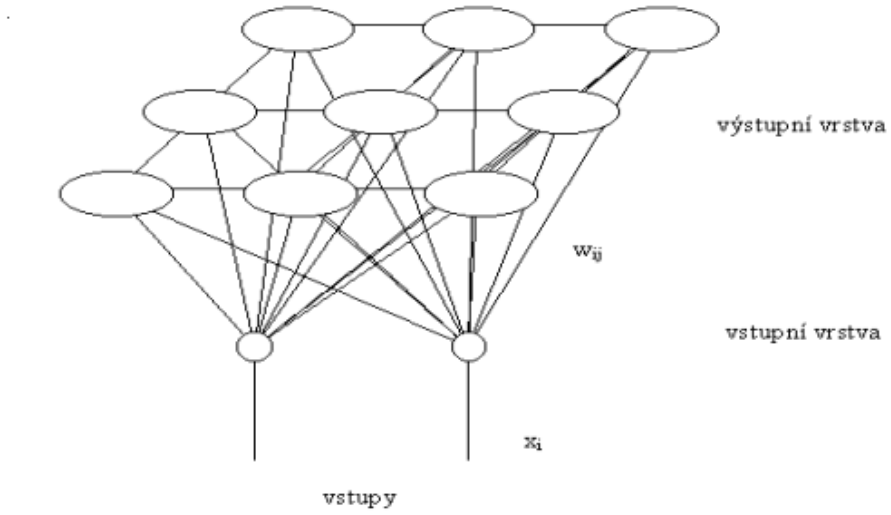
aktivní. Pravděpodobně nejdůležitější neuronovou architekturou, vycházející ze soutěžního učení, je Kohonenova samoorganizační mapa (Self Organizing Map).

2.3 Kohonenony samoorganizační mapy (SOM)

Jedná se o dvouvrstvou síť s úplným propojením jednotek mezi vrstvami. Vstupní vrstvu tvoří n neuronů pro přenos vstupních hodnot $\mathbf{x} \in \Re^n$. Výstupní vrstvu tvoří reprezentanti (codebook vectors) $\mathbf{w}_i \in \Re^n; (i = 1, \ldots, h)$. Jako reprezentanta přiřadíme ke každému vektoru $\mathbf{x}$ tu jednotku $\mathbf{w}_c$, která je mu nejbližší, tj.

$$c = arg\ min_{l=1,\ldots,h}\{\|\mathbf{x} - \mathbf{w}_l\|\}$$

Váhy náležející jedné výstupní jednotce určují její polohu ve vstupním prostoru. Výstupní jednotky jsou uspořádány do nějaké topologické struktury, kterou je dáno, které jednotky spolu sousedí. Pro naše účely použijeme dvojrozměrnou čtvercovou mřížku.



Obr. 4 Příklad topologie Kohonenovy neuronové sítě

Dále zavedeme pojem okolí neuronu: $N_s(c) = \{j;\ d(j,c) \leq s\}$ kde pro okolí $N_s(c)$ neuronu c platí, že vzdálenost ostatních neuronů v síti je menší nebo rovna s, $s \in N$.

Adaptivní dynamika je jednoduchá. Po předložení jednoho tréninkového vzoru proběhne mezi neurony sítě soutěž, která určí, který neuron je nejblíže předloženému vstupu. Učící algoritmus pro vítězný neuron má tvar:

$$w_{ji}^{(t)} = \begin{cases} w_{ji}^{(t-1)} + \theta(x_i^{(t)} - w_{ji}^{(t-1)}) & j \in N_s(c) \\ w_{ji} & \text{jinak} \end{cases}$$

kde $c = arg\ min_{l=1,\ldots,h}\{\|\mathbf{x}^{(t)} - \mathbf{w}_l\|\}$, parametr $\theta \in \Re$, $0 < \theta \leq 1$ určuje míru změny vah. Pro názornější představu je možná geometrická představa, kde vítězný neuron c posune svůj váhový vektor $\mathbf{w}_c$ o určitou poměrnou vzdálenost k aktuálnímu vstupu. Je snaha o to, aby vítězný neuron ještě zlepšil svoji pozici oproti ostatním neuronům vůči aktuálnímu vstupu.

3 Od SOM ke kreslení planárních grafů

Na nakreslení grafu jsou kladené podmínky, které mají učinit strukturu grafu přehlednou a esteticky příjemnou. Ale toto jsou kriteria, které jdou jen obtížně formalizovat. Některé obecné zásady pro nakreslení jsou např. minimální počet hran které se kříží, rovnoměrné rozdělení uzlů a délek hran. Ale optimalizace i takového kriteria, jako je minimum křížících se hran nebo minimalizace délek hran je NP-úplný problém [7]. Proto se používají heuristické metody pro aproximaci optimálního nakreslení.

Jestliže se díváme na váhový prostor ne jako na výsledek nějakého dotazu, ale vezmeme váhový prostor jako pohled na graf, dostáváme se k myšlence a postupu, použitému v tomto příspěvku. Topologická struktura ve tvaru mřížky se může podobat nakreslení nějakého grafu, který má tvar úrovňového grafu, pokud si představíme místo výstupních neuronů uzly a hrany mřížky nahradíme hranami grafu. První myšlenkový posun je tedy v pohledu na výsledek práce SOM, kde použijeme chování váhových vektorů a jejich pozici jako výsledek, tj. nakreslení grafu. Druhý krok v našem myšlenkovém posunu spočívá v tom, že místo trénování sítě k tomu, aby dávala správné výstupy vzhledem k zadaným vstupům, naše síť po natrénování nebude už nikdy použitá, tj. výsledek trénovacího procesu je v našem případě považován za výstup.

Ve standardním modelu SOM jsou sousední neurony ve výstupní vrstvě spojené do tvaru mřížky. V případě kreslení grafu v nakreslení, které je podobné úrovňovému grafu, nemohou být spojené všechny neurony, ale pouze ty, které odpovídají zadání grafu, tj. seznamu uzlů a hran mezi nimi.

Před samotným zpracováním je provedena úprava hran, které jsou mezi uzly na úrovních, které nejsou sousední. Takové hrany jsou fiktivně rozděleny, a je

mezi ně, na každou vynechanou úroveň, vložen fiktivní uzel. Na konci zpracování, po rozmístění uzlů, se odstraní fiktivní uzly a rozdělené hrany se nahradí původní, spojující opět dva původní uzly.

Dalším krokem je rozhodnout, co je v našem případě vítězný neuron a jaké je jeho okolí. V příspěvku je použita ukázka výstupu, kdy jako první je použit uzel (neuron) v nejširší vrstvě, tj. ve vrstvě, která obsahuje nejvíce uzlů. Okolí je tvořeno nejdříve uzly ve vyšších vrstvách, spojené hranami se zvoleným uzlem, až do zvolené úrovně. Takto se postupně prohlásí všechny uzly v nejširší vrstvě za vítěze a je pro ně uplatněn vztah pro vítězný neuron. Dále se postupuje stejně pro uzly ve vyšších vrstvách, až se vyčerpají všechny vrstvy. Analogicky se postupuje pro uzly v nižších vrstvách.

4 Příklad použití

Seznam uzlů a hran je zadán ve tvaru shodném s původním zadáním pro projekt Hypermedata.

Zadání uzlů:

200 7 1	200 5 7
200 6 2	200 5 8
200 6 3	200 5 11
200 6 4	200 5 12
200 6 5	200 3 9
200 5 6	200 3 13

První sloupec je souřadnice x, druhý sloupec je souřadnice y uzlu, jehož název je ve třetím sloupci. Pokud použijeme jako příklad první uzel, má jeho souřadnice x hodnotu 200, souřadnice y má hodnotu 7 a název uzlu je 1. Všechny uzly mají souřadnici x shodnou. Při skutečném trénování SOM se volí váhy spojů náhodné, většinou se používají hodnoty blízké nule. Zde byla pro názornost, jak se příslušné uzly posouvají ve váhovém prostoru během trénování sítě, zvolena velká hodnota souřadnice x. Souřadnice y reprezentuje úroveň uzlu, takže uzel, který je umístěn nejvýše, má nejvyšší hodnotu souřadnice y.

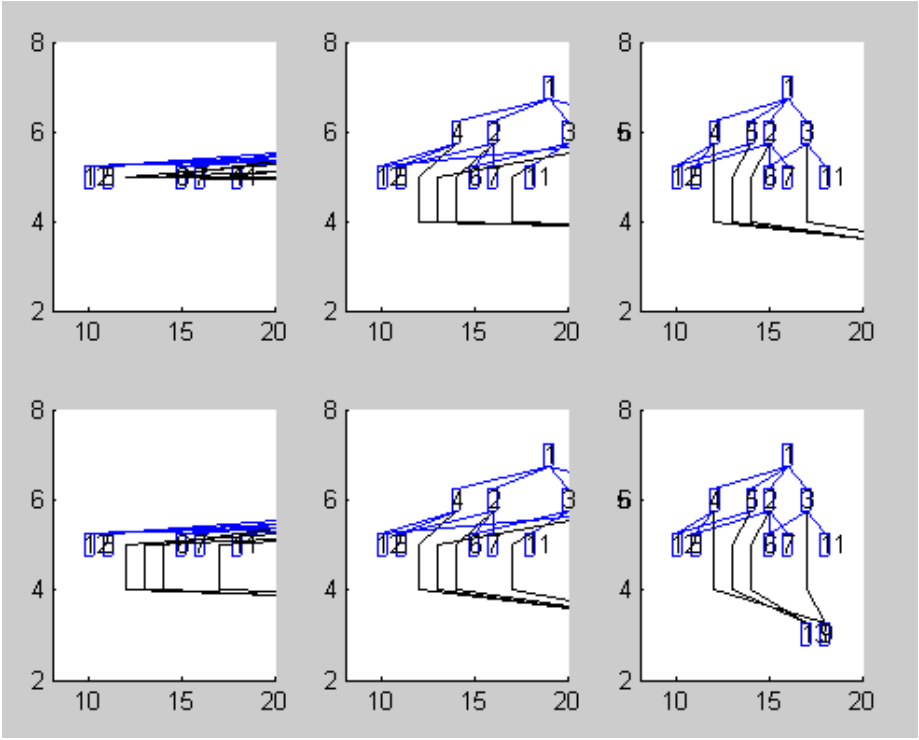
Zadání hran:

1 2	2 6	3 9	4 8
1 3	2 7	3 11	4 9
1 4	2 8	2 13	5 12
1 5	3 6	4 12	5 13

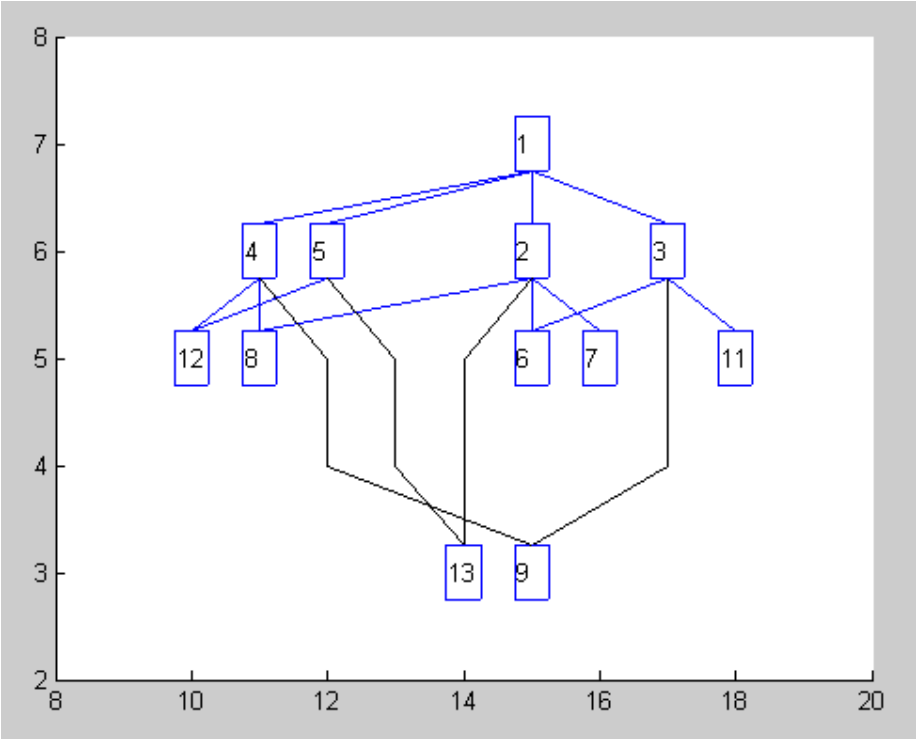
První sloupec je název uzlu z kterého hrana vychází, druhý sloupec je název uzlu, do kterého hrana vstupuje.

V názvu příspěvku je použitý termín planární graf, výsledné nakreslení ale není planární. Byl zvolen záměrně takový typ grafu, aby s planárním vykreslením byly problémy. Ale na druhou stranu je nutné pohlížet na příspěvek jako na první uplatnění myšlenky, kterou je ještě nutné propracovat. Bylo by možné např. volit jiný postup při výběru vítězného neuronu, jiné vstupní hodnoty souřadnic x a y, po nakreslení spočítat počet křížících se hran, provést nové rozdělení vstupních vektorů a vypočítat nové pozice, zda bude počet křížení menší apod.

Na obrázku 5 je vidět postup, jak je graf postupně vykreslován tak jak dochází k přepočítávání vah pro jednotlivé uzly. Stavy při nakreslení grafu se mění shora dolů a z leva do prava, takže vlevo nahoře je jeden z prvních stavů, vpravo dole je výsledné nakreslení. Pro lepší přehled je na obrázku 6 výsledné nakreslení grafu.



Obr. 5 Příklady trénování sítě



Obr. 6 Příklad nakreslení zadaného grafu

Reference

[1] Meyer B. Self-organizing graphs a neural perspective of graph layout, 1998 URL:http://welcome.to/bernd.meyer/

[2] CP 94-0943 HyperMeData (interní dokumentace kprojektu). HyperMeData Consortium 1998

[3] Šíma J., Neruda R. Teoretické otázky neuronových sítí. 1. vyd. Praha : Univerzita Karlova. Matfyzpress, 1996. ISBN 80-85863-18-9

[4] McCulloch W. S., Pitts W. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 5:115-133, 1943.

[5] Grossberg S. Adaptive pattern classification and universal recording: I. parallel development and coding of neural feature detectors. Biological Cybernetics, 23:121-134, 1976

[6] Grossberg S. On learning and energy-entropy dependence in recurrent and nonrecurrent signed networks. Journal of Statistical Physics, 1, 319-350, 1969

[7] DiBatista G., Eades P., Tamassia R., Tollis G. Algorithms for drawing graphs: an annotated bibliography. Journal of Computational Geometry Theory and Applications, 4:235-282, 1994