Zhlukovanie textových dokumentov modifikovaným algoritmom GHSOM

Peter Bednár, Peter Butka

Katedra kybernetiky a umelej inteligencie,
Fakulta elektrotechniky a informatiky,
Technická univerzita v Košiciach, Letná 9, 041 20, Košice,
Slovenská republika,
{Peter.Bednar, Peter.Butka}@tuke.sk

Abstrakt Jednou z možností automatickej organizácie množiny textových dokumentov je použitie algoritmov nekontrolovaného učenia na báze umelých neurónových sietí, konkrétne samoorganizujúcich sa máp (Self-Organizing Maps, SOM). Táto práca pojednáva o použití modifikovaného algoritmu tohto typu - GHSOM (Growing Hierarchical SOM) pri zhlukovaní textových dokumentov v slovenčine. Výsledkom boli hierarchie máp organizujúcich dané textové dokumenty do množín popísaných charakterizujúcimi termami (slovami).

1 Úvod

V súčasnosti existuje mnoho systémov pre zhlukovanie textových dokumentov a mnohé z nich fungujú na základe princípu samoorganizujúcich sa máp (SOM). Základnou vlastnosťou modelov na báze architektúry SOM je ich schopnosť zachovávať topológiu mapovania vstupného priestoru vo výstupnom priestore (mape). Často je problémom klasickej architektúry SOM (Kohonenova mapa) jej pevná štruktúra (po začatí učenia). Model blízky klasickému modelu SOM, Growing Grid (popísaný Fritzkem - [1]), umožňuje zväčšovanie pôvodnej SOM dynamicky počas procesu učenia (pridávaním riadkov resp. stĺpcov nových neurónov). Druhá adaptívna metóda je založená na použití hierarchickej štruktúry nezávislých máp, kde pre každý prvok mapy (neurón) je na novej úrovni pridaná nová mapa, ktorá potom podrobnejšie rozdeľuje príklady nadradeného (rodičovského) neurónu. Táto architektúra sa nazýva Hierarchical Feature Map (HFM, [2]). Algoritmus GHSOM (Growing Hierarchial SOM, [3]) kombinuje postupy týchto dvoch modelov neurónových sietí. Znamená to, že každá vrstva hierarchickej štruktúry pozostáva z množiny nezávislých máp, ktoré adaptujú svoju veľkosť s ohľadom na požiadavky vstupných dát (príkladov prislúchajúcich danej mape).

V nasledujúcej kapitole popíšeme klasický algoritmus GHSOM a prezentujeme prístup ku predspracovaniu textových dokumentov. V ďalších kapitolách budú popísané príčiny a návrhy modifikácií klasického algoritmu (a metód predspracovania), spôsob ich implementácie a experimenty s touto novou implementáciou.

2 Použitie algoritmu GHSOM

2.1 Predspracovanie

Predspracovanie textových dokumentov v našom prípade pozostávalo z nasledujúcich krokov:

1. Tokenizácia
2. Eliminácia neplnovýznamových slov
3. Úprava slov na základný tvar (stemming)
4. Výber termov na základe ich frekvencie výskytu

V prvom kroku (tokenizácia) je vstupný text transformovaný na tokeny. Tokenizácia je nevyhnutnou súčasťou predspracovania, ostatné kroky sú voliteľné (používajú sa kvôli redukcii priestoru termov). Takto získané tokeny predstavujú základný slovník spracovávanej množiny dokumentov. V nasledujúcom kroku sú z tejto množiny odstránené neplnovýznamové slová (stopwords - spojky, častice, zámená, ...) porovnaním so zoznamom slov v tzv. stop-liste. V ďalšom kroku sú zostávajúce tokeny transformované na ich základné tvary - stemy (ako napríklad "baníkov", "baníkom", "baník" sú všetky transformované na jeden koreň - "baník"). Dosiahneme tak opätovné zmenšenie slovníka (pre rôzne morfologické tvary zostane len koreň slova - stem). Posledný krok na základe frekvencie výskytu slov takto upraveného slovníka odstráni tie slová, ktoré sa vyskytovali v texte príliš často (nie sú zaujímavé pre rozlíšenie dokumentov), resp. tie slová, ktoré sa v texte vyskytli v príliš malom počte dokumentov. Preto sa ponechajú len tie tokeny - termy, ktorých frekvencia výskytu je niekde medzi dvojicou nastaviteľných prahov pre minimálnu a maximálnu povolenú hodnotu frekvencie. Skúmaná kolekcia dokumentov je potom reprezentovaná pomocou dokument-term matice. Pre váhovanie termov sme použili *tfidf schému*, ktorá priradzuje každej dvojici term-dokument reálne číslo (príslušnú váhu) nasledovne:

$$w_{ij} = tfidf_{ij} = tf_{ij} \log\left(\frac{ndocs}{df_i}\right), \tag{1}$$

kde tf_{ij} je frekvencia výskytu j-tého termu v i-tom dokumente, $ndocs$ je počet dokumentov v kolekcii a df_i je počet dokumentov obsahujúcich daný j-tý term (dokumentová frekvencia).

2.2 Algoritmus zhlukovania

Na začiatku procesu učenia je potrebné inicializovať štartovaciu mapu najvyššej úrovne. Mapa, ktorá pozostáva z $m \times n$ neurónov (centier zhlukov), je inicializovaná náhodne spomedzi vstupných vektorov.

Po inicializácii sú náhodne vyberané vstupné vektory prezentované na vstup neurónovej siete. Pre každý neurón sa vypočíta aktivácia vzhľadom k danému dokumentu. Neurón s najnižšou aktiváciou (použitie Euklidovskej metriky) sa označí ako víťaz. Tento vektor, ako aj vektory v okolí víťazného neurónu, sa adaptujú použitím nasledujúcich dvoch formúl:

$$influence = 1 - \frac{dist}{neighbourhood + 0.5} \tag{2}$$

$$center[i] = center[i] + learnrate \,.\, influence \,.\, vector[i] \,, \tag{3}$$

kde *dist* je vzdialenosť medzi víťazným neurónom a upravovaným neurónom *center*, *neighbourhood* je parameter susednosti. Potom *influence* je vypočítaná miera vplyvu daného vstupu *vector* na upravované váhy, *learnrate* je parameter učenia.

Počiatočný iteračný proces (jedna iterácia = každý dokument kolekcie je vybraný ako vstupný vektor) končí dosiahnutím *expandcount* iterácií. Potom sa vypočíta variabilita každého neurónu mapy (variabilita dokumentov priradených danému neurónu), ako aj variabilita celej mapy. Variabilita (stredná kvadratická odchýlka) neurónu sa vypočíta ako priemerná hodnota vzdialenosti centra daného neurónu od vstupných vektorov priradených tomuto neurónu. Potom variabilita mapy je priemerná variabilita neurónov danej mapy.

Po týchto krokoch je testovaná stredná kvadratická odchýlka (MQE) danej mapy. Ak podmienka

$$MQE \geq \tau_1 \,.\, mqe_0 \tag{4}$$

je splnená (reprezentuje vzťah variability danej mapy MQE k variabilite celej množiny vstupných vektorov mqe_0), potom je potrebné pridať neuróny. Preto zvolená konštanta τ_1 reprezentuje prah pre vkladanie neurónov. Pred pridaním nájdeme najvariabilnejší neurón (tzv. chybový neurón) a jeho najvzdialenejšieho suseda (podľa metriky vo vstupnom priestore). Potom vložíme celý blok neurónov pridaním riadku resp. stĺpca neurónov medzi túto dvojicu. Ak variabilita neurónov mapy poklesne dostatočne, vkladanie neurónov sa skončí. Inicializácia váh nových neurónov sa získava spriemerňovaním váh okolitých neurónov. Po každom vložení neurónov je potrebné mapu opäť preučiť.

Po skončení fázy rozširovania mapy sa testuje každý neurón mapy pre možnosť expanzie vo forme novej podmapy. Preto ak podmienka

$$mqe \geq \tau_2 \,.\, mqe_0 \tag{5}$$

je splnená (reprezentuje vzťah variability konkrétneho neurónu mqe k mqe_0), expanzia neurónu na novú úroveň hierarchie sa uskutoční. Parameter τ_2 predstavuje prah pre expanziu neurónu. Novovzniknutá mapa má veľkosť 2×2. Váhové vektory sú inicializované na základe neurónov v okolí expandovaného neurónu, a to spriemernením váh daného neurónu a troch jeho susedov v závislosti na pozícii nového neurónu v mape.

Algoritmus teda postupuje zhora-nadol a rozkladá množinu vstupných vektorov na stále menšie skupiny. Dostávame tak celú hierarchiu máp. Koniec algoritmu je vtedy, ak už neexistuje neurón (na žiadnej mape), ktorý by bolo potrebné expandovať na novú úroveň.

2.3 Vizualizácia

Výstupom celého procesu spracovania je množina HTML stránok, každá stránka zobrazuje jednu mapu hierarchie. Políčka mapy odpovedajú neurónom - centrám zhlukom. Každý zhluk je popísaný polohou na mape, počtom priradených vektorov a množinou charakteristických termov. Tieto sa získavajú na základe variability váh termov príkladov daného neurónu metódou LabelSOM. Táto metóda bola popísaná v [4].

3 Problémy a návrh modifikácií

Na základe skúseností s používaním algoritmu popísaného v predchádzajúcom príklade sme analyzovali jeho implementáciu popísanú v [5], pričom sme sa sústredili na niektoré konkrétne problémy.

Prvým problémom je pridávanie neurónov. V tomto prípade sa pridáva nie jeden neurón, ale aj množstvo ďalších neurónov v riadku resp. stĺpci. V dôsledku tohto procesu pridávania neurónov môže ľahko dôjsť k vzniku nepokrytých neurónov (neuróny bez podpory vstupných dát), ako aj k rozbitiu homogénnych zhlukov vo vzdialenej časti mapy. V našej modifikácii navrhujeme pridávať iba jeden neurón na miesto, kde je to najviac potrebné. Znamená to, že sledujeme počty vektorov, ktorým by viac vyhovoval akýsi virtuálny neurón v strede medzi dvojicou neurónov v mape. Hrana, ktorú reprezentuje tento stred medzi dvoma neurónmi, je nestabilná, ak je toto číslo veľké. Nový neurón pridávame do stredu najnestabilnejšej hrany chybového neurónu (teda jeden z dvojice neurónov musí mať najväčšiu variabilitu spomedzi všetkých neurónov skúmanej mapy).

Ďalším problémom je inicializácia nových máp po expandovaní neurónu. V dôsledku inicializácie popísanej v predchádzajúcej kapitole dochádza často k vzniku mapy (2×2) s jediným pokrytým neurónom (všetky vektory danej mapy sú klasifikované do jedného neurónu). Problém inicializácie môže byť riešený neinkrementálnym prístupom založenom na inicializácii pomocou vektorov klasifikovaných expandovaným neurónom.

Predspracovanie je podstatnou časťou celého procesu spracovania textových dokumentov, zvlášť ak sa jedná o spracovanie slovenčiny. V našich prvotných experimentoch sme pre slovenčinu použili veľmi jednoduchý algoritmus pre stemming založený na orezávaní prípon. Nedielnou súčasťou práce s textami, vzhľadom k vlastnostiam ich vektorovej reprezentácie, by mal byť aj spôsob ich ukladania a používania (malo by byť čo najefektívnejšie, pretože ide o veľmi riedke matice). K výraznému zlepšeniu najmä popisu neurónov by mal prispieť lepší algoritmus pre stemming (v prípade slovenčiny sa ani zďaleka nejedná o triviálny algoritmus, žiada sa použiť lingvistické postupy), použiť kvalitnejší zoznam neplnovýznamových slov, prípadne hľadať frázy a získať tak kompaktnejší slovník použitých termov.

4 Implementácia

Implementovaný systém pozostáva z troch základných blokov. V ďalšom sa sústredíme najmä na popis bloku zhlukovania. Vstupom celého systému je kolekcia dokumentov. Táto je indexovaná v bloku predspracovania. Jeho výstupom sú tfidf profily dokumentov, ktoré vstupujú do bloku zhlukovania, reprezentujúceho samotný modifikovaný algoritmus GHSOM. Výstupom zhlukovania je hierarchia máp. Táto vstupuje do bloku vizualizácie (LabelSOM) a dostávame výstupnú množinu HTML stránok. Podrobnejší opis celého systému nájdete v [6].

4.1 Proces učenia máp

Algoritmus učenia každej mapy pozostáva z týchto krokov:

1. Prvotné učenie (bez pridávania neurónov)
2. Fáza pridávania neurónov (rozširovanie mapy)
3. Expanzia neurónov na novú úroveň hierarchie (po fáze pridávania)

Jeden cyklus učenia predstavuje preučenie mapy všetkými priradenými vektormi. Prvotné učenie sa uskutočňuje v *expandCycles* cykloch cez všetky dokumenty. Výsledkom je naučená mapa, váhy centier neurónov sa teda prakticky nemenia, mapa je stabilná.

Pridávanie neurónov sa uskutočňuje v maximálne v *learnCycles* pokusoch. Znamená to, že sa pokúšame vložiť neurón na začiatku každého cyklu. Na konci cyklu sa testuje podmienka

$$(curCycle < learnCycles) \wedge (np < maxN) \wedge (MQE \geq \tau_1 \,.\, mqe_0) \qquad (6)$$

kde *curCycle* je aktuálny cyklus v procese pridávania neurónov, *maxN* je parameter maximálneho počtu neurónov mapy (užívateľská konštanta), *np* je aktuálny počet neurónov mapy. Tretia časť podmienky je ekvivalentná podmienke pre pridávanie neurónov v klasickom algoritme GHSOM. To znamená, že fáza pridávania neurónov končí, ak aspoň jedna zo subpodmienok nie je splnená, teda ak aspoň jeden z nasledujúcich bodov je splnený:

1. Dosiahneme maximálny počet cyklov *learnCycles*
2. Počet neurónov mapy dosiahne *maxN*
3. Variabilita mapy klesne pod požadovanú úroveň (ako klasický GHSOM)

Po fáze pridávania neurónov sa pre každý neurón testuje podmienka

$$(vectorCount \geq minExpandVectorCount) \wedge (mqe \geq \tau_2 \,.\, mqe_0) \qquad (7)$$

kde *vectorCount* je počet vektorov priradených kandidátovi na expanziu a *minExpandVectorCount* je zvolený prah pre tento počet, druhá podmienka je totožná s podmienkou expanzie klasického algoritmu GHSOM. Ak neurón spĺňa obidve podmienky, expanduje na novú úroveň, inicializuje sa nová mapa veľkosti 2×2. Po tomto kroku sa učenie prenáša na nové podmapy.

4.2 Adaptácia váh neurónov

Pre nájdenie víťaza môžeme použiť rôzne postupy, napríklad kvadrát Euklidovskej vzdialenosti. Potom víťazom pre daný vstupný vektor *vector* je taký neurón *centrum*, ktorý minimalizuje funkciu:

$$dist = \sum_{i=1}^{atr} (vector[i] - centrum[i])^2 \tag{8}$$

kde *atr* je počet atribútov (termov). V algoritmoch typu SOM je dôležité, ako učíme susedné neuróny. V klasickej dvojrozmernej mriežke je susedstvo jednoznačné, ak však pridávame samostatné neuróny, susedstvo je komplikovanejšie. Použili sme absolútnu vzdialenosť pre nájdenie k najbližších susedov, kde k je parameter susednosti. V snahe zachovať výhody zobrazenia na dvojrozmernej mriežke sme sa rozhodli indexovať neuróny tak, ako by sme v skutočnosti vkladali celé bloky neurónov, preto sa na výslednej mape objavia prázdne políčka, na mieste ktorých skutočne nie je alokovaný žiadny neurón. Dosiahneme tak kompromis medzi dvojrozmernou mriežkou a nejakou ťažšie zobraziteľnou štruktúrou.

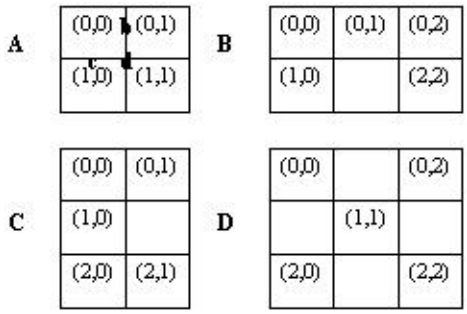
Potom sa adaptácia váh neurónu uskutočňuje na základe formuly:

$$c[i] = c[i] + learnrate \,.\, nFactor \,.\, (v[i] - c[i]) \,, \tag{9}$$

kde c je vektor váh centra neurónu, v je vstupný vektor a $nFactor$ je miera adaptácie susedov. Ak c je víťaz, $nFactor$ je 1. Inak, ak *neighbourhood* je 1, potom je použitý len jeden sused okrem víťaza, pričom má $nFactor = 0.5$. Ak je *neighbourhood* väčší ako 1, susedia sú usporiadaný podľa vzdialenosti, najbližší dostáva hodnotu 0.9, najvzdialenejší 0.1 . Hodnota pre ostatné neuróny sa vypočíta ako lineárna aproximácia medzi týmito prípadmi podľa vzdialenosti. Počas učenia sa taktiež znižuje parameter *learnrate*. Ak je pridaný nový neurón, hodnota parametra je opäť nastavená na maximálnu hodnotu a sieť sa preučí. Každá mapa má vlastnú hodnotu *learnrate*.

4.3 Pridávanie neurónov a máp, integrácia systému

Pre popis princípu vkladania neurónov sme použili vzťah "nestability" medzi dvoma neurónmi mapy - nestabilita hrany. Reprezentuje ju počet tých vstupov (vstupných vektorov prislúchajúcich tejto dvojici neurónov), pre ktoré je stred medzi nimi (aritmetický priemer váh) lepšou alternatívou, pričom každý vstup môže byť nestabilným len pre jednu hranu v mape (pre tú, ktorá má k nemu najbližší stred). Potom pridávame neurón nasledovne. Nájdeme opäť najvariabilnejší neurón (chybový) a k nemu jeho najnestabilnejšiu hranu. Potom na miesto stredu hrany položíme nový neurón, inicializačný vektor váh je potom práve tento stred. Posledný problém je indexovanie neurónov na mape, základný princíp preindexovania je na obrázku Obr.1.



Obr. 1. Reindexácia neurónov pri vkladaní neurónu - neurón (0,0) je chybový neurón a ďalšie tri neuróny predstavujú možnosti suseda s najnestabilnejšiou hranou b,c alebo d. Časť A reprezentuje mapu pred vložením nového neurónu. Časti B,C,D sú mapy po preindexovaní. Ekvivalentné hrany sú uvedené malými písmenami. Indexy neurónov sa menia tak ako v starom modeli (ako pridanie riadku, či stĺpca), avšak existuje len jeden nový neurón v riadku(C), stĺpci (B), alebo v oboch smeroch (D, môžeme pridávať aj po diagonále).

Základná diferencia medzi klasickým a modifikovaným algoritmom GHSOM pri vytváraní podmáp je v inicializácii neurónov novej podmapy. Proces expanzie začína výberom kandidátov na expanziu. Pre každý expandovaný neurón sa vytvorí a nalinkuje nová mapa 2×2. Neuróny podmapy sa však inicializujú príkladmi expandovaného neurónu. Nájdeme na nadradenej mape štyroch najbližších susedov (neuróny) a ku každému z nich nájdeme k nemu najbližší vstup expandovaného neurónu. Tieto príklady použijeme pre inicializáciu novej podmapy.

Poslednou časťou modifikácií bol prístup k predspracovaniu dokumentov v slovenčine a integrácia so systémom pre spracovanie a reprezentáciu textových dokumentov. V tomto prípade boli použité importované balíky *jbowl2* (systém pre efektívnu reprezentáciu dokumentov, autorom systému je jeden z autorov článku) a *dopmola* (lingvistické analýza slovenského textu, extrakcia fráz). Podrobnejšie informácie k balíku *dopmola* je možné nájsť v [7].

5 Experimenty

Prvou testovacou množinou bola množina krátkych správ zo servera *cassovia*. Jednotlivé súbory obsahovali maximálne 5 viet, zvyčajne jednu, dve. Množina obsahovala 356 dokumentov. Druhou množinou bolo 2015 článkov denníka *Sme*.

5.1 Predspracovanie

Predspracovanie bolo realizované použitím importovaných balíkov - *jbowl2* a *dopmola*. Po indexácii dokumentov boli orezané termy, ktoré sa nezmestili do medzí pre minimálnu a maximálnu dokumentovú frekvenciu ich výskytu. Pre

dokumenty *cassovia* sa počet použitých termov pohyboval medzi 150 a 700, pre články zo *Sme* medzi 300 a 2000.

5.2 Cassovia

Parameter pre maximálny počet neurónov mapy sa pohyboval medzi 8 a 12, $minExpandVectorCount$ medzi 15 a 20, τ_1 medzi 0,8 a 1,2 and parameter τ_2 medzi 0,2 a 0,5. Počet vrstiev, počet neurónov vrchnej mapy a hĺbka hierarchia závisí práve na týchto parametroch. Najvýraznejší vplyv na tvar hierarchie mali prvé dva parametre. Počet neurónov vrchnej mapy dosiahol maximálny povolený počet (je to pochopiteľné, na najvyššej úrovni sa príklady ešte nedokázali rozdeliť tak, aby splnili požiadavku pre variabilitu). Príklad vygenerovanej vrchnej mapy hierarchie je na obrázku Obr.2.

N: 0-0 V: 15 volba, komunálny, kandidát			N: 0-3 V: 13 víťaz, stať, čas, maratón	N: 0-4 V: 14 súd, navštíviť, ústavný
N: 1-0 V: 81 námestník, príležitosť, privítať			N: 1-3 V: 48 slovensko, veľvyslanec, návšteva, rokovať	N: 1-4 V: 28 predseda, vláda
				N: 2-4 V: 71 navštíviť, prezident, maďarský
	N: 3-1 V: 25 spolupráca, dohoda, podpísať, oblasť			
		N: 4-2 V: 29 zastupiteľstvo, poslanec, slávnostný		
N: 5-0 V: 14 poľský			N: 5-3 V: 10 navštíviť, magistrát	N: 5-4 V: 13 európsky, únia

Obr. 2. Príklad mapy na najvyššej úrovni hierarchie máp kolekcie *Cassovia*. Na mape sa objavuje niekoľko zaujímavých zhlukov, reagujúcich na návštevy prezidentov a veľvyslancov na magistráte, ústavnom súde, prípadne v prezidentskej kancelárii. Neurón 3-1 sa týka dohôd o spolupráci medzi rôznymi subjektami, ako technická univerzita, U.S.Steel a podobne. Neurón 0-0 patrí správam o komunálnych voľbách, neurón 0-3 má pre zmenu športový obsah, napríklad o maratóne, či tenisovom turnaji. Neuróny s väčším počtom príkladov sa rozdeľujú na ďalších úrovniach na menšie podskupiny.

5.3 Sme

Zaujímavou doménou pre zhlukovanie sú články denníka *Sme*. Použité parametre boli podobné ako pri *cassovii*, prípadné zmeny zohľadňovali rozdielnu mohutnosť skúmaných množín. Použili sme 2016 dokumentov. Pozitívny bol fakt, že približne 75% zhlukov malo veľmi dobrý a jasný popis (subjektíve ohodnotenie), napriek inicializačnej mape minimálnej veľkosti sa táto rozrastala bez efektu rozbíjania kompaktnosti nesúvisiacich častí mapy. Príklad vrchnej mapy hierarchie je na obrázku Obr.3.

N 0-0 V: 186 gól, strelec, diváK, tréner, majstrovstvo, hokej			N: 0-3 V: 149 zápas, hráč, diváK, bod, gól, liga		N: 0-5 V: 610 svetový, klub, život, dielo, planéta
		N: 1-2 V: 50 kolo, extraliga, bod			
	N: 2-1 V: 363 rusko, štát, člen, vláda, premiér, americký, rokovanie, dohoda				
				N:3-4 V: 24 film, kino	
N: 4-2 V: 29 strana, štát, európsky, únia, minister			N: 2-4 V: 71 milión, banka, úver, predaj, miliarda		N: 2-4 V: 71 noviny, šport, správa, film
N: 5-0 V: 14 zomrieť, narodiť			N: 5-3 V: 10 vojak, armáda, útok, izrael, rusko, lietadlo, teroristický, afganistan		N: 5-4 V: 13 rádio, hudba, hudobný, film

Obr. 3. Príklad mapy na najvyššej úrovni hierarchie máp kolekcie článkov *Sme*. Neuróny 0-0, 1-2 a 0-3 popisujú rôzne články týkajúce sa športu. Neuróny 4-0 a 2-1 sa sústreďujú na politiku, a to hlavne na politiku súvisiacu so Slovenskou republikou (vnútorná politika, zahraničná v rámci EÚ, či NATO, a podobne). Neuróny 3-4, 4-5 a 5-5 súvisia s programom kín, divadiel a rádií, neurón 4-3 zase s ekonomikou (napr. kurzy mien). Neurón 5-3 popisuje udalosti, ktoré sa vo svete vo všeobecnosti najviac objavovali, teroristické útoky a reakcie na nich.

6 Záver

Výhodou použitia uvedených modifikácií je precíznejšie pridávanie neurónov a schopnosť udržiavať zhluky na mape čo možno najkompaktnejšie. Ďalšou výhodou algoritmu je proces inicializácie založený na príkladoch expandovaného neurónu, ako aj zvýšená rýchlosť spracovania (vďaka integrácii so systémom *jbowl2*). Použitie balíka *dopmola* umožnilo získať kompaktnejšiu a čistejšiu vektorovú reprezentáciu (lepší stemming, extrakcia fráz), ako aj zlepšiť vďaka tomu popisy zhlukov extrahované pomocou metódy LabelSOM.

Problémom zostáva zatiaľ malá možnosť porovnania výsledkov s inými prácami popisujúcimi zhlukovanie slovenských dokumentov, ako aj kvantifikácia výsledkov zhlukovania na množine klasifikovaných dokumentov, ktorú sme v čase tvorby experimentov nemali k dispozícii. Problémom je predspracovanie slovenčiny, to bol aj dôvod použitia zdanlivo malého počtu dokumentov. Použitá implementácia pomocou externého volania programu *Ispell* je neefektívna, vyžaduje si to použiť vlastný, čo najlepší slovník, ktorý nie je voľne dostupný v elektronickej forme. Samotná metóda je z hľadiska škálovania použiteľná aj s počtom dokumentov o niekoľko rádov vyšším, čo napríklad dokumentuje [8].

Kvalita výstupných máp by mohla byť vyššia použitím lepšej adaptácie susedov, prípadne by sme mohli použiť určitú optimalizáciu (orezanie máp) so snahou dostať menšie a kompaktnejšie mapy. Ďalšou možnosťou je snažiť sa pracovať so všeobecnejšími architektúrami algoritmov na báze SOM (grafy) a snažiť sa ich čo najvhodnejšie zobraziť. Samozrejme, v prípade práce s textami v slovenčine, je dôležitá neustále aj snaha o čo najlepšie predspracovanie na báze NLP metód.

Literatúra

1. Fritzke, F.: Growing Grid - a self-organizing network with constant neighbourhood and adaptive strength. In Neural Processing Letters **2(5)** (1995)
2. Merkl, D.: Explorations of text collections with Hierarchical Feature Maps. In Proceedings of International ACM SIGIR Conference in Information Retrieval, Philadelphia (1997)
3. Dittenbach, M., Merkl, D., Rauber, A.: Using growing hierarchical self-organizing maps for document classification. In Proceedings of ESANN, Bruges, (2000) 7–12
4. Rauber, A.: On the labeling of self-organizing maps. In Proceedings of IJCNN, Washington, (2000)
5. Oraja, T.: Organizácia množiny textových dokumentov pomocou zhlukovania. Diplomová práca, KKUI FEI TU, Košice (2002)
6. Butka, P.: Zhlukovanie textových dokumentov. Diplomová práca, KKUI FEI TU, Košice (2003)
7. Blicha, M.: Spracovanie textových dokumentov v slovenskom jazyku s využitím nástrojov lingvistickej analýzy pre účely ich zhlukovania a kategorizácie. Diplomová práca. KKUI FEI TU, Košice (2003)
8. Kohonen, T., et al.: Self organization of a massive document collection. In IEEE Transactions on Neural Networks 11(3):574-585 (2000)

Core functions and statistical inference

Zdeněk Fabián *Academy of Sciences of the Czech republic,*

Institute of Computer Science,
Prague, Czech Republic
zdenek@cs.cas.cz

Abstract. In systems of transformed distributions with a fixed form of parametric space we introduce core functions. New characteristics of distributions based on the core functions can be used for new inference procedures in the point estimation and testing of hypotheses.

1 Introduction

In the last two workshops ITAT, a distance in the sample space ([2]) and a measure of information contained in a single observation ([3]) of continuous random variables were introduced. Both of them are based on the key concept, the core function.

In this paper we re-introduce the core function somewhat more carefully and explain why it seems to us to be an extremely useful concept.

2 Core function

Denote by $\Pi_{(a,b)}$ the class of continuous and, for the sake of simplicity, unimodal distributions on the real line R with differentiable densities supported by $(a, b) \subseteq R$. Let Y be a random variable with distribution $G \in \Pi_R$. Its density g has a mode (a maximum at an internal point of the support interval), the coordinate of which is interpreted as the centre of mass y^* of the distribution. The score function is function

$$S_G(y) = -\frac{g'(y)}{g(y)}, \qquad y \in R, \tag{1}$$

giving a relative rate at which the density g changes at point y. Apparently,

$$S_G(y^*) = 0. \tag{2}$$

A quite different situation occurs when $(a, b) \neq R$. Density f of a distribution $F \in \Pi_{(a,b)}$ may not have a mode (but for instance a supremum at

a or b) and the score function is queerish. It has turned out that the role of the score function plays for these distributions another function, which was called a core function. Let $\Psi_{(a,b)}$ be a set of monotone differentiable mappings from (a,b) onto R. For any $\psi \in \Psi_{(a,b)}$, the transformed random variable $X = \psi^{-1}(Y)$ has distribution $F \in \Pi_{(a,b)}$ with distribution function $F(x) = G(\psi(x))$ and density

$$f(x) = g(\psi(x))\psi'(x) \tag{3}$$

where $\psi'(x) = d\psi(x)/dx$. *Core function* of such transformed distribution is defined in [2] as

$$T_F(x) = S_G(\psi(x)), \tag{4}$$

i.e., as an image of the score function S_G of the "prototype" distribution G utilizing the mapping ψ. The function can be expressed without reference to the score function of its prototype by means of f and ψ only, according to [2], Theorem 1,

$$T_F(x) = -\frac{1}{f(x)}\frac{d}{dx}\left(\frac{1}{\psi'(x)}f(x)\right). \tag{5}$$

By using (4) and (2) it holds

$$T_F(\psi^{-1}(y^*)) = 0. \tag{6}$$

3 Systems of transformed distributions

Suppose we have a system $\mathcal{G}$ of some distributions $G \in \Pi_R$. Any $\psi \in \Psi_{(a,b)}$ generates system $\mathcal{F}_\psi = \{G \in \mathcal{G}, F = G\psi^{-1}\}$ of transformed (induced) distributions with support (a,b), densities (3) and core functions (4). The repertory of suitable mappings is limited. Mappings which are for different purposes used in statistics are listed in Table 1, together with formulas for the corresponding core functions T_F derived from (5) and expressed by the means of the "false score function" $U = -f'(x)/f(x)$ (log means the natural logarithm).

Table 1. Mappings $\psi : (a,b) \to R$ and the corresponding core function formulas

	(a,b)	$\psi(x)$	$\psi'(x)$	$T_F(x)$
1	$(0,1)$	$\log\frac{x}{1-x}$	$\frac{1}{x(1-x)}$	$2x + x(1-x)U - 1$
2	$(0,1)$	$-\log(-\log x)$	$\frac{-1}{x\log x}$	$1 + \log x(1 - xU)$
3	$(-\frac{\pi}{2}, \frac{\pi}{2})$	$\tan x$	$\frac{1}{\cos^2 x}$	$\sin 2x + U\cos^2 x$
4	$(0,\infty)$	$\log x$	$\frac{1}{x}$	$xU - 1$
5	R	$\sinh x$	$\cosh x$	$\cosh^{-1} x(U - \tanh x)$

Mappings 1, 4, and 5 are transformations used for purposes similar as ours by Johnson, mappings 1 and 2 are used as link functions of generalized linear models and mapping 3 is responsible for distributions expressed by trigonometric functions in the Burr system of distributions (we refer to [6] and [5]). All model distributions with support $(0, \infty)$ mentioned in statistical applications are members of system $\mathcal{F}_{\log}$ generated by mapping 4.

4 Core function of parametric distributions

Let $F \in \Pi_{(a,b)}, \Theta \subseteq R^m$ and $\mathcal{F} = \{F_\theta : \theta \in \Theta\}$ be a parametric family with parent F. Denote the density of F_θ by f_θ. A function of x and θ

$$s_{\theta_j}(x; \theta) = \frac{\partial}{\partial \theta_j} \log f_\theta(x)$$

is known as the score for parameter θ_j and $I(\theta) = Es_\theta^2$, where $s_\theta = (s_{\theta_1}, ..., s_{\theta_m})$, is the Fisher information matrix.

Let us fix the parametric vector of distributions from Π_R in the form $\theta_R = (\mu, \sigma, \lambda)$ where μ is the location, σ the scale and λ the (possibly $p-$dimensional vector of) shape parameters. Thus, $\theta_R \in \Theta_R$ where

$$\Theta_R = R \times (0, \infty) \times (0, \infty)^p.$$

By introducing a *pivotal function*

$$u_R = u_{\mu,\sigma}(y) = \frac{y - \mu}{\sigma}, \qquad y \in R,$$

the members of parametric family $\{G_{\theta_R}, \theta_R \in \Theta_R\}$ can be written in the form $G_{\theta_R}(y) = G_\lambda(u_R)$, and the corresponding densities as $g_{\theta_R}(y) = \sigma^{-1} g_\lambda(u_R)$.

Parametric distributions supported by arbitrary interval $(a, b) \subseteq R$ are constructed as follows. Let $G_\lambda(u_R) \in \Pi_R$ be a prototype parametric distribution and $\psi \in \Psi_{(a,b)}$. Set

$$\tau = \psi^{-1}(\mu) \tag{7}$$

so that $\tau \in (a, b)$ is the image of the location of the prototype distribution. Let us fix the parametric vector as $\theta = (\tau, \sigma, \lambda) \in \Theta$ where

$$\Theta = (a, b) \times (0, \infty) \times (0, \infty)^p$$

and define the *pivotal variable on* (a, b) by

$$u_\psi = u_{\psi(\tau),\sigma}(x) = \frac{\psi(x) - \psi(\tau)}{\sigma}, \qquad x \in (a, b). \tag{8}$$

Example Pivotal variable of system $(\mathcal{F}_{\log})_\theta$ on $(0, \infty)$ is

$$u_{\log} = \frac{\log x - \log \tau}{\sigma} = \log \left(\frac{x}{\tau}\right)^{1/\sigma}.$$

Proposition 1 Density of distribution $F_\theta(x) = G_\lambda(u_\psi)$ is

$$f_\theta(x) = \sigma^{-1} g_\lambda(u_\psi)\psi'(x).$$

Proof. Clear.

Definition 1 Let $(a,b) \subseteq R$, $\psi \in \Psi_{(a,b)}$ and $\theta = (\mu, \sigma, \lambda)$. Let $\mathcal{G} = \{G_{\theta_R}, \theta_R \in \Theta_R\}$ be a system of (prototype) distributions. A set

$$(\mathcal{F}_\psi)_\theta = \{F_\theta : G_\theta \in \mathcal{G}, F_\theta(x) = G_\lambda(u_\psi), \theta \in \Theta\},$$

where u_ψ is given by (8) and τ by (7), will be called an *induced system of distributions* on (a,b) and τ an *induced location.* A *core function* of distribution $F_\theta \in (\mathcal{F}_\psi)_\theta$ is

$$T_{F_\theta}(x) = S_{G_\lambda}(u_\psi) = -\frac{g'_\lambda(u_\psi)}{g_\lambda(u_\psi)}.$$

The core function is thus defined as the score function of the distribution expressed in terms of the pivotal variable. In [3] we have seen that

$$s_\tau(x;\theta) = \frac{\psi'(\tau)}{\sigma} T_{F_\theta}(x), \tag{9}$$

i.e., that the *core function is the inner part of the score for induced location.* By the use of(9), the core function of prototype distribution G_{θ_R} is $T_G(x) = \sigma s_\tau(x;\theta)$.

Example Core function of the normal distribution is $T_G(y) = u_R$.

5 New characteristics of distributions based on core functions

Within a system $(\mathcal{F}_\psi)_\theta$, the core function T_F of any distribution F is uniquely defined by (5). Let us list some characteristics of F based on it. For the sake of brevity we do not explicitly express (if not necessary) possible shape parameters.

Core moments are defined in [1] as

$$\mathcal{M}_k(F) = E_f T_F^k = \int_{(a,b)} T_F^k(x) f(x)\, dx, \qquad k = 1, 2, ...$$

i.e., as moments of the core function. It has appeared that the core function "match the density" and core moments exist for any $F \in (\mathcal{F}_\psi)_\theta$.

Since

$$\int_a^b T_{F_\theta}(x) f_\theta(x) dx = \int_{-\infty}^{\infty} S_G(u_\psi) g(u_\psi)\, du_\psi = 0,$$

$\mathcal{M}_1 = 0$. Core moments are thus the central moments around the point $x = \tau$. Moreover, core moments of the currently used simple model distributions

are simple functions of parameters (the usual moments are often expressed by means of non-elementary functions of parameters).

Information function i_F of distribution F was introduced in [3] as a squared core function, $i_F(x) = T_F^2(x)$. In a parametric case we have to set, according to (9), $i_F(x) = \left(\frac{\psi'(\tau)}{\sigma}\right)^2 T_F^2(x)$, since the mean value

$$I(\tau) = E_\theta i_F = \left(\frac{\psi'(\tau)}{\sigma}\right)^2 \mathcal{M}_2(F) \tag{10}$$

is the Fisher information for parameter τ. (10) can be interpreted as the mean information of distribution F.

The most popular characteristics of continuous random variables are the mean (a centre of mass) $m = \int_a^b xf(x)dx$ and the variance (a measure of dispersion) $s^2 = \int_a^b (x-m)^2 f(x)dx$. Unfortunately, they do not exist for many distributions since the integrals do not converge. Instead of the couple (m, s^2), distributions can be characterized by a couple (τ, V) where τ is the induced location (due to (6) a "core center of mass") and

$$V = I^{-1} \tag{11}$$

a "*core variance*", the reciprocal value of the mean information. Some examples are given in Table 2.

Table 2. Core center of mass and core variance of some distributions

Name	$f(x)$	$T_F(x)$	τ	V
Normal	$\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$	$\frac{x-\mu}{\sigma}$	μ	σ^2
Cauchy	$\frac{1}{\pi(1+(\frac{x-\mu}{\sigma})^2)}$	$\frac{2(\frac{x-\mu}{\sigma})}{1+(\frac{x-\mu}{\sigma})^2}$	μ	$2\sigma^2$
Weibull	$\frac{\beta}{x}(\frac{x}{\tau})^\beta e^{-(\frac{x}{\tau})^\beta}$	$(\frac{x}{\tau})^\beta - 1$	τ	τ^2/β^2
Log-logist.	$\frac{\beta}{x}\frac{(\frac{x}{\tau})^\beta}{(1+(\frac{x}{\tau})^\beta)^2}$	$\frac{(\frac{x}{\tau})^\beta - 1}{(\frac{x}{\tau})^\beta + 1}$	τ	$3\tau^2/\beta^2$
Gamma	$\frac{\gamma^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\gamma x}$	$\gamma x - \alpha$	α/γ	γ^2/α
Beta	$\frac{1}{B(p,q)} x^{p-1}(1-x)^{q-1}$	$(p+q)x - p$	$\frac{p}{p+q}$	$\frac{p}{q}\frac{(p+q)^2}{(p+q+1)}$

Core distance $d_T(x_1, x_2)$ of points x_1, x_2 *in the sample space* (a, b) of distribution F, suggested in [2], is

$$d_T(x_1, x_2) = \frac{1}{\sqrt{\mathcal{M}_2(F)}} |T_F(u_2) - T_F(u_1)|. \tag{12}$$

Core divergence $2D_T(F_1, F_2)$, as a distance of distributions $F_1, F_2 \in (\mathcal{F}_\psi)_\theta$, was introduced in [5] as a normed mean square distance of the corresponding core functions

$$2D_T(F_1, F_2) = \frac{1}{\mathcal{M}_2(F_1)} \int_a^b [T_{F_2}(u_2) - T_{F_1}(u_1)]^2 f_1(u_1)\, du_1. \tag{13}$$

Both distances $d_T(x_1, x_2)$ and $D_T(F_1, F_2)$ are in the cases of distributions with unbounded core functions unbounded, and in the cases of distributions with bounded core functions are bounded. Thus, they are not sensitive to the outlying values if the assumed model exhibits high probability of their occurrence and are sensitive if this probability is low. This property, trivial for the former distance, is a result of computations ([5]) for the latter one and indicates an "inherent robustness" of the core divergence.

Generally, in cases of distributions with a linear core function (in Table 2 the gamma and beta distributions), the center of mass is identical with the mean value. In the case of the normal distribution is couple (τ, V) identical with (μ, σ^2).

6 Some inference procedures

Let $(x_1, ..., x_n)$ be the observed values thought as outcomes of random variables $(X_1, ..., X_n)$ with unknown true distribution $F_0 = F_{\tau_0,\sigma_0}$. The true distribution is approximated by some $F \in (\mathcal{F}_\psi)_\theta$ (we thus suppose that ψ is known and families of the system reparametrized to have the induced location parameter). As before, we do not express λ explicitly. Denote

$$u_i = u_i(\tau, \sigma) = \frac{\psi(x_i) - \psi(\tau)}{\sigma}, \qquad i = 1, ..., n$$

the "latent values" of the pivotal variable. The values (functions of λ) $\mathcal{M}_k(F) = E_f T_F^k$ exist, so that they are supposed to be known.

6.1 Point estimation

Maximum likelihood estimates $(\hat{\tau}_L, \hat{\sigma}_L)$ of τ_0 and σ_0 are solutions of the system of likelihood equations, which can be expressed in terms of the core function as

$$\frac{1}{n}\sum_{i=1}^{n} T_F(u_i) = 0 \tag{14}$$

$$\frac{1}{n}\sum_{i=1}^{n} [u_i T_F(u_i) - 1] = 0. \tag{15}$$

By the substitution principle, the natural estimators of core moments are the core moments of the sample distribution function. Core moment estimates $(\hat{\tau}_C, \hat{\sigma}_C)$ are the thus solutions of system

$$\frac{1}{n}\sum_{i=1}^{n} T_F^k(u_i) = \mathcal{M}_k(F), \qquad k = 1, 2.$$

The estimates are consistent and asymptotically normal as well as the former ones ([1]), their asymptotic relative efficiencies are near to one if the core

function is bounded (this can be done artificially by methods of the robust statistics).

Since $\mathcal{M}_1 = 0$, in the case $m = 1$ and $\theta = \tau$ are both τ_C and τ_L identical. Let us denote them by $\hat{\tau}$. In Table 3, explicit formulas for $\hat{\tau}$ of some simple distributions are given.

Table 3. Estimates of the centre of mass of some distributions

Distribution	$f_(x)$	$T_F(x)$	$\hat{\tau}$
exponential	$\frac{1}{\tau}e^{-x/\tau}$	$x/\tau - 1$	$\frac{1}{n}\sum x_i$
lognormal	$\frac{1}{\sqrt{2\pi}x}e^{-\frac{1}{2}\log^2(x/\tau)}$	$\log(x/\tau)$	$\sqrt[n]{x_1 ... x_n}$
Frechet	$\frac{\tau}{x^2}e^{-\tau/x}$	$1 - \tau/x$	$n\left(\sum 1/x_i\right)^{-1}$

According the table, the geometric and harmonic means are thus the estimates of the centre of mass of the lognormal and Frechet distributions, respectively. By (10) and (11), the sample core variation of distribution F in system $(\mathcal{F}_{\log})_\theta$ is

$$\hat{V} = \frac{1}{M_2(F_{\hat{\theta}})}\hat{\tau}^2\hat{\sigma}^2. \tag{16}$$

It should be emphasized that $\hat{\sigma}$ is not the square root of the sample variance, but an consistent estimate of the scale parameter of the underlying distribution.

6.2 Testing of hypotheses

Let us test the hypothesis $H_0 : \theta = \theta_0$ against the alternative $H_1 : \theta \neq \theta_0$. H_0 is rejected in favor of H_1 when distance $d(\hat{\theta}, \theta_0) \geq c_n$ ($\hat{\theta}$ is usually the maximum likelihood estimate of θ). Denoting by P the probability, value c_n is to be determined from condition

$$P\{d(\hat{\theta}, \theta_0) \geq c_n\} = \alpha, \tag{17}$$

where α is a preassigned level of significance. Thus, one have to know the sampling distribution of $d(\hat{\theta}, \theta_0)$.

Consider first the case of single parameter $\theta = \tau$. The simplest classical distance is the Wald distance

$$d_W^2(\hat{\tau}, \tau_0) = (\hat{\tau} - \tau_0)^2 I(\hat{\theta}),$$

originated from the second term of the Taylor's expansion of the log-likelihood. However, this formula is oversimplified and renders symmetric confidence intervals. It may be a good approximation for large samples ($\hat{\tau}$ is asymptotically normal), but it is certainly not realistic in the cases of small samples. Other used "classical" distances are the likelihood ratio and the Rao distance, $d_R^2(\hat{\tau}, \tau_0) = (\sum_{i=1}^n s_\tau(x_i; \tau_0))^2 / I_\tau(\tau_0)$. Using (9), (10) and relation

$\sum T_F(u_i(\hat{\tau}_n, 1)) = 0$ following from (14), the Rao distance can be expressed as a squared sum of differences of core functions

$$n\mathcal{M}_2(F_{\tau_0})d_R^2(\hat{\tau}, \tau_0) = \left(\sum_{i=1}^{n}[T_F(u_i(\tau_0, 1)) - T_F(u_i(\hat{\tau}, 1))]\right)^2. \tag{18}$$

We suggest to use for testing of hypotheses the core distance of $\hat{\tau}$ and τ_0 in the sample space of F_{θ_0}. Writing T instead of T_F, it holds that $T_{\tau_0}(\psi(\tau_0)) = 0$ so that by (12)

$$d_T^2(\hat{\tau}, \tau_0) = \frac{1}{\mathcal{M}_2(\tau_0)}[T_{\tau_0}(\psi(\hat{\tau}))]^2. \tag{19}$$

This formula is often nearly as simple as the Wald distance (it equals the Wald distance in the cases of distributions with a linear core function).

If $m > 1$, distance (19) cannot be used, but it is possible to use the core divergence, either in form (13) or in the "sample form"

$$2n\mathcal{M}_2(F_{\theta_0})D_T(\hat{\theta}, \theta_0) = \sum_{i=1}^{n}\left[T(u_i(\hat{\tau}, \hat{\sigma})) - T(u_i(\tau_0, \sigma_0))\right]^2, \tag{20}$$

which seems to be better than the Rao's (18) since it does not fail in the cases of distributions with non-monotonous core functions. A comparison of the core distances with the Kullback-Leibler distances (representing the class of so called theoretic-informational distances) in some parametric families can be found in [5].

7 Example

Consider Lomax distribution with support $(0, \infty)$ and density

$$f_\alpha(x) = \frac{\alpha}{(1+x)^{1+\alpha}} \tag{21}$$

with single shape parameter $\alpha \in (0, \infty)$. Its absolute moments are

$$m_k = \int_0^\infty \frac{\alpha x^k}{(1+x)^{1+\alpha}}\, dx = \frac{\Gamma(k+1)}{\Gamma(\alpha+1)}\Gamma(\alpha - k),$$

where Γ is the gamma function; they exist only if $k < \alpha$. Lomax distribution with parameter $\alpha = 1$ has neither a mean nor a variance. The score for α is $s_\alpha(x) = 1/\alpha - \log(x+1)$ so that the maximum likelihood estimate of α is

$$\hat{\alpha}_L = \frac{n}{\sum_{i=1}^{n}\log(1+x_i)}.$$

Formula (21) can be rewritten into

$$f_\alpha(x) = \frac{1}{x}\frac{\alpha^{1+\alpha}\alpha x}{(\alpha + \alpha x)^{1+\alpha}} = \frac{1}{x}\frac{\alpha^{1+\alpha}e^u}{(e^u + \alpha)^{1+\alpha}}$$

where $u = \log(x/\tau)$ is the pivotal function and $\tau = 1/\alpha$ (Lomax density can thus be understood as having two parameters, the induced location $\tau = 1/\alpha$ and the shape parameter α.) By (1) and (4), the core function is

$$T_{F_\alpha}(x) = \alpha \frac{e^u - 1}{e^u + \alpha} = \frac{\alpha x - 1}{x + 1}.$$

The first two core moments are $\mathcal{M}_1 = 0$ and $\mathcal{M}_2 = \alpha/(\alpha + 2)$ (mean information of the Lomax distribution is thus equal to $\alpha^3/(\alpha + 2)$). The core moment estimate is

$$\hat{\alpha}_C = \frac{\sum_{i=1}^n 1/(x_i + 1)}{\sum_{i=1}^n x_i/(x_i + 1)}.$$

In a simulation experiment, the samples of length n were generated and average values of both $\hat{\alpha}_L$ and $\hat{\alpha}_C$ were calculated for 1000 estimates. Some results are given in Table 4. Although the average values of standard deviations of core moment estimates (not printed) are a little higher than those of the maximum likelihood estimates (the asymptotic relative efficiency of $\alpha_{n,C}$ is $\alpha(\alpha + 2)/(\alpha + 1)^2) < 1$), the values $\hat{\alpha}_C$ are systematically nearer to the true value $\alpha = 2$. In the case of Lomax distribution, it was shown in [1] that the core moment estimates are better for data contaminated by outliers. It is apparent from Table 4 that it holds even for non-contamined data.

Table 4. Average values of estimates

α	n	$\bar{\alpha}_L$	$\bar{\alpha}_C$
2	15	2.133	2.100
	30	2.082	2.060
	60	2.035	2.025
	100	2.018	2.016

The reason for this surprising fact is the bounded core function of the Lomax distribution, which suppresses outliers produced by the data generating the process.

Setting $\alpha = 1$ and introducing the induced location parameter τ, we obtain a log-logistic distribution F_τ with density

$$f_\tau(x) = \frac{1}{\tau} \frac{1}{(1 + x/\tau)^2}.$$

Its core function (let us write T instead of T_F again) is $T_\tau(x) = \frac{x/\tau - 1}{x/\tau + 1}$ and $\mathcal{M}_2(\tau) = 1/3$. Having a random sample $(X_1, ..., X_n)$ from F_{τ_0}, one obtains an estimate of τ_0, the estimate of τ_0 is given by

$$\hat{\tau}: \qquad \sum_{i=1}^{n} \frac{x_i/\tau - 1}{x_i/\tau + 1} = 0.$$

Let us test hypothesis $H_0 : \hat{\tau} = \tau_0$. The Wald distance is $d^2_{\mathrm{W}}(\hat{\tau}, \tau_0) = \frac{1}{3}(\hat{\tau} - \tau_0)^2$ and the core distance

$$d^2_T(\hat{\tau}, \tau_0) = \frac{1}{M_2(F_{\tau_0})}(T_{\tau_0}(\hat{\tau}))^2 = 3\left(\frac{\hat{\tau} - \tau_0}{\hat{\tau} + \tau_0}\right)^2.$$

Both d_W and d_T are plotted in Fig. 1.

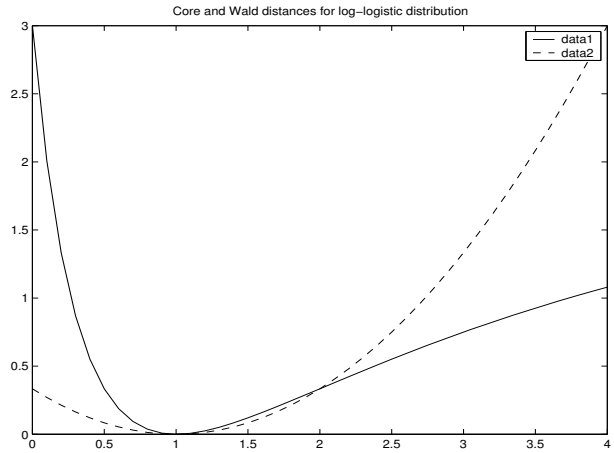


Fig.1 Core (1) and Wald (2) distances

Since $\hat{\tau}$ is asymptotically normal, d^2_W is distributed according to the chi-squared distribution and d^2_T according to the Fisher-Snedecor's. In both cases, one can easily find c_n in (17). It is apparent that the intersections of horizontal line $y = c_n$ with the core distance curve defines a more realistic region of the acceptance of H_0 than in the case of the Wald distance. Both the likelihood ratio and the Rao distance cannot be expressed in this case by means of simple formulas.

8 Remark

When considering an F supported by $(a, b) \neq R$, it is sometimes not clear to which system $(\mathcal{F}_\psi)_\theta$ it belongs. A solution to this problem is principally not unique. For instance, the uniform distribution is a member of any system with a finite interval support. On the other hand, the core function of the uniform distribution can be determined uniquely by a subsidiary requirement of its linearity (it uniquely specifies mapping 1 in Table 1). Further, we do not know how to use the theory in the cases of distributions with not

explicitly expressed densities and of distributions with support (a, b) where a or b is a parameter which is to be estimated.

Aknowledgements. The work was supported by grant IAA 1075403.

References

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

[2] Fabián, Z. (2002). Large relationship between probability measure and metric in the sample space. Proceedings ITAT'2001, 103-109.

[3] Fabián, Z. (2003). Information contained in an observed value. Proceedings ITAT'2002, 95-101.

[4] Fabián, Z. and Vajda, I. (2003). Core functions and core divergences of regular distributions. *Kybernetika*, **39**, 1, 29-42.

[5] Dobson, A. (2002). An introduction to generalized linear models. Chapman and Hall.

[6] Johnson, N.L., Kotz, S., Balakrishnan, N. (1994) Continuous univariate distributions 1, 2. Wiley, New York.

Vyhledávání častých frází pro generování uživatelských profilů

Petr Grolmus[1], Jiří Hynek[2] a Karel Ježek[1]

[1] Západočeská Univerzita, Univerzitní 8, 306 14 Plzeň, Česká Republika
[2] InSITE s.r.o., Rubešova 29, 326 00 Plzeň, Česká Republika

Abstrakt Systém ProGen (*Profile Generator*) je na Západočeské univerzitě v Plzni navržen a implementován s cílem zdokonalit vyhledávací služby pro vědeckovýzkumné účely. Předpokládá se jeho využití i v dalších oblastech (komerční sféra, vzdělávání, zábava). K tomu se využívá generování uživatelských profilů na základě dokumentů Internetu navštívených uživateli. Získání seznamu těchto dokumentů je podmíněno instalací vytvořeného paketového filtru na klientské stanici – tento způsob stírá morální problém, neboť uživatel není sledován bez jeho vědomí. Navštívené dokumenty jsou staženy off-line a z nich je vytvořen zájmový profil daného uživatele. K nalezení charakteristických frází z dokumentů je použit algoritmus *Suffix Tree Clustering*. Vytvořený uživatelský profil je použit ke splnění primárních cílů systému, kterými je: doporučování dokumentů na Internetu na základě nalezeného uživatelského profilu a vyhledávání doménových expertů.
Klíčová slova: generování profilu, zájmový profil, text mining, STC, suffix tree clustering, expert, vyhledávání, analýza sociálních sítí, doporučovací systém

1 Úvod

Systém ProGen (*profile generator*) je vyvíjen na Západočeské univerzitě v Plzni za účelem automatického generování uživatelského profilu. Nalezené uživatelské profily plánujeme využít pro naplnění dvou primárních cílů: doporučování dokumentů na Internetu a vyhledávání doménových expertů.

Systémy pro automatické generování uživatelských profilů nejsou ve světě elektronických dokumentů žádnou novinkou, nicméně nacházejí praktické uplatnění až v poslední době. Popularitu získávají zejména aplikace pro vyhledávání a doporučování expertů (*recommender systems*, *expert finding systems*). Doporučovací systémy lze obecně dělit na systémy založené na obsahu (*content-based*) a systémy tzv. kolaborativního filtrování. Systémy z první skupiny zjišťují podobnost mezi obsahem položek (např. článků, filmů apod.), které uživatel v minulosti označil za zajímavé a snaží se doporučovat jiné (dosud nenavštívené) položky s podobným obsahem. Využívá se zde algoritmů strojového učení, které klasifikují dostupné materiály jako zajímavé nebo naopak nezajímavé. Systémy z druhé skupiny naproti tomu nepředkládají doporučení na základě profilu uživatelových preferencí, ale podle podobností mezi profilem aktivního uživatele a

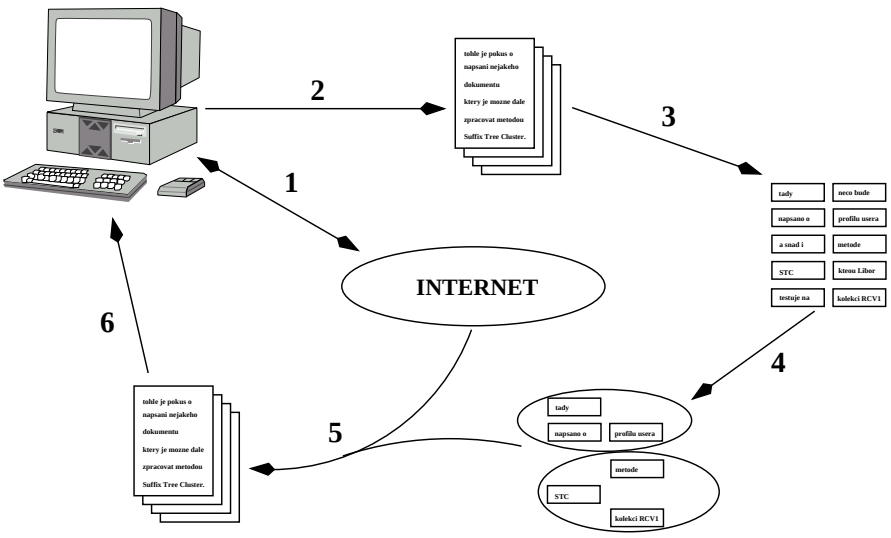
ostatních uživatelů registrovaných v systému. Oba přístupy se pak kombinují v hybridních doporučovacích systémech.

Jedním z prvních realizovaných systémů byl *HelpNet* (Maron a kol., 1986). Jeho odpovědí na žádost o informaci je seznam pracovníků setříděný podle jejich schopnosti zodpovědět dotaz. V poslední době se setkáváme s řadou dalších systémů, např. *Expert Finder* (Vivacqua, 1999) – hybridní doporučovací systém hledající odborníky v předdefinovaných komunitách uživatelů, *Expertise Recommender* (McDonald, Ackerman, 2000) – vyvinutý na University of California v Irvine, *RAAP* (Delgado, 2000) – hybridní doporučovací systém filtrující dokumenty a zajišťující on-line adaptabilitu uživatelských profilů nebo *XPERT-FINDER* (Sihn, Heeren, 2001) – analyzuje e-mailovou komunikaci uživatele a na jejím základě připravuje odpovídající znalostní profil.

Publikované postupy systémů pro automatické vytvoření profilu a generování shluků lze nalézt například v [1]–[4].

2 Popis systému

Struktura systému ProGen je znázorněna na obrázku 1.



Obrázek 1. Struktura systému

Prvním krokem při vytváření uživatelského profilu je sběr informací o uživatelem navštívených dokumentech v prostředí Internetu. Shromažďování těchto základních údajů lze realizovat několikerým způsobem, např. procházením logů WWW serveru (možné pouze v rámci intranetu) nebo proxy serveru (ne všichni

jej používají). Obě uvedené možnosti jsou zcela nevhodné pro univerzitní prostředí, neboť např. studenti nejsou v čase vázáni na konkrétní IP adresu. Převážně z tohoto důvodu systém ProGen využívá pro sběr údajů vlastní aplikaci, mající podobu paketového filtru, který filtruje požadavky uživatele na dokumenty v Internetu. Tato metoda navíc stírá morální problém, který by mohl vzejít z nedobrovolného monitorování uživatele na síti, neboť uživatel si je vědom funkce aplikace a má nad ní plnou kontrolu. Aplikaci může kdykoli vypnout, a zamezit tak poškození svého profilu, pokud by nechtěl, aby se některé dokumenty do jeho profilu začlenily. Nevýhody plynoucí z potřeby zapnutí monitorovací aplikace uživatelem považujeme vzhledem k její triviálnosti za nepodstatnou.

Po uplynutí konkrétního časového období, který závisí buď na počtu nasbíraných odkazů na dokumenty nebo velikosti prodlevy od poslední regenerace profilu uživatele, jsou dokumenty identifikované nasbíranými URL adresami načteny a upraveny pro další zpracování. V rámci lexikální analýzy jsou dokumenty převedeny do čistého textu (systém akceptuje dokumenty ve formátech PDF, postscript, MS Word a dalších). Z textu jsou následně vypuštěny nežádoucí nebo nadbytečné znaky (např. nevýznamová slova, značky HTML, interpunkční znaménka apod.) tak, aby každý dokument byl reprezentován jen proudem slov a čísel vzájemně oddělených znakem mezery. Vypouštěná nevýznamová slova dále slouží k automatické identifikaci jazyka zpracovávaného dokumentu. Vodítkem je počet nalezených (různých) nevýznamových slov z jednotlivých jazyků a také četnost jejich opakování. Zbylá slova jsou následně podrobena lemmatizaci prostřednictvím kombinace přesnější slovníkové metody (použit i-ispell) a Porterova algoritmu (odstraňování koncovek). Porterův algoritmus je aplikován pouze v případě slov, která nebyla nalezena ve slovníku. Určení jazyka v předchozím kroku nám umožňuje volit správný slovník a také množinu odtrhávaných koncovek pro Porterův algoritmus (obě metody jsou již ze své podstaty silně jazykově závislé, nicméně máme k dispozici slovníky pro všechny hlavní světové jazyky).

Tímto způsobem obdržíme „kolekci“ dokumentů příslušející jednomu uživateli. V této množině dokumentů v dalším kroku vytvoříme pomocí metody STC (*suffix tree clustering*) strom frází (viz např. [5]; metoda STC je popsána dále). Z charakteristických frází vybraných ze stromu STC jsou vytvořeny jejich shluky představující jednotlivé zájmy uživatele.

V nalezených shlucích představují charakteristické fráze klíčová slova, pomocí kterých můžeme vyhledat podobné dokumenty, např. pomocí vyhledávacího robota *Google*. Nalezené dokumenty ProGen následně porovná s uživatelským profilem a pokud považuje tyto dokumenty za relevantní a zároveň je uživatel dosud nenavštívil, doporučí je k prohlédnutí.

Po uživateli nežádáme ohodnocení systémem doporučených dokumentů, neboť uživatele tato činnost zpravidla obtěžuje a případný dotazník v praxi vyplňuje náhodným způsobem, což je kontraproduktivní. Je pouze na uživateli, aby svým chováním sám rozhodl o relevantnosti dokumentu. Pokud doporučené dokumenty navštíví, pak se tyto uplatní při příštím regenerování uživatelského profilu.

2.1 Suffix Tree Clustering

Algoritmus „*Suffix Tree*“ vznikl již v sedmdesátých letech, ale až koncem let devadesátých byl modifikován pro vyhledávání frází a nazván „*Suffix Tree Clustering*“. Publikované články uvádějí složitost metody *O(n)*, kde n představuje počet dokumentů v kolekci. Tuto lineární závislost prokazují experimenty na testovacích kolekcích. Výhodou metody je její nezávislost na pořadí zpracování dokumentů v kolekci. Další výhodou je jazyková nezávislost, která navíc umožňuje bezproblémové zpracování kolekcí obsahujících dokumenty v různých jazycích. V ideálním případě pak získáme zájmové shluky v těchto jazycích.

Postup vytváření stromu STC je velmi jednoduchý. Je nutné předem určit maximální délku hledaných frází. Délka hledané fráze mj. určuje hloubku vytvářeného stromu STC, čímž může významnou měrou ovlivnit paměťovou kapacitu nutnou pro výpočet. Délka fráze také představuje velikost „plovoucího okénka“ ve zpracovávaném textu.

Předpokládejme nyní, že maximální velikost hledané fráze je l. Strom STC vytvoříme takto:

1. Založení stromu (obsahuje pouze prázdný kořen);
2. Ze vstupního dokumentu načteme prvních l slov $w_1 \dots w_l$ (nebo méně, pokud jich již více není);
3. Jednotlivá vstupní slova w_i, kde $i = 1, \dots l$, zařadíme do stromu STC tak, že slovo w_i je umístěno do hloubky i tak, aby cesta od kořenu stromu do daného uzlu vedla přes uzly odpovídající slovům $w_1 \dots w_{i-1}$. Pro jednotlivé uzly (představují fráze) je nutné ukládat čísla dokumentů, ve kterých je daná fráze obsažena;
4. Ze vstupu vyřadíme první slovo a pokud je vstup neprázdný, postup opakujeme od kroku 2;
5. Načteme další dokument a pokračujeme bodem 2.

Např. pro následující tři věty (dokumenty) a maximální délku fráze $l = 3$:

1. Dědek tahá řepu.
2. Bába tahá dědka.
3. Bába také tahá řepu.

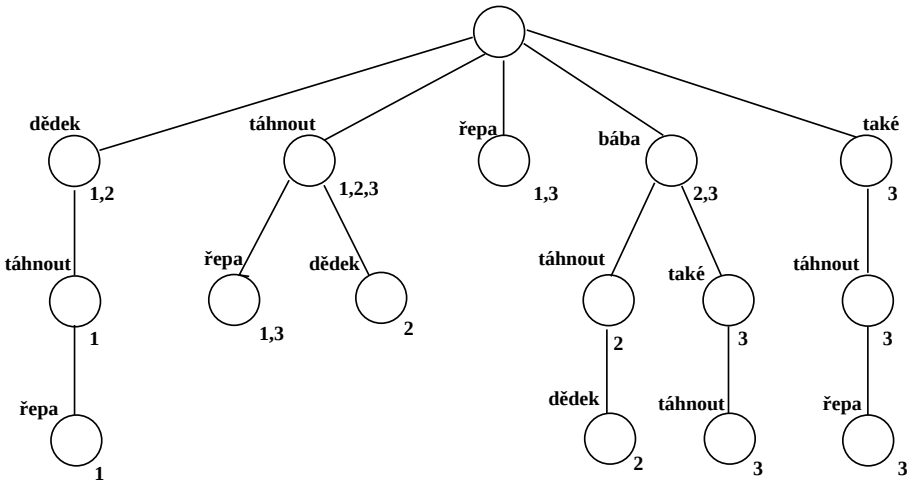
bude strom vygenerovaný po lemmatizaci mít tvar uvedený na obrázku 2.

2.2 Algoritmus generování shluků

Ve vzniklém stromu STC určíme váhu každé fráze takto:

$$w(f_i) = N(f_i) \times L(f_i)^2 \times \sum_{j=1}^{m} \left(tf_{ij} \times log \frac{m}{df_i} \right),$$

kde $L(f_i)$ je délka zpracovávané fráze, $N(f_i)$ počet výskytů fráze, tf_{ij} je počet výskytů fráze f_i v dokumentu j, m je celkový počet dokumentů v kolekci a df_i je



Obrázek 2. Příklad stromu STC

celkový počet dokumentů obsahujících frázi f_i. Umocnění $L(f_i)$ má za následek zvýhodnění méně často zastoupených delších frází.

Ze vzniklého stromu STC vybereme p charakteristických frází s největší vahou. Výsledný počet charakteristických frází určíme takto:

$$p = r \times \frac{s}{m} + \frac{m}{k},$$

kde m je počet zpracovávaných dokumentů, s je počet všech výskytů slov ve všech dokumentech, podíl $\frac{s}{m}$ tudíž představuje průměrný počet slov v jednom dokumentu. r určuje číselnou konstantu, která odpovídá procentuálnímu zastoupení průměrné délky dokumentu vůči celkovému počtu všech slov s. Tato konstanta umožňuje ovlivnit počet charakteristických frází v závislosti na průměrném počtu slov v dokumentech. Jelikož chceme zohlednit ve výběru také celkovou mohutnost množiny zpracovávaných dokumentů, k výsledku připočítáváme podíl $\frac{m}{k}$, zvyšující vybraný počet charakteristických frází právě o jednu na každých k dokumentů v kolekci. Výsledků, které jsou popisovány v další kapitole, bylo dosaženo pro nastavené hodnoty $r = 0.3$, $k = 50$ (tj. cca 200 charakteristických frází v následujících experimentech).

V takto získané množině charakteristických frází hledáme v dalším kroku podobnosti těchto frází pro vytvoření shluků frází, které představují jednotlivé zájmy uživatele. Nutno předeslat, že počet těchto shluků není předem dán, neboť nelze jakkoli odhadnout počet oblastí zájmu jednotlivých uživatelů.

Podobnost frází je měřena vždy pro dvě různé fráze na základě množiny dokumentů obsahujících obě fráze:

$$\left(\frac{|D_m \cap D_n|}{|D_m|} \geq \phi\right) \wedge \left(\frac{|D_m \cap D_n|}{|D_n|} \geq \phi\right)$$

kde ϕ určuje stanovenou minimální mez podobnosti dvou frází, D_n množinu dokumentů obsahujících frázi f_n a D_m množinu dokumentů obsahujících frázi f_m. V našich experimentech bylo dosaženo nejlepších výsledků při zvolené hodnotě $\phi = 0.5$.

Pokud si představíme jednotlivé fráze jako uzly a podobnost dvou frází jako hranu neorientovaného grafu, pak lze hledání shluků převést na klasickou úlohu z teorie grafů – hledání souvislých komponent grafu. Nalezené shluky představují jednotlivé oblasti zájmu uživatele.

3 Experimenty

Vzhledem ke skutečnosti, že v současné době nemáme statisticky významný počet testovacích uživatelů, prováděli jsme testování navrženého systému na následujících kolekcích – anglické Reuters Corpus Volume One (RCV1) a české kolekci poskytnuté Českou tiskovou kanceláří (ČTK). V experimentech s RCV1 jsme vytvářeli simulovaný uživatelský profil z tříd *sports*, *soft commodities*, *domestics politics* a *war*. V případě ČTK byl profil reprezentován pouze třídou *politika*.

	ČTK	RCV1
počet dokumentů	130 955	806 791
počet slov	29×10^6	193×10^6
průměrná délka dokumentu	159,0	88,4
délka nejkratšího dokumentu	10	10
délka nejdelšího dokumentu	5 721	3 996
průměrný počet tříd zařazení jednoho dokumentu	1,7	3,2
počet tříd	42	103

Tabulka 1. Srovnání testovacích kolekcí

Testování jsme prováděli pro obě kolekce shodně. Nejprve jsme z vybraných tříd kolekcí vybrali 2/3 dokumentů, ve kterých jsme za pomoci metody STC nalezli charakteristické fráze a v nich shluky. Tato množina dokumentů simuluje dokumenty navštívené uživatelem. Zbylou 1/3 dokumentů jsme smíchali se shodným počtem dokumentů z jiných tříd – tato množina pak simuluje dokumenty nalezené vyhledávacím robotem a obsahuje dokumenty relevantní i nerelevantní z pohledu uživatelského profilu.

Následně jsme pro každý kandidátní dokument na doporučení vypočítali jeho podobnost s uživatelským profilem. K porovnání byl vytvořen vektor frází jednotlivých shluků a pro každý dokument byl tento vektor naplněn hodnotami odpovídajícími počtu výskytů dané fráze v porovnávaném dokumentu (tedy i 0). Z takto určeného vektoru lze následně určit podobnost dokumentu D se shlukem profilu S_i pomocí kosinové míry:

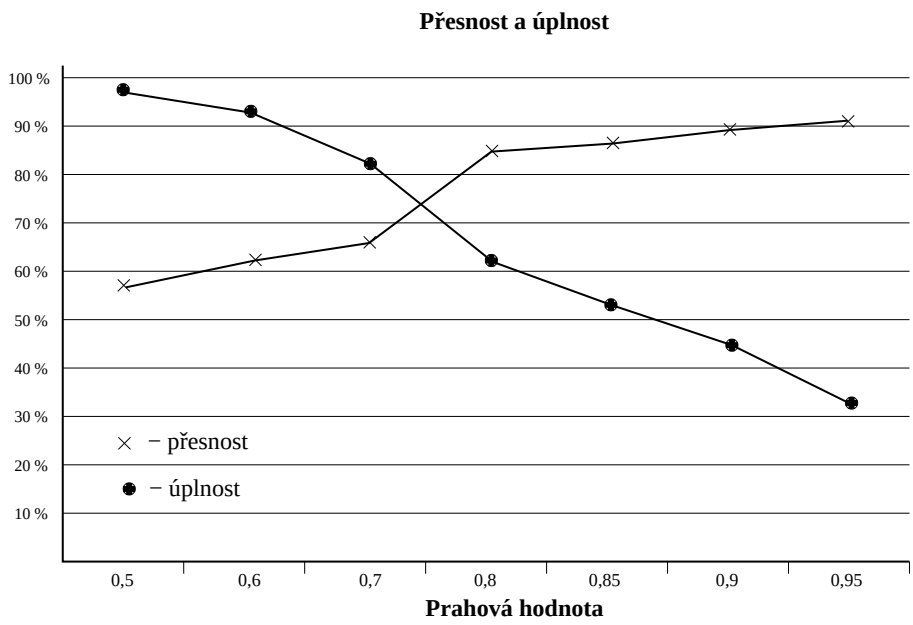
$$Sim(S_i, D) = \frac{\sum_1^H (w_h \times d_h)}{\sqrt{\sum_1^H (w_h)^2 \times \sum_1^H (d_h)^2}},$$

kde H je počet charakteristických frází shluku S_i, w_h udává váhu h-té fráze a d_h počet opakování h-té fráze v dokumentu D. Pokud vypočítaná míra podobnosti překročí stanovenou prahovou hodnotu τ alespoň pro jeden shluk, pak rozhodneme, že jde o dokument relevantní pro daného uživatele.

Podle počtu vybraných relevantních (V_R), nevybraných relevantních (N_R) a vybraných nerelevantních (V_N) dokumentů lze snadno určit přesnost (*precision* – P) a úplnost (*recall* – R) použité metody pomocí následujících vzorců:

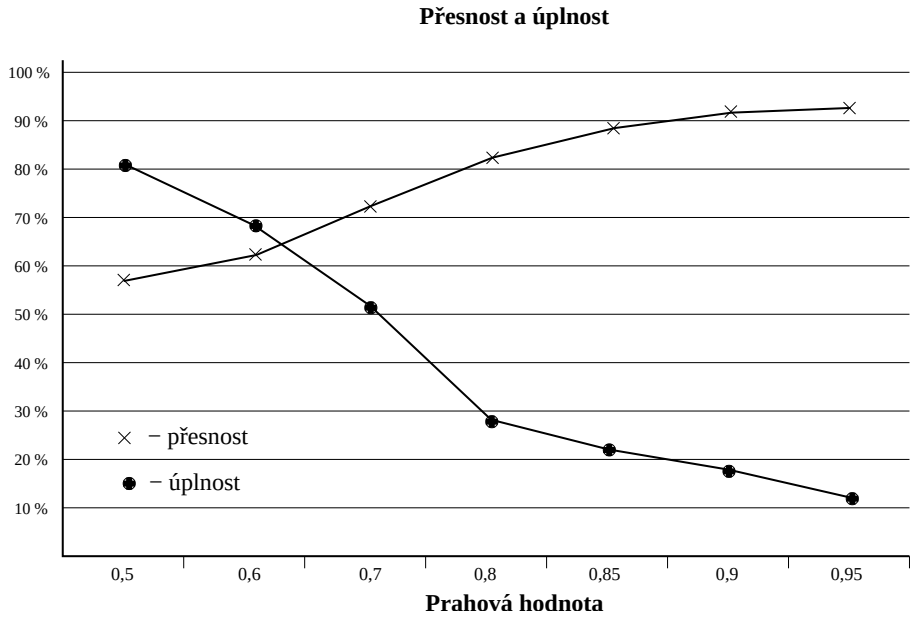
$$P = \frac{V_R}{V_R + V_N} \qquad R = \frac{V_R}{V_R + N_R}$$

Výsledky pro různá nastavení prahové hodnoty τ lze vidět na následujících grafech – pro kolekci ČTK a kolekci RCV1 na obrázcích 3 a 4. Podotýkáme, že počítaná úplnost (R) má smysl pouze v případě, že simulujeme chování uživatele kolekcí dokumentů.



Obrázek 3. Přesnost a úplnost v české kolekci ČTK

Jak je z grafů patrné, dosažené výsledky se pro stejné prahové hodnoty pohybují ve shodných rozmezích pro obě kolekce.



Obrázek 4. Přesnost a úplnost v anglické kolekci RCV1

3.1 Nalezené shluky

Příklad nalezených shluků lze demonstrovat na následujících výsledcích získaných z tématu „Politika“ z české kolekce ČTK:

- **C1**: irák, irácký
- **C2**: bělehrad, jugoslávský, kosovský, srbský, albánie, kosovský albánie
- **C3**: izrael, izraelský
- **C4**: tiskový konference, konference, tiskový
- **C5**: moskva, ruský, rusko
- **C6**: zahraniční, ministr zahraničí

Vzhledem k nalezeným shlukům je nutné upozornit, ze kolekce ČTK obsahuje tiskové zprávy za rok 1999 – což je doba, kdy na území bývalé Jugoslávie probíhaly boje.

Charakteristiky zpracování uvedeného příkladu shluků ČTK je možné nalézt v následující tabulce :

počet dokumentů	6 362
slov v dokumentech	1 003 072
uzlů stromu STC	1 397 413
doba zpracování	cca 10 minut

Testování na kolekci RCV1 bylo prováděno na srovnatelné množině anglických dokumentů, jako v případě české kolekce ČTK (tj. cca 6300 referenčních dokumentů použitých pro vygenerování shluků, resp. profilu).

4 Plánovaná rozšíření systému

Důležitým rozšířením popisovaného systému ProGen je zavedení „stárnutí“ profilu. Zájmy každého člověka se s časem vyvíjí a mění, a proto je nutné tomu přizpůsobit i generovaný profil tak, aby systém nedoporučoval uživateli dokumenty, o které již nemá zájem.

Dalším uvažovaným adaptačním mechanismem je přizpůsobování prahové podobnosti τ jednotlivým uživatelům tak, aby počet doporučovaných stránek vyhovoval jejich individuálním potřebám.

Zajímavé by jistě také bylo modifikovat nějaký stávající vyhledávací systém tak, aby nalezené dokumenty řadil s ohledem na relevantnost dokumentu vůči profilu uživatele, který vyhledávací dotaz zadal.

Závěrem bychom rádi poděkovali České tiskové kanceláři za poskytnutí kvalitní české kolekce dokumentů pro testovací účely.

Reference

1. Philip K. Chan – Constructing Web User Profiles: A Non-invasive Learning Approach; příspěvek konference Web Usage Analysis and User Profiling – International WEBKDD'99 Workshop San Diego, USA
2. R.Koval, P.Návrat – Intelligent Support for Information Retrieval in the WWW Environment; příspěvek konference Advances in Databases and Information Systems – 6th East european Conference, ADBIS 2002, Bratislava, Slovensko
3. C.C.Chen, M.C.Chen, Y.Sun: A Web Document Clustering: A Feasible Demonstration; `http://ants.iis.sinica.edu.tw/was/publication.htm`
4. C.C.Chen, M.C.Chen, Y.Sun: PVA: A Self-Adaptive Personal View Agent; `http://ants.iis.sinica.edu.tw/was/publication.htm`
5. Oren Zamir, Oren Etzioni – Web Document Clustering: A Feasible Demonstration; nalezeno na CiteSeer – `http://citeseer.org/` (září 2003)

Struktura reálných XML dokumentů a metody indexování*

Jiří Kosek[1], Michal Krátký[2], Václav Snášel[2]

[1]LISP, VŠE Praha
jirka@kosek.cz

[2]Katedra informatiky, VŠB–Technická univerzita Ostrava
michal.kratky@vsb.cz vaclav.snasel@vsb.cz

Česká republika

Abstrakt. Značkovací jazyk *XML* (*Extensible Markup Language*) je v současné době chápán jako nový přístup pro reprezentaci dat. Slovy databázové technologie je XML *jazyk pro modelování dat. Správně strukturovaný* (*well-formed*) XML dokument nebo množina dokumentů je XML databáze a příslušné DTD jejími schématy. Implementace systémů vhodných pro efektivní uložení a dotazování XML dokumentů (tzv. *nativní XML databáze*) vyžaduje vývoj nových technik a je dnes jednou z klíčových otázek světa informačních technologií. Pro efektivní uložení a dotazování XML dat není možné použít existující databázové modely (ať již relační nebo objektově–relační). Při vykonávání dotazu daného regulárním výrazem cesty je nutné procházet XML stromem. V tomto případě konvenční přístupy jako například SQL nebo OQL selhávají nebo nejsou příliš efektivní. V současné době existuje několik přístupů pro indexování XML dokumentů. Jelikož se všechny přístupy snaží modelovat definovaný XML strom tak, aby jím bylo možné efektivně procházet, dokumenty s velmi velkou maximální hloubkou zhoršují efektivitu vyhledávání. Stejně tak jako dokumenty s velmi rozdílnou hloubkou. Ačkoli jazyky pro popis struktury dokumentu umožňují definovat rekurzivní zanoření elementů, v tomto příspěvku chceme ukázat, že u prakticky používaných dokumentů se maximální a průměrná hloubka pohybuje v nízkých hodnotách. Efektivita přístupů pro indexování XML dat není tedy příliš často omezována indexováním hlubokých dokumentů.

Klíčová slova: XML, indexování XML dat, struktura XML dokumentů

1 Úvod

Značkovací jazyk *XML* (*Extensible Markup Language*) [36] je v současné době chápán jako nový přístup (respektive již pět let starý) pro reprezentaci dat. Slovy databázové technologie je XML *jazyk pro modelování dat* [25]. *Správně*

* Práce je částečně podporována grantem GAČR 201/03/1318

strukturovaný (*well-formed*) XML dokument nebo množina dokumentů je XML databáze a příslušné DTD jejími schématy. Struktura dokumentu může být popsána sofistikovanějšími jazyky jako je např. *XML Schema* [37, 38]. Implementace systémů vhodných pro efektivní uložení a dotazování XML dokumentů (tzv. *nativní XML databáze*) vyžaduje vývoj nových technik [25, 6] a je dnes jednou z klíčových otázek světa informačních technologií.

XML dokument je obyčejně modelován jako graf [9], ve kterém jsou korespondující uzly abstraktní objekty a hrany jsou označeny názvy elementů. Nejčastěji je tento graf stromem (tzv. *XML strom*). Pro získání dat z XML databáze byly vyvinuty různé dotazovací jazyky, např. *Lorel* [1], *XML-QL* [9], *XPath* [35, 33], *XQL* [28], *XQuery* [34]. Společným rysem těchto jazyků je použití regulárních výrazů pro vyjádření cesty grafem. Kde cesta je sekvence názvů elementů (nebo atributů) od kořenového elementu k listovému elementu. Uživatel se potom v XML dokumentu naviguje pomocí různě dlouhé cesty v XML stromu vyjádřené regulárním výrazem.

Pro efektivní uložení a dotazování XML dat není možné použít existující databázové modely (ať již relační, objektový nebo objektově–relační). Při vykonávání dotazu daného regulárním výrazem cesty je nutné procházet XML stromem. V tomto případě konvenční přístupy jako například SQL nebo OQL selhávají nebo nejsou příliš efektivní. V současné době existuje několik přístupů pro indexování XML dokumentů nebo obecně semistrukturovaných dat. Tyto přístupy je možné rozdělit do zhruba tří skupin:

1. *přístupy založené na relační dekompozici*,
2. *přístupy založené na trie reprezentaci XML dokumentu*,
3. *vícerozměrné přístupy*.

Do první skupiny náleží techniky, které se snaží využít mnoha lety vývoje prověřenou databázovou technologii pro indexování XML dokumentů. Abychom byli schopni rozhodnout zda jsou efektivnější přístupy založené na relační dekompozici či přístupy nativní, nebo vůbec posoudit efektivitu jednotlivých přístupů, musíme mít k dispozici jednotnou testovací kolekci dat a dotazů. Efektivitu (nejčastěji počet diskových přístupů) jednotlivých technik pak můžeme měřit při vykonávání definovaných dotazů nad testovací kolekcí dat. V současné době již začaly vnikat projekty (např. [29, 32]), které definují testovací data a často také množinu dotazů pro testovaní přístupů indexování XML.

V kapitole 2 jsou krátce popsány některé současné přístupy pro implementaci nativních XML databází. Jelikož se všechny přístupy snaží modelovat definovaný XML strom tak, aby jím bylo možné efektivně procházet, dokumenty s velmi velkou maximální hloubkou zhoršují efektivitu vyhledávání. Stejně tak jako dokumenty s velmi rozdílnou hloubkou. Ačkoli jazyky pro popis struktury dokumentu umožňují definovat rekurzivní zanoření elementů, v kapitole 3 ukazujeme, že u prakticky používaných dokumentů se maximální a průměrná hloubka pohybuje v nízkých hodnotách. Efektivita přístupů pro indexování XML dokumentů není tedy příliš často omezována indexováním hlubokých dokumentů.

2 Aktuální stav implementace nativních XML databází

V této kapitole jsou krátce popsány některé přístupy indexování XML dat. Úplnější přehled nejdeme např. v [2].

2.1 Přístupy založené na relační dekompozici

Během mnohaletého vývoje byly databázové technologie velmi dobře teoreticky popsány, ale zároveň i implementačně zvládnuty. I když pro uložení XML dat je nutné vyvinout nové přístupy, existují techniky které tuto mnoha lety ověřenou technologii využívají i pro uložení strukturovaných a semistrukturovaných dat. Jedna z prvních původních technik pro indexování XML dat je *Lore* publikovaný v [20]. Tento přístup používá tři vyhodnocovací strategie, pro podporu těchto strategií jsou použity tyto indexovací struktury: hodnotový a textový index, index odkazů a index cest. Většina indexovacích schémat ukládá záznamy existujících cest v databázi. Další z těchto přístupů jsou např. *XISS* [18], *STORED* [10] a *Hybrid Storage Model* [31]. Pro urychlení vykonávání dotazů založených na výrazech cest, je důležitá schopnost rychle určit předka/následníka nějaké dvojice uzlů v XML grafu. V případě použití kompletního k-árního stromu roste počet uzlů takového stromu se vzrůstající aritou. Proto je vhodné použít některá číslovací schémata [18].

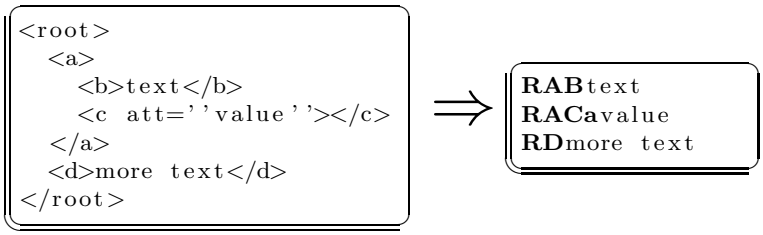
Klasickým přístupem založeným na relační dekompozici je *STORED* [10], který se snaží automaticky vytvořit dobré relační schéma pro XML data. Generování schématu je založeno na hledání vzorů v dokumentech. Dotazy kladené na semistrukturovaná data jsou transformovány na SQL dotazy. Data která se nepodaří uložit v relační databázi, jsou transformována do grafu, který je uložen externě. Autoři tvrdí, že tento graf je většinou malý a efektivita celého systému je tak vysoká. Testy s kolekcí DBLP [17] ukazují, že 60–90% semistrukturovaných dat je možno uložit do relační databáze. Efektivita STORED ukazuje, že je možné použít databázovou technologii pro uložení semistrukturovaných dat, ovšem nativní přístupy indexování XML poskytují více flexibilnější podporu pro uložení těchto dat.

2.2 Přístupy založené na trie reprezentaci XML dokumentu

Trie je datová struktura určená pro uložení řetězců. Vrchol v trie koresponduje k pozici v uloženém řetězci a hrany jdoucí z vrcholů jsou přiděleny k možným znakům na daných pozicích. Přístupy indexování XML založené na trie reprezentaci jsou např. *Index Fabric* [8], *RegXP* [3], *DataGuide* [26], *1-index*, *template index* (*T-index*) [21] a *A(k)-index* [14]. *DataGuide* reprezentuje XML dokument jako strom - trie strukturu všech cest v dokumentu. V nejhorším případě je počet uzlů exponenciální s velikostí původního XML stromu. V [21] autor konstatuje, že pro nepravidelné a hluboké dokumenty, roste DataGuide nade všechny meze,

proto navrhuje strukturu *reserved DataGuide*. *1-index* je také graf, který kóduje všechny cesty v XML stromu. V porovnání s DataGuide potřebuje méně paměti pro reprezentaci grafu. Další vylepšení velikosti indexovacího grafu přináší *A(k)-index*. Všechny tyto přístupy jsou použitelné pro dotazy specifikované jedním výrazem cesty. V [21] autor uvádí více obecnou strukturu zvanou *T-index*. Tato struktura umožňuje použít více cest v dotazu a zavádí třídy cest.

Index Fabric [8] se snaží redukovat počet diskových přístupů při vyhledávání použitím datové struktury *PATRICIA Trie* [22]. Tato struktura nebyla navržena jako perzistentní, ale Index Fabrix vytváří vyvážený tvar datové struktury, který je možné uložit do diskových bloků. Tento přístup ukládá XML data jako zakódované řetězce. Každému uzlu nalezenému v dokumentu je přidělen symbol a cesty jsou pak zakódovány spojením těchto symbolů (viz obrázek 1). Takto zakódované cesty jsou vloženy do PATRICIA trie do příslušných diskových bloků. Každý blok reprezentuje nejdelší společný prefix kódovaných cest. Štěpením bloků vznikají nové vrstvy trie a dochází k jejich hierarchizaci. Bloky jsou uloženy ve vyvážené stromové struktuře s jedním kořenovým uzlem. Tato struktura je efektivní pro vykonávání dotazů specifikovaných přesnou cestou, pro dotazy specifikované složitějším regulárním výraz je většinou nutné projít celou datovou strukturu.



Obr. 1. Index Fabric: Kódování cest do řetězců.

2.3 Vícerozměrné přístupy

Tyto přístupy jsou založené na myšlence, že XML dokument může být reprezentován jako množina prvků vícerozměrného prostoru. V [4] je XML dokument modelován jako množina bodů ve 3-rozměrném prostoru, kde jednotlivé dimenze prostoru představují: cesty v XML stromu, hodnoty cest a jednoznačná čísla dokumentů. *XPath Accelerator* [12] využívá úplné uspořádání definované na uzlech XML stromu. Každá z os XPath [35, 33] je pak definována pomocí tohoto uspořádání. Pro každý element je vytvořen 5 rozměrný bod, který je vložen do datové struktury *R-strom* [13]. XPath dotazy jsou potom vykonávány jako dotazy nad touto datovou strukturou.

Přístup publikovaný v [16, 15] modeluje XML dokument (nebo dokumenty) jako množinu bodů ve vícerozměrném prostoru, dimenze tohoto prostoru je určena maximální hloubkou XML stromu. Tento přístup využívá datových struktur pro indexování vektorových prostorů (např. *UB-stromy* [5, 19] nebo *BUB-stromy* [11]) i metrických prostorů (např. *M-stromy* [7]). V tomto přístupu je indexována nejen struktura (pro rychlý přístup k elementům), ale i řetězcové hodnoty elementů. XML strom je modelován jako množina všech cest od kořene k listům. V [30] autor ukazuje, že počet všech cest je sice exponenciální, ovšem s malým exponentem, je tedy semiexponenciální. V následující kapitole ukážeme, že i v reálných XML dokumentech počet všech cest neroste nade všechny meze a proto je možné tento přístup použít i pro velké kolekce dokumentů.

3 Struktura reálných XML dokumentů

Rozšiřování formátu XML přináší zvýšené požadavky na vyhledávání v dokumentech XML. Klasické indexovací a vyhledávací metody se tak rozšiřují o možnost postihnout v dotazu strukturu XML dokumentu. Pro zhodnocení úspěšnosti a použitelnosti jednotlivých indexovacích metod je potřeba znát variabilitu zpracovávaných dokumentů. Vhodnost určité indexovací metody přitom závisí zejména na hloubce zanoření elementů a počtu různých kontextů (kombinací zanoření elementů) vyskytujících se v dokumentech. Na první pohled by se mohlo zdát, že pro určitou doménu dokumentů můžeme jejich charakteristiky odvodit čistě teoreticky na základě znalosti jejich schématu. Schéma dokumentu můžeme chápat jako gramatiku popisující všechny přípustné instance dokumentu. Nejčastěji se pro zachycení schématu používá DTD, ale postupem času se prosazují i novější jazyky jako *XML Schema* [37, 38] a *Relax NG* [23].

Problém tohoto přístupu spočívá v tom, že mnoho typů dokumentů obsahuje rekurzivní struktury. Maximální teoretická hloubka zanoření elementů je proto nekonečná. To nám pro vyladění indexovacích metod příliš nepomůže. Nezbývá nám proto než empiricky na vzorku skutečných dokumentů vypozorovat jejich obvyklé charakteristiky. Dokumenty XML se většinou dělí do dvou základních typů *datově* a *dokumentově orientované*. Datově orientované dokumenty mají obvykle pravidelnou a plochou strukturu a používají se jako přenosový formát mezi různými informačními systémy. Pro klasické indexování tak tyto dokumenty nejsou často zajímavé, a i kdyby byly, jejich plochá struktura neklade příliš vysoké nároky na indexovací metodu.

Budeme se tedy zajímat o dokumentově orientované XML. Asi nejvíce dokumentů z této kategorie dnes tvoří webové stránky v XHTML. Zkoumání proto podrobíme vzorek 601 XHTML stránek, které vznikly konverzí stránek z `http://www.kosek.cz` do XHTML. DTD pro XHTML má střední složitost. Na světě existuje mnoho dokumentů, které používají mnohem komplexnější DTD. Mezi nejznámější a nejrozšířenější patří rozhodně formát pro ukládání dokumentace *DocBook* [24]. Druhý vzorek, na kterém jsme testovali variabilitu znač-

kování, tak byly tři reálné dokumenty v DocBooku: *DocBook: The Definitive Guide*[1], *PHP Manual*[2] a český tutoriál o *DocBook*[3]. Třetí vzorek jsou naopak dokumenty založené na poměrně jednoduchém DTD. Jedná se o přepisy Shakespearových her do XML od Jona Bosaka[4].

U zkoumaných kolekcí dokumentů nás zajímalo několik charakteristik. První z nich byla maximální hloubka zanoření elementů. Představíme-li si dokument XML jako strom, odpovídá tento údaj právě výšce stromu. Související charakteristikou je i průměrná hloubka zanoření elementů. Druhý ukazatel variability dokumentu je počet unikátních kontextů. *Kontextem* přitom rozumíme cestu složenou z názvů elementů, která ve stromu dokumentu vede od jeho vrcholu ke každému uzlu. Počet unikátních kontextů tak vyjadřuje kolik různých vzájemných kombinací zanoření elementů se v dokumentu vyskytuje. Zjištěné charakteristiky pro tři testovací kolekce dokumentů i pro jejich sjednocení přináší tabulka 1. Pro zajímavost je tabulka doplněna ještě o další údaje, jako je počet dokumentů v kolekci, celková velikost kolekce a celkový počet kontextů, který odpovídá celkovému počtu elementů. Počet všech cest od kořenového uzlu ke všem listům je tedy hodnota mezi celkovým počtem kontextů a počtem unikátních kontextů. Vidíme, že počet těchto cest neroste nade všechny meze.

Tabulka 1. Charakteristiky typických XML dokumentů.

Sada dokumentů	**Počet dokumentů**	**Celková velikost [MB]**	**Maximální hloubka**
DocBook	3	13.9	16
Shakespeare	37	7.5	6
XHTML	601	3.6	15
Celkem	641	25	16

Sada dokumentů	**Průměrná hloubka**	**Celkový počet kontextů**	**Počet unikátních kontextů**
DocBook 3	8.56	289 532	3 327
Shakespeare	4.77	179 689	57
XHTML	5.47	49 678	874
Celkem	6.95	518 899	4 258

Z naší jednoduché analýzy můžeme usuzovat, že i velmi složité dokumenty založené na schématech s možností rekurze, v praxi dosahují poměrně malé hloubky zanoření elementů. V našem případě 16, průměrná hloubka přitom nepřesahuje hodnotu 10. Při optimalizaci indexovacích metod tedy nemusíme počítat s pří-

[1] http://www.docbook.org
[2] http://www.php.net/manual/
[3] http://www.kosek.cz/xml/db/
[4] http://www.ibiblio.org/xml/examples/shakespeare/

liš hlubokými dokumenty. Variabilita dokumentu vyjádřená počtem unikátních kontextů se i v kolekcích dokumentů s různými schématy pohybuje v řádech tisíců.

V tabulce 2 jsou prezentovány charakteristiky největších XML dokumentů z projektu *XML Data Repository* [32]. Jedná se o dokumenty datově orientované. V rámci tohoto projektu byly vytvořeny XML dokumenty pro potřeby testování přístupů indexování XML dat. Vidíme, že i pro tyto poměrně rozsáhlé dokumenty se maximální a průměrná hloubka pohybuje v nízkých hodnotách.

Tabulka 2. Charakteristiky největších dokumentů z projektu *XML Data Repository*.

Dokument	**Velikost [MB]**	**Maximální hloubka**	**Průměrná hloubka**	**Počet elementů**
Protein Sequence Db.	683	7	5.2	21 305 818
SwissProt	109	5	3.6	2 977 031
DBLP CS Bibliography	127	6	2.9	3 332 130
Celkem	919	6	3.9	27 614 979

4 Závěr

V tomto příspěvku prezentujeme některé přístupy pro implementaci nativních XML databází a strukturu reálných XML dokumentů. Jelikož přístupy pro indexování XML dat často modelují XML strom tak, aby v něm bylo možné rychle procházet (určovat předky, následníky, atd.), dokumenty s velmi velkou maximální hloubkou či velmi rozdílnou hloubkou zhoršují efektivitu vyhledávání. Některé přístupy také modelují XML strom jako množinu všech cest od kořenového uzlu ke všem listovým uzlům. Ačkoli jazyky pro popis struktury dokumentu umožňují definovat rekurzivní zanoření elementů, v příspěvku ukazujeme, že maximální a průměrná hloubka reálných XML dokumentů, stejně tak jako počet všech cest v XML stromu se pohybuje v rozumných mezích. Jelikož je hloubka dokumentů v praktických případech poměrně malá, můžeme pro dotazování XML dokumentů použít modely umožňující vyhledávání na neúplnou shodu, které by v případě hlubokých dokumentů nebylo možné použít (např. [27]). Hloubka XML stromu často nabývá hodnoty maximálně deset. Při optimalizaci indexovacích metod tedy nemusíme počítat s příliš hlubokými dokumenty.

Reference

1. S. Abiteboul and et al. The Lorel Query Language for Semistructured Data. *Journal of Digital Libraries*, 1(1):68–88, 1997.
2. D. Barashev, M. Krátký, and T. Skopal. Modern Approaches to Indexing XML Data. *Sborník vědeckých prací VŠB-Technická univerzita Ostrava, accepted*, 2003.
3. D. Barashev and B. Novikov. Indexing XML Data to Support Regular Expressions. In *Proceedings of Advances in Databases and Information Systems, ADBIS 2002, 6th East European Conference, Bratislava, Slovakia*, volume Research Commmunications, pages 1–10, September 8-11, 2002.
4. M. G. Bauer, F. Ramsak, and R. Bayer. Indexing XML as a Multidimensional Problem. Technical Report TUM-I0203, Technical University München, 2002.
5. R. Bayer. The Universal B-Tree for multidimensional indexing: General Concepts. In *Proceedings of World-Wide Computing and its Applications'97, WWCA'97, Tsukuba, Japan*, 1997.
6. R. Bourret. XML and Databases, 2001, `http://www.rpbourret.com/xml/XMLAndDatabases.htm`.
7. P. Ciaccia, M. Pattela, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of 23rd International Conference on VLDB*, pages 426–435, 1997.
8. B. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, and M. Shadmon. A Fast Index for Semistructured Data. In *Proceedings of the VLDB Conference*, pages 341–350. VLDB, 2001.
9. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. Technical report, WWW Consortium, August, 1998.
10. A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with STORED. In *Proceedings of 1999 ACM SIGMOD international conference on Management of data*, pages 431–442. ACM Press, 1999.
11. R. Fenk. The BUB-Tree. In *Proceedings of 28rd VLDB International Conference on VLDB*, 2002.
12. T. Grust. Accelerating XPath Location Steps. In *Proceedings of ACM SIGMOD 2002, Madison, USA*, June 4-6, 2002.
13. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD 1984, Annual Meeting, Boston, USA*, pages 47–57. ACM Press, June 1984.
14. R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting Local Similarity to Efficiently Index Paths in Graph-Structured Data. In *Proceedings of ICDE'02*, 2002.
15. M. Krátký, J. Pokorný, T. Skopal, and V. Snášel. The Geometric Framework for Exact and Similarity Querying XML Data. In *Proceedings of First EurAsian Conferences, EurAsia-ICT 2002, Shiraz, Iran*. Springer–Verlag, LNCS 2510, October 27-31, 2002.
16. M. Krátký, J. Pokorný, and V. Snášel. Indexing XML data with UB-trees. In *Proceedings of Advances in Databases and Information Systems, ADBIS 2002, 6th East European Conference, Bratislava, Slovakia*, volume Research Commmunications, pages 155–164, September 8-11, 2002.
17. M. Ley. How to mirror DBLP, 1998, `http://www.informatik.uni-trier.de/~ley/db/about/instr.html`.
18. Q. Li and B. Moon. Indexing and Querying XML Data for Regular Path Expressions. In *Proceedings of 27th VLDB International Conference*. Morgan Kaufmann, 2001.

19. V. Markl. Mistral: Processing Relational Queries using a Multidimensional Access Technique. In *Ph.D. thesis, Technical University München, Germany*, 1999, `http://mistral.in.tum.de/results/publications/Mar99.pdf`.
20. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: a database management system for semistructured data. *ACM SIGMOD Record*, 26(3):54–66, 1997.
21. T. Milo and D. Suciu. Index structures for path expressions. In *In Proceedings of 7th the International Conference on Database Theory, ICDT'99*, pages 277–295. Springer–Verlag, LNCS 1540, 1999.
22. D. R. Morrison. PATRICIA - Practical Algorithm To Retrieve Coded in Alphanumeric. *JACM*, 15(4):514–534, 1968.
23. OASIS Relax NG TC. Relax NG - XML schema languages, 2002.
24. O'Reilly. Docbook: The definitive guide, 2003, `http://www.docbook.org`.
25. J. Pokorný. *XML: a challenge for databases?*, pages 147–164. Kluwer Academic Publishers, Boston, 2001.
26. J. W. R. Goldman. DataGuides: enabling query formulation and optimization in semistructured databases. In *Proceedings of 23rd International Conference on Very Large Data Bases, VLDB'97*, pages 436–445, 1997.
27. V. Rejlek. Podobnost XML dokumentů. Master's thesis, MFF UK, Katedra softwarového inženýrství, Praha, 2003.
28. J. Robie, J. Lapp, and D. Schach. XML Query Language (XQL), 1998, `http://www.texcel.no/whitepapers/xql-design.html`.
29. A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April, 2001, `http://monetdb.cwi.nl/xml/`.
30. D. Shasha. Algorithmics and Applications of Tree and Graph Searching, tutorial. In *ACM Symposium on Principles of Database Systems (PODS) 2002*, 2002.
31. D. Shin. XML Indexing and Retrieval with a Hybrid Storage Model. In *Proceedings of Knowledge and Information Systems (2001) 3, Springer–Verlag London*, 2001.
32. University of Washington's database group. The XML Data Repository, 2002, `http://www.cs.washington.edu/research/xmldatasets/`.
33. W3 Consortium. XML Path Language (XPath) Version 2.0, W3C Working Draft, 15 November 2002, `http://www.w3.org/TR/xpath20/`.
34. W3 Consortium. XQuery 1.0: An XML Query Language, W3C Working Draft, 15 November 2002, `http://www.w3.org/TR/xquery/`.
35. W3 Consortium. XML Path Language (XPath) Version 1.0, 16 November 1999, `http://www.w3.org/TR/xpath/`.
36. W3 Consortium. Extensible Markup Language (XML) 1.0, 1998, `http://www.w3.org/TR/REC-xml`.
37. W3 Consortium. XML Schema Part 1: Structure, 2001, `http://www.w3.org/TR/xmlschema-1/`.
38. W3 Consortium. XML Schema Part 2: Datatypes, 2001, `http://www.w3.org/TR/xmlschema-2/`.

Document Clustering based on Terms

László Kovács[1], Tibor Répási[2]

[1] Department of Information Technology, University of Miskolc
kovacs@iit.uni-miskolc.hu
[2] Department of Information Technology, University of Miskolc
repasi@iit.uni-miskolc.hu

Abstract. Document clustering is an important component in current information retrieval systems. The WEB is a huge pool of documents in very many different formats and with permanently changing content. There is a great demand on efficient document clustering methods. The paper gives first a general overview of typical clustering algorithms. The special characteristics of document clustering are also described in details. In the second part of the paper, the application of term thesaurus for document clustering is presented. The proposal demonstrates the usage of term thesaurus in dimension reduction of the document vector space.

1 Overview of Clustering Techniques

Clustering is a key component in any sophisticated information retrieval system. The main task of clustering is to group the objects where the objects within the same group are similar and the objects of the different groups are un-similar. There are many ways to perform clustering. The best-known methods are the K-means and the hierarchical clustering [6]. K denotes a natural number that denotes the desired number of clusters. To construct the clusters, initially all K clusters are created at once and then their positions are improved step by step with moving objects from one cluster to the other.

The benefits of the K-means algorithm are the simplicity and the relatively good performance. The execution cost is $O(I*N*K*M)$ where N denotes the number of elements, K is the number of clusters, M denotes the number of attributes, and I is the iteration count. The major disadvantage of this method is that it requires from the user to specify K in advance, which may be impossible in some cases. If K is too high, the execution cost is too high as well, and if it is too low, the accuracy of clustering decreases. Another disadvantage is that it prefers clusters with spherical shape and it does not support clusters of complex form.

The hierarchical methods work by grouping objects into a tree of clusters. A parent cluster contains all of its descendant clusters. The generation of the cluster tree can be done in an agglomerative or divisive way. A main benefit of the cluster tree is that it can be used as a search tree to locate the position of any arbitrary element. The cluster tree enables a grouping of the elements at several

layers. On the other hand, an important disadvantage of this method is that the clustering decisions made at an early level cannot be rolled back. A step that seems optimal at level L is not always optimal in a later step. The complexity of the HAC (Hierarchical Agglomerative Clustering) is equal to $O(N*N*M*log(N))$.

As can be seen, there are some pros and cons for both main clustering approaches, but there is a common belief that the hierarchical agglomerative algorithm is better and superior to partitioning, to the K-means algorithm. This belief is based on experiments with low dimensional datasets as well as on case studies. On the other hand, a new study [21] shows that partitioning algorithms always generate better hierarchical clustering solutions than agglomerative algorithms. The reasons for poor performance of the agglomerative method are the errors made during the early agglomeration.

There are several other candidates for clustering methods beside the two main variants. The proposal in [22] suggests using a neural network for clustering. The best-known NN variant for unsupervised processing is the Kohonen's SOM neural network. The main goal of the Bickshot algorithm [1] is to reduce the overall execution cost of the clustering algorithm. In the first step a subset of the original element set is selected randomly. The size of the initial subset is $sqrt(N)$. In the next step, clustering is performed for the selected subset using any known clustering method resulting in K clusters. It then uses the centroids of the clusters as the initial K centroids of the K-means clustering. The complexity of the Buckshot is equal to $O(N*K*M)$.

If there are more than one element-set to be clustered and the different clustering results are correlated to each other, information-theory-based co-clustering is proposed among others in [26]. The training set is given with a matrix. Co-clustering is defined as a pair of mapping from rows to row-clusters and from columns to column-clusters. It can be shown that the optimal co-clustering method is one that leads to the largest mutual information between the clustered random variables. The mutual information $I(X,Y)$ between the random variables X and Y is given by

$$I(X,Y) = \Sigma_{ij}\ p_{ij}\ log(p_{ij}/p_{*j}p_{i*})\ldots$$

where $p_{ij} = Pr\{X = X_i,\ Y{=}Y_j\}$, $p_{*j} = Pr\{Y{=}Y_j\}$, $p_{i*} = Pr\{X{=}X_i\}$. This can be considered as a statistical measure of the information that variable X contains about variable Y. The total mutual information value can be used as a target function during the clustering process.

2 Process of Document Clustering

Information retrieval systems accept a set of initial keywords from the user and return a set of documents having similar description keyword set. The quality of the answer was measured basically with two parameters: recall and precision. The recall is equal to N_1/N_2 where N_1 denotes the number of documents of interest found in the result set. The N_2 denotes the total number of documents of interest in the total document pool. The other parameter, precision is given by: N_1/R_1 where R_1 is the total number of documents in the result set. If both

precision and recall are equal to 1, the result set contains all relevant documents and all of the relevant documents are in the result set. As both parameters are usually lower than 1, the result set is not a perfect result. To improve the quality, users can usually alter the query parameters and re-query the document set. The manual refinement process is usually very time-consuming.

Document clustering was initially proposed for improving the precision and recall of information retrieval systems. Organizing the documents into hierarchical groups or clusters may support exploration of document archives. A document is usually treated as a list or bag of words. The bag representation ignores the position of the words within the document. The documents are characterized by the contained words. If an importance value is assigned to every document-word pair, every document can be described with vectors. The words are the dimensions, and the importance values are the dimension values. In some other approaches, not only the words are considered to describe the document content but the word sequences, too. The idea behind this extension is the experience that a word alone may have several meanings, it is ambiguous, but word sequences usually carry better and sharper meanings. Thus both words and word sequences should be taken into account. Both words and word sequences are called terms.

An M*N term-by-document matrix A can describe a database containing N documents and M words. The a_{ij} element of the matrix is the importance value of the w_i term in the d_j document. A variety of schemas are proposed to measure the importance value. The importance value is usually a product of two factors. The first factor is a global weight that reflects the overall value of the term in the whole document pool. It is widely accepted that the terms that are very frequent are not useful in the clustering process. Thus the global factor depends usually on the term frequency value. The second factor is a local component that reflects the importance of the term in the document. The local weight value may vary from simple binary values to functions involving logarithms of term frequencies. In the binary representation, the 0 value denotes that the term does not occur in the document. Otherwise the weight value is equal to 1.
The most widely used form of weighting is the TF-IDF model, i.e. the term frequency value combined with the inverse document frequency:

$$\text{Local weight} = t_{ij}$$

$$\text{Global weight} = \log(N / (d_j + 1))$$

$$a_{ij} = t_{ij} \log(N / (d_j + 1))$$

where t_{ij} is the number of occurrences of term j in document i (TF value) and d_j is the number of documents in which the term w_j occurs. There are some modifications in the literature for the base TF-IDF value. A refined version can be found for example in [24] where the local weight value is defined in the following way:

$$a_{ij} = (t_{ij} / (sqrt(sum(t_{ij} * t_{ij}))) * \log(N / (d_j + 1)).$$

Another popular term weighting scheme, beside the basic frequency-based methods, is the entropy-based measures [9]. In this weighting, the local weight is the logarithm of the term frequency. Global weighting uses the entropy value.

Local weight = $log(t_{ij}+1)$
Global weight = $1-\Sigma_i(\ p_{ij}\,log(p_{ij})/log(N))$

where

$p_{ij} = t_{ij}/\Sigma_j t_{ij}$.

The similarity value can be measured in a different way for document clusters. Beside the usual Euclidian distance in the vector space, some other interesting proposals can be found in the literature. In [26] for example, an association rule hypergraph partitioning method is presented. The algorithm is based on the association rule discovery technique used in data mining. An itemset is frequent if it occurs in many transactions. The best-known method for generating the frequent itemsets is the Apriori method. In the representations, documents are the items and a term is a transaction. A frequent itemset is a set of documents having some common terms. Itemsets are presented in a hypergraph where vertices are the documents and a hyperedge denotes a frequent itemset.

A key problem of agglomerative document clustering is the large number of similarity values to be stored. The size of the document-document similarity matrix is usually $N*N$. To reduce this size, only the most significant, highest K values are stored in the matrix, where $K << N$ [19].

A more sophisticated method for size reduction is the Latent Semantic Indexing (LSI) [2], where reduction is related to the term-document matrix. With this method, the number of attributes or objects can be drastically decreased. The LSI method uses the Singular Value Decomposition (SVD) technique to reduce the rank of the original term-document matrix. The elements of the A term-document matrix are the importance values of the terms in the different documents. In the SVD method, the $A\ (M*N)$ term-document matrix is decomposed into three matrices, written as

$A = U\ S\ V^T$

where U is a $M*M$ orthogonal matrix having left singular vectors of A as its columns, V is a $N*N$ orthogonal matrix having right singular vectors as its columns and S is a $M*N$ diagonal matrix having the singular values $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \ldots \geq 0$. The number of nonzero singular values is the rank of A. This factorization exists for any matrix A. The key motivation in LSI is to keep only the first K largest singular values in S and set others to zero. If the reduced S matrix is denoted by S' then

$A' = U'\ S'\ V'$

is a rank-K approximation matrix of A. U' is a $M*L$ matrix and V' is a $N*L$ matrix. A classic theorem of Eckart and Young states that the distance between A and its rank-K approximations is minimized by the SVD A' approximation. The SVD decomposition can be generated in the following way:

1. Compute $B = A*A$.
2. The eigenvectors of B are the right singular vectors of A.
3. The eigenvalues of B are the squares of the singular values of A.
4. The left singular vectors are generated using the V and S matrices.

A similar dimension reduction method is the PCA method where new dimensions are generated instead of a set of correlated old dimensions. Thus the

number of dimensions can be reduced in every iteration step. Also the PCA method calculates the eigenvalues and eigenvectors of $A*A'$ to find out which dimensions are correlated.

Beside the sophisticated mathematical methods there exist some heuristic methods for dimension reduction, too. The usual approach is to reduce the set of words used in the clustering process. It is a common experience that a word that occurs in every document cannot be used for clustering. Thus it is usual to remove those words that are too frequent or too sparse. The usual range of acceptance is between 5%-50%.

Another widely used method for word reduction is the elimination of the words with a non-topic oriented meaning. These stop-words are removed from the term-set. There exists usually a stop-word dictionary. The main drawback of this method is that it is language-dependent. The different languages have different stop-words.

3 Document Clustering with TERM Thesaurus

The proposed construction algorithm is intended for information systems with a relatively narrow problem area. In this case, an attribute hierarchy can be generated within an acceptable time and effort. It is assumed that the attribute hierarchy contains only those attributes that are relevant for the problem area in question. In this case, the size of the attribute thesaurus and the intent part of the concepts will be manageable. The elements of the attribute hierarchy are denoted by Ti. There exists a parent-child (specialization) relationship among the Ti terms in the hierarchy. According to our simplification, every term node may have only one parent term. The specialization relationship is denoted by the < symbol. The

$$T_1 < T_2$$

holds if T_2 is an ancestor of T_1, i.e. T_1 is a specialization of T_2. We assume there exists a common term hierarchy for all terms included in the document classification process. Thus there exists a unit term which is the generalization of all of the terms. This unit term is denoted by T_u. The T_u may be an abstraction, without correspondence to any real term.

In the proposed algorithm the main goal is to reduce the involved attributes for the document description. To achieve a better performance three methods are combined:

- term thesaurus is based reduction
- clustering of the terms (the clusters are used as new terms instead of its members)
- detection of the implication relations among the terms.

The first method is based on the semantic aspects of the terms, while the two other steps use statistical methods. The key role of the applied thesaurus is to reduce the attribute set used in the document representation. Basically, every word found in one element of the document pool considered as an attribute.

The usual number of different words in a document pool may be some hundred thousands. This huge size of the attribute space can not be managed with the available computing power. The number of description attributes should be reduced to some hundreds or less to perform the computations within an acceptable time. In our method, the attribute thesaurus is used to perform dimension reduction.

In the reduction based on thesaurus, the words with low frequency are not erased from the documents but they are replaced with the generalized attributes. Using this method, the number of different attributes in the document pool will be reduced, but some part of the original information is saved at the same time. For example, if both the words football and tennis were rear, they would be eliminated in the traditional processing way. On the other hand, if both words are replaced with the generalized term sport, this generalized attribute may be frequent enough to remain in the final attribute set. During the substitution steps, new terms may appear in the attribute set, but the total number of attributes decreases. If a generalized word gets too frequent during the substitution steps, this word is also removed from the attribute set. Only those generalized terms remain in the final set whose frequency lies between the given acceptance interval. If the generalized term of an attribute has low frequency, this term will be replaced with its parent in the next loop. The key steps of the attribute reduction algorithm:

1. Calculate the term frequency values (TF) for every attribute.
2. Order the attributes by the frequency values.
3. Eliminate the attributes with $TF_i > TFMAX$.
4. Let L be the set of attributes with $TF_i < TFMIN$.
5. If L is empty stop the reduction procedure.
6. For each w_i element in L, replace w_i with its parent based on the attribute thesaurus.
7. Calculate the new frequency values for the actual attribute set.
8. Go back to step 2.

The algorithm ends in final steps as all of the investigated terms all contained in a common hierarchy. Let N_i denote the number of terms at level i. As there is only one root element, $N_0 = 1$. Let m denote the maximal level of the initial terms. In the algorithm step 6, each term is replaced with its parent. According to our definition, every terms except T_u has exactly one parent, thus the replacement can be performed. The maximal level value is decreased by one in this step. As the initial level is a final integer number, the maximal level is decreased to 0 in final steps. Thus in final steps all of the terms are replaced with only one term which terminates the algorithm. Of course, the termination condition may be met in one earlier step, too.

After finishing the reduction process, only the attributes with accepted frequency values remain in the attribute set. The size of the resulted attribute set is larger compared to the result of the basic reduction method, but this set stores more information about the original document content. To make the computation efficient, the remained attribute set should be reduced in further steps. The

reduction here is also done with the use of the attribute thesaurus. The terms are replaced with the generalized terms. In the first step, the existing generalization relationships are detected in the initial term-set. If there exists T_1 and T_2 in the term-set where $T_1 < T_2$ then T_1 is eliminated from the term-set. For the generated term-set holds that no term is an ancestor of some other terms.

If the number of terms is above the given threshold, further reduction is required. In this step, the terms are replaced with their parent term, similarly to the first reduction step. After this phase, the number of attributes is reduced to the desired level. This ensures better performance and computational efficiency.

The assumption that every word is contained in the attribute hierarchy is very rarely met. Most of the terms in a document pool are not categorized previously. To cope with this kind of problem as well, the proposed system merges the methods based on concept lattice and statistical calculations. To create a thesaurus manually is a very time-consuming task. Another problem is that a word has usually more than one meaning. The goal to create the thesaurus automatically is a very important research area in information technology.

Beside the reduction based on thesaurus, a statistical reduction method is applied in our test system. In this phase, the clustering of the terms is performed using the distance values among the terms. For this purpose, the words are represented in a vector space:

$$\mathrm{V}_i = (\mathrm{v}_{ij})$$

where v_{ij} denotes the frequency of term V_i in the D_j document. The similarity between two terms is measured with the usual Euclidean distance value. According to this interpretation, the terms that occur in similar documents are similar. Similar terms usually belong to the same topic. The distance value between two terms is defined by

$$\mathrm{D}_{xy}{}^2 = \Sigma\ (\mathrm{v}_{xj} - \mathrm{v}_{yj})^2$$

The terms with similar vector representations denote terms with similar usage, these terms are strongly correlated to each others. To reduce the representation space, the correlated attributes may be eliminated from the attribute space. In order to determine the correlated terms, a term clustering is performed. To cluster the terms, a hierarchical merging clustering method is used. The clusters near to each other will be merged as long as one of the stop conditions is met:

- the minimal number of clusters is reached;
- the distance between the nearest clusters is higher than a given threshold.

After finishing this primary clustering process, a significant number of terms is bound to one of the clusters. According to our experience, the generated clusters correspond to the different key document topics. Using these term-clusters, the documents can be assigned to one or more topic clusters if they contain terms from the clusters. The outlier terms are not useful in this process. According to our experience, most of the terms are outliers. To reduce the number of outliers, a near cluster is searched where the term can be bound to. The implication rule is used to bind the outlier terms to clusters. Only those outlier terms are assigned to clusters where there exists an implication from the term to the cluster.

The T_1 implies T_2 if the document set of T_1 is a subset of document set of T_2. The implication relation is denoted by $\rightarrow$. The concept of implication between two terms T_1 and T_2 can be defined as follows:

$$T_1 \rightarrow T_2 \text{ if } D_1 \subseteq D_2$$

where D_1 denotes the set of documents containing T_1, and D_2 is the document set containing T_2. Using the implication lattice, the clusters can be extended with new terms to provide better clustering accuracy for documents.

4 Test system

To support further research in the field of document clustering we created a test system. The document source is an integral part of this test system. By choosing an appropriate document set as source we will be able to solve some initial problems of data mining (e.g. data cleaning, uniforming, validation, etc.). Our choice was a standard documents set, collected, preprocessed and published to the research community by Reuters. The Reuters Corpus Vol. 1 is a comprehensive collections of news articles (news stories) originating from the newswire. To have a picture about this corpus, it contains about 810.000 news stories written in English, each in the same format. The NewsML standard stands for the storage for newswire stories in an XML based format over the whole life cycle of the documents. Advantages of this format are clear: ease data access by using a standard XML parser, strict structure of the documents, a high amount of meta-data, containing author, title, identification token, and a validated classification.

The test system was implemented around a database containing only the necessary metadata of the documents. At the beginning, the first step was to collect metadata of the documents. For this task we used a Java program implementing an XML parser which was collecting the words an their occurrence in the source documents[3].

A calculation resulting in a correlation matrix was done. The values contained in the matrix are the correlation values for each word pair of a selected subset of the words. The selection was made on the simple occurrence frequency of the words, based on the consideration that words which are occurring in lots of documents are not really relevant, and words occurring in very few documents only are not selective enough to create document clusters. We calculated the correlation value of a given word pair as the normalized element number of the symmetric difference of those subsets of documents in which the given words are occurring. Possible correlation values for word pairs are 1, in the case if the two words are present in exactly the same subset of documents, 0 if there is no document containing both words, and all possible values in between. A remarkable note to this value is that this kind of correlation is symmetric for each word pair.

[3] Due to calculation time limitations and the finiteness of storage capacity we used to have just a small part of the corpus processed. There where around 2500 documents in processing

At the next step we have analyzed the correlation matrix. We are using to use the term distance of words as the opposite of correlation; 1 - correlation value. For this reason we created 20 distance classes with an even distribution over the correlation value range.

Table 1.

Distance	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Count	3	2	3	1	1	3	3	9	9	15	20	30	66	131	288	760	2758	15249	159956	2366657

Based on the distance between the words we can cluster them. This clustering is also suitable for the document, as a cluster of words can be understood as a subset of documents containing the words. After this, we are going to search for the non symmetric correlation in the documents. As known, some words are relevant in a special context.

Table 2.

Cluster ID	42	453	334	188	206	393	70	72	101	17	2	139	3	18	7
Size	6	6	7	8	8	8	9	9	9	10	11	20	42	47	228

After term reduction based on term frequency pre-clustering was performed. Taking only statistical clustering, only a few terms can be found that have similar description vectors. Only about 20% of the total term-set was involved in one of the generated clusters. When this was done, about 20 clustered.

Conclusion

The document clustering is an important area in the development of human oriented information retrieval systems. One of the key elements in document clustering is the efficiency of the computations. The paper proposed a combined method for the attribute reduction. The net effect of the three methods, which include the reduction based on term thesaurus, based on clustering of the terms and on the detection of the implication relations among the terms, is significant better than the effect of the single methods. In order to provide a real-application oriented solution, the thesaurus based on hierarchy should be replaced with the thesaurus based on lattice in the future investigations.

References

[1] K. Lin – R. Kondadadi: A word-based soft clustering algorithm for documents, *Tech.Rreport of Univ. of Memphis*

[2] M. Hasan and Y. Matsumoto: Document Clustering Before and after the singular value decomposition, *Tech. Rep*

[3] S. Dumais and H. Chen: Hierarchical classfification of WEB content, *Proc. Of 23rd ACM Int. Conf. on Research and Development in Information Retrieval,* Athens, 2000

[5] Y. Yang: An Evaluation of Statistical Approaches to Text Categorization, *Information retrieval*, Kluwer Academic Publishers , 1999

[6] J. Han – M. Kamber: *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 1999

[7] K. Hu and Y. Lu and C Shi: Incremental Discovering Association Rules: A Concept Lattice Approach, *Proceedings of PAKDD99*, Beijing, 1999, 109-113

[8] L. Kovacs – P Baranyi: Document Clustering Using Concept Set Representation, *INES02*, Opatija, Croatia, 2002

[9] D. Lewis - R. Schaipe - J. Callan – R. Papka: Training Algorithms for linear text classifiers, Proceedings of SIGIR'96, 1996

[12] K. Hu and Y. Lu and C Shi: Incremental Discovering Association Rules: A Concept Lattice Approach, *Proceedings of PAKDD99*, Beijing, 1999, 109-113

[13] C. Lindig: Fast Concept Analysis, *Proceedings of the 8th ICCS*, Darmstadt, 2000

[14] L. Nourine and O. Raynaud: A Fast Algorithm for Building Lattices, *Information Processing Letters,* 71, 1999, 197-210

[15] S. Radeleczki and T. Tóth, Fogalomhálók alkalmazása a csoporttechnológiában, *Kutatási jelentés, OTKA kutatási jelentés,* Miskolc, Hungary, 2001.

[16] G. Stumme and R. Taouil and Y. Bastide and N. Pasquier and L. Lakhal: Fast Computation of Concept Lattices Using Data Mining Techniques, *7th International Workshop on Knowlegde Representation meets Databases (KRDB 2000)*, Berlin, 2000

[17] M. Zaki and M. Ogihara: Theoretical Foundations of Association Rules, *Proceedings of 3 rd SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'98)*, Seattle, Washington, USA, June 1998.

[18] L. Kovacs: Efficiency Analysis of Building Concept Lattice, *Proceedings of 2nd ISHR on Computational Intelligence*, Budapest, 2001

[19] A. Smeaton and etc: An Architecture for Efficient Document Clustering and Retrieval on Dynamic Collection of Newspaper Texts, *Proceedings of IRSG*, 1998

[20] J. Silva and J. Mexia and A. Coelho and G. Lopez: Document Clustering and Cluster Topic Extraction in Multilingual Corpora, *Proc. of the 2001 IEEE Int. Conference on Data Mining*, IEEE Computer Society, pp. 513-520

[21] Y. Zhao and G. Karypis: Evaluation of Hierarchical Clustering Algorithms for Document Datasets, *Techn. Reports of Univ. Minnesota*, http://www.cs.umn.edu/∼karypis

[22] D. Merkl: Text Data Mining, *Handbook of Natural Language Processing*, 1998, New York Marcel Dekkler Publ.

[23] G. Jones and P. Willet, etc: Non-hierarchical document clustering using genetic algorithm, *Onformation Research, Vol 1. April 1995*

[24] B. Mandhani and S. Joshi and K. Kummamuru: A metrix-density based algorithm to hierarchically co-cluster documents and words, file://cdrom/papers/refereed/p704/rpsa.html

[25] D. Tikk and J. Yang and S. Bang: Text categorization using fuzzy relational thesauri, *Technical report, Chonbuk National University*, Chonju, Korea, 2001

[26] D. Boley and etc: Partitioning-based clustering for WEB document categorization, *Decision Support Systems,* Vol 27, 1999, pp 329-341

Induktívne štruktúry

Stanislav Krajči
krajci@science.upjs.sk
Ústav informatiky
Prírodovedecká fakulta UPJŠ Košice

8.12.2003

Abstrakt

V mnohých disciplínach matematickej informatiky sa v najrôznejších formách stretávame s indukciou. Jej použitie sa spravidla považuje za triviálne a nevenuje sa mu žiadna pozornosť. Napriek tomu máme pocit, že táto problematika je povšimnutiahodná. V tomto článku sa po vzore úvodnej kapitoly knihy [GJ] pokúsime o vystihnutie tejto spoločnej črty použitím pojmu induktívna štruktúra. Sforumulujeme a dokážeme verziu vety o matematickej indukcii pre takúto induktívnu štruktúru. Naviac definujeme rozšírenie tohoto pojmu – induktívnu štruktúru s jednoznačnou konštrukciou, ktorá, ako dokážeme využitím myšlienok z [BŠ], je nutnou podmienkou korektného použitia induktívnej definície pozdĺž prvkov tejto štruktúry.

1 Motivácia

Každý z nás si iste z detstva (alebo z detstva svojich detí) pamätá na čarovnú hračku s názvom Lego. Každá séria obsahuje kúsky ľudovo zvané „kocky“ rôznych tvarov, veľkostí a farieb. Dômyselne členený povrch zabezpečuje možnosť ich mechanického spájania do zložitejších útvarov. Nie je náhoda, že táto hračka sa používa na školách ako pomôcka na znázornenie atómov. Čo majú tieto dve veci spoločné? Ako jednotlivé kocky lega, tak i atómy predstavujú akési *základné prvky.* V oboch prípadoch tiež platí, že spájanie týchto základných prvkov do väčších celkov má svoje pravidlá, nemožno spojiť hocičo s hocičím. Tieto pravidlá spájania kociek do väčších konštrukcií či zlučovania atómov do molekúl a molekúl do makromolekúl môžeme chápať ako metódy alebo *operátory* vzniku. Obe tieto štruktúry,

jedna tvorená všetkými možnými konštrukciami z legových kociek, druhá všemožnými chemickými zlúčeninami, sú príkladom *induktívnej štruktúry.*

Induktívna štruktúra je teda množina všetkých objektov, ktoré vzniknú z istých základných prvkov pomocou istých operátorov. Uvedomme si, že s takýmto princípom sa stretávame aj na mnohých iných miestach. Pekným príkladom induktívnej štruktúry je sama príroda, a to ako život jedinca (za základné prvky môžeme považovať dve rodičovské bunky a za operátory okrem iného predpisy zakódované v jeho DNA), tak život vo všeobecnosti (základnými prvkami sú prvé živé bunky a operátormi trebárs princípy evolučnej teórie), ale i svet ako celok (základným prvkom je situácia po veľkom tresku s dobre nastavenými konštantami, operátormi sú prírodné zákony).

V každej induktívnej štruktúre sú potrebné ako základné prvky, tak operátory. Ak by sme nemali základné stavebné jednotky, darmo by sme ovládali metódy konštrukcie, nič by sa nám skonštruovať nepodarilo. Rovnako bezradní by sme boli, keby sme síce základné prvky mali, ale nepoznali by sme operátory. Pekne to vidieť na ďalšej „hravej“ induktívnej štruktúre – na domine. Keď dostatočne blízko vedľa seba postavíme kocky domina a potom do prvej z nich cvrnkneme, nespadne len ona, ale vyvolá domino efekt – popadá celý rad. Základným prvkom je tu pád prvej kocky, operátor tu predstavuje fakt, že pád ľubovoľnej (nielen prvej) kocky spôsobí pád kocky nasledujúcej. Darmo by sme však kocky akokoľvek ukladali, keď sa žiadna nezvalí, domino efekt neuvidíme. Neúspešní by sme boli i vtedy, keď by sme nedodržali vhodnú vzdialenosť medzi ľubovoľnými dvoma susednými kockami – pád kociek sa na takom mieste preruší a ostatné kocky nespadnú.

Domino efekt v sebe ukrýva jednu dôležitú induktívnu štruktúru – prirodzené čísla. Pád každej kocky (za predpokladu, že ich je nekonečne veľa) reprezentuje jedno prirodzené číslo. Pád prvej kocky, čiže základný prvok, je najmenšie prirodzené číslo – nula (`0`), súvislosť medzi pádmi susedných kociek zasa reprezentuje operátor – funkciu nasledovník (`S`). Všimnime si, že táto induktívna štruktúra je najjednoduchšia, a pre túto svoju jednoduchosť aj najdôležitejšia. Má totiž jediný základný prvok a jediný operátor.

2 Induktívna štruktúra

Induktívny charakter prirodzených čísel umožňuje používať v dôkazoch princíp matematickej indukcie. Je založený na tejto vete:

Veta 1 (o matematickej indukcii)
Nech $V(n)$ je tvrdenie o prirodzenom čísle n a nech platí:

1. *$V(0)$ platí.*
2. *Pre každé prirodzené k z platnosti $V(k)$ vyplýva platnosť $V(k+1)$.*

Potom $V(n)$ platí pre všetky prirodzené n.

Ďalším využitím induktívnosti prirodzených čísel je definícia matematickou indukciou. Najprv (v 1. indukčnom kroku) explicitne definujeme hodnotu funkcie pre základný prvok `0` a potom (v 2. indukčnom kroku) sme povedali, ako sa k tejto funkcii zachová operátor `S`. Opäť vzniká otázka: Stačia tieto dve rovnosti na jednoznačné definovanie celej funkcie? Odpoveď dáva nasledujúca veta:

Veta 2 (o definícii matematickou indukciou)
Nech A je množina, c jej prvok a g funkcia z množiny $\mathbb{N} \times A$ do množiny A. Potom existuje jediná funkcia $f : \mathbb{N} \to A$ taká, že platí:

1. *$f(0) = c$.*
2. *$f(k+1) = g(k, f(k))$.*

Takéto princípy môžeme vysloviť pre každú (dostatočne presne definovanú) induktívnu štruktúru. Pokúsme sa však najprv presne zadefinovať, čo to induktívna štruktúra je:

Definícia 1
Nech M je množina, nech $B = \{b_i : i \in I\}$ je množina niektorých jej rôznych prvkov a $O = \{o_j : j \in J\}$ je množina (vzhľadom na M parciálnych) funkcií $o_j : D_{o_j} \to M$, kde n_j sú prirodzené čísla a $D_{o_j} \subseteq M^{n_j+1}$. Nech U je najmenšia podmnožina množiny M taká, že platí:

1. *Každé b_i je prvkom U.*

2 Ak $j \in J$ a $t_0, \ldots, t_{n_j}$ sú prvky U a $\langle t_0, \ldots, t_{n_j} \rangle \in D_{o_j}$, tak aj $o_j(t_0, \ldots, t_{n_j})$ je prvkom U.

Potom množinu U nazývame induktívna štruktúra *daná* základnými prvkami B a operátormi O.

Pod slovom najmenšia rozumieme prienik všetkých množín vyhovujúcich podmienkam 1 a 2 – ľahko vidíme, že prienik takýchto množín musí mať tiež tieto dve vlastnosti.

Ako sme už spomínali, špeciálne pre množinu $\mathbb{N}$ všetkých prirodzených čísel (chápanú ako podmnožinu množiny $\mathbb{R}$ všetkých reálnych čísel) máme $M = \mathbb{R}$, I a J sú jednoprvkové, napríklad $I = J = \{0\}$, základný prvok je jediný – $b_0 = \mathtt{0} \in \mathbb{R}$ a operátor tiež jediný – $o_0 = \mathtt{S} : \mathbb{R} \to \mathbb{R}$ (definovaný $\mathtt{S}(x) = x+1$).

Induktívne štruktúry sú v matematike i teoretickej informatike často využívané. Popri už rozoberaných prirodzených číslach spomeňme ďalšie príklady:

V teórii vypočítateľnosti tvoria induktívnu štruktúru (primitívne, všeobecne i čiastočne) rekurzívne funkcie. Základnými prvkami sú tu funkcie $\mathtt{Z}$, $\mathtt{S}$ a $\mathtt{P}_i^n$, kde $\mathtt{Z}(x) = 0$, $\mathtt{S}(x) = x + 1$ a $\mathtt{P}_i^n(x_1, \ldots, x_n) = x_i$, operátormi metódy vzniku nových funkcií – substitúcia, rekurzia a prípadne (regulárna) minimalizácia. (Všimnime si, že v tomto prípade nie sú operátory totálne funkcie.)

Relačná algebra databázových systémov je tiež induktívna štruktúra. Základnými prvkami sú databázové tabuľky, operátormi operácie nad nimi – selekcia, transformácia stĺpcov, projekcia, spojenie, zjednotenie či tranzitívny uzáver a ďalšie – ktorých výsledkom je pohľad alebo odpoveď na dopyt.

S databázou spriazneným príkladom je prologovský program. Jeho základnými prvkami sú fakty (z databázového hľadiska sú to riadky v príslušných tabuľkách), operátormi pravidlá (tie sú pendantmi pohľadov resp. odpovedí na dopyty).

Pre matematickú logiku sú podstatnými príkladmi induktívne štruktúry výrokov, termov, formúl i dokázateľných výrokov a formúl.

Pre každý prvok takejto induktívnej štruktúry vieme vysledovať v jednotlivých krokoch históriu jeho vzniku:

Definícia 2
Vytvárajúcou postupnosťou *prvku p induktívnej štruktúry U nazývame konečnú postupnosť $\langle p_0, \ldots, p_m \rangle$ prvkov U takú, že platí:*

1 $p_m = p$ *(t. j. postupnosť sa končí práve prvkom* p*).*

2 Pre každé $i \leq m$ *(t. j. pre každý prvok vytvárajúcej postupnosti) platí jedna z podmienok:*

2.1 p_i *je základný prvok štruktúry* U*.*

2.2 Existujú indexy $j_0, \ldots, j_k < i$ *a operátor* o *štruktúry* U *taký, že* $p_i = o(p_{j_0}, \ldots, p_{j_k})$ *(t. j.* p_i *vznikne aplikáciou nejakého operátora na niektoré z prvkov, ktoré sú vo vytvárajúcej postupnosti pred ním).*

Že to vieme urobiť naozaj pre každý prvok, ukazuje nasledujúca veta:

Veta 3
Každý prvok induktívnej štruktúry U *má (aspoň jednu) vytvárajúcu postupnosť.*

Dôkaz. Označme V množinu tých prvkov z U, ktoré majú nejakú vytvárajúcu postupnosť. Ukážeme, že táto množina spĺňa podmienky 1 a 2 z definície induktívnej štruktúry:

1 Zrejme každý základný prvok b_i je z U a má vytvárajúcu postupnosť, a to napríklad $\langle b_i \rangle$, teda patrí do V.

2 Nech $j \in J$ a $t_0, \ldots, t_{n_j}$ sú prvky V, teda pre ne existujú vytvárajúce postupnosti, pre t_k nech je to $\langle p_0^k, p_1^k, \ldots, p_{m_k}^k \rangle$. Potom (samozrejme, za predpokladu, že $\langle t_0, \ldots, t_{n_j} \rangle$ patrí do definičného oboru funkcie o_j) je $\langle p_0^0, \ldots, p_{m_0}^0, p_0^1, \ldots, p_{m_1}^1, \ldots, p_0^{n_j}, \ldots, p_{m_{n_j}}^{n_j}, o_j(t_0, \ldots, t_{n_j}) \rangle$ vytvárajúca postupnosť prvku $o_j(t_0, \ldots, t_{n_j})$ (uvedomme si, že $p_{m_k} = t_k$), takže platí aj $o_j(t_0, \ldots, t_{n_j}) \in V$.

Keďže U bola najmenšia množina spĺňajúca podmienky definície 1 a 2, nutne $U \subseteq V$ (U je najmenšia taká množina), teda $V = U$, čo sme chceli dokázať. $\square$

Spomedzi všetkých vytvárajúcich postupností určite existuje najkratšia (môže ich byť aj viac, ba dokonca, ako uvidíme neskôr, nemusia obsahovať ani rovnaké prvky).

Definícia 3
Ak $\langle p_0, \ldots, p_m \rangle$ je najkratšia vytvárajúca postupnosť prvku p, tak m budeme nazývať zložitosť *prvku p.*

Vyslovme teraz zovšeobecnenie vety 1:

Veta 4 (o matematickej indukcii v induktívnej štruktúre)
Nech U je induktívna štruktúra daná základnými prvkami $\{b_i : i \in I\}$ a operátormi $\{o_j : j \in J\}$ a nech $V(p)$ je tvrdenie o jej prvku p. Nech ďalej platí:

1. *Pre ľubovoľné $i \in I$ platí $V(b_i)$.*
2. *Pre ľubovoľné $j \in J$ a prvky $t_0, \ldots, t_{n_j}$ také, že $\langle t_0, \ldots, t_{n_j} \rangle$ patrí do definičného oboru o_j, z platnosti $V(t_0), \ldots, V(t_{n_j})$ vyplýva platnosť $V(o_j(t_0, \ldots, t_{n_j}))$.*

Potom $V(p)$ platí pre všetky $p \in U$.

Dôkaz. Vezmime množinu P všetkých prvkov p z U, pre ktoré neplatí $V(p)$, chceme ukázať, že je prázdna. Ak prázdna nie je, vezmime ľubovoľný jej prvok $p_{\min}$ s najmenšou zložitosťou m (vzhľadom na vlastnosti prirodzených čísel (na metaúrovni) určite aspoň jeden taký existuje). Nech $\langle q_0, \ldots, q_m \rangle$ je jeho najkratšia vytvárajúca postupnosť. Zrejme $q_m = p_{\min}$ nie je základný prvok (tie totiž podľa bodu 1 do P patriť nemôžu), musí mať teda tvar $o_j(q_{i_0}, \ldots, q_{i_{n_j}})$ pre nejaké $j \in J$ a indexy $i_0, \ldots, i_{n_j} < m$. Uvedomme si však, že každý prvok q_{i_k} (pre $k \in \{0, \ldots, n_j\}$) má zložitosť menšiu než m (veď preň máme vytvárajúcu postupnosť $\langle q_0, \ldots, q_{i_k} \rangle$), preto nepatrí do P (najmenšia zložitosť prvkov z P je totiž m). Platí teda $V(q_{i_0}), \ldots, V(q_{i_{n_j}})$, teda podľa predpokladu 2 platí aj $V(o_j(q_{i_0}, \ldots, q_{i_{n_j}}))$, čiže $V(p_{\min})$, čo je však spor. Žiadne také $p_{\min}$ teda neexistuje, čiže P je naozaj prázdna. $\square$

Ak teda budeme chcieť dokázať nejakú vlastnosť pre všetky prvky induktívnej štruktúry, bude stačiť dokázať ju pre základné prvky a to, že každý operátor ju zachováva.

Všimnime si tiež, že veta o klasickej matematickej indukcii je špeciálnym prípadom tejto vety: V prvom indukčnom kroku ukazujeme platnosť V pre jediný základný prvok (`0`), druhý hovorí, že jediný operátor (`S`) platnosť V zachováva.

3 Induktívna štruktúra s jednoznačnou konštrukciou

Ostáva nám ešte zovšeobecniť vetu o definícii matematickou indukciou. Tu však musíme byť opatrní, lebo nie každá induktívna štruktúra bude mať také vyhovujúce vlastnosti ako prirodzené čísla s jediným základným prvkom a jediným operátorom, kde môžeme každý prvok skonštruovať jediným spôsobom, a to niekoľkonásobnou aplikáciou operátora na základný prvok.

Napríklad pri množine slov so základnými prvkami `A` a `B` a operátormi o_1 a o_2 takými, že o_1 „prilepí“ na koniec svojho jediného vstupu znak `B` a o_1 na jeho začiatok znak `A`, je možné slovo `AB` vyjadriť dvoma rôznymi spôsobmi: $\texttt{AB} = o_1(\texttt{A}) = o_2(\texttt{B})$. Takýmto induktívnym štruktúram sa budeme snažiť vyhnúť, čo vyjadríme nasledujúcimi definíciami:

Definícia 4
Hovoríme, že prvok p induktívnej štruktúry U má jednoznačnú konštrukciu, *ak platí:*

1. *Ak p je základný prvok, nemožno ho vyjadriť v tvare $o(t_0, \ldots, t_n)$, kde o je operátor a $t_0, \ldots, t_n$ sú prvky U.*
2. *Zo vzťahu $p = o_1(t_0^1, \ldots, t_{n_1}^1) = o_2(t_0^2, \ldots, t_{n_2}^2)$, kde o_1 a o_2 sú operátory a $t_0^1, \ldots, t_{n_1}^1$ a $t_0^2, \ldots, t_{n_2}^2$ sú prvky U, vyplýva $o_1 = o_2$, $n_1 = n_2$ a $t_i^1 = t_i^2$ pre každé $i \in \{0, \ldots, n_1\}$.*

Hovoríme, že induktívna štruktúra má jednoznačnú konštrukciu, *ak každý jej prvok má jednoznačnú konštrukciu.*

Podmienka 1 síce vyjadruje intuitívne zrejmú myšlienku, že základný prvok má právo na svoj názov, má však svoje opodstatnenie: Napríklad prirodzené čísla možno považovať aj za induktívnu štruktúru s dvoma základnými prvkami 0 a 1 a jediným operátorom `S`, potom však základný prvok 1 možno vyjadriť aj v „nezákladnom“ tvare $\texttt{S}(0)$.

Najjednoduchším príkladom induktívnej štruktúry s jednoznačnou štruktúrou sú, ako sme už naznačili, prirodzené čísla (s `0` a `S`). Ďalšími príkladmi sú výroky z výrokového počtu a termy či formuly z predikátového počtu. Naproti tomu napríklad relačná algebra nemá jednoznačnú konštrukciu, veď hľadanie

najefektívnejšej vytvárajúcej postupnosti je podstatou práce databázového stroja.

Uvedomme si, že z tejto definície vyplýva, že v induktívnej štruktúre s jednoznačnou konštrukciou musia ľubovoľné dve najkratšie vytvárajúce postupnosti toho istého prvku obsahovať rovnaké členy (možno v inom poradí). Hovorí o tom dôsledok nasledujúcej lemy:

Veta 5
Nech U je induktívna štruktúra s jednoznačnou konštrukciou a p jej prvok. Ak je $\langle p_0, \ldots, p_k \rangle$ nejaká najkratšia vytvárajúca postupnosť prvku p a $\langle q_0, \ldots, q_m \rangle$ je nejaká ďalšia (nie nutne najkratšia) vytvárajúca postupnosť prvku p, tak platí vzťah $\{p_0, \ldots, p_k\} \subseteq \{q_0, \ldots, q_m\}$.

Dôkaz. Urobíme ho indukciou (t. j. podľa vety 4):

Ak p je základný prvok, jeho jediná najkratšia vytvárajúca postupnosť je $\langle p \rangle$. Každá iná jeho vytvárajúca postupnosť však ho zrejme musí obsahovať (a to na svojom poslednom mieste). Tvrdenie teda pre základný prvok platí.

Nech p má tvar $o(t_0, \ldots, t_n)$ a nech $\langle q_0, \ldots, q_m \rangle$ je ľubovoľná jeho vytvárajúca postupnosť. Z definície vytvárajúcej postupnosti muselo $p = q_m$ vzniknúť aplikáciou nejakého operátora o' na prvky $q_{i_0}, \ldots, q_{i_r}$, t. j. $p = o'(q_{i_0}, \ldots, q_{i_r})$. Vzhľadom na jednoznačnosť konštrukcie p musí platiť $o' = o$, $r = n$ a $q_{i_j} = t_j$. Postupnosť $\langle q_0, \ldots, q_{i_j} \rangle$ je teda vytvárajúcou postupnosťou prvku t_j. Nech $\langle p_0^j, \ldots, p_{k_j}^j \rangle$ je (ľubovoľná) najkratšia vytvárajúca postupnosť prvku t_j, podľa indukčného predpokladu je $\{p_0^j, \ldots, p_{k_j}^j\} \subseteq \{q_0, \ldots, q_{i_j}\} \subseteq \{q_0, \ldots, q_m\}$. Keďže $p = q_m$, platí aj $\{p_0^0, \ldots, p_{k_0}^0\} \cup \cdots \cup \{p_0^n, \ldots, p_{k_n}^n\} \cup \{p\} \subseteq \{q_0, \ldots, q_m\}$. Pričom zrejme postupnosť $\langle p_0^0, \ldots, p_{k_0}^0, \ldots, p_0^n, \ldots, p_{k_n}^n, p \rangle$ a tiež postupnosť $\overline{p}$, ktorá z nej vznikne odstránením prípadných nie prvých výskytov jej členov, sú vytvárajúce postupnosti prvku p. Ukázali sme teda, že všetky členy vytvárajúcej postupnosti $\overline{p}$ prvku p sú členmi ľubovoľnej inej jeho vytvárajúcej postupnosti.

Treba ešte ukázať, že $\overline{p}$ je najkratšia vytvárajúca postupnosť prvku p a že má rovnaké členy ako ľubovoľná iná najkratšia vytvárajúca postupnosť $\overline{r}$ prvku p. Ak označíme P množinu všetkých členov postupnosti $\overline{p}$ a R množinu všetkých členov postupnosti $\overline{r}$, stačí ukázať, že $P = R$. V prechádzajúcom odseku sme ukázali, že platí $P \subseteq R$. Z toho, že $\overline{r}$ je najkratšia, zasa vyplýva, že $|P| \geq |R|$. Pretože P aj R sú konečné, platí požadované $P = R$.

Tvrdenie je tak dokázané aj pre nezákladné prvky. $\square$

A dôsledok:

Veta 6
Každé dve najkratšie vytvárajúce postupnosti prvku z induktívnej štruktúry s jednoznačnou konštrukciou obsahujú rovnaké členy.

Teraz už môžeme vysloviť a dokázať zovšeobecnenie vety 2 pre induktívne štruktúry s jednoznačnou konštrukciou:

Veta 7 (o definícii matematickou indukciou)
Nech M a A sú množiny a nech $U \subseteq M$ je induktívna štruktúra s jednoznačnou konštrukciou daná základnými prvkami $\{b_i : i \in I\}$ a operátormi $\{o_j : j \in J\}$, kde $o_j : D_{o_j} \to M$ pre nejaké prirodzené n_j a $D_{o_j} \subseteq M^{n_j+1}$. Nech $\{c_i : i \in I\}$ sú konštanty z A a $\{g_j : j \in J\}$ sú funkcie také, že $g_j : D_{o_j} \times A^{n_j+1} \to A$. Potom existuje jediná funkcia $f : U \to A$ taká, že platí:

1 Ak $p = b_i$, tak $f(p) = c_i$.

2 Ak $p = o_j(t_0, \ldots, t_{n_j})$, tak $f(p) = g_j(t_0, \ldots, t_{n_j}, f(t_0), \ldots, f(t_{n_j}))$.

Dôkaz. Vezmime množinu F všetkých zobrazení h takých, že:

0 definičný obor D_h funkcie h je podmnožinou U a s každým prvkom D_h patria do D_h aj všetky členy jeho najkratšej vytvárajúcej postupnosti (podľa predchádzajúcej vety ktorejkoľvek),

1 ak $p = b_i \in D_h$, tak $h(p) = c_i$,

2 ak $p = o_j(t_0, \ldots, t_{n_j}) \in D_h$, tak $h(p) = g_j(t_0, \ldots, t_{n_j}, h(t_0), \ldots, h(t_{n_j}))$.

Množina F je neprázdna, lebo do nej určite patrí napríklad prázdna funkcia, ale i zobrazenie $\{\langle b_i, c_i \rangle : i \in I\}$.

Všimnime si, že pre každé dve zobrazenia h_1 a h_2 z F platí, že na prieniku svojich definičných oborov sa zhodujú. To dokážeme indukciou podľa zložitosti prvku p:

Ak $p = b_i$ je v oboch definičných oboroch, tak (podľa bodu 1 pre h_1 aj h_2) platí $h_1(p) = c_i = h_2(p)$. Ak $p = o_j(t_0, \ldots, t_{n_j})$, tak zrejme všetky t_k majú menšiu zložitosť, preto pre ne platí $h_1(t_k) = h_2(t_k)$. Potom však (podľa bodu 2 pre h_1 i h_2) máme $h_1(p) = g_j(t_0, \ldots, t_{n_j}, h_1(t_0), \ldots, h_1(t_{n_j})) = g_j(t_0, \ldots, t_{n_j}, h_2(t_0), \ldots, h_2(t_{n_j})) = f_2(p)$.

Ak teda vezmeme zjednotenie f všetkých zobrazení z F, bude to tiež funkcia. Ukážeme teraz, že do definičného oboru tohoto f patrí každý prvok z množiny U. Ak nie, nech $p_{\min}$ je prvok minimálnej zložitosti, ktorý tam nepatrí (ak by ich bolo viac, môžeme vziať ľubovoľný z nich). Zrejme $p_{\min}$ nie je základný prvok (ukázali sme totiž, že zobrazenie $\{\langle b_i, c_i\rangle : i \in I\}$ patrí do F, teda je podmnožinou f), preto musí byť $p = o_j(t_1, \ldots, t_{n_j})$. Nech pre každé $k \in \{0, \ldots, n_j\}$ je h_k funkcia z F, ktorá má v definičnom obore prvok t_k. Teraz stačí definovať funkciu h ako zjednotenie všetkých h_k zväčšené o dvojicu $\langle p_{\min}, g_j(t_0, \ldots, t_{n_j}, h_0(t_0), \ldots, h_{n_j}(t_{n_j}))\rangle$. Keďže každý člen najkratšej vytvárajúcej postupnosti prvku p okrem samotného p je členom vytvárajúcej postupnosti niektorého t_k (inak by sme ho z postupnosti vylúčili, výsledná postupnosť by bola kratšia a ostala by vytvárajúcou pre p), ľahko vidieť, že táto funkcia h spĺňa všetky podmienky patrenia do F – jej definičný obor je podmnožinou U, s každým prvkom definičného oboru do definičného oboru patria aj všetky členy jeho najkratšej vytvárajúcej postupnosti, pre jeho základné prvky platí bod 1 a pre ostatné bod 2, a to včítane $p_{\min}$, lebo platí $h(p_{\min}) = g_j(t_0, \ldots, t_{n_j}, h_0(t_0), \ldots, h_{n_j}(t_{n_j})) = g_j(t_0, \ldots, t_{n_j}, h(t_0), \ldots, h(t_{n_j}))$. Našli sme teda funkciu z F, ktorej definičný obor obsahuje $p_{\min}$, čo znamená, že $p_{\min}$ patrí aj do definičného oboru celého f, a to je spor. Definičný obor zobrazenia f je teda naozaj celá množina U.

Dokážeme ešte sporom, že táto funkcia f je jediná: Nech existuje ešte iná vyhovujúca funkcia, označme ju h. Nech $p_{\min}$ je (ľubovoľný) prvok najmenšej zložitosti taký, že $f(p_{\min}) \neq h(p_{\min})$. Zrejme $p_{\min}$ nie je základný prvok, lebo pre tie (podľa bodu 1 pre f i h) platí $f(b_i) = c_i = h(b_i)$. Ak však $p_{\min} = o_j(t_0, \ldots, t_{n_j})$, nutne všetky t_k majú zložitosti menšie ako $p_{\min}$, a teda pre ne platí $f(t_k) = h(t_k)$. Potom však (podľa bodu 2 pre f i h) máme $f(p_{\min}) = f(o_j(t_0, \ldots, t_{n_j})) = g_j(t_0, \ldots, t_{n_j}, f(t_0), \ldots, f(t_{n_j})) = g_j(t_0, \ldots, t_{n_j}, h(t_0), \ldots, h(t_{n_j})) = h(o_j(t_0, \ldots, t_{n_j})) = h(p_{\min})$, čo je spor. $\square$

Ak teda v induktívnej štruktúre s jednoznačnou konštrukciou budeme chcieť definovať nejakú funkciu pre všetky jej prvky, stačí definovať ju pre jej základné prvky a ukázať, ako určiť hodnotu v nezákladnom prvku z hodnôt

v jeho podprvkoch.

Všimnime si opäť, že klasická veta o definícii matematickou indukciou je špeciálnym prípadom tejto vety. V prvom indukčnom kroku definujeme hodnotu v jedinom základnom prvku `0`, v druhom správanie jediného operátora `S`.

Literatúra

BŠ Balcar B., Štěpánek P.: Teorie množin, Academia, Praha, 1986

GJ Goldstern M., Judah H.: The Incompleteness Phenomenon, A New Course in Mathematical Logic, A K Peters, Wellesley, Massachusetts, 1995

Image Compression Using Space-Filling Curves

Michal Krátký, Tomáš Skopal, Václav Snášel

Department of Computer Science, VŠB-Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava, Czech Republic
{michal.kratky, tomas.skopal, vaclav.snasel}@vsb.cz

Abstract. The classical approaches within the area of image compression treat the image as a set of rows or columns. In our paper we introduce an idea of using space-filling curves for the purposes of image compression and generally for purposes of image processing. We show that topological properties of space-filling curves can affect the compression ratios as well as the image quality. We present an application of space-filling curves for both the lossless and the lossy type of image compression.

1 Introduction

The main task of the research within the area of Image Compression and Image Processing is to find and exploit a semantics and structure hidden in the images. Such a valuable information is necessary for advanced image processing.

So far, the research efforts were usually directed to various heuristic algorithms, more or less "made-to-measure" for a specific task. There was developed a plenty of algorithms of image compression and processing. However, they are rarely based on some general formal method. In this paper we want to focus on the topological properties of space filling curves and use its formal model for the image compression purposes.

2 Space-Filling Curves

Space-filling curves became a topic of interest to Peano and Hilbert in the late 19th century. The initial motivation for space-filling curves was a question (studied in the set theory) if there exists a bijective mapping between a square and a finite line. The answer was 'yes'. Every such mapping defines a space-filling curve (square filling curve respectively). In fact, we can say that a space-filling curve just linearly orders all the points within the space. For more details about space-filling curves see the monograph by Hans Sagan [6].

In computation models, data are stored within finite and discrete spaces. Discrete versions of space-filling curves are widely used in various areas of computer science, especially in the domain of Information Retrieval where the space-filling curves serve as a tool for data clustering [4]. We have also tried to use the space-filling curves in Image Recognition [7].

An example of six space-filling curves is depicted in Figure 1. Note that every discrete finite space-filling curve is just a geometric interpretation of a linear order on the points within a discrete finite space.

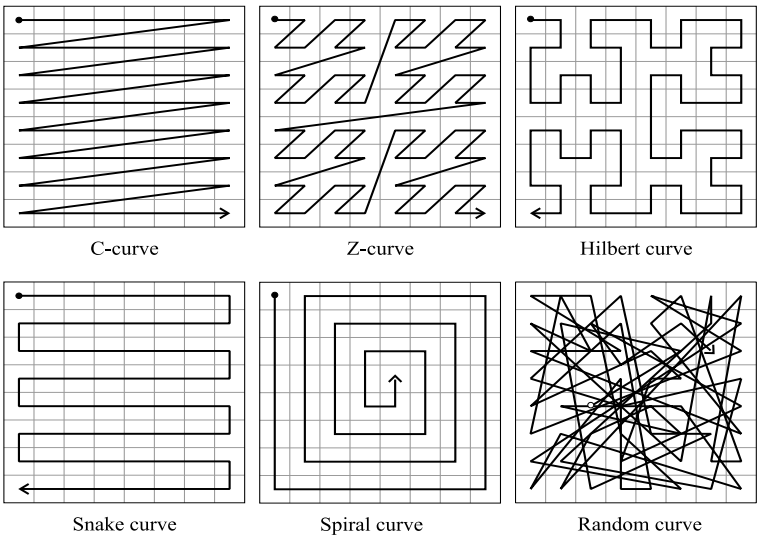


Fig. 1. Two-dimensional space-filling curves.

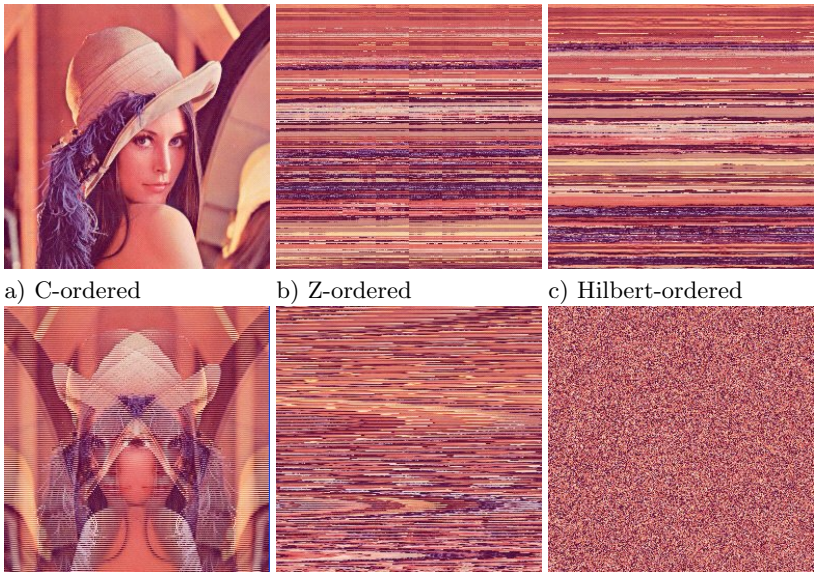
2.1 Our Motivation

Why do we focus on the space-filling curves? Well, they have an interesting property – they can make a multi-dimensional space single-dimensional. In other words, an arbitrary point of a multi-dimensional space can be characterized by a single integer, *address*, without loss of an information (in sense of bijective mapping).

In the area of image processing we can use the space-filling curves as a function that transforms a two-dimensional image to a linear sequence of values, i.e. to a single-dimensional signal. Regardless of a concrete method, the process of image compression must in the final always scan the image pixel by pixel. This is usually realized by scanning the image row by row (column by column respectively). If we realize, this "classical" scanning method corresponds with scanning the image along the C-curve (see the first curve in Figure 1).

The space-filling curves allow us to generalize the image scanning process, since the pixels could be scanned according to various types of space-filling curves, not only the C-curve. In the following sections we will present some experimental results showing that there exist more suitable space-filling curves for both – the lossless and the lossmaking type of compression.

In Figure 2 is visualized the pixel ordering of the classic "Lena" image (24-bit RGB, $256 \times 256 = 65536$ pixels). The sequences of pixel ordered by the curves are horizontaly wrapped as if they were images. This wrapped images serve solely for the visualization purposes – for image compression the sequences remain single-dimensional. Note that C-ordered Lena is in the "wrapped mode" identical to the original image.



a) C-ordered b) Z-ordered c) Hilbert-ordered

d) Snake-ordered e) Spiral-ordered f) Random-ordered

Fig. 2. Visualization of the pixel ordering.

2.2 Measuring the Quality of Space-Filling Curves

In general, every bijective mapping of a two-dimensional space Ω onto a single-dimensional interval I can be interpreted as a space-filling curve. Even the random ordering of points within a space actually defines a space-filling curve. However, not every such curve can be useful for computation purposes, e.g. the above mentioned random ordering will hardly have any application due to its low topological quality (in terms of digital topology, see [5]). But what is the quality differentiating the space-filling curves? Actually, we will figure out when we look a little bit closer.

The fundamental property of many space-filling curves is that they partially preserve a metric (in this paper we will consider the Euclidean L_2 metric[1]). The random curve does not preserve metric at all but (as we will show later) e.g. the Hilbert curve satisfies high degree of metric preservation. The metric preservation requirement can be formulated as follows:

Every two points are "close" in space Ω if and only if they are "close" on the space-filling curve.

This requirement is often measured through *locality* of space-filling curve. In [2], authors pay attention to general metric properties of space-filling curves, refering mainly to methods of measuring the locality.

[1] Common distance measure $L_2 = d(o^1, o^2) = \sqrt{(o_x^1 - o_x^2)^2 + (o_y^1 - o_y^2)^2}$

In order to calculate the degree of metric preservation (locality) for our specific purposes, we decided to observe possible anomalies within the shape of a discrete space-filling curve. The anomalies are of two types – *distance shrinking* and *distance enlargement* (see Figure 3).

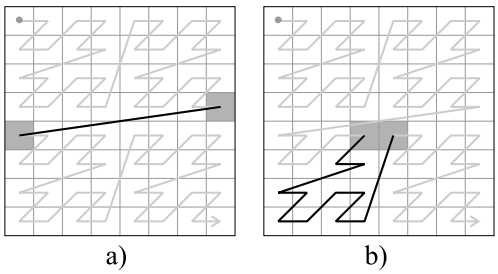


Fig. 3. a) Dist. shrinking – points that are "close" on the curve are "distant" in the space. b) Dist. enlargement – points that are "close" in space are distant on the curve.

In fact, the *distance shrinking* (a kind of "curve jumping") violates the "$\Leftarrow$" direction while the *distance enlargement* violates the "$\Rightarrow$" direction of the above formulated preservation requirement. Some number of the *distance enlargement* occurrences must appear in the shape of every space-filling curve. However, there exist space-filling curves without an occurrence of the *distance shrinking* anomaly, e.g. the Hilbert curve.

Symmetry of Space-Filling Curves. The first criterion in our paper attempting to materialize the concept of space-filling curve quality is the *symmetry*. We have borrowed this criterion from the area of Information Retrieval where data clustered using a highly symmetric space-filling curve are retrieved faster.

The basis of the symmetry measuring relies in overall calculation of the anomalies separately according to both dimensions. Very loosely speaking, if the number of anomalies in one dimension is close to the number of anomalies in the second dimension then the curve is symmetric. For comprehensive discussion on the space-filling curves symmetry we refer to [3].

The result is that some space-filling curves are less or more symmetric and thus less or more suitable for clustering purposes. We show an ordering for our six curves according to the symmetry:

C-curve (less symmetric)= Snake curve < Random curve < Z-curve < Spiral curve < Hilbert curve (most symmetric)

Later in this paper, we show experimental results saying how can highly symmetric curves improve the image compression.

Jumping Factor. In our approach we have proposed an alternative indicator of space-filling curve quality based on overall calculation of the distance shrinking

anomalies. This indicator – *jumping factor* – actually measures the number of "jumps" while scanning the image along a given space-filling curve. Note that the "jumps" appear in the Figure 1 as oblique lines.

We show an ordering for our six curves according to the jumping factor:

$$\text{Hilbert curve} = \text{Spiral curve} = \text{Snake curve (no jump)} < \text{C-curve} < \text{Z-curve} < \text{Random curve (most jumps)}$$

Later in this paper, we show experimental results saying how can the curves with low jumping factor improve the image compression.

3 Lossless Compression

Our experiments present the result of using space-filling curves for lossless compressions. We have tried to show the benefits of using the space-filling curves for image processing, our focus was not aimed to develop a sophisticated stand-alone compression method. We have tried to show the meaning of the curves for image processing. We have realized two experiments for the lossless compressions. First, the measurement of a redundancy in image and second, the LZW compression. For both experiments we chose the "Lena" image.

3.1 Image Redundancy

The first experiment measures the redundancy in the image. The image is being scanned according to the space-filling curve and the number of consecutive pixels with the same color is returned. It corresponds to the well-known RLE compression method. The image is processed for the whole 24-bit color and separately for each of the color components. The result size is a sum for a color with the same number of repeatings.

3.2 Experimental Results

The Figure 4 presents the results of the experiments. The Table 1 shows the values for each space-filling curve. Except of the size, the values in the table represent the relative differences to the values for the C-curve (δ_c [%]). The result of the experiments is a "preferability" order of the space-filling curves. The best results (the bold values) achieve the Hilbert and Spiral curves, the worst are the Z-curve and Random curve. Are we able to say which space-filling curve is better? Let us follow to the second experiment.

3.3 LZW

We have used the well-known LZW compression method for the second experiment. The method was applied to each of the color components. The Table 2 presents the size of encoded files. The best results show the Hilbert and Spiral curves and the worst, again, is the Random curve.

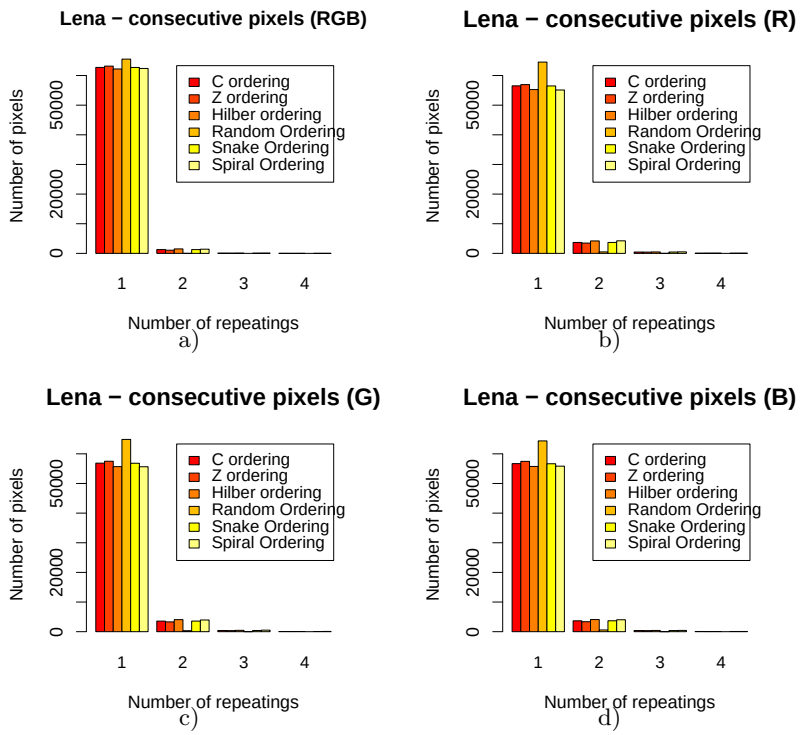


Fig. 4. The number of consecutive repeating colors for a) RGB b) red component c) green component d) blue component.

3.4 Summary

Let us try to resume the results. Which space-filling curve is the best? In other words – which curve does the best approximation of the topology? It obviously depends on the encoded image but we may say that the choice of space filling curve is important. The best result shows the Hilbert and the Snake orderings. Is it possible to determine the quality of the space-filling curve more deeply? The experiments on the lossy image compression could answer our questions.

4 Lossy Compression

In this section we want to demonstrate how the usage of space-filling curves can affect the techniques of lossy image compression. Again, we did not intend to propose a completely new method of lossy image compression which can be generally applicable. We just wanted to reveal the advantages of scanning the images along certain space-filling curves.

Curve (Ordering)	RGB			Sum of color components		
	Size	δ_c [%]	Order	Size	δ_c [%]	Order
C-curve	192.258		4th	182.445		4th
Z-curve	192.939	+0.35	5th	183.409	+0.53	5th
Hilbert curve	**191.430**	**-0.43**	**1st**	**180.180**	**-1.24**	**1st**
Random curve	196.599	+2.26	6th	194.576	+6.65	6th
Snake curve	192.240	-0.01	3rd	182.384	-0.03	3rd
Spiral curve	191.688	-0.30	2nd	180.541	-1.04	2nd

Table 1. The results of measuring the redundancy in lena image.

Curve (Ordering)	Lena		
	Size	δ_c [%]	Order
C-curve	205.463		4th
Z-curve	201.415	-1.97	3rd
Hilbert curve	**194.031**	**-5.56**	**1st**
Random curve	253.771	+23.51	6th
Snake curve	205.967	+0.25	5th
Spiral curve	201.351	-2.00	2nd

Table 2. The results for the LZW compression.

4.1 Delta Compression

As a representative, we have chosen a variant of the DPCM (Differential Pulse Code Modulation) coding [8], the lossy delta-compression respectively. The *delta compression* is based on a simple assumption that a signal changes smoothly over the time. Then storing the differences (deltas) between individual sample values (instead of storing the sample values themselves) leads to better entropy of the output sequence of values.

Modification for the images relies just in the transformation of the two-dimensional raster to a single-dimensional signal. In this subsection we inspect the phase of "transformation" by using space-filling curves. It is necessary to say that image delta compression is suitable only for images like "photos" where the pixel neighbourhood changes only a little.

The *lossy* delta compression is limited with a maximal absolute size (maximum delta parameter) of the difference value – greater/lower values are aligned to the maximal/minimal allowed difference. This limitation may cause some losses that appear as error pixels by the image reconstruction (decoding). On the other hand, reasonable reduction of the number of possible differences leads to better entropy.

Average entropy of a signal is computed as the average number of bits per value. For a sequence $\mathcal{S}$ the entropy is computed as:

$$entropy(\mathcal{S}) = -\sum_{v \in \mathcal{S}} \frac{freq(v, \mathcal{S})}{|\mathcal{S}|} * log_2(\frac{freq(v, \mathcal{S})}{|\mathcal{S}|})$$

where $freq(v, \mathcal{S})$ is the number of occurrences of value v in the sequence $\mathcal{S}$

4.2 Experimental Results

The experiments were realized on several photos but for the lack of space in this paper we present only one test – the red component of the "Lena" image. The "maximum delta" parameter was set to ± 31.

The testing process consisted of three steps for each space-filling curve. First, the image was ordered using a space-filling curve. Second, the delta encoding was applied on the sequence of pixels. Finally, the histogram was created. See the process schema in Figure 5.

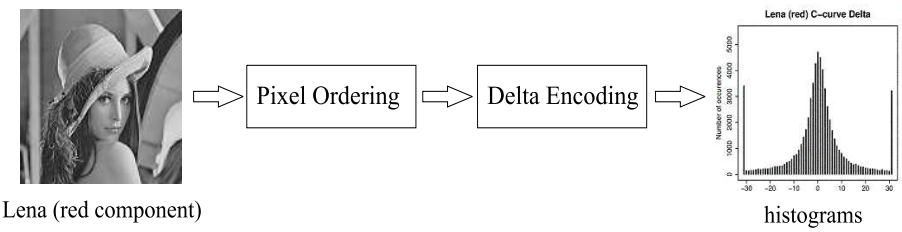


Fig. 5. Process of the delta encoding.

The output histograms carry the two characteristics we wanted to investigate (see Figure 6) – the entropy and the image quality. Entropy can be observed from the height and sharpness of the histogram "bell". The image quality is determined by the border values of the histograms. Frequent occurrences of the maximal/minimal differences prompt worse image quality because many values were aligned (and thus distorted) by the delta encoding.

The overall results are overviewed in Table 3. The first two columns contain the properties for the space-filling curves. Next two columns show error pixels (total count and percentage) and the last columns show achieved entropy (bits per pixel per color, total bytes per color and percentage). The bold values are the best. The visualization of error pixels is depicted in Figure 7 and provides

Lena Delta ±31 (red component)	**Curve Characteristics**		**Errors Pixels**		**Compression Rate (entropy)**			Winning order
	jumps	*symmetry*	*pixels*	*ratio*	*bpp*	*bytes*	*ratio*	
C-curve	little	low	6647	10.14%	5.14	42130	64,29%	5th
Z-curve	much	high	5096	9.01%	5.19	42542	64,91%	3rd
Hilbert curve	**none**	**very high**	**4211**	**6.42%**	5.03	41251	62,94%	**1st**
Snake curve	**none**	low	6365	9.71%	5.14	42069	64,19%	4th
Spiral curve	**none**	high	4493	6.86%	**4.99**	**40914**	**62,43%**	2nd
Random curve	very much	moderate	39205	59.82%	3.95	32331	49,33%	6th

Table 3. Results for the lossy compression.

subjective view.

4.3 Summary

The experiments have shown that different types of space-filling curves have different influence on both the image quality and the compression ratio (entropy respectively). In general, we can say that space-filling curves with high degree of symmetry and with low jumping factor report good results when applied on the lossy image compression.

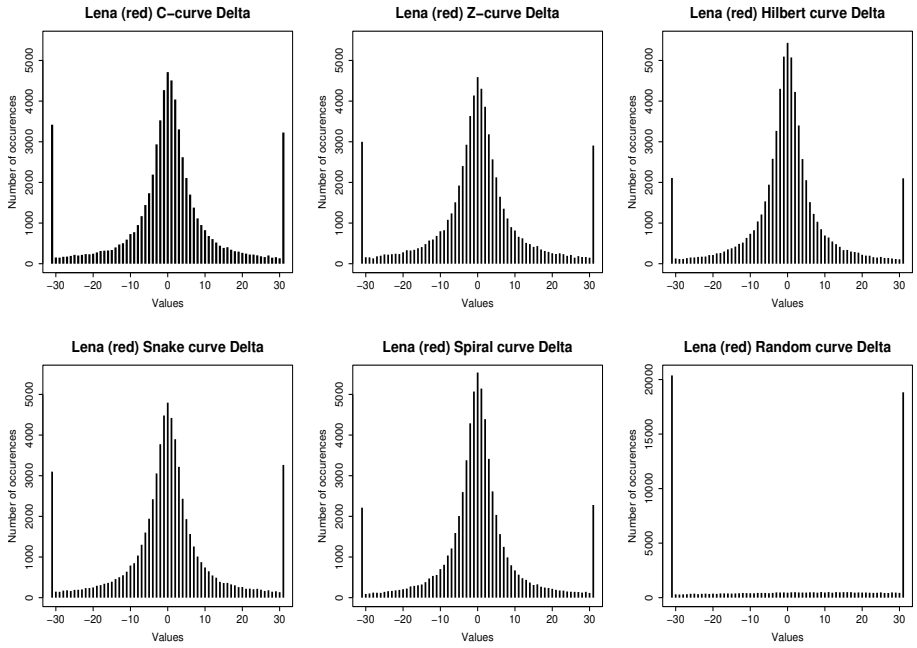


Fig. 6. Histograms of delta values.

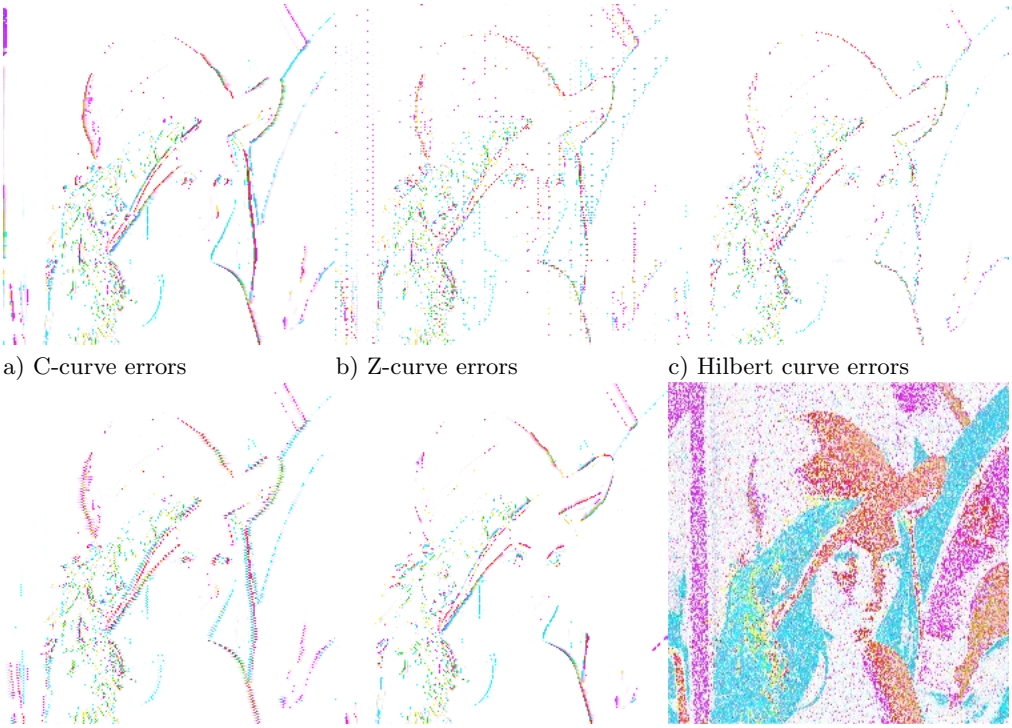
5 Conclusions and Outlook

We have shown that usage of the space-filling curves can positively affect the compression ratio and image quality. This is because they can reflect some common structure hidden in the images. Experimental results have shown that space-filling curves satisfying the high degree of symmetry together with the low jumping factor approximate well the metric. We offer a hypothesis that the "good" metric preservation causes also "good" reflection of the image structure.

However, there remain some questions open. Does exist an ideal space-filling curve that achieves good results for all images? Can be mathematically formalized the relation between digital topologies and space-filling curves? This questions are the subject of our future work.

References

1. Bayer R.: The Universal B-Tree for multidimensional indexing: General Concepts. In: *Proc. Of World-Wide Computing and its Applications 97 (WWCA 97)*. Tsukuba, Japan, 1997.



a) C-curve errors b) Z-curve errors c) Hilbert curve errors

d) Snake curve errors e) Spiral curve errors f) Random curve errors

Fig. 7. Errors in reconstructed images.

2. Gotsman C., Lindenbaum M.: On the Metric Properties of Discrete Space-Filling Curves, *IEEE International Conference on Pattern Recognition*, Jerusalem, Israel, 1994
3. Markl, V.: *Mistral: Processing Relational Queries using a Multidimensional Access Technique*, Ph.D. thesis, Technical University Munchen, `http://mistral.in.tum.de`, 1999
4. Moon B., Jagadish H.V., Faloutsos C., Saltz J.H.: Analysis of the Clustering Properties of Hilbert Space-filling Curve, *IEEE Transactions on Knowledge and Data Engineering*, 1996
5. Marchand-Maillet S., Sharaiha M.Y.: *Binary Digital Image Processing.* Academic Press, 2002.
6. Sagan H.: *Space-Filling Curves*, Springer-Verlag, 1994
7. Skopal T., Krátký M., Snášel V.: Image Recognition Using Finite Automata. *Prague Stringology Conference '02*, Prague, Czech Republic, September 2002
8. Subramanya S.R., Youssef A.: Performance Evaluation of Lossy DPCM Coding of Images using Different Predictors, *ISCA CAINE98 Conference*, Las Vegas, Nevada, November 1998.

Indexovanie pre plnotextový stroj využívajúci relačný databázový model

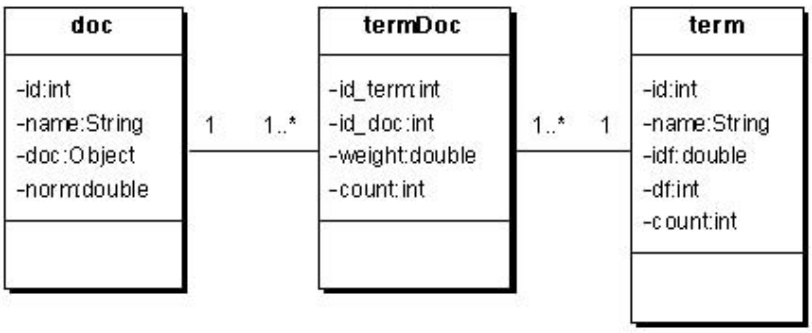
Rastislav Lencses

Ústav informatiky, Prírodovedná fakulta,
Univerzita P. J. Šafárika v Košiciach,
Jesenná 5, 040 01, Košice,
Slovenská republika,
lencses@science.upjs.sk

Abstrakt V príspevku sa zaoberáme indexovaním kolekcie dokumentov pre plnotextový stroj. Popíšeme pôvodný algoritmus na generovanie indexu, ktorý porovnáme so známymi algoritmami a odhadneme jeho časovú zložitosť. Taktiež podáme presné definície pojmov používaných v oblasti plnotextového vyhľadávania.

1 Úvod

Už v čase vytvorenia prvej komerčnej relačnej databázy vznikli prvé práce týkajúce sa využitia relačného databázového modelu podporovaného jazykom SQL pre potreby plnotextového vyhľadávania : [9] - Macleod v roku 1978, [2] - Crawford v roku 1981. Novšia práca je napríklad [8]. Základ spočíva vo vytvorení troch relačných tabuliek, nad ktorými sa pomocou SQL dá dopytovať (obr. 1).



Obrázok 1. Databázové relácie pre plnotextové vyhľadávanie

Typický SQL dopyt nad uvedenou schémou potom vyzerá nasledovne - nájdenie dokumentov, ktoré obsahujú termy 'Adam' alebo 'Afrika':

```
SELECT doc.name from doc, term, termDoc td
    where term.id=td.id_term and doc.id = td.id_doc and
    (term.name='Adam' or term.name='Afrika')
```

V druhej časti príspevku podávame presné definície pojmov používané v oblasti plnotextového vyhľadávania (information retrieval). V tretej časti definujeme index - invertovaný zoznam, ktorý sa používa na podporu rýchleho vyhľadávania. V štvrtej časti popisujeme známe algoritmy tvorby indexu a uvádzame pôvodný algoritmus, ktorý nepotrebuje niektoré fázy nutné pre iné algoritmy.

2 Definície

Plnotextový stroj (fulltext engine) musí vykonávať dve základné činnosti: vytvoriť si index nad kolekciou dokumentov a vedieť nad touto kolekciou za pomoci indexu rýchlo vyhľadávať dokumenty relevantné voči užívateľovej otázke. Dokumenty sú neštruktúrované, obsahujú slová.

Slová zo všetkých dokumentov dajme do postupnosti S

$$S = s_h, h \in [1, m_S]$$

kde m_S je počet jedinečných slov cez všetky dokumenty. Kolekcia dokumentov je formálne postupnosť

$$D_i, i \in [1, n]$$

kde n je počet dokumentov. Dokument D_i je postupnosť

$$(s_h^i)_k, k \in [1, m_{D_i}]$$

kde m_{D_i} je počet slov v dokumente D_i. $(s_h^i)_k$ je teda h-te slovo postupnosti s_h nachádzajúce sa v dokumente D_i na k-tej pozícii.

V plnotextovom stroji dokumenty sú reprezentované pomocou postupnosti termov. Term je zvyčajne slovo nachádzajúce sa v dokumente, môže však byť redukovaný na základný tvar, s oddelenou predponou alebo príponou. Všetky termy teda umiestnime do postupnosti T

$$T = t_j, j \in [1, m_T]$$

kde m_T je počet jedinečných termov v celej postupnosti dokumentov D_i. Vzťah medzi slovami a termami určuje funkcia g:

$$g : S \rightarrow T,\ g(s_h) = t_j$$

ak t_j je základný tvar slova s_h. Zatiaľ neuvažujeme oddelenie prípony alebo predpony, takže $T \subseteq S$.

Existuje viacero teoretických modelov, ktoré sa môžu využiť na návrh reprezentácie dokumentov a výpočet funkcie priradzujúcej otázke množinu dokumentov. Podľa vektorového modelu ([13]) sa dokumenty reprezentujú ako vektory

v priestore s dimenziou rovnou počtu jedinečných termov. Každá zložka tohto vektora obsahuje váhu, ktorou daný term prislúchajúci tejto zložke prispieva ku reprezentácii dokumentu. Najprv definujme inverznú frekvenciu dokumentu

$$\mathrm{idf}_j = \log(\frac{n}{\mathrm{df}_j})$$

kde df_j je počet dokumentov (document frequency), ktoré obsahujú term t_j. Ďalej definujme maticu A veľkosti $m_T \times n$, ktorej zložky w_{ij} sú definované nasledovne

$$w_{ij} = \mathrm{tf}_{ij} \,.\, \mathrm{idf}_j$$

kde tf_{ij} je počet výskytov (term frequency) termu t_j v dokumente D_i.

Platí teda, že w_{ij} je váha termu t_j v dokumente d_i, čiže riadky matice A predstavujú (riedke) vektory zodpovedajúce dokumentom a stĺpce zodpovedajú termom. Uvedený vzťah na výpočet váhy termu v dokumente je základný. Neskôr boli nájdené (a experimentálne overené) zložitejšie vzťahy, napríklad [12]:

$$w_{ij} = \frac{(\log \mathrm{tf}_{ij} + 1.0) \,.\, \mathrm{idf}_j}{\sum_{j=1}^{n}((\log \mathrm{tf}_{ij} + 1.0) \,.\, \mathrm{idf}_j)^2}$$

Najdôležitejšou zmenou je tu prekonanie príliš veľkého vplyvu termu s vysokou frekvenciou.

Nájdenie dokumentov relevantných voči otázke znamená nájsť dokumenty obsahujúce termy otázky a vypočítať podobnosť týchto dokumentov voči otázke. Otázka sa tiež chápe ako dokument vnorený do m_T dimenzionálneho priestoru, ktorého váhy sa vypočítajú podobne, ako váhy ostatných dokumentov. Napr. podľa [12] sa zložky vektora zodpovedajúceho otázke vypočítajú nasledovne:

$$w_{qj} = (0.5 + \frac{0.5\,\mathrm{tf}_{qj}}{\max_{l \in L_q}(\mathrm{tf}_{ql})}) \,.\, \mathrm{idf}_j$$

kde $L_q = \{x \in [1, m_T], t_x \text{ je term patriaci otázke } q\}$

Pri určení relevancie teda ide o podobnosť dvoch vektorov. Tá sa môže vypočítať napr. ako ich vzdialenosť, alebo ako ich uhol. Podobnosť vypočítaná ako kosínus uhla dvoch vektorov:

$$\mathrm{sim}(q, D_i) = \frac{\sum_{j=1}^{m_T} w_{qj}.w_{ij}}{\sqrt{\sum_{j=1}^{m_T}(w_{ij})^2 \sum_{j=1}^{m_T}(w_{qj})^2}}$$

Definujme Q ako množinu možných otázok. Ďalej definujme funkciu f, ktorá pre danú otázku nájde množinu relevantných dokumentov.

$$f : Q \rightarrow \mathcal{P}(D_i)$$

kde $f(q) = \{D_i, \mathrm{sim}(q, D_i) > p, 0 \leq p \leq 1\}$. Prah p je zvyčajne rovný 0. Dokumenty v množine $f(q)$ sa dajú usporiadať vzhľadom ku ich relevancii ku otázke q.

3 Index

Dokumenty relevantné voči otázke môžeme vyhľadávať priamo prehľadávaním plného textu. Existuje viacero algoritmov, ktoré sú schopné účinne prehľadávať - Knuth-Moris-Pratt algoritmus, Boyer-Moore algoritmus a ďalšie. Tento prístup je však vhodný len pre malé množstvá dokumentov. Pre väčšie si musí plnotextový stroj vybudovať externú dátovú štruktúru - index, ktorý bude urýchľovať vyhľadávanie.

Existujú tri typy bežne využívaných indexov - invertované zoznamy, signatúry a stromy s príponami (suffix trees). Invertované zoznamy umožňujú menší vyhľadávací čas a menšie priestorové požiadavky než sa dá dosiahnuť u iných typov indexov.

Invertovaný zoznam obsahuje zoznam termov a pre každý term eviduje informáciu o vzťahu term - dokument. Invertovaný zoznam môže byť binárny - booleovský (eviduje sa výskyt termu v dokumente), rozšírený (eviduje sa počet výskytov) a úplný (eviduje sa presné miesto termu v dokumente).

Invertovaný zoznam I je

$$(t_j, Z_j), j \in [1, m_T]$$

Pre binárny invertovaný zoznam I môžeme Z_j definovať

$$Z_j = D_l^j, l \in [1, m_{Z_j}]$$

kde D_l^j sú tie dokumenty, v ktorých sa vyskytuje term t_j a m_{Z_j} je ich počet. Je tiež možné indexovať nie dokumenty, ale bloky. Celý zoznam dokumentov sa vtedy rozdelí na bloky rovnakej dĺžky (pričom blok teda môže obsahovať časti viacerých dokumentov). Dosiahne sa tým zníženie priestorovej náročnosti pre index (nižší počet bitov pre číslo označujúce index bloku).

Pre rozšírený invertovaný zoznam I môžeme Z_j definovať

$$Z_j = (D_l^j, k_l^j), k \in [1, m_{D_l}]\}$$

k_l^j je počet výskytov termu t_j v rámci dokumentu D_l^j.

Pre úplný invertovaný zoznam I môžeme Z_j definovať

$$Z_j = (D_l^j, (d_l^j)_r), r \in [1, m_{D_l}]$$

kde $(d_l^j)_r$ je postupnosť pozícií termu t_j v dokumente D_l^j.

Invertovaný zoznam musí ďalej mať nasledovné vlastnosti:

- neobsahuje zhodné prvky : ak $t_i = t_j$ tak $i = j$.
- index je utriedený : ak $i < j$ tak $t_i < t_j$.

V ďalšom sa budeme zaoberať rozšíreným invertovaným zoznamom.

Termy sa získajú z dokumentov rozobratím (parse) slov dokumentov, pričom jednotlivé slová sú oddelené štandardnými oddeľovačmi (medzera, čiarka, bodka

a podobne). Už v tejto fáze môžeme termy zhlukovať na základe morfologickej podobnosti. Ak budeme namiesto všetkých skloňovaných a časovaných tvarov termov evidovať iba ich zástupcu v základnom tvare, sémantika textu dokumentu sa zníži len minimálne. Výsledkom tohto zhlukovania bude nižší počet termov a nižšia časová náročnosť ďalších algoritmov.

Pre morfologické zhlukovanie sme využili voľne dostupný program *ispell*, pre ktorý existuje slovenský slovník [5]. Pomocou tohto programu sme pre všetky jemu známe základné tvary slov vygenerovali morfologicky odvodené tvary (skloňovaním, časovaním, stupňovaním).

S využitím tohto zoznamu sme potom schopní pre slovo v texte nájsť jeho základný tvar. Program ispell to vie samozrejme nájsť sám tiež, nie je však vhodné ho priamo využívať z dôvodu veľkej časovej réžie.

Vyvstáva tu samozrejme problém rozlíšenia základného tvaru slova pre zhodné odvodené tvary. Napríklad tvar *bankami* má dva základné tvary s rôznou sémantikou, ktoré sú rozlíšiteľné len v rámci kontextu - *bank* a *banka*. Po analýze frekvencie výskytov možnosti takýchto zámen (viď nižšie) sme tento problém zanedbali.

3.1 Konkrétne výsledky pri tvorbe indexu

Pre kolekciu novinových článkov denníka SME sme dosiahli nasledovný výsledok: kolekcia má okolo 25 000 dokumentov, veľkosť okolo 60 MB, počet jedinečných termov je 305 000, počet jedinečných termov po použití morfologického zhlukovania je 193 000 (63 percent).

Slovenský slovník programu *ispell* má približne 51 000 slov v základnom tvare, typická slovná zásoba človeka je okolo 20 000 slov. Pre 51 000 slov ispell vie vygenerovať vyše 1 360 000 tvarov. Z tohto počtu je okolo 10 500 prekrytí - pre nejaký skloňovaný/časovaný/stupňovaný tvar slova existuje viac základných tvarov. Z nich je okolo 60 percent prekrytí týkajúcich sa prísloviek a prídavných mien (tvar *biologickej* zodpovedá základným tvarom *biologicky* a *biologický*). Ďalšie prekrytia sú medzi podstatným menom skloňovaným podľa vzoru vysvedčenie a prídavným menom (tvar *chytení* zodpovedá základným tvarom *chytenie* a *chytený*). Ak by sme neuvažovali tieto prekrytia, v ktorých dochádza ku minimálnemu zníženiu sémantiky, ostalo by menej než 3000 prekrytí, ktoré sa zväčša viažu na sémanticky odlišné základné tvary. Pomer počtu prekrytí ku počtu všetkých tvarov by bol teda okolo 2.2 promile.

4 Vytváranie indexu

Invertovaný zoznam možno priamočiaro vytvoriť v pamäti použitím utriedeného poľa. Pole obsahuje termy a pre každý term evidujeme pole jeho výskytov (v ktorom dokumente sa koľkokrát nachádza). Počas procesu indexovania dokumentu z kolekcie je pre utriedené pole termov potom možné binárnym vyhľadávaním zistiť, či sa tam term z dokumentu vyskytuje. Ak áno, prehľadá sa pole jeho

výskytov a pridá/doplní sa výskyt termu v aktuálnom dokumente. Ak sa nevyskytuje, pridá sa do poľa termov a tiež sa pridá výskyt termu v dokumente. Algoritmus je na obr. 2.

V prípade väčších kolekcií dokumentov však tento prístup zlyhá kvôli nedostatku operačnej pamäti. Vzhľadom na to, že operačný systém bez znalosti algoritmu nedokáže poskytovať virtuálnu pamäť stránkovaním efektívne, je nutné použiť perzistenciu.

V literatúre [1] je popísaný algoritmus, ktorý buduje invertovaný zoznam vyššie popísaným algoritmom, kým nie je vyčerpaná operačná pamäť. Potom uloží vybudovaný zoznam termov a ich výskytov na disk, vyprázdni pamäť a opakuje to, kým nie sú spracované všetky dokumenty z kolekcie. Nasleduje fáza hierarchického zlučovania uložených čiastočných indexov. Ide o zlučovanie utriedených polí termov. Pre rovnaké termy stačí zoznamy výskytov jednoducho zlúčiť, keďže medzi nimi nie je prienik. Táto fáza zlučovania zaberá podľa výpočtov i testov 20 až 30 percent celkového času.

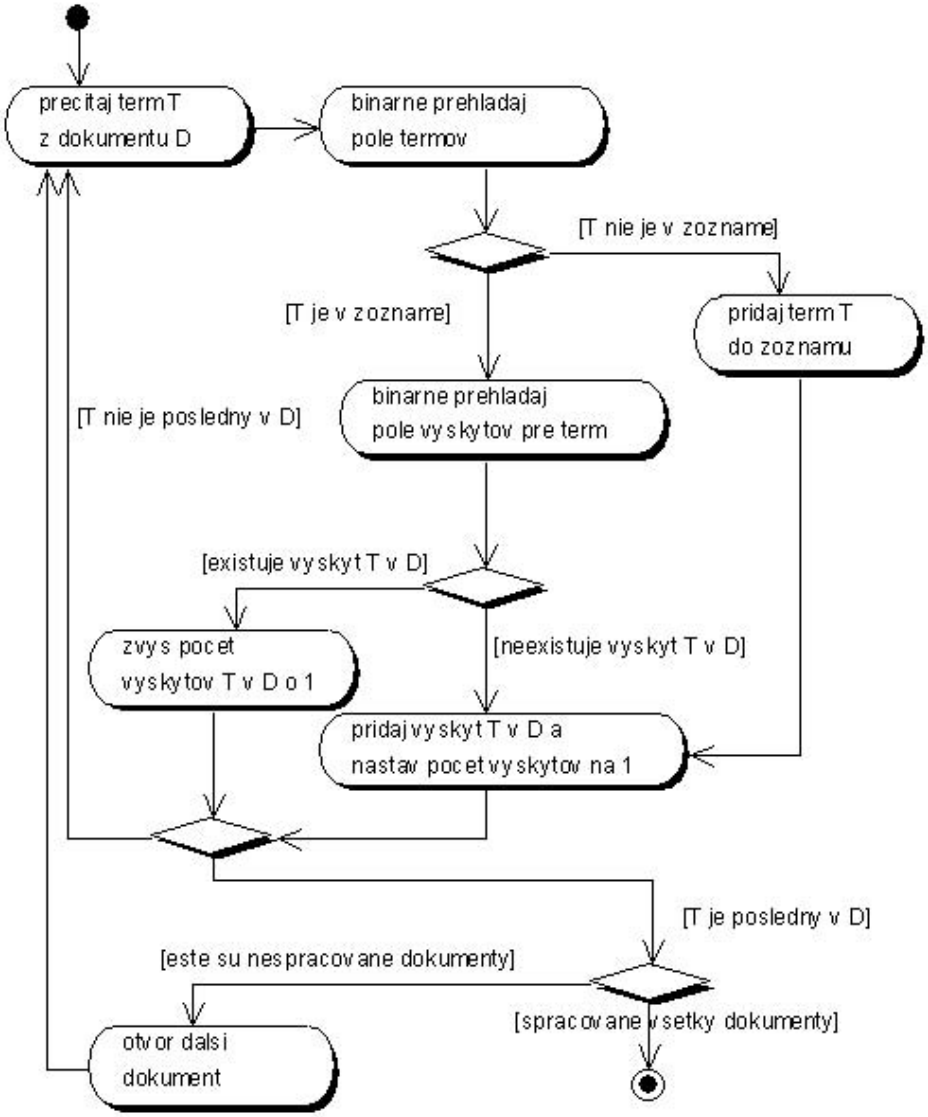
Už vytvorený invertovaný zoznam sa často ukladá do B-stromu, prípadne príbuznej štruktúry, ktorá dokáže efektívne vyhľadávať v utriedenom zozname, ktorý sa nezmestí do pamäti.

Ďalší algoritmus ([3]) bol navrhnutý priamo pre plnotextový stroj podporovaný relačnou databázou. Algoritmus pri spracúvaní dokumentu buduje invertovaný zoznam, ten po spracovaní dokumentu transformuje na INSERT príkazy do tabuľky *termDoc* a jeden INSERT do tabuľky *doc* z obr. 1. Keďže po spracovaní každého dokumentu je invertovaný zoznam zmazaný, nehrozí nedostatok operačnej pamäti. Tabuľka *term* sa naplní po spracovaní všetkých dokumentov pomocou SQL dopytu, ktorý získa zoznam jedinečných termov. Tiež je nutné vypočítať pre každý term frekvenciu dokumentov, v ktorých sa vyskytuje a inverznú frekvenciu dokumentu.

Využitím myšlienok v [3] a [1] získame pôvodný algoritmus Index(D, M). Algoritmus v [3] zapisuje invertovaný zoznam do databázy, nevyužíva dostupnú operačnú pamäť, nazvime ho teda Index(D). Algoritmus v [1], nezapisuje do databázy, využíva dostupnú pamäť, nazvime ho Index(M). Algoritmus Index(M) musí vo svojej záverečnej fáze zlučovať utriedené polia výskytov termov, na rozdiel od Index(D, M). Algoritmus Index(D) musí v závere počítať globálne váhy a tabuľku Term, na rozdiel od Index(D, M). Taktiež zápis výskytov termov pre algoritmus Index(D) trvá podstatne dlhšie, než pre Index(D, M), keďže zápis názvu termu trvá dlhšie, než len zápis jeho číselného identifikátora. [1]

Analyzujme štruktúru invertovaného zoznamu. Z definície invertovaného zoznamu vidno, že pozostáva zo zoznamu termov, pričom na každý term sa viaže zoznam výskytov tohto termu. Samotný zoznam termov nie je veľký, rastie tým menej, čím viac rastie počet dokumentov v kolekcii (sublineárne podľa Heapsovho zákona [4]) a v podstate je to konštanta pre použitý jazyk (v jazyku existuje obmedzené množstvo existujúcich slov). Naproti tomu zoznam výskytov termu rastie lineárne v závislosti od počtu dokumentov.

[1] Algoritmus Idex(D) bol pri testovaní približne o 20 percent pomalší než algoritmus Index(D,M) najmä kvôli zápisu výskytov termov



Obrázok 2. Vytváranie invertovaného zoznamu v operačnej pamäti

Na začiatku nášho algoritmu načítame do pamäti utriedený (aby sme mohli vyhľadávať s logaritmickým časom) zoznam vygenerovaný pomocou ispellu obsahujúci všetky jemu známe tvary slov a ich základné tvary. Tento zoznam zaberie v operačnej pamäti okolo 25 MB. Pre každé načítané slovo z dokumentu tak vieme rýchlo určiť jeho základný tvar, ak ho máme v zozname. Ak sa v zozname nenachádza, jednoducho ho zaevidujeme do zoznamu bez určenia jeho základného tvaru. Pri budovaní indexu budeme mať stále v pamäti zoznam termov. Zoznam výskytov termov prenesieme do databázy ihneď po vyčerpaní pamäte a pokračujeme v budovaní. Na konci uložíme do pamäte aj všetky termy, prepočítame ich váhy termov pre jednotlivé dokumenty a vypočítame normy dokumentov.

Keďže sme dekomponovali termy od ich výskytov (tabuľka Term a tabuľka TermDoc), nezlučujeme čiastočné zoznamy ako Index(M). Pre termy evidujeme počas celého algoritmu frekvenciu dokumentu a inverznú frekvenciu dokumentu, čiže to nevypočítavame ako v Index(D), ale priamo zapisujeme do databázy. Algoritmus Index(D,M) je na obr. 3, sú z neho vypustené časti už spomenuté v algoritme na obr. 2.

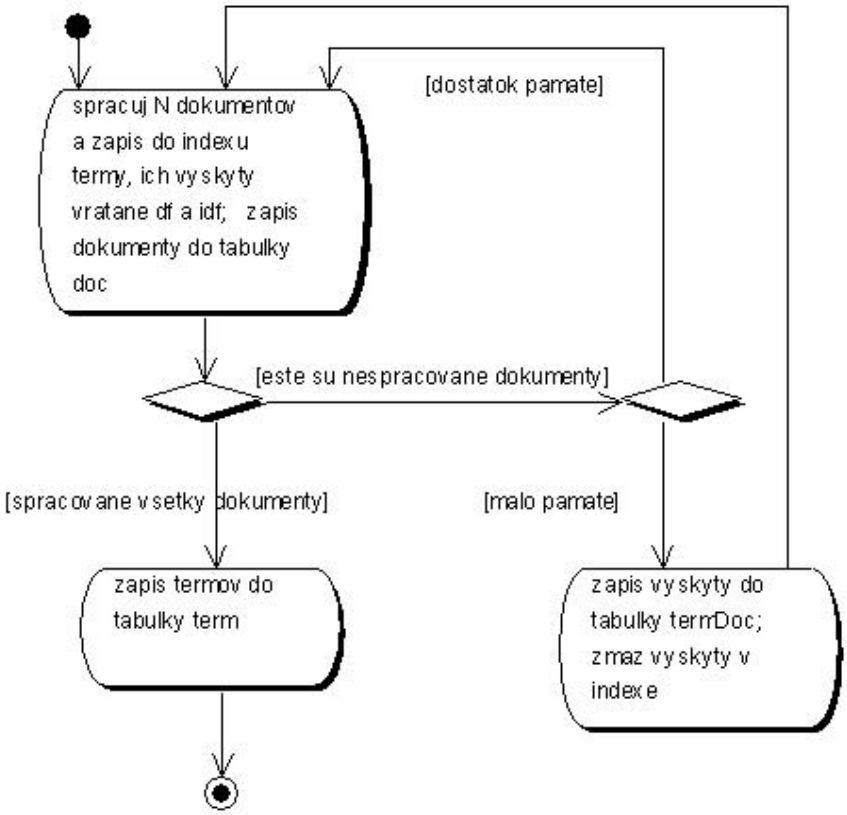
5 Výsledky testov indexovania

Pre kolekciu novinových článkov denníka SME sme dosiahli nasledovný výsledok: kolekcia má veľkosť 53 MB, čas na indexovanie týchto dokumentov bol 10 minút (počítačová konfigurácia obsahovala procesor 1,2 GHz a 512 MB operačnej pamäte, ako databáza bola použitá MySQL). Pre porovnanie: indexovanie pomocou open-source fulltextového stroja Lucene [7] napísaného v Jave trvalo 9 minút. Treba však poznamenať, že Lucene nie je založený na vektorovom modeli, a teda nevykonáva napríklad výpočet normy vektorov, ďalej sa nevykonáva žiadne morfologické zhlukovanie. Ďalšie porovnanie bolo s nerelačnou komerčnou databázou Lotus Notes [10], ktorá má zabudovaný fulltextový stroj (pravdepodobne booleovský model). Načítanie a indexovanie našej kolekcie trvalo 8 minút.

Pre porovnanie sme tiež testy vykonali na väčšej kolekcii LATimes [6](anglické články novín Los Angeles Times, 130000 dokumentov, veľkosť 390 MB), kde celý proces indexovania nášmu algoritmu trval 60 minút.

6 Odhady časovej zložitosti

Stanovme najprv vzťah medzi veľkosťou kolekcie n_{MB} (napríklad v megabajtoch) a počtom (opakujúcich sa) termov v nej obsiahnutých, aby sme potom mohli odvodiť časové zložitosti len na základe veľkosti kolekcie. Tieto opakujúce sa termy sú v postupnostiach $g((s_h^i)_k), k \in [1, m_{D_i}]$, označme ich počet ako m_K. Kolekcie obsahujúce dokumenty z jedného zdroja sú zväčša homogénne z hľadiska priemerného počtu slov v dokumente a vzťah medzi veľkosťou kolekcie n_{MB} a počtom slov m_K je lineárny. To sa potvrdilo aj v našom prípade. Na obr. 4 vidíme priemerný počet slov v 1 MB pre kolekcie SME a LATimes. Všimnime si väčší počet slov pre anglické články (slová sú tu kratšie).



Obrázok 3. Generovanie invertovaného zoznamu do databázy

Vzťah medzi počtom termov m_K a počtom jedinečných termov m_T (tzv. slovníkom) pre nejakú kolekciu je vyjadrená exponenciálne pomocou Heapsovho vzorca [4]

$$m_T = O((m_K)^\beta)$$

kde $0 < \beta < 1$ je konštanta závislá od kolekcie. Experimentálne zistená hodnota tejto konštanty pre kolekciu SME je $\beta = 0.77$. Keďže už vieme, že počet termov je lineárne závislý od veľkosti kolekcie, môžeme hodnotu konštanty β určovať vzhľadom na veľkosť kolekcie a nie počet termov:

$$m_T = O((m_{MB})^\beta)$$

Algoritmus indexovania Index(D, M) sa skladá z niekoľkých fáz, z ktorých časovo najdôležitejšie sú:

- A Vytvorenie/načítanie slovníka morfologických tvarov
- B Parsovanie dokumentov
- C Zápis termov do databázy
- D Zápis výskytov termov do databázy
- E Výpočet váh pre výskyty termov v dokumentoch
- F Výpočet noriem dokumentov

Fáza A trvá konštantný čas, označme ho t_A.

Fáza B je závislá od počtu termov m_K (všetky treba spracovať). Samotný algoritmus ďalej pre každý term vyhľadáva v utriedenom poli morfologických tvarov a vkladá výskyty termov do utriedeného poľa termov (postupnosť $t_j, j \in [1, m_T]$). Vyhľadávanie/vkladanie v utriedenom poli má logaritmickú časovú zložitosť vzhľadom na veľkosť poľa. Obe polia sú však v pamäti a časová náročnosť je oproti čítaniu z disku minimálna. Pole morfologických tvarov má navyše konštantnú veľkosť, čiže vyhľadávanie v ňom tiež trvá konštantný čas. Pole jedinečných termov má ďalej podstatne menšiu veľkosť než počet všetkých termov ($m_T << m_K$). Časovú zložitosť tejto fázy teda vieme vyjadriť ako

$$t_B = O(m_K \cdot \log(m_T)) \approx O(m_K) = O(m_{MB})$$

Potom teda

$$t_B = k_B \cdot n_{MB}$$

Experimenty tento vzorec potvrdili. Na obr. 5 vidno hodnotu konštanty k_B, ktorá má pre väčšie hodnoty skutočne minimálnu odchýlku. Táto zložitosť $O(n_{MB})$ je potvrdená aj inými prácami, napr. [11]. Pre kolekciu LaTimes je konštanta k_B o niečo menšia. Keďže ide o kolekciu anglických slov, nepoužíval sa tu slovník slovenských morfologických tvarov (tým pádom nebolo nutné zisťovať, či sa slovo v slovníku nachádza), dostupnej pamäte teda bolo o niečo viac. Taktiež slovenská kolekcia má viac jedinečných termov oproti anglickej, čo spomalilo parsovanie.

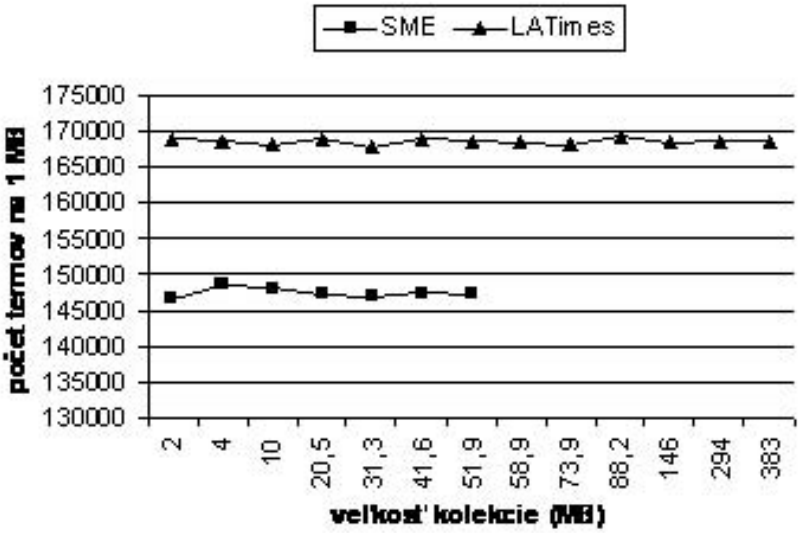
Fázu A a B sme mali plne pod kontrolou a vieme presne odhadnúť časovú zložitosť aj pre väčšie kolekcie. Fázy C až F sa týkajú práce s databázou a i keď tam v princípe vieme tiež stanoviť časovú zložitosť (poznáme základné algoritmy, ktoré sa tam používajú), experimentálne výsledky môžu byť iné kvôli prípadným optimalizáciám a využitiu cache pamäti. Typicky najmä vďaka využitiu dostatočnej cache pamäti pre indexy sme experimentom zistili zložitosť v týchto fázach $O(n_{MB})$. Pre kolekcie, ktorých indexy sa nezmestia do operačnej pamäti (respektíve sú malé veľkosti cache pamätí databázy), sme namerali zložitosť blížiacu sa ku $O((n_{MB})^2)$.

7 Záver

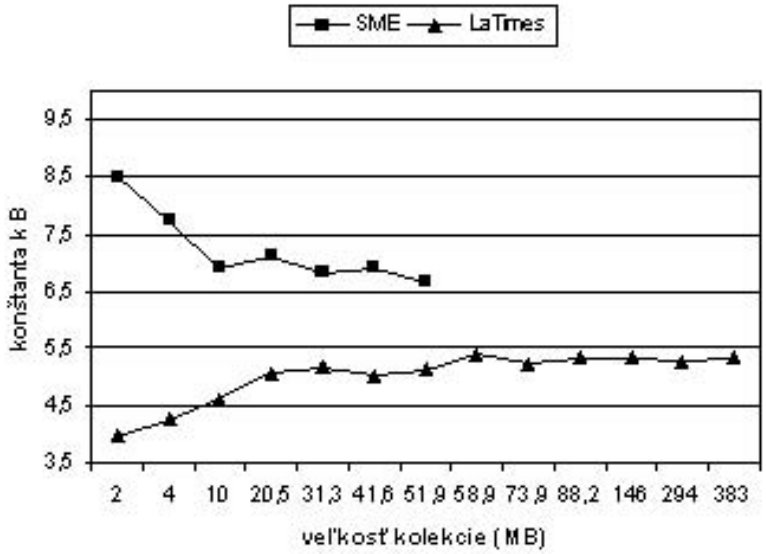
Algoritmus Index(D, M) bol úspešne implementovaný v jazyku Java, využitá bola databáza MySQL (v princípe možno použiť ľubovoľnú databázu podporujúcu JDBC a spĺňajúcu štandard SQL99). Časová zložitosť vytvárania invertovaného zoznamu je lineárna vzhľadom na veľkosť kolekcie. Zložitosť zápisu tohto zoznamu do databázy a výpočet váh sa nachádza medzi lineárnou a kvadratickou zložitosťou, v závislosti od veľkosti dostupnej pamäti a použitej databázy. Ďalšia práca sa bude zaoberať distribuovaním algoritmu Index(D, M) medzi viac počítačov s cieľom zrýchliť výpočet.

Referencie

1. *Baeza-Yates, R., Ribeiro-Neto, B.* : Modern information retrieval. Addison Wesley (1999), 197-198
2. *Crawford, R.* : The relational model in information retireval. journal of the American Society for Information Science (1981) 51-64
3. *Grossman, D. A., Frieder, O.* : Information Retrieval: Algoritms and heuristics. Kluwer Academic Publishers (2000) 169-170
4. *Heaps, J.* : Information Retrieval - Computational and Theoretical aspects. Academic Press, 1978
5. http://spell.linux.sk
6. http://trec.nist.gov
7. http://jakarta.apache.org/lucene
8. *Lundquist, C.* : Relational information retrieval: Using Relevance Feedback and Parallelism to Improve Accuracy and Performance. PhD thesis, George Mason University (1997)
9. *Macleod, I.* : A relational approach to modular information retrieval systems design. In Proceedings of the ASIS Annual Meeting (1978) 83-85
10. http://www.ibm.com/notes
11. *Ribeiro-Neto, B., Moura, E. S., Neubert, M. S., Ziviani, N.* : Efficient distributed algorithms to buildinverted files, In 22th ACM Conf. on R&D in Information Retrieval, 1999
12. *Salton, G., Buckley, C.* : Term-weighting approaches in automatic text retrieval. Information Processing and Management 24, 513-523 (1988)
13. *Salton, G., Yang, C., Wong, A.* : A vector-space model for automatic indexing. Communication of ACM 18 (1975) 613-620



Obrázok 4. Priemerný počet termov v 1 MB pre rôzne veľkosti kolekcie SME a LATimes



Obrázok 5. Hodnoty konštanty k_B v závislosti od veľkosti kolekcie, pre kolekcie SME a LaTimes

Testování konzistence a úplnosti valenčního slovníku českých sloves*

Markéta Lopatková and Zdeněk Žabokrtský

Centrum komputační lingvistiky, MFF UK, Praha
{lopatkova,zabokrtsky}@ckl.mff.cuni.cz

Abstrakt Na moderní valenční slovník klademe řadu požadavků. Kromě strojové čitelnosti a dostatečné explicitnosti použitého popisu jde zejména o kvalitu dat ve slovníku obsažených. V článku přibližujeme nástroje navržené pro testování konzistence a úplnosti slovníku VALLEX. Rozebíráme metody využívané pro zvýšení jeho kvality od odstraňování technických chyb přes porovnání s existujícími lexikografickými zdroji po testování vnitřní konzistence budovaného slovníku.

Valence je jeden ze základních jazykových jevů, se kterým je třeba počítat při tvorbě většiny aplikací v oblasti počítačového zpracování přirozeného jazyka a jehož zkoumání je zajímavé i pro „tradičního“ lingvistu. Valenční vlastnosti sloves (i některých ostatních slovních druhů) jsou ovšem velmi rozmanité. Nelze je odvodit obecnými pravidly, je třeba je popsat v podobě valenčního slovníku, který obsahuje popis valence jednoho slova po druhém. Z těchto důvodů vzniká v Centru komputační lingvistiky od roku 2001 elektronický valenční slovník českých sloves VALLEX, http://ckl.mff.cuni.cz/zabokrtsky/vallex/1.0/. V tuto chvílu je v něm obsaženo zhruba 1400 sloves, probíhá jeho další rozšiřování. Budování slovníku je úzce spjato s vytvářením Pražského závislostního korpusu.[1]

Na moderní valenční slovník je kladena řada požadavků – kromě strojové čitelnosti a dostatečné explicitnosti použitého popisu jde zejména o kvalitu dat ve slovníku obsažených. Slovník by neměl obsahovat chyby, a to ani z technického, ani z lingvistického úhlu pohledu. Mezi měřítka kvality slovníku řadíme konzistenci (důsledné zachycování „stejných věcí stejně“) a úplnost (pokrytí všech významů, kterých dané sloveso může v jazyce nabývat).

V sekci 1 přiblížíme základy použité podkladové teorie, valenční teorii Funkčního generativního popisu češtiny. Dále bude popsána struktura hesel slovníku VALLEX (sekce 2). Jádro článku tvoří sekce 3 a 4, ve kterých přibližujeme navržené nástroje (sekce 3) a rozebíráme metody testování kvality slovníku (sekce 4) – zejména porovnávání s existujícími zdroji (4.2), testování „vnitřní“ konzistence slovníku (4.3) a ověřování na autentických větách (4.4). V sekci 5 uvedeme příklady aplikací, ve kterých se komplexní valenční slovník VALLEX s úspěchem využívá.

* Valenční slovník českých sloves VALLEX je vytvářen v Centru komputační lingvistiky při MFF UK, které vzniklo jako výzkumné centrum LN00A063 na základě programu MŠMT ČR, s podporou grantu GAČR 405/04/0243.

[1] http://ufal.mff.cuni.cz/pdt

1 Trocha teorie – co je valence?

Pokud aspirujeme na vytvoření konzistentního jazykového zdroje (language resource), který by byl využitelný pro aplikace v NLP i pro podrobná lingvistická zkoumání, potřebujeme důkladně rozpracovanou podkladovou teorii. VALLEX je budován na základě Funkčního generativního popisu (FGD, viz zejména [6]), což je závislostně orientovaný stratifikační systém, v jehož rámci je teorie valence studována od sedmdesátých let (viz zejména [5]).

Co je to tedy valence? Podle autorů valenčního slovníku Slovesa pro praxi [7]: *„Valencí rozumíme v lingvistice schopnost lexikální jednotky, především slovesa, vázat na sebe jiné výrazy a mj. tak zakládat větné struktury.“* Tato schopnost se týká primárně významové reprezentace, promítá se i do povrchové realizace věty.

Informace o valenčním chování lexikální jednotky je uchovávána ve valenčních rámcích – každému slovesu odpovídá soubor **valenčních rámců**, které ve FGD v zásadě odpovídají jednotlivým významů slovesa. Valenční rámec se skládá z **vnitřních doplnění slovesa** (aktantů, též participantů nebo argumentů), obligatorních i fakultativních, a dále z **obligatorních volných doplnění** (adverbiální doplnění, adjunkty).

FGD rozlišuje pět vnitřních doplnění (aktor, patient, adresát, původ, výsledek; v aktivní větě aktor typicky odpovídá subjektu, patient přímému objektu, adresát nepřímému objektu) a řadu volných doplnění (odpovídají příslovečným určením, např. místa, času, způsobu, prostředku, podmínky – viz tabulku 1). Vnitřní i volná doplnění mohou být buď obligatorní (povinně přítomny ve významové reprezentaci věty), nebo fakultativní.[2]

*Matka.*ACT *předělala loutku.*PAT *z Kašpárka.*ORIG *na čerta.*EFF.
*Petr.*ACT *včera.TWHEN v novinách.LOC četl o katastrofě.*PAT.
*Děti.*ACT *přišli pozdě.TWHEN.* (=*domů, sem.*DIR1)
*Venku.*LOC *prší.*
V Praze.LOC se sejdeme na Hlavním nádraží.LOC u pokladen.LOC.
*Kniha.*ACT *vyšla.*
*Chlapec.*ACT *vyrostl v muže.*PAT.

Klasifikaci FGD obohacujeme o tzv. typická doplnění,[3] z nichž některá mohou být obligatorní (*přijít kam.*DIR3).

[2] Následující příklady a tabulka 1 umožní sledovat článek i čtenáři, který není obeznámen s příslušnými lingvistickými teoriemi. Příklady částečně přebíráme z článků J. Panevové. Členy valenčních rámců sloves jsou označeny verzálkami; fakultativní volná doplnění, která nejsou součástí rámce, označujeme kurzívou (přesněji – vyznačujeme jméno příslušné sémantické relace mezi slovesem a jeho valenčním doplněním). V tabulce jsou polotučným písmem vyznačeny větné členy, které odpovídají příslušnému funktoru.

[3] Typická doplnění jsou fakultativní volná doplnění (tudíž nepatřící do „klasického“ valenčního rámce), která dané sloveso „zpravidla“ rozvíjejí; navíc taková doplnění obvykle rozvíjejí celou třídu sémanticky blízkých sloves. Např. slovesa pohybu jsou typicky rozvíjena volnými doplněními směru (*jít jet/běžet/spěchat do kina.DIR3 /přes les.DIR2 /z domova.DIR1*).

Funktor	Příklad
ACT (aktor)	**Petr** čte knihu.
ADDR (adresát)	Petr dal **Marii** knihu.
EFF (výsledek, efekt)	Zvolili Petra **předsedou**.
ORIG (původ, origo)	Upekla **z jablek** koláč.
PAT (patient)	Viděl jsem **Petra** venku.
DIFF (rozdíl)	Jejich počet vzrostl **o 200**.
OBST (překážka)	Zakopl **o kámen**.
INTT (záměr)	Jana šla **nakoupit**.
ACMP (doprovod)	Matka přišla **s dítětem**.
AIM (účel)	Jan došel do pekárny **pro housky**.
BEN (prospěch)	Udělala to **pro své děti**.
CAUS (příčina)	Lucie to udělala, **protože to po ní chtěli**.
COMPL (doplněk)	Petr pracuje **jako učitel**.
DIR1 (směr-odkud)	Petr se vracel **ze školy** pěšky.
DIR2 (směr-kudy)	Petr se loudal **parkem**.
DIR3 (směr-kam)	Petr spěchal **do práce**.
DPHR (frazém)	Bloudil **křížem krážem** lesem.
EXT (míra)	Petr měří **180 cm**.
HER (dědictví)	Josífek se jmenoval **po otci**.
LOC (místo)	Narodil se **v Itálii**.
MANN (způsob)	Psal **bezchybně**.
MEANS (prostředek)	Petr přijel **na kole**.
NORM (norma)	Petr sestavil model **podle instrukcí**.
RCMP (náhrada)	Jana si koupila nové tričko **za 200 Kč**.
REG (zřetel)	**Co se týká Petra**, je vše v pořádku.
RESL (účinek)	Matka brání děti **před vším nepohodlím**.
SUBS (zastoupení)	Jana šla **za svou sestru** na zkoušku.
TFHL (čas-na jak dlouho)	Petr přerušil školu **na jeden semestr**.
TFRWH (čas-ze kdy)	**Z dětství** si nepamatuje nic.
THL (čas-jak dlouho)	Četl **půl hodiny**.
TOWH (čas-na kdy)	Odložil schůzku **na příští týden**.
TSIN (čas-ze kdy)	**Od té doby** jsem o něm neslyšel.
TWHEN (čas-kdy)	Jeho syn Jan se naridil **loni**.

Tabulka 1. Funktory pro syntakticko-sémantickou anotaci.

2 Co valenční slovník obsahuje?

Každé sloveso ve slovníku VALLEX je reprezentováno jako soubor valenčních rámců s doplňujícími syntakticko-sémantickými informacemi (vztaženými vždy k danému rámci); homonymní slovesa jsou popsána více soubory. Typicky jeden rámec odpovídá jednomu významu slovesa, příslušný význam je vždy určen glosou a příklady použití. Valenční rámec slovesa, který tvoří jádro zachycované informace, definujeme jako kombinaci prvků rámce (slovesných doplnění). U každého prvku rámce jsou zachyceny jeho tři vlastnosti:

– funktor, tj. jméno sémantické relace mezi slovesem a jeho příslušným doplněním (aktantem nebo volným doplněním);

– morfematické vyjádření příslušného doplnění (číslo pádu, předložka+číslo pádu, infinitiv nebo podřadící spojka);
– typ doplnění, tj. zda jde o obligatorní (obl) nebo fakultativní (opt) valenční doplnění, příp. doplnění typické (typ).

Cílem VALLEXu je poskytnout uživateli komplexní syntakticko-sémantickou informaci. Proto je jádro slovníku – soubor valenčních rámců – obohaceno o další informace využitelné v NLP (tyto údaje jsou vždy vztaženy k jednotlivým valenčním rámcům, nikoli k celému slovesu – výjimku tvoří vidová charakteristika, která je vlastní celému slovesu):[4]

– reflexivita* (výčet možných syntaktických funkcí zvratného zájmena *se/si*);
– reciprocita* (možnost členu valenčního rámce vstupovat do symetrické relace s jiným členem);
– kontrola (u sloves s doplněním ve formě infinitivu; jde o vzájemný vztah mezi některým členem valenčního rámce a subjektem infinitivu);
– vid, příp. vidový protějšek (odkaz na příslušný valenční rámec);
– syntakticko-sémantická třída;*
– pointer na odpovídající synset české větve sémantické databáze EuroWordNet.*

Ve valenčním slovníku VALLEX 1.0 je obsaženo přes 1400 českých sloves – prvních zhruba 1000 sloves bylo vybráno podle frekvence v Českém národním korpuse (s výjimkou pomocného slovesa *být*, které vyžaduje zvláštní zpracování), k nim byly posléze doplněny jejich vidové protějšky (pokud ještě nebyly zpracovány).

3 Jaké nástroje lze využít při testování konzistence a úplnosti VALLEXu?

Při budování slovníku je nutno klást maximální důraz na systematičnost a konzistenci v zachycování jednotlivých jazykových jevů, neboť konzistence zpracování patří k základním požadavkům kladeným na každý zdroj jazykových dat.

Přestože při testování konzistence slovníku mají a budou mít nezastupitelnou úlohu vzájemné ruční kontroly anotátorů (každé heslo procházejí nejméně tři lidé v různých fázích zpracování), jejich úsilí mohou podstatným způsobem zefektivnit navržené nástroje umožňující vyhledávání údajů a třídění hesel podle jednotlivých atributů a jejich kombinací.

Vyhledávací rozhraní pro WWW. Vyhledávací rozhraní pro WWW umožňuje vyhledávat rámce podle toho, zda daný rámec nebo jeho vybrané atributy obsahují určité podřetězce nebo odpovídají regulárnímu výrazu. (Např. „najdi všechna slovesa kontroly“, „najdi všechna slovesa obsahující v rámci funktor EFF“, „najdi všechna slovesa s reflexivním zájmenem *se*“, případně „zobraz celý slovník“ (dotaz bez omezovacích podmínek).)

[4] Údaje označené hvězdičkou jsou zpracovány zatím pouze částečně.

Dále je možné zjišťovat rozvržení hodnot jednotlivých atributů. (Např. „zobraz všechny hodnoty atributu reciprocity a jejich rozložení“, „zobraz valenční rámce všech sloves kontroly“). K vyhledaným hodnotám lze vždy zobrazit informaci o příslušných valenčních rámcích, případně o jejich vybraných atributech.

Toto rozhraní je grafické, umožňuje klást dotazy anotátorům, kteří nejsou zběhlí v programování.

Vyhledávání v dostupných elektronických zdrojích. Tato aplikace umožňuje rychle nahlédnout, jak je dané sloveso zpracováno v existujících slovnících. K dispozici máme slovníky Slovesa pro praxi a Slovník spisovného jazyka českého, dále případné zpracování slovesa v české větvi EuroWordNetu a 100 náhodných výskytů v Českém národním korpusu.

Vyhledávání v XML-reprezentaci dat. Datová reprezentace slovníku je založená na XML, lze tedy využít řady existujících nástrojů. Jde zejména o editor XSH (*XML Editing Shell*)[5] P. Pajase, který umožňuje klást dotazy přesahující možnosti grafického rozhraní (např. „zjisti počet sloves / rámců / prvků v rámcích“, „zobraz slovesa, která mají více než 5 rámců“, „najdi primární reflexiva tantum“). Užívání XSH vyžaduje základní znalost XML technologií, více viz [3].

4 Jaké metody lze využít při testování konzistence a úplnosti VALLEXu

Testování konzistence a úplnosti slovníku je metodologicky i časově náročná činnost.[6] Neznáme obecně přijatou metodologii testování systematičnosti a konzistence slovníku, která by byla dostatečně efektivní a komplexní a kterou bychom mohli přejmout, proto jsme byli nuceni vypracovat vlastní metody testování.

Testování konzistence bylo částečně provedeno po základním zpracování tisíce českých sloves, druhé kolo masivního testování (a následné opravy) proběhlo po zpracování všech 1400 sloves obsažených ve verzi slovníku VALLEX 1.0.

4.1 Odstranění technických nedostatků

Slovník VALLEX má striktně definovanou notaci, prohřešky proti ní (např. chybějící závorka) lze většinou nalézt automaticky. Dalším typem čistě technické chyby je překlep ve funktoru nebo použití neexistující morfématické formy (např. *u*+4 – předložka *u* se nepojí s akuzativem).

4.2 Porovnání s jinými lexikografickými zdroji

Již při základním zpracování sloves jsme využívali valenční slovník BRIEF a Slovník spisovného jazyka českého (SSJČ). Při následném testování jsme obsah

[5] http://xsh.sourceforge.net

[6] Hrubý odhad času vynaloženého na testování konzistence a úplnosti slovníku se pohybuje okolo 1/3 času věnovaného vytváření slovníku.

slovníku VALLEX porovnávali s tím, jak jsou slovesa zpracována ve slovníku Slovesa pro praxi (SPP) a částečně i v české větvi databáze EuroWordNet (EWN). Toto porovnání bylo přínosné zejména pro vyčlenění jednotlivých významů zpracovávaných sloves a pro doplnění případných chybějících významů sloves, přitom ovšem bylo potřeba brát v úvahu rozdílné přístupy uplatněné v jednotlivých zdrojích.

Slovník BRIEF. Valenční slovník povrchových realizací ve formátu BRIEF [4], který vznikl kompilací několika tištěných slovníků, především SSJČ, byl využit již při základním zpracování sloves zejména jako zdroj morfematických forem, které se pojí s jednotlivými slovesy.

Slovník spisovného jazyka českého. SSJČ a jeho elektronická podoba[7] sloužila jako základní zdroj informací o významech sloves. Vyčlenění jednotlivých významů sloves v SSJČ však neodpovídá jednotlivým valenčním rámcům (tuto zásadu jsme převzali z podkladové teorie FGD), proto bylo přepracováno s důrazem na syntaktická a sémanticko-syntaktická kritéria.

Obecně jsou v SSJČ významy členěny jemněji (např. *bát se*), existují ovšem i příklady opačné relace (např. *pocházet*), viz tabulky 2 a 3.

Významy v SSJČ označené za zastaralé nebyly ve VALLEXu zpracovávány.

SSJČ – *bát se*	VALLEX – *bát se*
1. mít strach ∼ *byla sama doma a bála se*	1. ACT (PAT), mít strach ∼ *bát se tmy/učitele* */aby se v labyrintu vyznal* */že bude pršet;*
2. mít strach něco udělat ∼ *bojí se jít za tmy do lesa*	*bojí se létat*
3. mít strach z někoho/něčeho ∼ *bát se otce, samoty*	
4. mít starost, že někdo/něco je ohrožen(o) ∼ *bát se o otce, o výsledky své práce;* *bojím se, abych neupadl*	2. ACT PAT, obávat se o někoho/něco ∼ *bála se o syna*

Tabulka 2. Vyčlenění významů slovesa *bát se* v SSJČ a ve VALLEXu.

SSJČ – *pocházet*	VALLEX – *pocházet*
1. vzít původ, vznik; vzniknout, vzejít, povstat, zrodit se	1. ACT PAT, ∼ *nemoc pochází z viru* 2. ACT DIR1, ∼ *Jan pochází z venkova* 3. ACT TFRWH, ∼ *rukopis pochází z roku 1352*

Tabulka 3. Vyčlenění významů slovesa *pocházet* v SSJČ a ve VALLEXu.

Slovesa pro praxi. Slovník SSP poskytuje podrobné údaje o valenčním chování vybraných sloves (767 sloves), které byly využity při testování VALLEXu.

[7] Aplikace GSlov byla poskytnuta Laboratoří zpracování přirozeného jazyka, FI MU Brno.

Vyčlenění jednotlivých významů ovšem opět zcela neodpovídá kritériím přijatým ve VALLEXu – sporná je především možnost přiřazování konkrétních užití slovesa jednotlivým rámcům (viz např. pět významů slovesa *bát se*, v tabulce 4).

SPP – *bát se*
1. mít pocit ohrožení ∼ *Když ten pes pozná, že se ho bojíš, kousne tě docela určitě. Koně se báli biče jako čert kříže.*
2. mít obavu z něj. vlastní činnosti ∼ *Nakonec se našel nakladatel, který se nebál český překlad vydat. Z chlapce se stává muž. Nebojí se žádné práce.*
3. mít nelibý pocit plynoucí z očekávání něčeho nepříjemného ∼ *Hlavně se bojím toho, že budu nemocná. Psi zalezli do boudy, báli se, že je tentokrát výprask nemine. Ponejvíc se lidé bojí, aby je někdo neošidil.*
4. mít obavu o někoho, něco ∼ *O výsledky své práce se nebojíme. Bezpečnostní situace v hlavním městě je taková, že se obyvatelé právem bojí o svůj majetek a někteří i o své životy.*
5. být bojácný ∼ *Pojď, neboj se, nejsi přece malé dítě. Míša se nebojí, jaképak bání! Co je to za hlídacího psa, když se bojí!*

Tabulka 4. Vyčlenění významů slovesa *bát se* v SPP.

EuroWordNet. EuroWordNet[8] je multilinguální lexikální databáze; průnik sloves v její české větvi (cca 3 000 sloves) a sloves zpracovaných ve VALLEXu představuje zhruba 500 sloves. EWN neobsahuje žádné informace o valenci, pokusili jsme se jej částečně využít jako pomůcku pro rozlišování významů slovesa (s plným vědomím, že členění významů v EWN, jehož základem je zpracování anglických sloves, zcela neodpovídá češtině). Nicméně výhody i nedokonalého navázání jednotlivých valenčních rámců na synsety (tj. základní významové jednotky EWN) jsou zřejmé.

4.3 Testování konzistence uvnitř VALLEXu

Mezi hlavní měřítka kvality slovníku je potřeba řadit konzistenci zpracování dat, nutnost stanovit jasnou koncepci (která může být pro různé účely různá) a v jejím rámci zpracovávat „stejné věci stejně“. Proto je ve VALLEXu kladen velký důraz na odstranění nezdůvodnitelné různorodosti, která vzniká při budování slovníku „zdola“.

Vidové protějšky. Valenční rámce sobě odpovídajících vidových protějšků jsou často totožné. Protože vidové protějšky byly zpracovávány nezávisle na sobě, lze jejich porovnání (a případné následné sjednocení) považovat za masivní test konzistence zpracování. Stejně jsme ve VALLEXu využili podobnosti předponových a bezpředponových sloves.

Sjednocení anotace příbuzných sloves je přínosné zejména pro slovesa s mnoha významy – např. vidové protějšky *brát* a *vzít* mají 13 totožných rámců zachycu-

[8] http://www.hum.uva.nl/∼ewn/

jících primární a posunuté užití a 9 totožných rámců pro idiomy, *brát* má navíc 4 idiomatické rámce, *vzít* rámce 2.

Sémantické třídy. Slovník VALLEX obsahuje u přibližně jedné třetiny rámců informaci o syntakticko-sémantické třídě. I když jde zatím pouze o předběžné třídění a seskupení slovesných rámců, má velký význam pro konzistenci zpracování – předpokladem je, že slovesa patřící do jedné třídy se budou chovat i z pohledu valence velmi podobně. Zatím byla systematicky provedena anotace u sloves pohybu (třídy motion, transport), sloves pravení (třídy communication, mental action, perception, social interaction) a částečně u sloves výměny (exchange).

Tímto způsobem bylo například u sloves pohybu (motion, transport) systematicky doplněno typické doplnění záměru (funktor INTT) vždy k primárnímu významu zpracovaných sloves, *jít na houby, přivedl mu ukázat přítelkyni* (původně 24 intuitivně anotovaných výskytů INTT bylo rozšířeno na 48 výskytů).

Morfématické formy. Systematicky byly zpracovány některé morfématické formy – byly porovnány všechny funktory s konkrétními formami i celé valenční rámce. Tyto testy byly účelné zejména pro zpracování předložkových skupin *o*+4 (zejména s ohledem na zachycení funktorů DIFF (rozdíl) a OBST (překážka)) a *za*+4 (systematické zpracování sloves výměny). Dále byla zkoumána doplnění vyjádřená infinitivem a výrazem *jako* (konzistentní rozlišování funktorů COMPL (doplněk) a EFF (efekt)).

Kromě toho byly zkoumány možné kombinace morfématických forem u jednotlivých funktorů (např. u funktoru INTT (intence, záměr) u sloves pohybu byla forma sjednocena na *na*+4, inf).[9]

Další možností je porovnávat kombinace forem bez ohledu na funktor (zejména např. pro úplný soubor podřadících spojek, zatím zpracováno částečně).

Typická doplnění. Systematicky jsou zpracovávána též fakultativní volná doplnění, která lze označit jako typická (viz poznámka 3). Byly porovnány všechny rámce, ve kterých se vyskytuje některé ze specifických volných doplnění (např. MEANS, BEN, CAUS).

Typická doplnění byla sjednocena také u sloves již zpracovaných sémantických tříd. Například slovesa pohybu (třídy motion a transport) jsou typicky rozvíjena volnými doplněními směru – pro určení obligatorního doplnění směru dává kritéria FGD, fakultativní doplnění jsou zpracována systematicky v rámci tříd; slovesa vyjadřující pohyb pomocí dopravního prostředku mají typicky volné doplnění prostředku, MEANS.

Četnost. Jako obecně užitečná se ukázala technika „co je málo časté, to je podezřelé“. Tuto techniku lze s výhodou využít napříč slovníkem, u všech zachycovaných informací. Například u morfématické formy lze tímto způsobem

[9] Výjimku tvoří sloveso *nést* ve významu *nese rozdat handouty*, kde není možná předložková skupina *na*+4 (nejednotnost je tedy v tomto případě opodstatněná).

odhalit nejen překlepy, ale i idiomatičnost některých spojení. U funktorů, které se ve VALLEXu vyskytly pouze několikrát, je potřeba zkontrolovat jejich účelnost, případně správné rozlišování anotátory (konkrétně např. funktory NORM, norma a CRIT, kritérium). Také ověřování anotace kontroly a reciprocity vedlo k omezení neopodstatněné různorodosti (málo četné hodnoty v těchto atributech vedly k odhalení technických nedostatků i faktických chyb).

Technika „co je málo časté, to je podezřelé“ byla (zatím částečně) použita i na celé valenční rámce – pokud se některý rámec vyskytne v celém VALLEXu jen jednou, je vhodné ověřit, zda se v něm nevyskytuje nějaká chyba nebo neopodstatněná variace.

4.4 Ověřování na Českém národním korpusu

Zpracování sloves ve VALLEXu je ověřováno na autentických příkladech užití slovesa v ČNK.[10] Pro každé zpracované sloveso jsme použili 60 (pro nejsložitější slovesa 100) náhodně vybraných příkladových vět[11] z ČNK a ověřovali, zda lze výskytům daného slovesa přiřadit valenční rámec z VALLEXu. Přínosem této metody je především ověření vhodného rozčlenění slovesných rámců – důležitým kritériem pro vyčlenění jednotlivých valenčních rámců je shoda anotátorů v jejich přiřazování konkrétním výskytům slovesa –, případně doplnění chybějících rámců.

Například pro sloveso *nalézat*^{I} byly původně vyčleněny 4 rámce – 1. hledáním získávat, objevovat (*nalézat zlato na Aljašce*), 2. získávat (*nalézat přítele, potěšení v práci, pochopení*), 3. odhalovat (*nalézat na studiu kladné stránky*), 4. ohodnotit (*nenalézal na něm nic dobrého*); testy na příkladech ukázaly nemožnost rozlišovat mezi 2. a 3. rámcem, proto byly tyto dva rámce sloučeny (v souladu se SSJČ). Naopak, na základě vět z ČNK byl pro sloveso *přijmout* vyčleněn nový rámec s glosou *schválit* (*parlament přijal zákon*).

5 K čemu valenční slovník?

Při budování VALLEXu je kladen důraz na skutečnost, aby byl slovník snadno a rychle čitelný, na snadnou orientaci a na srozumitelnost. To jsou základní předpoklady, které jsou nezbytné pro efektivní manuální zpracovávání jednotlivých sloves a pro možnost odhalování chyb a nekonzistencí. Na druhou stranu je takový formát podmínkou pro využití slovníku v dalším lingvistickém výzkumu. Nicméně hlavní přínos VALLEXu se předpokládá v automatických procedurách NLP.

V současné době se VALLEX testuje v následujících aplikacích:

– automatická syntaktická analýza („shallow parsing“);

[10] http://ucnk.ff.cuni.cz

[11] Pro časovou náročnost těchto kontrol (60 x 1000 vět = 60 000 přiřazených výskytů valenčního rámce) bylo zatím použito pouze omezeného vzorku ČNK, předpokládáme další ověřování.

– „tektogrammatický parser“, tj. automatický systém pro vytváření významové reprezentace české věty;
– zdrojová data pro budování valenčního slovníku substantiv.

Valenční slovník VALLEX je pro nekomerční účely volně k dispozici, více informací viz http://ckl.mff.cuni.cz/zabokrtsky/vallex/1.0/.

6 Shrnutí a otevřené otázky

Vytváření valenčního slovníku českých sloves VALLEX je úzce spojeno s budováním Pražského závislostního korpusu, jeho koncept vznikl v souvislosti s potřebou zajistit konzistentní zachycení valence v PDT. Zásadní důraz je přitom kladen na systematičnost zpracování všech jevů ve slovníku obsažených.

V tomto příspěvku jsme představili nástroje pro vyhledávání údajů a třídění hesel podle jednotlivých atributů, které byly navrženy pro testování konzistence a úplnosti slovníku. Dále jsme přiblížili řadu metod již použitých i v současné době aplikovaných – tyto metody jednak využívají existující jazykové zdroje, jednak se soustřeďují na eliminaci neopodstatněné různorodosti a na dosažení jednotného zpracování jevů ve slovníku obsažených.

Metody zde stručně popsané je možno chápat jako příspěvek k vytváření metodologie testování konzistence a úplnosti jazykových zdrojů. Zatím otevřenou otázkou zůstává metodologie evaluace slovníku, kvalifikovaný odhad možného množství chyb a mezianotátorské shody.

Reference

1. Hajič, J., Panevová, J., Urešová, Z., Bémová, A., Kolářová, V., Pajas, P. 2003. PDT-VALLEX: Creating a Large-coverage Valency Lexicon for Treebank Annotation. In: Proceedings of The Second Workshop on Treebanks and Linguistic Theories. pp. 57–68. Vaxjo University Press.
2. Hajičová, E., Panevová, J., Sgall, P. 2001. Manuálů pro tektogramatické značkování. ÚFAL/CKL TR-2001-12.
3. Lopatková, M., Žabokrtský, Z., Skwarska, K., Benešová, V. 2002. Tektogramaticky anotovaný valenční slovník českých sloves. ÚFAL/CKL TR-2002-15.
4. Pala K., Ševeček, P. 1997. Valence českých sloves (Valency of Czech verbs). In: *Sborník prací FFBU*. volume A45.
5. Panevová, J. 1994. Valency Frames and the Meaning of the Sentence. In: Ph. L. Luelsdorff (ed.) *The Prague School of Structural and Functional Linguistics*. Amsterdam-Philadelphia, John Benjamins, pp. 223-243.
6. Sgall, P., Hajičová, E., Panevová, J. 1986. The Meaning of the Sentence in Its Semantic and Pragmatic Aspects (ed. by J. Mey). Dordrecht:Reidel and Prague:Academia.
7. Svozilová, N., Prouzová, H., Jirsová, A. 1997. Slovesa pro praxi. Academia, Praha.
8. Slovník spisovného jazyka českého. Praha. 1964.

Restarting Automata and Combinations of Constraints

František Mráz[1] ⋆, **Friedrich Otto**[2], and **Martin Plátek**[1]

[1] Charles University, Department of Computer Science
Malostranské nám. 25, 118 00 PRAHA 1, Czech Republic
platek@ksi.ms.mff.cuni.cz, mraz@ksvi.ms.mff.cuni.cz

[2] Fachbereich Mathematik/Informatik, Universität Kassel
D-34109 Kassel, Germany
otto@theory.informatik.uni-kassel.de

Abstract. Recently, j-left-, j-right- and j-right-left-monotone restarting automata ($j \geq 1$) have been introduced and studied. We introduce combinations of these degrees of monotonicity (together with their strong variants). Concentrating on deterministic restarting automata, we show some interesting differences between one-way and two-way restarting automata with respect to several combinations of degrees of monotonicity.

1 Introduction

Restarting automata (RA) were introduced as a model for the so-called *analysis by reduction* of natural languages. They can do a bottom-up syntactic analysis for natural and formal languages with a sufficient generality and *'explicative power'* (cf. [3]). The notions developed during the study of restarting automata (cf. [2]) give a rich taxonomy of constraints for various models of analyzers (parsers).

Several programs based on the idea of restarting automata have already been used in Czech and German (corpus) linguistics (cf. [1, 3]).

Analysis by reduction is based on the idea of stepwise simplification of a given sentence until we get a correct simple sentence (and accept) or until we get a sentence which cannot be simplified anymore (it contains the 'core' of the error, and we reject it). One simplification (or reduction) consists in rewriting some fixed length part of the sentence by a shorter string. Analysis by reduction can serve as a tool for deciding dependency relations in a given sentence. Let us illustrate it on an artificial example. Let $L = \{ a^i b^i c^j d^j \mid i, j \geq 0 \}$. Words from this language can be accepted by analysis by reduction with two reduction rules: if the word is of the form $a^*abb^*c^*d^*$, then ab can be rewritten by the empty word λ, in words of the form $a^*b^*c^*cdd^*$ the subword cd can be rewritten by λ. A word is accepted iff it can be simplified (reduced) to λ. All the possible

⋆ The first and the third author were partially supported by the Grant Agency of the Czech Republic, Grant-No. 201/02/1456, and by Grant-No. 201/04/2102. The work of the second author was supported by a grant from the Deutsche Forschungsgemeinsschaft.

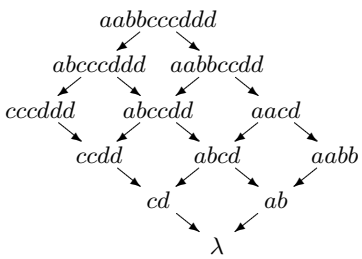


Fig. 1. All possible reductions on the word *aabbcccddd*.

reductions on a given word create a reduction system (see Fig. 1). If we want to construct a recognizer for the language L, we may use only some of the possible reductions, e.g., we can restrict attention to leftmost or rightmost reductions. But if we want to know whether there are some independent parts in the word considered, then we need the whole reduction system. In the words from L we can identify two places in which reductions can be performed independently (that is, the border between a's and b's, and the border between c's and d's). Later we will say that the corresponding restarting automaton has degree 2 of (left) (non)monotonicity.

A two-way restarting automaton (RLWW-automaton) M is a device with a finite state control and a read/write window of fixed size. This window moves (in both directions) on a tape containing a word delimited by sentinels until its control decides (nondeterministically) that the contents of the window should be rewritten by some shorter string. After rewriting, M can continue to move its window until it either halts and accepts, or halts and rejects, or restarts, that is, it moves its window to the leftmost position, enters the initial state, and continues with the computation. Thus, each computation of M can be described through a sequence of cycles. In this way M can be viewed as a *recognizer* and as a (regulated) *reduction system*, as well. An overview of research about restarting automata can be found in [2].

Several subclasses of RLWW-automata were studied. Among others, one-way versions (called RRWW-automata), several subclasses without working symbols (which are in general allowed) and (right-)monotone automata have been considered. A restarting automaton M is called right-monotone if during each computation of M, the places of rewriting do not increase their distances to the right end of the current word. In [4] some degrees and types of monotonicity were studied. The (degrees of) monotonicities describe important topological constraints for the rewriting systems (parsers) defined by RA. We are looking for such constraints in order to improve the modelling of the analysis by reduction of the individual languages and of syntactic phenomena.

Here we study the power of combinations of constraints, which can essentially restrict the 'analysis by reduction' defined by RA's. An interesting asymmetry between the right and left (strong) monotonicities will be shown for the one-way deterministic RA's. On the other hand the two-way deterministic RA's shows a

symmetric behavior in corresponding cases. The first aim of these results is to show the 'expressiveness' of the introduced 'system of constraints.' We use (invariantly) only one sequence of sample languages in order to obtain a (relatively) rich set of hierarchies of formal languages. Of course, this set is far from being complete. A second aim is to present the results for deterministic RA's, which in a way complete the results for nondeterministic RA's from [4].

2 Definitions

A *two-way restarting automaton*, RLWW-automaton for short, is a one-tape machine that is described by an 8-tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, where Q is the finite set of states, Σ is the finite input alphabet, Γ is the finite tape alphabet containing Σ, $¢, \$ \notin \Gamma$ are the markers for the left and right border of the work space, respectively, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the *read/write window*, and

$$\delta : Q \times \mathcal{PC}^{(k)} \to \mathfrak{P}((Q \times (\{\mathsf{MVR}, \mathsf{MVL}\} \cup \mathcal{PC}^{\leq(k-1)})) \cup \{\mathsf{Restart}, \mathsf{Accept}\})$$

is the *transition relation*, where $\mathfrak{P}(S)$ denotes the powerset of the set S.

$$\mathcal{PC}^{(k)} := (¢ \cdot \Gamma^{k-1}) \cup \Gamma^k \cup (\Gamma^{\leq k-1} \cdot \$) \cup (¢ \cdot \Gamma^{\leq k-2} \cdot \$)$$

is the set of *possible contents* of the read/write window of M, where $\Gamma^{\leq n} := \bigcup_{i=0}^{n} \Gamma^i$, and $PC^{\leq(k-1)} := \bigcup_{i=0}^{k-1} PC^{(i)}$.

The transition relation describes five different types of transition steps:

1. A *move-right step* is of the form $(q', \mathsf{MVR}) \in \delta(q, u)$, where $q, q' \in Q$ and $u \in \mathcal{PC}^{(k)}$, $u \neq \$$. If M is in state q and sees the string u in its read/write window, then this move-right step causes M to shift the read/write window one position to the right and to enter state q'.

2. A *move-left step* is of the form $(q', \mathsf{MVL}) \in \delta(q, u)$, where $q, q' \in Q$ and $u \in \mathcal{PC}^{(k)}$, $u \notin ¢ \cdot \Gamma^*$. It causes M to shift the read/write window one position to the left and to enter state q'.

3. A *rewrite step* is of the form $(q', v) \in \delta(q, u)$, where $q, q' \in Q$, $u \in \mathcal{PC}^{(k)}$, $u \neq \$$, and $v \in \mathcal{PC}^{\leq(k-1)}$ such that $|v| < |u|$. It causes M to replace the contents u of the read/write window by the string v, and to enter state q'. Further, the read/write window is placed immediately to the right of the string v. However, some additional restrictions apply in that the border markers ¢ and \$ must not disappear from the tape nor that new occurrences of these markers are created. Further, the read/write window must not move across the right border marker \$.

4. A *restart step* is of the form $\mathsf{Restart} \in \delta(q, u)$, where $q \in Q$ and $u \in \mathcal{PC}^{(k)}$. It causes M to move its read/write window to the left end of the tape, so that the first symbol it sees is the left border marker ¢, and to reenter the initial state q_0.

5. An *accept step* is of the form $\mathsf{Accept} \in \delta(q, u)$, where $q \in Q$ and $u \in \mathcal{PC}^{(k)}$. It causes M to halt and accept.

A *configuration* of M is a string $\alpha q \beta$, where $q \in Q$, and either $\alpha = \lambda$ and $\beta \in \{¢\} \cdot (\Sigma \cup \Gamma)^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot (\Sigma \cup \Gamma)^*$ and $\beta \in (\Sigma \cup \Gamma)^* \cdot \{\$\}$; here q is the current state, $\alpha\beta$ is the current contents of the tape, and the head scans the first k symbols of β or all of β when $|\beta| < k$. A *restarting configuration* is of the form $q_0 ¢ w \$$, where $w \in \Gamma^*$; it is an *initial configuration* if $w \in \Sigma^*$.

In general, the automaton M is *nondeterministic*, that is, there can be two or more instructions with the same left-hand side (q, u). If that is not the case, the automaton is *deterministic*.

An input *word* $w \in \Sigma^*$ *is accepted by* M, if there is a computation which, starting with the initial configuration $q_0 ¢ w \$$, finishes by executing an Accept instruction. By $L(M)$ we denote the language consisting of all words accepted by M; we say that M *recognizes (accepts) the language* $L(M)$.

We observe that any finite computation of a two-way restarting automaton M consists of certain phases. A phase, called a *cycle*, starts in a (re)starting configuration, the head moves along the tape performing MVR, MVL, and $\mathsf{Rewrite}$ operations until a $\mathsf{Restart}$ operation is performed and thus a new restarting configuration is reached. If no further restart operation is performed, any finite computation necessarily finishes in a halting configuration – such a phase is called a *tail*. We assume that M performs *exactly one* rewrite operation during any cycle – thus each new phase starts on a shorter word than the previous one. During a tail at most one rewrite operation may be executed.

We use the notation $u \vdash_M^c v$ to denote a cycle of M beginning with the restarting configuration $q_0 ¢ u \$$ and ending with the restarting configuration $q_0 ¢ v \$$; the relation $\vdash_M^{c^*}$ is the reflexive and transitive closure of $\vdash_M^c$. We often (implicitly) use the following obvious fact.

Fact 1 (Correctness preserving property).
Let M be a deterministic restarting automaton, and u, v be words in its input *alphabet. If $u \vdash_M^{c^*} v$ and $u \in L(M)$, then $v \in L(M)$.*

An RLW-*automaton* is an RLWW-automaton whose working alphabet coincides with its input alphabet. Note that each restarting configuration is initial in this case.

An RL-*automaton* is an RLW-automaton whose rewriting instructions can be viewed as deletions, that is, if $(q', v) \in \delta(q, u)$, then v is obtained by deleting some symbols from u.

An RRWW-*automaton* is an RLWW-automaton which does not use any MVL instructions. Analogously, we obtain RRW-*automata* and RR-*automata*.

An RWW-*automaton* is an RRWW-automaton which restarts immediately after rewriting. Analogously, we obtain RW-*automata* and R-*automata*.

Finally we come to the various notions of *monotonicity*. Any cycle C contains a unique configuration $¢\alpha q \beta \$$ in which a rewrite instruction is applied. Then $|\beta\$|$ is the *right distance* of C, denoted by $D_r(C)$, and $|¢\alpha|$ is the *left distance* of C, denoted by $D_l(C)$.

We say that a *sequence of cycles* $Sq = (C_1, C_2, \cdots, C_n)$ is *monotone* (or *right-monotone*) if $D_r(C_1) \geq D_r(C_2) \geq \ldots \geq D_r(C_n)$, and we say that this sequence is *left-monotone* if $D_l(C_1) \geq D_l(C_2) \geq \ldots \geq D_l(C_n)$. Finally, we call it right-left-*monotone* if it is both right- and left-monotone. We speak about *strong* monotonicities if we consider strong inequalities instead of $\geq$ in the above sequences.

For each of the prefixes $X \in \{\mathsf{right}, \mathsf{left}, \mathsf{right\text{-}left}\}$, a *computation* is X-*monotone* iff the corresponding sequence of cycles is X-monotone. Observe that the tail of the computation does not play any role here.

Let j be a natural number. We say that a sequence of cycles $Sq = (C_1, C_2, \cdots, C_n)$ is j-*monotone* (or j-*right-monotone*) if there is a partition of Sq into j (scattered) subsequences that are monotone. Formally we say that the sequence of cycles Sq is j-*monotone* iff there is a partition of Sq into j subsequences

$$\begin{aligned} Sb_1 &= (C_{1,1}, C_{1,2}, \cdots, C_{1,p_1}), \\ \cdots \quad & \quad \cdots \\ Sb_j &= (C_{j,1}, C_{j,2}, \cdots, C_{j,p_j}), \end{aligned}$$

where each Sb_i, $1 \leq i \leq j$, is monotone. Analogously, the notions of j-*left-monotonicity*, of j-right-left-*monotonicity* and the variants of strong monotonicities are defined.

Let $j \geq 1$, and let $X \in \{\mathsf{right}, \mathsf{left}, \mathsf{right\text{-}left}, \mathsf{strong\text{-}right}, \mathsf{strong\text{-}left}, \mathsf{strong\text{-}right\text{-}left}\}$. A *computation* is j-X-*monotone* iff the corresponding sequence of cycles is j-X-monotone. Again the tail of the computation does not play any role here. Finally, an RLWW-automaton M is called j-X-*monotone* iff all its computations are j-X-monotone. The prefixes j-X-mon- and sometimes X-mon- for 1-X-mon- are used to denote the corresponding classes of restarting automata. Observe that 1-X-monotonicity coincides with X-monotonicity.

Notation. For brevity, the prefix det- will be used to denote the property of being deterministic. For any class A of automata, $\mathcal{L}(\mathsf{A})$ will denote the class of languages recognizable by the automata from A. By CFL we denote the class of context-free languages, DCFL is the class of deterministic context-free languages. By $\subset$ we denote the proper subset relation. Further, we will sometimes use regular expressions instead of the corresponding regular languages. Finally, throughout the paper λ will denote the empty word and $\mathbb{N}_+$ will denote the set of all positive integers. The prefix j-left-mon will be often shortened to j-lm, similarly j-right-mon to j-rm, j-right-left-mon to j-rlm and strong to str. Combinations of constraints are denoted by concatenating them, e.g., an RRW-automaton which is 1-left-monotone and simultaneously strong-3-right-left-monotone will be called an (1-lm)-(str-3-rlm)-RRW-automaton.

The following fact will sometimes be used.

Fact 2 (Pumping Lemma). *For any* RLWW*-automaton M (with initial state q_0), there exists a constant p such that the following holds. Assume that $uvw \vdash_M^c uv'w$, where $u = u_1u_2u_3$ and $|u_2| = p$. Then there exists a factorization $u_2 = z_1z_2z_3$ such that z_2 is non-empty, and $u_1z_1(z_2)^iz_3u_3vw \vdash_M^c u_1z_1(z_2)^iz_3u_3v'w$ for*

all $i \geq 0$, that is, z_2 is a 'pumping factor' in the above cycle. Similarly, such a pumping factor can be found in any factor of length p of w. Such a pumping factor can also be found in any factor of length p of a word accepted in a tail computation.

3 Results

We start with an enhancement of a result from [4] which creates the basis for our further considerations.

Theorem 1. *For $j \in \mathbb{N}_+$, $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$ and $\mathsf{Y} \in \{\mathsf{str}, \lambda\}$:*

$$\begin{aligned}
&(1.)\quad \mathsf{DCFL} = \mathcal{L}(\mathsf{det\text{-}1\text{-}rm\text{-}X}) = \mathcal{L}(\mathsf{det\text{-}}j\mathsf{\text{-}rm\text{-}X}),\\
&(2.)\ \mathcal{L}(\mathsf{det\text{-}(Y\text{-}}j\mathsf{\text{-}rm)\text{-}(Y\text{-}1\text{-}lm)\text{-}X}) = \mathcal{L}(\mathsf{det\text{-}Y\text{-}1\text{-}rlm\text{-}X}),\\
&(3.)\quad \mathcal{L}(\mathsf{det\text{-}str\text{-}}j\mathsf{\text{-}rm\text{-}X}) = \mathcal{L}(\mathsf{det\text{-}str\text{-}1\text{-}rm\text{-}X}).
\end{aligned}$$

Remark. Let us present the previous theorem in an informal way. The first assertions says that any of the introduced classes of the one-way deterministic restarting automata, which are j-right-monotone, characterizes the class DCFL for any natural number j.

Similarly, any studied class of the one-way deterministic restarting automata, which are 1-left-monotone and at the same time j-right-monotone characterizes the same class of languages as the corresponding class of one-way deterministic restarting automata, which are 1-left-monotone and at the same time 1-right-monotone.

Similar assertions holds also for the strong variants of monotonicities.

Proof. We outline the idea of the proof. Let us suppose that M is a det-j-mon-X-automaton (or the strong variant) for some $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$, and k is the size of its read/write window. Then M scans in each cycle at least one symbol which was written in the previous cycle, that is, the right distance increases by less than k per cycle.

Let us recall that a sequence of cycles $(C_1, C_2, \cdots, C_n)$ is not j-monotone iff there exist $1 \leq i_1 < i_2 < ... < i_{j+1} \leq n$ such that $D_r(C_{i_1}) < D_r(C_{i_2}) < \cdots < D_r(C_{i_{j+1}})$. We can see from the previous two observations that $D_r(C_{i_1+i_2}) - D_r(C_{i_1}) \leq j \cdot k$ for $1 \leq i_1 \leq i_1 + i_2 \leq n$. This means that any sequence of (at most j) consecutive cycles with increasing right distances can be simulated by a single cycle of a det-rm-X-automaton $M_{j,k}$ that has a read/write window of size $j \cdot k$. We can see that $M_{j,k}$ can be constructed in such a way that it is right-monotone and recognizes the same language as M. If M is strongly j-right-monotone then $M_{j,k}$ is strongly right-monotone. If M is j-right-monotone and 1-left-monotone at the same time then $M_{j,k}$ is 1-right-left-monotone. Since $\mathcal{L}(\mathsf{det\text{-}mon\text{-}X}) = \mathsf{DCFL}$ for each $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}, \mathsf{RWW}, \mathsf{RRWW}\}$ (cf. [4]), our proof is complete. □

In the rest of the paper we will deal with a generalization of the example from Section 1. For $j \geq 2$, let $L^{(j)}$ denote the language

$$L^{(j)} := \{\, a^{m_1} b^{m_1} \dots a^{m_j} b^{m_j} \mid m_1, \dots, m_j > 0 \,\}.$$

The language $L^{(j)}$ can be recognized by a deterministic R-automaton $M^{(j)}$ that works in the following way:

- $M^{(j)}$ directly accepts (without restart) the word $(ab)^j$.
- If the current word has a prefix of the form $(ab)^i a^*$ for some $(0 \leq i \leq j-1)$ followed by abb, then $M^{(j)}$ deletes ab out of the factor abb and restarts.
- All words of any other form are rejected by $M^{(j)}$.

It is easily verified that $M^{(j)}$ accepts the language $L^{(j)}$.

Lemma 1. *$M^{(j)}$ is a* det-(str-j-rlm)-(str-1-rm)-R-*automaton.*
In other words $M^{(j)}$ is a deterministic R-*automaton, which is* strongly j-right-left-*monotone and at the same time* strongly 1-right-*monotone.*

Proof. It is easily seen that every computation of $M^{(j)}$ is a 1-str-rm one.

Further we will show that each computation of $M^{(j)}$ on a word of the form $a^{m_1} b^{m_1} \dots a^{m_j} b^{m_j}$, where $m_1, \dots, m_j > 1$ can be divided into j strong-right-left-monotone (continuous) subsequences as follows. For each $r = 1, \dots, j$, the r-th subsequence consists of those cycles that delete factors of the form ab, where the corresponding prefix is of the form $¢(ab)^{r-1} a^p$, where $0 \leq p < m_r$. The i-th cycle in the r-th subsequence has left distance $1 + 2(r-1) + m_r - i$, and hence, each of these subsequences is strong-left-monotone.

As the union of these subsequences is the complete computation of $M^{(j)}$, we see that $M^{(j)}$ is indeed strong-j-right-left-monotone. □

Remembering our illustrative example from Section 1, we can say that $M^{(j)}$ makes leftmost reductions only. Actually, when we have a deterministic one-way restarting automaton without working symbols, i.e., an RRW-automaton (or one of its more restricted variants), then we can accept the language $L^{(j)}$ only in this way. This is shown in the following lemma.

Lemma 2. $L^{(j)} \notin \mathcal{L}$(det-$(j-1)$-lm-RRW).
In other words $L^{(j)}$ cannot be recognized by a deterministic (j - 1) - left - monotone RRW-*automaton.*

Proof. Assume to the contrary that M is a deterministic RRW-automaton for $L^{(j)}$, and that M is $(j-1)$-left-monotone. As M has no auxiliary symbols, each rewrite operation of M that is applied during an accepting computation must transform a word from $L^{(j)}$ into another (shorter) word of $L^{(j)}$ by Fact 1. Let r be a sufficiently large constant, larger than the size of M's read/write window, and also larger than the constant p from the pumping lemma, and let w be the following element of $L^{(j)}$:

$$w := a^r b^r a^{2r} b^{2r} \dots a^{jr} b^{jr}\,.$$

No word from $L^{(j)}$ containing a subword a^p can be accepted by M without a restart, because otherwise we can apply the pumping lemma on the subword a^p to get an accepting tail for a word that does not belong to $L^{(j)}$.

There is only one type of rewrite transition that M can possibly apply to w: M can replace a factor $a^m b^m$ by $a^{m-i} b^{m-i}$ for some $0 < i \leq m$.

If M does not use such a transition on the first syllable of the form $a^p b^p$ to which it is applicable, then it will never be able to apply a transition of this form to that syllable, because M is deterministic, and it cannot move left again on realizing that no transitions are applicable to any factor to the right of this particular syllable. Thus, M must apply these transitions at the first possible position. This means that M rewrites the syllables $a^{ir} b^{ir}$ strictly from left to right, that is, M behaves essentially like the R-automaton $M^{(j)}$ above, that is, M is in particular not $(j-1)$-lm. □

Hence,

$L^{(j)} \in \mathcal{L}(\text{det-(str-}j\text{-rlm)-(str-1-rm)-R}) \smallsetminus \mathcal{L}(\text{det-}(j-1)\text{-lm-RRW})$.

Lemma 1 and Lemma 2 have the following consequences showing that j-left- and j-right-left-monotone deterministic one-way restarting automata are stronger with increasing j. This is in contrast to the situation for j-right-monotone deterministic one-way automata (without working symbols) (cf. Theorem 1).

Corollary 1. *For* $j \in \mathbb{N}_+$, $\mathsf{X} \in \{\mathsf{R}, \mathsf{RR}, \mathsf{RW}, \mathsf{RRW}\}$, *and* $\mathsf{Y}, \mathsf{Z} \in \{\mathsf{str}, \lambda\}$,

(1.) $\mathcal{L}(\text{det-Y-}j\text{-lm-X}) \subset \mathcal{L}(\text{det-Y-}(j+1)\text{-lm-X})$.
(2.) $\mathcal{L}(\text{det-Y-}j\text{-rlm-X}) \subset \mathcal{L}(\text{det-Y-}(j+1)\text{-rlm-X})$.
(3.) $\mathcal{L}(\text{det-(Y-}j\text{-lm)-(Z-1-rm)-X}) \subset \mathcal{L}(\text{det-(Y-}(j+1)\text{-lm)-(Z-1-rm)-X})$.
(4.) $\mathcal{L}(\text{det-(Y-}j\text{-rlm)-(Z-1-rm)-X}) \subset \mathcal{L}(\text{det-(Y-}(j+1)\text{-rlm)-(Z-1-rm)-X})$.

Deterministic two-way restarting automata can recognize the languages $L^{(j)}$ also using only rightmost reductions. Actually, for each $j \geq 2$, we can construct a det-RL-automaton $M_1^{(j)}$ for the language $L^{(j)}$. At the beginning of each cycle, $M_1^{(j)}$ moves to the right end of the current word and then it continues in a way symmetrical to the behavior of $M^{(j)}$.

Using the symmetry in the computations of $M_1^{(j)}$ in relation to the computations of $M^{(j)}$, it is easily seen that $M_1^{(j)}$ is a det-(str-j-rlm)-(str-1-lm)-RL-automaton (cf. Lemma 1). Thus, we have the following.

Lemma 3. $L^{(j)} \in \mathcal{L}(\text{det-(str-}j\text{-rlm)-(str-1-lm)-RL})$.
It means that $L^{(j)}$ *is recognized by a deterministic RL-automaton, which is* strongly j-right-left-*monotone and at the same time* strongly 1 - left-monotone.

When we require that a deterministic RLW-automaton M recognizing $L^{(j)}$ is 1-left-monotone, then it is easily seen that M is not $(j-1)$-right-monotone, that is, we have the following result.

Lemma 4. $L^{(j)} \notin \mathcal{L}(\mathsf{det}\text{-}((j-1)\text{-}\mathsf{rm})\text{-}(1\text{-}\mathsf{lm})\text{-}\mathsf{RLW})$.
It means that $L^{(j)}$ *is not recognized by any deterministic* RL-*automaton, which is* $(j-1)$-right-*monotone and at the same time* 1 - left-monotone.

Hence,

$$L^{(j)} \in (\mathcal{L}(\mathsf{det}\text{-}(\mathsf{str}\text{-}j\text{-}\mathsf{rlm})\text{-}(\mathsf{str}\text{-}1\text{-}\mathsf{lm})\text{-}\mathsf{RL})) \setminus \mathcal{L}(\mathsf{det}\text{-}((j-1)\text{-}\mathsf{rm})\text{-}(1\text{-}\mathsf{lm})\text{-}\mathsf{RLW}).$$

Thus, we obtain the following consequence.

Corollary 2. *For each* $j \in \mathbb{N}_+$, $\mathsf{X} \in \{\mathsf{RL}, \mathsf{RLW}\}$, *and* $\mathsf{Y}, \mathsf{Z} \in \{\mathsf{str}, \lambda\}$,

(1.) $\mathcal{L}(\mathsf{det}\text{-}(\mathsf{Y}\text{-}1\text{-}\mathsf{lm})\text{-}(\mathsf{Z}\text{-}j\text{-}\mathsf{rm})\text{-}\mathsf{X}) \subset \mathcal{L}(\mathsf{det}\text{-}(\mathsf{Y}\text{-}1\text{-}\mathsf{lm})\text{-}(\mathsf{Z}\text{-}(j+1)\text{-}\mathsf{rm})\text{-}\mathsf{X})$.
(2.) $\mathcal{L}(\mathsf{det}\text{-}(\mathsf{Y}\text{-}j\text{-}\mathsf{rlm})\text{-}(\mathsf{Z}\text{-}1\text{-}\mathsf{lm})\text{-}\mathsf{X}) \subset \mathcal{L}(\mathsf{det}\text{-}(\mathsf{Y}\text{-}(j+1)\text{-}\mathsf{rlm})\text{-}(\mathsf{Z}\text{-}1\text{-}\mathsf{lm})\text{-}\mathsf{X})$.
(3.) $\mathcal{L}(\mathsf{det}\text{-}(\mathsf{Y}\text{-}1\text{-}\mathsf{rm})\text{-}(\mathsf{Z}\text{-}j\text{-}\mathsf{lm})\text{-}\mathsf{X}) \subset \mathcal{L}(\mathsf{det}\text{-}(\mathsf{Y}\text{-}1\text{-}\mathsf{rm})\text{-}(\mathsf{Z}\text{-}(j+1)\text{-}\mathsf{lm})\text{-}\mathsf{X})$.
(4.) $\mathcal{L}(\mathsf{det}\text{-}(\mathsf{Y}\text{-}j\text{-}\mathsf{rlm})\text{-}(\mathsf{Z}\text{-}1\text{-}\mathsf{rm})\text{-}\mathsf{X}) \subset \mathcal{L}(\mathsf{det}\text{-}(\mathsf{Y}\text{-}(j+1)\text{-}\mathsf{rlm})\text{-}(\mathsf{Z}\text{-}1\text{-}\mathsf{rm})\text{-}\mathsf{X})$.

Remark. It follows from [4] that all the classes of languages from the previous theorem are included in CFL and $\mathcal{L}(\mathsf{det}\text{-}\mathsf{str}\text{-}2\text{-}\mathsf{rlm}\text{-}\mathsf{RL})$ is incomparable with CFL.

4 Conclusion

We have seen that based on the degree of (non)monotonicity, we obtain some new infinite hierarchies of languages that are obtained by combinations of constraints for deterministic restarting automata without auxiliary symbols. We have used only one sequence of sample languages from DCFL with some simple symmetry properties. It was shown that the degrees of right-monotonicities and left-monotonicities lead for the one-way deterministic restarting automata to asymmetric (different) types of results. On the other hand a 'symmetric' behavior was obtained for the two-way deterministic restarting automata. We believe that we have sufficiently illustrated the power of the introduced combinations of (topological) constraints for restarting automata (considered as syntactical analyzers). However, it is clear that the strength of the combinations of constraints needs a more complete investigation.

References

1. K. Oliva, P. Květoň and R. Ondruška: The Computational Complexity of Rule-Based Part-of-Speech Tagging. To appear in: *Proceedings of TSD 2003*, to be published in the LNAI Series by Springer Verlag.
2. F. Otto: Restarting Automata And Their Relations to the Chomsky hierarchy. In: Z. Esik, Z. Fülöp (eds.): *Developments in Language Theory, Proceedings of DLT'2003*, LNCS 2710, Springer Verlag, Berlin, 2003, 55 - 74.
3. M.Plátek, M. Lopatková, K. Oliva: *Restarting Automata: Motivations and Applications.* In: M. Holzer (ed.), Workshop 'Petrinetze' and 13. Theorietag 'Formale Sprachen und Automaten', Proceedings, Institut für Informatik, Technische Universität München, 2003, 90-96.
4. M. Plátek, F. Otto, F. Mráz: *Restarting Automata and Variants of j-Monotonicity.* Proceedings of DCFS' 2003, MTA SZTAKI, Budapest, (2003) 303 - 312.

Syntax naruby

Karel Oliva

Ústav pro jazyk český
Akademie věd České republiky
Letenská 4, CZ-110 00 Praha 1 - Malá Strana,
e-mail: oliva@ujc.cas.cz
a
Österreichisches Forschungsinstitut für Artificial Intelligence
Freyung 6/6
A-1010 Wien
e-mail: karel@oefai.at

Abstract. Článek se zabývá problémem explicitního formálního popisu negramatických konstrukcí, který byl až dosud zanedbáván. V první části článku jsou uvedeny důvody, proč je jak teoreticky významné tak prakticky důležité negramatickým konstrukcím věnovat pozornost. Další části potom motivují a na příkladech ilustrují metodu, jak negramatické konstrukce systematicky zachycovat pomocí formálního aparátu.

1 Úvahy o negramatičnosti a jejím formálním popisu

Ve formální lingvistice je (přirozený) jazyk obvykle chápán jako množina řetězců sestávajících ze slovních forem tohoto jazyka. Formálně se tento předpoklad vyjadřuje tak, že se říká, že (přirozený) jazyk L je podmnožinou množiny T^* ($L \subset T^*$) všech možných řetězců utvořených z prvků množiny T (což je právě množina všech možných forem slov jazyka - množina všech slovních forem odpovídajících různým pádům, číslům, časům, osobám, ...). O jakou podmnožinu se přitom jedná, je (nebo se předpokládá, že by mělo být) exaktně vymezeno formální gramatikou G, jejíž množinou terminálních symbolů je právě T. Vztah mezi jazykem L a gramatikou G se obyčejně vyjadřuje zápisem $L = L(G)$, jehož smysl je vyznačit, že L je jazyk definovaný gramatikou G.

Kromě určení, zda určitý řetězec σ patří do jazyka L ($\sigma \in L$), má gramatika jazyka L obvykle ještě další úkol: pokud je σ prvkem L, má gramatika přiřadit řetězci σ (a obecně každému řetězci, který je prvkem jazyka L) jeho (syntaktickou) strukturu (příp. struktury, pokud je řetězec syntakticky víceznačný). Myšlenka, která se za tím skrývá, by se dala vyjádřit zhruba tak, že možnost či nemožnost přiřazení syntaktické struktury je právě to, co odlišuje správně tvořené řetězce (věty) od řetězců nesprávných (negramatických), jinými slovy, že úloha určit, zda je řetězec σ prvkem jazyka $L(G)$, a úloha přiřadit řetězci σ strukturu podle gramatiky G jsou vlastně identické.

Pokud se však nad problémem hlouběji zamyslíme, nebude těžké dospět k závěru, že takový pohled je přespříliš zjednodušující a kromě toho omezuje možné aplikace. Zjednodušující je proto, že předpokládá, že každý řetězec σ, kterému není možné podle nějaké gramatiky G popisující (tj. v praxi: snažící se popsat) přirozený jazyk L přiřadit strukturu, je negramatický i v neformálním smyslu (tj. nepřijatelný, nesprávný podle názoru rodilých mluvčí). Takový předpoklad je však nerealistický, tak přesné popisy žádného přirozeného jazyka neexistují (a vůbec není jisté, že kdy existovat budou - jazyk je v neustálém vývoji, přespříliš těsně souvisí s lidským myšlením atd. na to, aby bylo zřejmé, že je vůbec možné jej popsat jako fixně danou množinu vět). Omezující v oblasti aplikací je takový pohled proto, že vymezuje pouze linii mezi větami, které jsou spolehlivě gramatické (totiž těmi, kterým gramatika G přiřazuje strukturu) a ostatními, o kterých se neví:

– zda jsou skutečně negramatické (v preteoretickém smyslu),
– zda jsou to správně tvořené věty příslušného přirozeného jazyka, které ale obsahují jevy, jež zatím nebyly podrobeny formálnímu popisu,

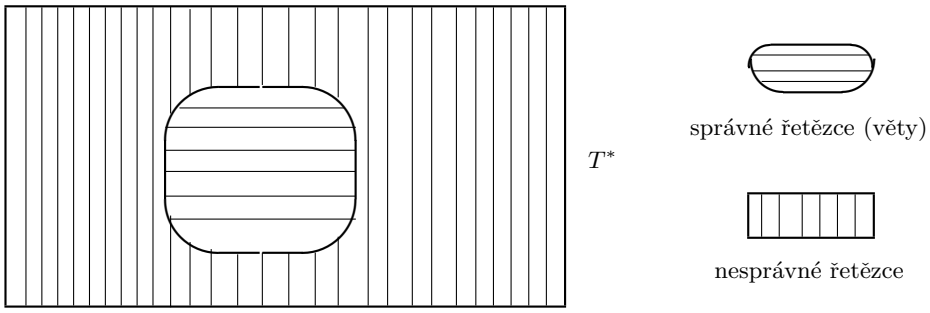
nebo

– zda jsou to věty, jejichž přijatelnost je sporná (jsou "na hraně" mezi intuitivně chápanou správností a nesprávností).

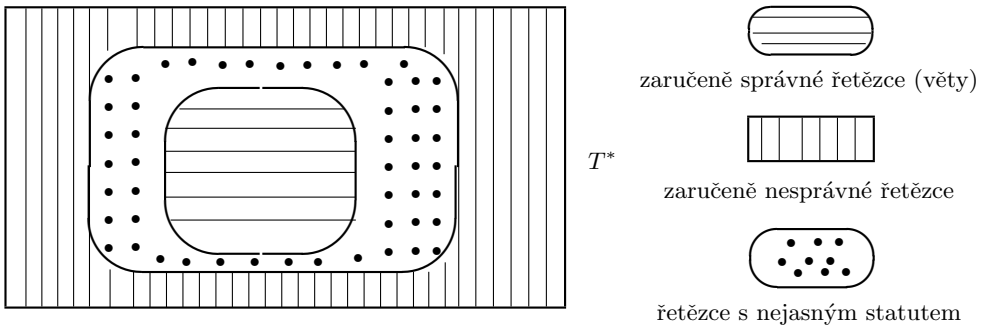
Pro některé aplikace - např. pro automatickou kontrolu syntaktické správnosti (grammar-checking) či na pravidlech založenou disambiguaci textu (značkování jazykového korpusu) - je tento pohled navíc nevhodný proto, že z úhlu pohledu těchto aplikací je vymezení množiny "zaručeně" nesprávných vět daleko důležitější než vymezení vět "zaručeně" správných.

Dovedeme-li předchozí úvahy do důsledků, dostaneme místo obvyklého bipartitního modelu, který dělí řetězce na gramatické a negramatické (viz obr. 1), model tripartitní, ve kterém je množina všech řetězců rozdělena do tří skupin:

– na řetězce (zaručeně) správné,
– řetězce (zaručeně) nesprávné,
– řetězce, o jejichž správnosti či nesprávnosti není (at' už z jakéhokoliv důvodu) jasno (viz obr. 2).



Obr.1



Obr. 2

Pokud dále přijmeme pohled, že obrázek 2 je správný (míní se: správnější, více odpovídající jazykové realitě než obr. 1) a že tedy má sloužit jako základ pro formalizaci jazyka, vyplyne z toho nutnost formálně popsat alespoň dvě ze tří disjunktních množin řetězců, které zde vystupují (třetí množina bude pak výsledkem množinového rozdílu mezi

sjednocením těchto dvou množin a množinou T^*). V principu je výběr, které dvě ze tří množin popíšeme, samozřejmě zcela otevřený, prakticky je však zřejmé, že je možné formálně popisovat jen ty řetězce, jejichž gramatický status je jasně definovaný, tj. buď řetězce, kde můžeme o všech jevech, které se v nich vyskytují, s jistotou tvrdit, že jsou gramatické (a popsat je jako takové), nebo řetězce, kde dochází k jasnému a nepochybnému porušení gramatičnosti (a je tedy možné explicitně popsat toto porušení).

Zatímco popis gramatických konstrukcí je zcela standardní záležitostí a nástroje k jeho realizaci (formální gramatiky) patří k běžné výbavě formální lingvistiky, úloha popisovat negramatické konstrukce se zdá být zcela nová - a zejména není zřejmá žádná metoda, jak takový úkol systematicky plnit. Hlavním úkolem tohoto článku tedy bude nastínit základy jednoho možného přístupu k tomuto problému.

Výchozím bodem všech úvah, které v dalším povedeme, je uvědomit si, že:

- gramatičnost a negramatičnost jsou vždy definovány pro celé věty (nejen pro jejich části - alespoň ne v obecném případě)
- negramatičnost je vždy důsledkem porušení určitého jazykového jevu (tj. porušení zákonitostí, které výskyt tohoto jevu s sebou přináší: např. nutnost shody podmětu s přísudkem v osobě a čísle apod.).

Z toho plyne, že je rozumné, aby popisu "zřetelných" chyb předcházela systematická klasifikace jazykových jevů, tak, jak se vyskytují ve větách. Jako možná (a rozumná) se přitom jeví klasifikace jazykových jevů do následujících tří tříd:

– **jevy selekční:** V širokém porozumění je selekce (chápaná jako zobecnění pojmu valence) požadavek nutného výskytu určitého prvku (slova) E_1 ve větě, pokud se v téže větě vyskytuje slovo E_2 (příp. skupina slov $E_2, E_3, ..., E_n$); tj. jestliže se ve větě vyskytuje slovo E_2 (resp. skupina slov $E_2, E_3, ..., E_n$), ale slovo E_1 se v ní nevyskytuje, je zákonitost vyžadovaná příslušným selekčním jevem porušena a věta je negramatická.

Příklad: v češtině vyžaduje sloveso *potřebovat* jako svůj předmět podstatné jméno ve 4. pádě (příp. infinitiv či vedlejší větu); ve větách, které obsahují finitní tvar slovesa *potřebovat* a přitom nejsou eliptické (např. obsahují ještě podmět), se proto nutně musí vyskytovat i takový předmět - podstatné jméno (příp. jiné vyjádření předmětu): srv. negramatičnost (nepřijatelnost, nesprávnost) věty *Jan potřebuje.*

– **jevy slovního pořádku:** Jde o jazykové zákonitosti, které definují vzájemný pořádek dvou nebo více slov $E_1, E_2, ...$ ve větném kontextu. Pokud tento pořádek není dodržen, je příslušná zákonitost porušena a věta je pokládána za negramatickou.

Příklad: pokud dva příklonné (klitické) tvary osobních zájmen, z nichž jedno je ve 3. pádě a jedno je ve 4. pádě, "patří" ke stejnému slovesu (např. jako - po řadě - jeho nepřímý a přímý předmět), pak musí stát zájmeno ve 3. pádě před zájmenem ve 4. pádě. Např. v řetězci slov *Jan ho mu dal* je tato zákonitost porušena, a proto jej nelze považovat za správně tvořenou větu.

– **jevy souvýskytu rysů** (souvýskytem rysů se přitom rozumí zobecnění pojmu shoda): Jevy tohoto typu jsou charakterizovány vlastností, že pokud se v kontextu věty spolu vyskytují dvě nebo více slov $E_1, E_2, ...$, pak některé jejich morfologické charakteristiky musejí být určitém systematickém vztahu (často je takovým vztahem identita, tedy nejjednodušší možný systematický vztah, existují však i případy vztahů daleko složitějších). Pokud tento požadavek není splněn, je věta - stejně jako v minulých případech - nutně pokládána za negramatickou.

Příklad: pokud spolu tvar pomocného slovesa *být* a tvar minulého příčestí významového slovesa tvoří tvar minulého času (např. *přišel jsem, viděly jste*), musejí se pomocné sloveso a příčestí shodovat v (gramatickém) čísle. Např. tedy řetězec slov *Do kina jsem včera přišli pozdě* je tedy nutné pokládat za negramatický.

Podrobnější úvahy o výše uvedeném přehledu tříd jazykových jevů naznačují, že na každý řetězec, který porušuje zákonitosti nějakého jevu, lze pohlížet jako na rozšíření (extenzi) nějakého minimálního negramatického řetězce, tj. řetězce, který obsahuje pouze materiál nutný právě pro takové porušení gramatičnosti. Například tedy řetězec *Viděl jsem v tom broušeném zrcadle náhle mě* lze zřejmě pokládat za extenzi řetězce *jsem mě*, který porušuje zákonitosti o reflexivitě vážící se k subjektu v češtině (správně by mělo být *jsem sebe*), a to zcela stejným způsobem jako delší řetězec původní.

To znamená, že v každém negramatickém řetězci lze najít (alespoň) jeden minimální negramatický (pod)řetězec a že tedy nástroje k popisu negramatických konstrukcí se mohou značně podobat (regulárním) gramatikám. Konkrétně jde o to, že

- každý minimální negramatický řetězec lze zřejmě popsat jako posloupnost jeho prvků (tj. regulárním výrazem)
- každý negramatický řetězec lze popsat jako rozšíření (prodloužení zleva, zprava i "zevnitř") nějakého minimálního řetězce, konkrétně jako prodloužení pomocí takových prvků, jejichž vložení nezpůsobí, že se řetězec stane gramatickým.

V obou případech nelze samozřejmě zanedbat, že se vždy musí jednat o řetězce větné délky, tj. o řetězce pokrývající větu od začátku do konce. Pro formální vyjádření tohoto požadavku zavedeme v následujícím speciální symboly $'\{'$ (pro abstraktní pozici "začátek věty", tj. před prvním slovem) a $'\}'$ (pro abstraktní pozici "konec věty", tj. typicky za tečkou), o kterých budeme předpokládat, že se nikde jinde nevyskytují.

Příklad: Minimálním negramatickým řetězcem, který lze přiřadit jako zdroj negramatičnosti k řetězci *Viděl jsem v tom broušeném zrcadle náhle mě*, je (v obecné podobě) konfigurace (1), kde je znaménko $'+'$ použito jako separátor a abstraktní lingvistické kategorie jsou vyznačeny seznamem svých význačných rysů uzavřeným do hranatých závorek.

(1) { + [cat: verb, num: sg, osoba: 1] + mě + }

Takový (abstraktní) regulární výraz reprezentuje řetězce (věty) jako např. *Vidím mě, Nenávidím mě*, ale samozřejmě i další, kde je zdrojů negramatičnosti více (např. *Spím mě, Bojím mě)*; mezi reprezentované řetězce patří i konkrétní minimální negramatický řetězec příkladové věty, tj. řetězec slov *Jsem mě*. Lingvisticky je základem negramatičnosti konfigurace (1) porušení požadavku, aby v jevu anaforického odkazu k subjektu (jev typu souvýskyt) bylo použito zvratné zájmeno (a nikoliv zájmeno osobní). Z toho lze snadno odvodit podmínku, kterou musí splňovat jakékoliv rozšíření této konfigurace takové, že bude nutně negramatické: nesmí být přidáno žádné slovo, které by mohlo vázat slovní tvar *mě* (tj. řetězec by se mohl stát gramatickým, kdyby se v něm takové slovo objevilo - proto je pro spolehlivé zachování negramatičnosti právě potřeba klást podmínku, že se takové slovo objevit nesmí). Obecně je vymezení takové podmínky lingvisticky značně složitá úloha, spokojíme se zde proto s velmi jednoduchým (a velmi neúplným) řešením, totiž že povolíme, aby v "mezerách" mezi prvky minimální konfigurace stály pouze adverbia, předložky nebo spojky (možnost řetězců složených z těchto elementů vyznačíme Kleeneho '*'). Zobecněná negramatická konfigurace bude tedy mít tvar regulárního výrazu v (2), kde '$\vee$' je symbol disjunkce.

(2) { + ([cat: adv] $\vee$ [cat: prep] $\vee$ [cat: conj])*
+ [cat: verb, num: sg, osoba: 1]
+ ([cat: adv] $\vee$ [cat: prep] $\vee$ [cat:conj])*
+ mě
+ ([cat: adv] ([cat: prep] ([cat: conj])* + }

(2) je tak výsledný tvar jedné negramatické konfigurace. Každý řetězec slov, která svou morfologickou charakteristikou či tvarem jednoznačně odpovídají řetězci (2), je spolehlivě negramatická věta (v češtině).

Z předchozího vyplývá, že významnou podmnožinu (nebo dokonce všechny prvky) množiny všech negramatických řetězců lze popsat jako rozšířené negramatické řetězce. Ty lze získat extenzí z minimálních negramatických řetězců. Problémem, který zatím zůstává neřešen, je, jak získat tyto minimální řetězce pokud možno systematicky, tj. tak, aby žádný nebyl vynechán. Jako základní myšlenka se zde nabízí postupovat podle délky - nejprve zjistit (a popsat) všechny minimální negramatické řetězce délky 1, potom všechny takové řetězce délky 2 atd. Naskýtají se přitom dvě zásadní otázky:

- zda je pro určitou pevně danou délku počet příslušných minimálních negramatických řetězců omezen
- zda existuje horní hranice hranice délky minimálních negramatických řetězců tj. že žádný negramatický řetězec, který je delší než toto maximum, není minimálním negramatickým řetězcem a lze jej tedy redukovat na kratší negramatický řetězec).

Odpovědi na tyto otázky nejsou nijak samozřejmě jasné, pokud však budeme předpokládat, že je konečný počet jazykových jevů a že konkrétní realizace každého jevu zasahuje jen konečný počet slov (slovních tvarů), dospějeme k hypotéze, že obě odpovědi jsou kladné a že tedy množina minimálních negramatických řetězců je konečná. (Jistotu ovšem nemáme, protože uvedené předpoklady konečnosti nemusejí být správné, byť se intuitivně správné být zdají.)

Třetí, přidruženou otázkou pak je, zda (či za jakých předpokladů) je konečná i množina všech rozvolněných negramatických řetězců. Pokud předpokládáme konečnost množiny minimálních negramatických řetězců, redukuje se tato otázka na problém, zda existuje alepoň jeden minimální negramatický řetězec takový, že není možné všechna jeho rozvolnění na rozvolněné negramatické řetězce popsat konečným počtem regulárních výrazů. V tomto článku budeme ve shodě s intuicí předpokládat, že takový minimální řetězec neexistuje, tj. že libovolný minimální řetězec lze rozvolnit jen konečně mnoha způsoby. Pokusme se tedy nyní na příkladu několika nejjednodušších negramatických řetězců ilustrovat, jak by taková "ne-gramatika" (a práce na jejím vývoji) mohla vypadat.

2 Minimální negramatické řetězce délky 1 a jejich rozvolnění

Jevy, jejichž zákonitosti mohou být porušeny v řetězci délky 1, jsou pouze jevy typu selekce. Je totiž zřejmě vyloučeno, aby jedno jediné slovo, které se v takovém řetězci vyskytuje, porušovalo zákonitosti jakéhokoliv jevu typu slovní pořádek (nelze "vzájemně uspořádat" jeden prvek, a tedy nelze takové uspořádání ani porušit), a podobně není ani možné, aby rysy samostatného (jediného) slova porušovaly jakýkoliv je typu souvýskyt, prostě proto, že se o žádný souvýskyt morfologických rysů v různých slovech zřejmě nejedná.

Typickým selekčním jevem je slovesná valence - příklad jejího porušení (konkrétně nenaplnění valence slovesa *potřebovat*) byl uveden již výše. Problém se slovesnou valencí jako s příkladem porušení gramatičnosti v jednoslovných řetězcích je v tom, že v případě rozsáhlých elips je (povrchové) nenaplnění možné (otázka:*Potřebuješ něco ?* odpověď:*Potřebuju.*) a prohlásit takové řetězce za negramatické v obecném případě tedy nelze. Je proto potřeba se při hledání příkladu minimálního negramatického řetězce porozhlédnout jiným směrem: jako dobrý příklad se nabízejí klitiky (příklonky, předklonky), tedy slova, která nemohou existovat samostatně, ale musejí se "foneticky opírat" o slovo jiné, a tím tedy vynucují jeho přítomnost. Příkladem klitik v češtině jsou některé "krátké" tvary osobních zájmen *(mi, ti, mu, ho)* a "reflexivních zájmen" *(si, se)* - u *se* je přitom ještě potřeba vzít v úvahu, že jde o slovo homonymní (*se* je i vokalizovaná forma předložky *s*). Jako příklad minimálního negramatického řetězce je tedy možno uvést (3).

(3) { + (mi ∨ ti ∨ mu ∨ ho ∨ [cat: pron, form: se] ∨ si) + }

Všechna uvedená slova jsou enklitiky, příklonky, tj. slova, která se "foneticky opírají" o slovo, které jim bezprostředně předchází. Minimální negramatickou konfiguraci (3) můžeme tedy rozvolnit na konfiguraci (4), kde za příklonkou následuje libovolný počet libovolných prvků (to je poněkud zjednodušeně vyznačeno speciálním symbolem ANY). (4) je spolehlivě negramatická: její negramatičnost vyplývá z toho, že není splněn požadavek, že příklonku musí alespoň jeden prvek (jedno slovo) předcházet.

(4) { + (mi ∨ ti ∨ mu ∨ ho ∨ [cat: pron, form: se] ∨ si) + ANY* + }

3 Minimální negramatické řetězce délky 2 a jejich rozvolnění

V řetězcích o délce dvou slov již je možné nalézt porušení jazykových jevů všech typů. Jako příklady je možné si zde uvést následující řetězce (je přitom samozřejmé, že budou uvedeny jen takové příklady negramatických řetězců délky 2, které nejsou rozšířením minimálních negramatických řetězců délky 1).

3.1 Porušení selekce

Jde o případ, který jsme již stručně uvedli výše: pokud má některé sloveso (povrchově) obligatorní valenci pro předmět, ten však není ve větě přítomen a přitom nejde o větu eliptickou (např. je ve větě přítomen podmět), jedná se o negramatickou konstrukci. Takové minimální negramatické řetězce mají abstraktní strukturu popsanou formální v (5)a,b.

(5) a. { + [cat: n, case: nom] + [cat: v,oblig_val: objekt] + }
b. { + [cat: v,oblig_val: objekt] + [cat: n, case: nom] + }

Rozvolnění je možné přidáním jakýchkoliv prvků, které spolehlivě nemohou být předmětem slovesa, triviálně tedy např. libovolným počtem příslovcí a/nebo podstatných jmen v nominativu. Takové prvky přitom mohou být při rozvolňování přidány na libovolné místo původního řetězce, tj. rozvolnění (5)a,b mohou mít tvar (6)a,b (jedná se vskutku o rozvolnění velmi jednoduchá, obecně by bylo možné formulovat podmínky podstatně volnější - a tím ovšem složitější).

(6) a. { + ([cat: n, case: nom] ∨ [cat: adv])*
+ [cat: n, case: nom]
+ ([cat: n, case: nom] ∨ [cat: adv])*
+ [cat: v,oblig_val: objekt]
+ ([cat: n, case: nom] ([cat: adv])* + }

b. (+ ([cat: n, case: nom] ∨ [cat: adv])*
+ [cat: v,oblig_val: objekt]
+ ([cat: n, case: nom] ∨ [cat: adv])*
+ [cat: n, case: nom]
+ ([cat: n, case: nom] ∨ [cat: adv])* + }

3.2 Porušení slovního pořádku

Porušení slovního pořádku v minimálním negramatickém řetězci o délce 2 nastane tehdy, když řetězec sestává ze dvou slov S_1, S_2 (v tomto pořadí), která - z nějakých důvodů, které mohou být obecně i velmi komplikované - mohou v gramatické větě stát těsně vedle sebe pouze v pořadí S_2, S_1 (tj. pořadí S_1, S_2 je vždy negramatické, zatímco pořadí S_2, S_1 je možné:

pokud by nebylo možné ani jedno z nich, šlo by zřejmě buď o porušení jevu jiného typu, nebo by nešlo o řetězec mimimální délky). Jako příklad takového negramatického řetězce může posloužit (7).

(7) { + [cat: v, vform: fin] + [cat: subord_conj] + }

Konfigurace, ve které těsně za sebou stojí finitní tvar slovesa a podřadicí spojka, je zřejmě v češtině vyloučená - přitom ovšem neplatí, že podřadicí spojka musí stát vždy na začátku (jednoduché) věty, srv. příklady v (8), kde spojce postupně předchází podstatné jméno, přídavné jméno, infinitiv a příslovce (a je možno najít i příklady, kdy podřadicí spojce předchází více takových prvků - stejného či různého druhu - (9); všechny doklady jsou z korpusu SYN2000 Českého národního korpusu).

(8) *a. ... že byly studené jako led <u>a tělo že</u> měl studené jako železo ...*
b. ... že je z něho cítit chlast ... <u>a červený že</u> je jak tulipán...
c. ... musel jsem je přečíst, vrátil jsem je i s recenzí, <u>vydat že</u> to nejde ...
d. Je velký a klidný, a <u>hned že</u> pomůže tátovi ...

(9) *a. ... mám jít balit čistý papír do sklepení ..., <u>nic jiného že</u> nebudu balit než čistý papír ...*
b. <u>Kvůli tomu vašemu lovení abych</u> přihřívala kuřecí prsíčka, ...

3.3 Porušení zákonitostí souvýskytu

Asi vůbec nejtriviálnějším porušením zákonitostí jazykového jevu typu souvýskyt je porušení shody podmětu s přísudkem v osobě, čísle a rodu. Pokud uvážíme, že podmět je typicky tvořen podstatným jménem v prvním pádě a naopak, podstatné jméno v prvním pádě musí být (téměř) vždy podmětem, můžeme jako jednu z možností porušení shody podmětu s přísudkem v řetězci délky 2 vzít minimální negramatický řetězec (10).

(10) { + [cat: n, case: nom, num: pl] + [cat: v, v_form: fin, num: sg] + }

(Je samozřejmě možné najít celou řadu dalších minimáních řetězců porušujích shodu podmětu s přísudkem a bylo by dokonce možné nad touto množinou provést generalizaci, jejímž výsledkem by byl jediný abstraktní řetězec. V momentě, kdy by se začalo jednat o rozšíření tohoto řetězce, by však nastal závažný praktický problém: takové rozšíření by pro generalizovanou podobu bylo velmi obtížné, podmínky, které by s ním byly spojeny, by musely být velmi komplexní, nepřehledné a snadno by se v nich vyskytly chyby).

Základní myšlenka rozvolnění řetězce (10) spočívá (celkem samozřejmě) v tom, že nikam do řetězce nesmí být přidáno žádné podstatné jméno, které by mohlo stát v pozici podmětu slovesa v jednotném čísle, a ani se v rozvolněném řetězci nesmí nikde vlevo od podstatného jména v prvním pádě plurálu objevit jakýkoliv element, který by "zpochybnil" roli tohoto podstatného jména jako podmětu (takovým elementem je např. spojka *jako* - srv. negramatičnost věty *Hrdinové kráčel hrdě (vstříc svému osudu)* na straně jedné a nepochybnou správnost věty *Jako hrdinové kráčel hrdě (vstříc svému osudu)* na straně druhé). Poněkud zjednodušeně můžeme tedy jako rozvolnění negramatického řetězce (10) zformulovat jako (11).

(11) { + (not([cat: n, case: nom, num: sg] ∨ jako))*
+ [cat: n, case: nom, num: pl]
+ (not([cat: n, case: nom, num: sg]))*
+ [cat: v, v_form: fin, num: sg]
+ (not([cat: n, case: nom, num: sg]))* + }

Podobně by bylo možno pokračovat v definici minimálních negramatických řetězců a jejich rozvolňování až do dosažení (předpokládané, viz výše) hranice. V tomto příspěvku by to však bylo zbytečné, základní myšlenky postupu již byly dostatečně ilustrovány.

References

1. Isa Khader: *Evaluation of an English LFG-based grammar as error checker.* Master s thesis, University of Manchester Institute of Science and Technology (2003).
2. Wolfgang Menzel, Inge Schroeder: *Error diagnosis for language learning systems.* In: Special edition of the Re-CALL journal (1999) 20 - 30.
3. M.Plátek, M. Lopatková, K. Oliva: *Restarting Automata: Motivations and Applications.* Proceedings of 13.Theorietag, Automaten und Formale Sprachen, Herrsching, (2003) 90 - 96.
4. M. Plátek, F. Otto, F. Mráz: *Restarting Automata and Variants of j-Monotonicity.* Proceedings of DCFS' 2003, MTA SZTAKI, Budapest, (2003) 303 - 312.
5. David Schneider, Kathleen McCoy: 1998. Recognizing syntactic errors in the writing of second language learners. In: Proc. of Coling-ACL 98, Montreal (1998) 1198 - 1204

Two-dimensional on-line tessellation automata: Recognition of NP-complete problems

Daniel Průša*

Faculty of Mathematics and Physics, Charles University in Prague
Malostranské nám. 25, 118 00 Prague, Czech Republic
prusa@barbora.mff.cuni.cz

Abstract. *Two-dimensional on-line tessellation automata* ($2OTA$) are kind of non-deterministic cellular automata. The class of picture languages recognizable by them is considered by some authors to be a good candidate for the ground level of the picture languages theory.
We show that, in some sense, these automata are able to simulate one-dimensional bounded cellular automata (BCA). As a consequence, it gives us a generic way, how to design $2OTA$ recognizing modifications of NP-complete problems recognizable by BCA's.

1 Introduction

The two-dimensional on-line tessellation automaton was introduced by K. Inoue and A. Nakamura in 1977. Two-dimensional arrays of symbols (so called *pictures*) are taken to be inputs to this model. A computation is performed in a restricted way – cells do not make transitions at every time-step, but rather a "transition wave" passes once diagonally across them. Each cell changes its state depending on two neighbors – the top one and the left one.

The class of picture languages recognizable by $2OTA$ satisfies many important properties that are analogous to properties of the regular languages (e.g. closure properties). Moreover, the class can be defined in many various ways – the following classes of picture languages coincide: $L(2OTA)$, languages expressed by formulas of existential monadic second order logic, languages generated by finite tiling systems, languages corresponding to regular expressions of a special type.

Based on the given facts, $L(2OTA)$ is considered to be a good candidate for the "ground" level of the picture languages theory, even better than the class of languages recognizable by two-dimensional finite state automata (which are bounded, non-rewriting Turing machines), see [1]. However, there are some properties of the class that speak against this suggestion. The possibility to recognize some NP-complete picture languages is one of them (this is a known result).

* Supported by the Grant Agency of the Czech Republic, Grant-No. 201/02/1456 and by the Grant Agency of Charles University, Prague, Grant-No. 300/2002/A-INF/MFF.

In this paper we list basic properties of $L(2OTA)$ and compare them to properties of the class of picture languages recognizable by two-dimensional finite-state automata. After that we show that, in some sense, $2OTA$ can simulate bounded one-dimensional cellular automata. As a consequence, this simulation gives us a generic way how $2OTA$ can recognize modifications of one-dimensional NP-complete problems which are recognizable by BCA's (these modifications extend strings to pictures to provide sufficient space for the simulation).

2 Picture languages, two-dimensional automata

We start with a brief overview of the picture languages theory. If needed, more details can be found, e.g., in [2].

Definition 1. *A* picture *over a finite alphabet Σ is a two-dimensional rectangular array of elements of Σ. Moreover, Λ is a special picture called the* empty picture.

Σ^{**} denotes the set of all pictures over Σ. A *picture language* over Σ is a subset of Σ^{**}. Let $P \in \Sigma^{**}$ be a picture. Then, $\text{rows}(P)$, resp. $\text{cols}(P)$ denotes the number of rows, resp. columns of P. The pair $\text{rows}(P) \times \text{cols}(P)$ is called the *size* of P. The empty picture Λ is the only picture of the size 0×0. There are no pictures of sizes $0 \times k$ or $k \times 0$ for any $k > 0$. For integers i, j such that $1 \leq i \leq \text{rows}(P)$, $1 \leq j \leq \text{cols}(P)$, $P(i, j)$ denotes the symbol in P at the coordinate (i, j). Row, resp. column concatenation is defined for two pictures having the same number of columns, resp. rows. The result is the union, where the second picture is placed bellow, resp. after the first picture.

The most general two-dimensional recognizing device is a Turing machine (TM) working over a two-dimensional tape, which is allowed to move the head in four directions – up, down, left and right. The tape is infinite in both directions (vertical, horizontal). T can rewrite the scanned symbol. In the initial configuration, an input is embedded in the tape. Fields that do not store the input consist of the background symbol # and finally, the head of T scans the top-left corner of the input.

Let NP_{2d} be the class consisting of all picture languages that can be recognized by a two-dimensional Turing machine in a polynomial time (an analogy to NP). Then, NP_{2d}-complete languages can be defined. It holds that each NP-complete (one-dimensional) language is also NP_{2d}-complete (a two-dimensional TM can be simulated on a one-dimensional TM if an input picture is encoded into a string, e.g. row by row, having rows separated by a special symbol).

T is *bounded* if whenever it moves out of the area storing an input, then it immediately returns back in the following step and it does not perform rewriting in this step. T is *finite* if it does not perform any rewriting at all. And finally, T is a *two-dimensional finite state automaton* ($2FSA$) if it is bounded and finite.

We list some of the most important properties of $L(2FSA)$ and $2FSA$'s.

1. $L(2FSA)$ is not closed under concatenation (neither row or column)

2. The class of languages recognizable by deterministic $2FSA$'s is strictly included in $L(2FSA)$.
3. The emptiness problem is not decidable for $2FSA$.

Comparing to the class of regular (one-dimensional) languages, all these properties are different.

3 Two-dimensional on-line tessellation automaton

Definition 2. *A* (non-deterministic) two-dimensional on-line tessellation automaton A *is a tuple* $(\Sigma, Q, q_0, Q_F, \delta)$*, where*

- Σ *is an input alphabet*
- Q *is a finite set of states and it holds* $\Sigma \subseteq Q$
- $q_0 \in Q$ *is the initial state*
- $Q_F \subseteq Q$ *is a set of accepting states*
- $\delta : Q \times Q \times Q \to 2^Q$ *is a transition function*

Let $P \in \Sigma^{**}$ be an input to A. The automaton consists of an array of cells. In case of P, the size of the array is $\text{rows}(P) \times \text{cols}(P)$. For $i \in \{1, \ldots, \text{rows}(P)\}$ and $j \in \{1, \ldots, \text{cols}(P)\}$, let $c(i, j)$ denote the cell at the coordinate (i, j). Moreover, let $c(0, j)$ and $c(i, 0)$ denote fictive cells that stay in the state # during the whole computation.

In the initial configuration, each cell $c(i, j)$ is in the state $P(i, j)$. The computation has the character of a wave passing diagonally across the array. A cell changes its state exactly once. The cells $c(i, j)$, where $i + j - 1 = t$, compute in the t-th step – $c(1, 1)$ in the first step, $c(1, 2)$ and $c(2, 1)$ in the second step, etc. When $c(i, j)$ performs a transition, its change depends on the current state of the top and left neighbor. Thanks to fictive cells mentioned above, each considered cell has these two neighbors. Let $c(i, j)$ be in state q and the left, resp. right neighbor of $c(i, j)$ be in state q_l, resp. q_t. Then, the transition of $c(i, j)$ is given by states in $\delta(q, q_l, q_t)$. Figure 1 shows a scheme related to this change.

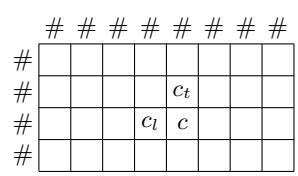


Fig. 1. Example of a $2OTA$ working on an input of the size 4×8. The transition of the cell c depends on its own state and states of c_l and c_t. #'s denote fictive cells neighboring to cells in the first row and column.

The computation consists of $\text{rows}(P) + \text{cols}(P) - 1$ steps. When it is finished, the set of states reachable by $c(\text{rows}(P), \text{cols}(P))$ determines whether A

accepts P (a state in Q_F can be reached) or rejects it (otherwise). $2OTA$ A is deterministic ($2DOTA$) if $|\delta(q_1, q_2, q_3)| \leq 1$ for each triple q_1, q_2, q_3 in Q. The classes of picture languages recognizable by a $2OTA$ and a $2DOTA$ are denoted by $L(2OTA)$ and $L(2DOTA)$, respectively. We list some important properties of these classes.

1. $L(2OTA)$ is closed under row and column concatenation, union and intersection
2. $L(2DOTA)$ is closed under complement, $L(2OTA)$ is not closed under complement
3. $L(2DOTA)$ is strictly included in $L(2OTA)$
4. $L(2FSA)$ is strictly included in $L(2OTA)$

4 Simulation of one-dimensional cellular automaton

In this section, we will work with one-dimensional bounded cellular automata (BCA). Such an automaton is a tuple (Q, Q_I, Q_A, δ), where Q is a set of states, $Q_I \subseteq Q$ a set of initial states, $Q_A \subseteq Q$ a set of accepting states and $\delta : Q \times Q \times Q \to 2^Q$ a transition function. Each transition of a cell depends on its own state and states of the left and right neighbor. We consider the model, where all cells start at the same moment and an input is accepted if the leftmost cell can reach some state in Q_A.

In [2], there is described a procedure of how to synchronize cells of a BCA by the assumption the leftmost (resp. rightmost) cell is in some distinguished state q_a and the other cells in a passive state q_p (meaning these cells stay in this state until they are activated by a signal send by the leftmost cell). In the end of the process all cells enter a state q_s first time (which can be interpreted they are synchronized). The procedure requires $3 \cdot n$ computational steps maximally (n denotes the number of cells).

Let Σ be an alphabet and \$ a special symbol not contained in Σ. We define function $\tau : \Sigma^* \times \mathbf{N}^+ \times \mathbf{N}^+ \to (\Sigma \cup \{\$\})^{**}$, where $P = \tau(w, m, n)$ is of size $m \times n$ and, for $w = a_1 \ldots a_k$, it satisfies

$$P(i,j) = \begin{cases} a_j & \text{if } i = 1 \text{ and } j \leq |w| \\ \$ & \text{otherwise} \end{cases}$$

Theorem 1. *Let $C = (Q, \Sigma, Q_A, \delta)$ be a deterministic one-dimensional bounded cellular automaton recognizing a language L_1 in time t. There is a deterministic $2OTA$ automaton A recognizing L_2 such that a picture P is in L_2 if and only if the following conditions are fulfilled:*

1) *P is equal to $\tau(w, m, n)$ for some suitable positive integers m, n and a string $w \in \Sigma^*$.*
2) *Let $k = |w|$, where w is the string from point 1). Then*

$$\text{rows}(P) \geq \frac{k+1}{2} + 3 \cdot k + t(k) \qquad \text{cols}(P) \geq k + 1 + 3 \cdot k + t(k)$$

Proof. We will construct $2OTA$ automaton A simulating C. Let P be an input to A and $P = \tau(w, m, n)$ for some positive integers m, n and $w \in \Sigma^*$ (not necessary in L_1), where $|w| = k \leq n$, $m \geq \frac{k+1}{2} + 3 \cdot k + t(k)$, $n \geq k + 1 + 3 \cdot k + t(k)$. The other inputs will be discussed separately in the end of the proof.

Figure 4 outlines the main idea of the simulation. The computation of A is divided into four phases. During the first phase, w is moved to the diagonal - it is represented in $\lceil \frac{k+1}{2} \rceil$ cells, where each of the cells stores two input characters. An exception is the cell corresponding to the end of w which can possibly store one character only. When the first phase is completed it is possible to simulate one step of C during two steps of A. However, before the whole process can be launched it is necessary to synchronize cells of A so that they are able to start at the same moment. This is done during the second phase. The third phase consists of a simulation of C and finally, the fourth phase just delivers information about the result of the simulation to the bottom-right cell of A so that A can correctly accept or reject the input.

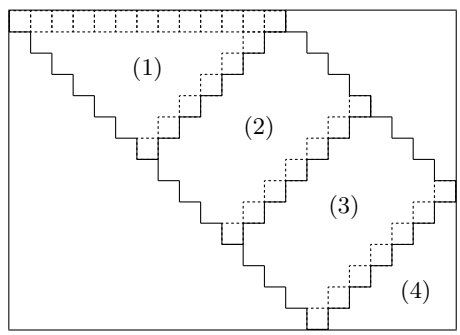


Fig. 2. Simulation of one-dimensional cellular automaton on $2OTA$. Groups of cells involved in one of the four computational phases are distinguished.

A description of the phases in more details follows.

Let $w = a_1 a_2 \ldots a_k$. For $i = 1, \ldots, m$ and $j = 1, \ldots, n$, let c_{ij} denote the cell of A at the coordinate (i, j). The cells $c_{1,1}, c_{2,1}, \ldots, c_{k,1}$ store w at the beginning. The goal of the first phase is to represent w in cells $d_i = c_{k+1-s+i, s-i+1}$ for $i = 1, \ldots, s$ and $s = \lceil \frac{k+1}{2} \rceil$. The representation should be as follows. For each $i < s$, d_i stores $a_{2 \cdot i-1}$ and $a_{2 \cdot i}$, d_s stores a_k if k is odd, else it is marked as the right end of w only. In addition, d_1 is marked as the left end of w.

To reach the desired configuration, A computes according to the following rules:

- During each odd computational step of A – each cell checks if the top neighbor is in a state that represents a pair of symbols. If so, the symbols are copied to the cell (otherwise the state is not changed). Cells in the first row

do not perform any changes except the cell $c_{1,1}$, which records the presence of the left end of w. This mark is copied together with symbols in next steps.
- During each even step – each cell checks the state of its left neighbor. If the neighbor represents a pair of symbols, the cell copies them. If the cell is placed in the first row and stores a_i, then it reads a_{i-1} stored in the state of the left neighbor and represents pair (a_{i-1}, a_i).
- The only exception from the previous two rules is the cell $c_{1,k+1}$ which stores \$ following the last symbol of w. If $k+1$ is odd $c_{1,k+1}$ copies a_k and marks the right end of w, else it marks the end only. After A finishes the $k+1$-st computational step the desired diagonal representation of w is formed.

The second phase simulates the synchronizing procedure we have already mentioned. The rightmost cell of C, which is represented in $c_{1,k+1}$ or $c_{2,k}$, is considered to be the only cell in an active state. Thanks to the synchronization, a simulation of all cells of C (working over w) can be started at the same moment (i.e. in one computational step of A). Both phases ((2), (3)) are similar, in the next paragraphs we need not to distinguish between them. Comparing to the first phase, cells of A are required to behave differently during the second and third phase, thus there must be some notification about the change. The notification is done by spreading signal 'phase one ended'. The signal is generated by the cell c_{k+1}. In following steps, each cell that detects a neighbor marked by the signal, marks itself as well. When a cell is marked, it computes as it is described in the next paragraphs. Since the signal is being spread faster than the signals related to the synchronization process (during two steps signal 'phase one ended' notifies two additional cells of A simulating together 4 cells of C, while A performs one step only), this consecutive notification is sufficient.

Let us assume some configuration of C is represented in a diagonal (as shows Figure 4). One computational step of C is simulated during two steps of A. Let the diagonal $\mathcal{D}_1$ be formed of s cells c_{x_1-i,y_1+i}, where $i = 0, \ldots, s-1$ and (x_1, y_1) is the coordinate of the bottom-left cell of $\mathcal{D}_1$, A be performing two computational steps that should produce the next configuration of C. In the first step, each cell which has the left and top neighbors in $\mathcal{D}_1$ records states represented by the neighbors. The diagonal consisting of these cells is $\mathcal{D}_2 = \{c_{x_1+1+i,y_1-i} \mid i \in \{0, \ldots, s-2\}\}$. In the second step, each cell that has at least one of the neighbors in $\mathcal{D}_2$, i.e. each cell in $\mathcal{D}_3 = \{c_{x_1+1+i,y_1-1-i} \mid i \in \{0, \ldots, s-1\}\}$, simulates one computational step of represented cells of C. For example, let us consider some $c_{ij} \in \mathcal{D}_1$ storing states of two cells of C (denoted c'_1 and c'_2) at the moment when the l-th computational step of C is done. Then, the cell $c_{i+1,j-1} \in \mathcal{D}_3$ can retrieve states of c'_1, c'_2 (recorded in the top neighbor of $c_{i+1,j-1}$) as well as states of neighbors of c'_1, c'_2 in C (recorded in the left and the top neighbor), thus $c_{i+1,j-1}$ can simulate the $l+1$-st computational step of c'_1 and c'_2 and record the resulting states.

The fourth phase is started at the moment the leftmost cell of C reaches an accepting state. If w is in L_1, it occurs after the simulation of at most $t(k)$ steps. In this case, signal 'input accepted' is started to be spread. First of all, the cell of A that detects C has accepted w is marked to carry the signal. Then,

each cell of A having a neighbor carrying the signal is marked as well. Using this principle, the signal reaches the bottom-right cell of A, which indicates the input should be accepted. If C does not accept, no signal is generated and A rejects. Note that, during the fourth phase, some cells of A still work as it was given in the description of the second and third phase, however, this process does not interfere with spreading of the signal, thus it has no affect on the result.

It remains to discuss, how A detects an input that cannot be written as $\tau(w, m, n)$ for some suitable values w, n, m, where $m \geq \frac{k+1}{2} + 3 \cdot k + t(k)$ and $n \geq k + 1 + 3 \cdot k + t(k)$ ($k = |w|$ as used before). First of all, A can easily detect, if the only symbols different to \$ are placed in the first row as a contiguous sequence starting at the first field – if a cell of A contains this symbol, it checks if the top neighbor is in state # and if the left neighbor was originally (before it performed its computational step) in a state different to \$. A verification of m and n is done automatically. If one of these values is less than it should be then A has not a space large enough to perform all four phases, thus 'input accepted' signal cannot be generated implying the input is rejected. Note that phase one requires $\lceil \frac{k+1}{2} \rceil$ rows of cells and $k+1$ columns to be performed, phase two additional $3 \cdot k$ rows and the same number of columns and finally, phase three additional $t(k)$ rows and also $t(k)$ columns. □

The simulation we have presented can be improved in the case of non-deterministic $2OTA$ automata.

Theorem 2. *Let $C = (Q, \Sigma, Q_A, \delta)$ be a non-deterministic one-dimensional bounded cellular automaton recognizing a language L_1 in time t. There is a non-deterministic $2OTA$ automaton A recognizing L_2 consisting of pictures that can be written as $\tau(w, m, n)$, where $w \in L_1$, $m \geq \frac{|w|+1}{2} + t(|w|)$ and $n \geq |w| + 1 + t(|w|)$.*

Proof. We will modify the computation presented in the proof of Theorem 1. It is possible to remove the second phase (the synchronization). An equivalent configuration, where all cells are synchronized, can be guessed (using non-determinism) immediately after the representation of w is created in a diagonal. The other phases remain the same, so we describe a modification of the synchronizing process only. Let cells of A be denoted by $c_{i,j}$ again and let $r = \lceil \frac{k}{2} \rceil$.

Let each cell taking part in the first phase non-deterministically guess if it belongs to diagonal $\mathcal{D} = \{c_{k-r+i, r+1-i} \mid i \in \{1, \ldots, r\}\}$ or not. Note that cells in this diagonal compute during the k-th step of A, i.e. one step before the representation of w is created. If a cell decides it belongs to $\mathcal{D}$, it records this fact in its state (let us call such a cell to be 'synchronized'). Each cell related to the first phase checks the presence of the 'synchronized' marker in its left and top neighbor. According to the position of a cell c and according to its state we distinguish the following cases:

1) c is placed in the first row, it is in the state \$ and this is the first \$ following the last symbol of w (i.e. $c = c_{k+1}$) – c checks if the left neighbor is 'synchronized'. If not, the cell generates 'bad synchronization' signal, that is spread

to the bottom-right cell and causes that the correspondent computational branch of A does not accept.

2) c is placed in the first row and it is in some state in Σ – if the left neighbor is 'synchronized' c generates 'bad synchronization' signal.
3) c is not a cell of the first row and both its neighbors represent some symbols of w. Now, if exactly one of the neighbors is 'synchronized', then 'bad synchronization' is generated.

It should be evident that no 'bad synchronization' signal is generated if and only if all cells in $\mathcal{D}$ are the only cells that guess they are 'synchronized' – conditions 1) and 2) force c_k to be the first 'synchronized' cell in the first row, condition 3) inducts that a cell in $\mathcal{D}$ precessing 'synchronized' cell is 'synchronized' as well.

Let us consider a computational branch of A, where the synchronization is correctly guessed. Cells computing in the $k+1$-st step verify this fact and become 'active'. From now, no more guesses related to the synchronization are needed ('active' status is spread during the remaining parts of the computation), the simulation of C can begin according to the same scenario as it was described for the deterministic variant. □

Example 1. Let us consider a well known NP-complete problem – the problem of a knapsack. Informally, an instance of the problem is a finite sequence of positive integers $n_0, n_1, \ldots, n_k$ $(k \geq 1)$ represented in binary. These numbers are commonly interpreted as a knapsack of the size n_0 and k items of the sizes $n_1, \ldots, n_k$. The question is if it is possible to select some of the available items so that the selected items exactly fit into the knapsack. In another words, if there is a subset of indices $I \subseteq \{1, \ldots, k\}$ such that $n_0 = \sum_{i \in I} n_i$.

We show that the knapsack problem can be decided by a non-deterministic cellular automaton in time $t(n) = n$. For our purposes, we will use special format of inputs over the alphabet $\Sigma = \{0, 1, a, b\}$. We code one sequence $n_0, n_1, \ldots, n_k$ by the string $w = w_0 a w_1^R b w_2^R b \ldots b w_k^R$, where w_i is n_i written in binary. It means $w_i \in \{0,1\}^+$, $w_i = a_{i,1} a_{i,2} \ldots a_{i,l_i}$, where $a_{i,1} = 1$ and $\sum_{j=1}^{l_i} a_{i,j} = n_i$. Note that w contains exactly one symbol a. Each w_i^R $(i > 0)$, except the last one, is followed by b (a and b serve as delimiters).

We define L to be the language over Σ containing exactly all strings encoding an instance of the knapsack problem that has a solution.

Lemma 1. *L can be recognized by a non-deterministic bounded CA in time $t(n) = n$.*

Proof. Let us consider a well formed input of the form $w_0 a w_1^R b w_2^R b \ldots b w_k^R$. Let the cell storing a in the initial configuration be denoted by c_a.

The idea of the presented algorithm is to choose a subset of items non-deterministically and subtract their sizes from the size of the knapsack. c_a plays an important role in this process. The cells storing w_0 in the beginning configuration are used during the computation as a counter which is being decremented by the sizes of selected items. The cells positioned after c_a shift string $w_1^R b \ldots w_k^R$ left only, in each step by one character. These cells cannot be distinguished from

the cells precessing c_a (except the rightmost cell), thus they behave in the same way. Each cell keeps information, which symbol of the input it represents in the beginning configuration, however this information is relevant in case of the cells forming the counter only. The last character of the shifted string is immediately followed by a special signal E_1 indicating the end. The signal is generated in the first computational step by the rightmost cell.

As we have already mentioned c_a is consecutively feeded by one character of $w_1^R b \dots w_k^R$ in each step. If the currently received character is the first character of some w_i^R, i.e. the least significant bit of w_i, c_a non-deterministically decides if w_i will be added to the knapsack or not. In the second case, w_i is absorbed by c_a, it means, no signals are sent to the cells on the left. In the first case, for each received bit 1, resp. 0 of w_i^R, c_a sends left signal S_1, resp. S_0 representing subtraction of 1, resp 0. Moreover, if the received bit is the least significant (i.e. the first bit of w_i^R), c_a sends together with S_i signal N announcing the beginning of a new substraction. The idea of counting is based on delivering a signal generated by the i-th bit of w_j^R to the correspondent cell representing the i-th bit of the counter. Note that to achieve this we have decided to code all w_i, $i > 0$ in the reversed form. The last thing to be mentioned is that c_a changes E_1 to E_2 when the signal is received.

Let us take a closer look at the behavior of cells forming the counter at the moment they receive one of the previously defined subtraction signals. We consider a cell c keeping a bit b. Moreover, we assume each cell has a boolean flag (denoted f in case of c). A value of the flag is stored in states of cells. It is used to control delivering of S_i signals. If f is true, it indicates that c awaits signal S_i – when the signal is received f is changed to false meaning that next signals S_i should be passed to next cells. One more signal is still needed – signal D will be generated when a carriage from a bit to higher bits occurs during a subtraction. Signal N is sent together with signal S_0 or S_1. We consider N to be processed by a cell, which receives it, first (before S_i). A detailed description of how c processes received signals follows.

N) c sets f to be true (meaning c is the target cell for the firstly received signal S_i), N is sent to the left neighbor.

S_0) If f is false c sends S_0 to the left, else c changes f to false – since 0 is subtracted from the bit b kept by the cell, no additional changes are needed to be performed.

S_1) Again, if f is false c sends S_1 to the left, else c changes f to false. If $b = 1$, b is changed to 0, otherwise b is changed to 1 and a signal D is generated and sent to the left neighbor (a decrement is needed in higher bits to complete the subtraction).

D) If $b = 1$, b is changed to 0, else b is changed from 0 to 1 and D is sent to the left neighbor.

If the leftmost cell is about to send S_i or D to its left neighbor, it indicates that a negative value has been reached by the counter, thus the input is rejected. While moving to the leftmost cell, signal E_2 checks if all bits of the counter are 0.

If so, it means the knapsack problem has been solved successfully and the input is accepted. To verify correctness of the presented algorithm it is sufficient to realize that signals handling subtractions of the counter never mutually interfere.

It remains to discuss how the automaton detects badly formatted inputs. If there is no symbol a contained in the input, the leftmost cell receives E_1 instead of E_2. If there are two or more symbols a, one of the correspondent cells receives E_2 instead of E_1. If one of the cells forming the counter stores b at the beginning, it is detected when the cell receives some of the substraction signals or E_2. And finally, it can be checked by c_a if each binary value representing the size of an item starts with bit 1 (c_a needs always to remember the bit received in the previous step) and the leftmost cell of the automaton checks the highest bit of the knapsack size.

Time complexity of the computation is exactly $|w|$ for any $w \in \Sigma^+$ – signal E_1 is generated in the first step, after next $|w| - 2$ steps reaches the second cell (possibly changed to E_2), the first cell detects it and finishes the computation in one more step. □

Let L be the language in Example 1. As a consequence of Theorem 2 and Lemma 1, we get it is possible to construct $2OTA$ recognizing

$$L_K = \{\tau(w, m, n) \mid w \in L \wedge m \geq \frac{|w| + 1}{2} + 4 \cdot |w| \wedge n \geq 5 \cdot |w| + 1\}$$

which is an NP_{2d}-complete language.

References

1. D.Giammarresi, A.Restivo: Two-dimensional Languages, In A.Salomaa and G.Rozenberg, editors, Handbook of Formal Languages, volume 3, Beyond Words, pp. 215–267. Springer-Verlag, Berlin, 1997.

2. A.Rosenfeld: Picture Languages - Formal Models of Picture Recognition, Academic Press, New York, 1979.

Synchronisation of Large Heterogenous Data Using DataPile Structure

David Bednárek, David Obdržálek, Jakub Yaghob, Filip Zavoral †

Department of Software Engineering,
Faculty of Mathematics and Physics, Charles University Prague
†also Department of Computer Science, Institute of Finance and Administration
{david.bednarek, david.obdrzalek, jakub.yaghob, filip.zavoral}@mff.cuni.cz

Abstract. V článku se budeme zabývat možným způsobem návrhu informačního systému zastřešujícího stávající provozní systémy. Zaměříme se na požadavky kladené na takového systémy a popíšeme návrh architektury centrálního skladu dat s použitím organizace dat do datového stohu. Dále popíšeme synchronizační a replikační mechanismy, které umožní efektivní replikaci dat i mezi provozními systémy, jejichž fyzická a logická organizace dat je velmi rozdílná.
Základem navrženého řešení je organizace centrálních dat formou datového stohu. Tato metoda spočívá ve vertikalizaci dat, tj. nevyužívá tradiční 'řádkové' pojetí databázové tabulky, kdy jedna řádka představuje nějakou množinu spolu souvisejících atributů nějaké entity. Místo toho je každý atribut 'tradiční' řádky představován jedním řádkem datového stohu a odpovídající si atributy jsou pak spojeny identifikací entity, které tyto atributy náleží.
Replikační mechanismy slouží k předávání dat mezi provozními systémy a centrální databází. Jedná se o distribuovanou aplikaci, jejímž úkolem je předávat obousměrně data od centrální databáze k replikačnímu serveru, který dále data v jednotném formátu distribuuje typicky k lokálním replikačním aplikacím nebo je od nich sbírá.

1 Motivace

1.1 Problém

Velké společnosti, úřady, univerzity apod. často provozují množství oddělených postupně nasazovaných informačních systémů. Při překročení určité míry heterogenity a velikosti dat vzniká potřeba tyto systémy integrovat a jejich data vzájemně synchronizovat. Základní možnosti řešení tohoto problému jsou dvě. První možností je nahradit množství dosavadních systémů jedním systémem všeobjímajícím. Druhou možností je nad stávajícími provozními systémy vytvořit zastřešující informační systém, který by shromažoval data provozních systémů, udržoval mezi nimi potřebné vazby a prováděl potřebné synchronizace a aktualizace.

Tam, kde jsou již nějaké takovéto provozní systémy nasazeny, nastávají při plánování propojení těchto systémů problémy:

– různé systémy uchovávají data, jejichž průnik je neprázdný, takže při synchronizaci může dojít k nekonečnému opravování dat mezi hodnotami pocházejícími z různých zdrojů,
– data jsou uložena v různých formátech v různých úložištích, takže jejich přímá synchronizace je obtížná

V případě, kdy mají být ve výsledném řešení zapojeny již hotové aplikace, nastává tedy problém s předáváním dat mezi těmito „starými“ aplikacemi navzájem i mezi starými aplikacemi a novými částmi informačního systému. Z tohoto hlediska můžeme stávající aplikace rozdělit na tyto druhy:

a) Aplikace, které jejich autor upraví tak, aby se zúčastnily synchronizace stejně, jako nově vytvářené aplikace,
b) Aplikace, jejichž změna není možná, ale která data ukládají v datovém úložišti se známým formátem dat nebo přístupovým rozhraním, nebo jsou schopny data v nějakém známém formátu alespo explicitně exportovat a importovat,
c) Aplikace, jejichž změna není možná a které ukládají data v proprietárním úložišti bez známého formátu.

Aplikace ze skupiny c) není možno do celkového informačního systému ze zjevných důvodů začlenit; způsob začlenění aplikací ze skupin a) a b) je možný a bude v tomto článku popsán.

1.2 Tradiční řešení

Tradiční řešení zastřešujícího informačního systému spočívá v návrhu jednotné relační struktury, ve které vznikne sjednocení všech existujících vazeb a struktur. Takové řešení je ale typicky spojeno s nevýhodami a problémy:

– sjednocení je velmi složité a struktura těžkopádná
– výsledný návrh má malou flexibilitu a není snadno udržovatelný (často téměř vůbec)
– přidání dalších vztahů může znamenat nutnost zásadní změny celé struktury
– standardní návrh obvykle nezachycuje původ dat (ze které aplikace) a jejich historii (jakým způsobem a kdy byla data změněna)
– není snadné definovat, jaká data jsou při změně jednotlivým provozním systémům předávána, a to jak co se týče výběru atributů, tak co se týče výběru celých záznamů

2 Stoh

2.1 Vertikalizace dat

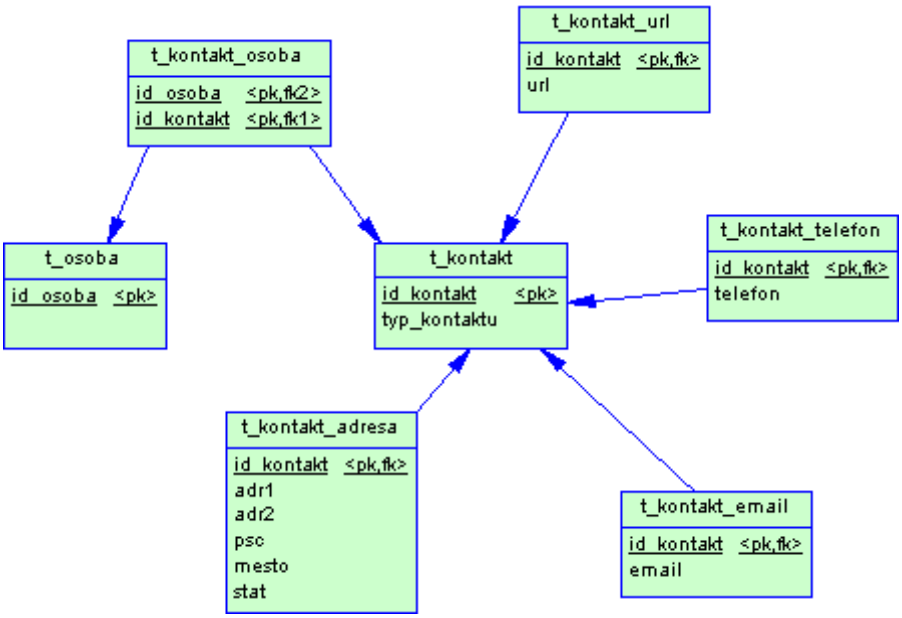
Tato metoda spočívá ve vertikalizaci dat, tj. nevyužívá tradiční horizontální pojetí databázové tabulky, kdy jedna řádka představuje nějakou množinu spolu

souvisejících atributů nějaké entity. Místo toho je každý atribut 'tradiční' řádky představován jedním řádkem datového stohu a odpovídající si atributy jsou pak spojeny identifikací entity, které tyto atributy náleží. Tento způsob organizace dat se osvědčil zejména u systémů s požadavky na časovou evidenci aktualizací záznamů [1,2], vysokou heterogenitu shromažovaných dat a co nejjednodušší rozšiřitelnost systému:

- každý atribut tradiční řádky je představován jedním řádkem datového stohu
- atributy, z nichž je tvořen tradiční řádek, jsou ve stohu spojeny pomocí identifikace entity, k níž náleží

Celé datové schéma všech zúčastněných aplikací je nahrazeno dvěma tabulkami

- hodnoty atributů
- struktura entit



Obr. 1. Struktura úložiště s použitím stohu

S využitím dalších metatabulek (viz obr. 1) pak budou poměrně snadno dosažitelné všechny výše uvedené cíle, tj. maximální konfigurovatelnost (jedná se pouze o naplnění dat metatabulek), možnost uchovávat historii modifikací,

možnost u vybraných atributů libovolného objektu sledovat období platnosti v reálném světě, možnost definovat libovolné přirozené identifikační atributy, možnost určovat míru relevantnosti daného provozního systému jako zdroje konkrétního typu dat.

Dále tato datová struktura triviálně umožní vytvoření jednoznačné identifikace libovolného objektu spravovaného IS, tato identifikace bude dále označována jako ENTID. Jako entitu pak definujeme jakýkoliv objekt, na který se může odkazovat jiný objekt, nebo pro jehož popis je nutno využít více atributů, které však nejsou vícečetné.

Hlavní přínosy takto navržené struktury jsou:

- udržovatelnost struktury dat
- jednoznačná indentifikace objektu v celém systému
- určování míry relevantnosti zdroje dat
- uchovávání historie modifikací
- sledování období platnosti
- možnost synchronizace jen podmnožiny záznamů s vybranou provozní aplikací

2.2 Cache

Zřejmou nevýhodou datového stohu je nevhodnost pro pokládání dotazů; jednalo by se o velmi komplikované a zárove relativně pomalé dotazy. Proto je navržen mechanismus dvojúrovových cache, který však bude sloužit jen pro pokládání dotazů.

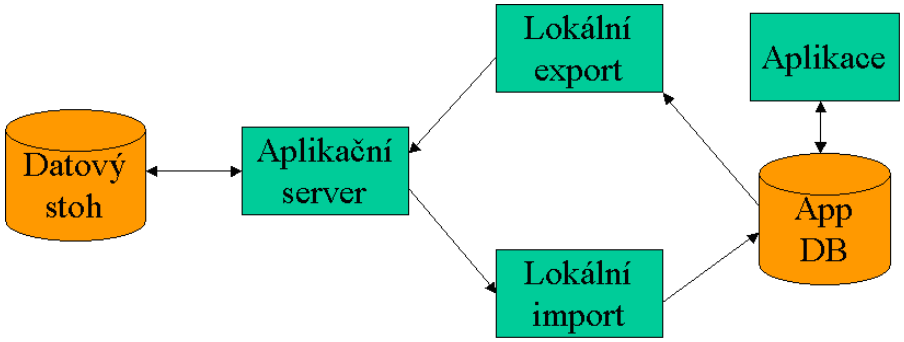
První vrstva cache nazvaná *primární cache* bude převedením vertikálního formátu dat do horizontálního pro stejnou třídu entit. Toho lze dosáhnout poměrně jednoduše opět pomocí metatabulek popisujících strukturu tabulek primární cache a jim odpovídajících atributů z datového stohu.

Druhá vrstva cache nazývaná *sekundární cache* pak bude vytvářet dojem tabulek vhodných pro běžné aplikace. Zde se nabízí několik možných řešení implementace, která zahrnují např. využití materializovaných pohledů dostupných v Oracle 9i, nebo implementaci pomocí dalších metatabulek, nebo kombinace obojího, to vše v závislosti na četnosti a rychlosti změn.

3 Technické provedení

3.1 Schéma datové synchronizace

Provozní aplikace pracuje se svou lokalní databází. Úkolem lokálního exportu je přesunout změněná data v lokální databázi k aplikačního serveru. Opačným směrem synchronizace je distribuce změněných dat aplikačním serverem do databází aplikací. Tento směr je nazýván z pohledu provozní aplikace lokální import.



Obr. 2. Schéma synchronizace

3.2 Aplikační server

Z pohledu centra vykonává aplikační server tyto funkce (viz obr. 2):

- příjem dat z lokálního exportu;
- kontrolu správnosti dat (kompletnost, neporušenost) – bude uskutečněna pomocí standardních metod kryptografie;
- kontrolu oprávněnosti přidání nebo změny dat (podle nastavených přístupových rolí) – provádí část *replikace dovnitř* ve spolupráci s částmi *mapování aplikací a uživatelů* a *systém práv*;
- vytváření diferenčních dat porovnáním obsahu centrální databáze k zadanému datu a přijatých dat – to je opět úkolem části *replikace dovnitř*;
- evidenci výskytu datových objektů v provozních databázích – to je součástí *mapování aplikací a uživatelů*;
- mapování identifikátorů objektů v provozních systémech na centrální identifikátory včetně rozpoznávání nových entit – i to je součástí *mapování aplikací a uživatelů*;
- aktualizaci centrálních databází zpracovanými přijatými daty podle váhy– provádí replikační server podle *popisu atributů* a *replikace dovnitř*;
- generování seznamu cílových provozních systémů pro zpětnou aktualizaci přijatými a centrálně zpracovanými daty – to je úkolem části *replikace ven*;
- distribuci aktualizací dat – i to je úkolem části *replikace ven*;
- archivaci informací o průběhu datových synchronizací a identifikovaných problémech (protokolování) – to je úkolem části *replikace dovnitř*;
- řízení průběhu datových přenosů pomocí definice jednotlivých akcí a jejich plánování v čase.

3.3 Lokální export

Může probíhat ve dvou režimech podle typu provozní aplikace:

Online režim – vhodný zejména pro lokální SQL databáze ve známém formátu ve spolupráci s dodavatelem provozního systému:

přímé vložení změn do synchronizačního mechanismu

Offline režim – u ostatních aplikací:

Převod dat do XML v jednotném XML schématu

Zaslání po síti aplikačnímu serveru

3.4 Lokální import

Zobrazí změněná data provozní aplikace, po odsouhlasení autorizovaným uživatelem proběhne lokální import odsouhlasených dat. U provozních systémů, které nedovolují strojový lokální import, musí uživatel takového systému provést změnu dat ručně.

4 Rozpoznávání a vážení dat

Rozpoznávání nových entit a vážení importovaných dat jsou algoritmy použité v průběhu synchronizace na straně aplikačního serveru.

4.1 Rozpoznávání

Všechny atributy budou rozřazeny do tří tříd:

Určující – zcela jednoznačně určuje danou entitu (např. u osoby ve valné většině případů rodné číslo, příp. připravované jednoznačné číslo sociálního pojištění). Navíc slouží jako základ při generování ENTID pro zcela novou entitu.

Relevantní – důležitější atributy, které pomohou jednoznačně rozpoznat shodu entit (např. u osoby jméno a příjmení).

Vedlejší – nemají vůbec vliv na rozpoznávání stejných entit.

Pro rozpoznávání shody entit bude použit následující algoritmus:

Shodují se určující i relevantní atributy – zcela bezproblémová shoda, je možné pokračovat dál se změnou atributů pro danou entitu bez ukládání pomocné informace do záznamu o replikaci.

Shodují se určující atributy, ale neshodují se relevantní – tento případ může např. u osob nastat vdáním se ženy a převzetím manželova příjmení. V takovém případě se vytvoří nová entita, aby mohla být přijímaná data uložena v datovém stohu. Do záznamu o replikaci se uloží informace o této kolizi s odkazy na nově vzniklý i pravděpodobný (určený určujícími atributy) ENTID. Navíc tato informace musí být zpětně doručena uživateli provozní databáze, který tuto kolizi vyvolal, s navrhovanou změnou na pravděpodobnou entitu podle určujících atributů. Uživatel pak sám rozhodne, jestli skutečně žádá změnu relevantních atributů.

Jeden z určujících atributů je jiný a ostatní určující a relevantní se shodují – může opět u osob nastat např. překlepem v zápisu rodného čísla. řešení je obdobné jako v předchozím případě.

Všechny ostatní neshody už definují různé entity, řešení případných konfliktů bude umožovat administrátorská aplikace.

Algoritmus rozpoznávání stejných entit je důležitý pro sjednocení záznamů z různých provozních systémů z různých pracoviš.

Pokud se uživatel u provozního systému rozhodne, že skutečně žádá změnu relevantního nebo i určujícího atributu (nastává po potvrzení uživatele při neshodě atributů v případě b) a c)), budou provedeny kroky pro sloučení dat z nově vzniklé a originální entity na straně centrální databáze. Dále je zapotřebí nově změněné určující a relevantní atributy rozdistribuovat do ostatních provozních systémů, které s nimi také pracují.

4.2 Vážení dat

U každé datové položky v datovém stohu bude uloženo číslo, které udává výši relevance dat. Tato hodnota se při replikaci dat směrem dovnitř průběžně vypočítá podle metody popsané níže. Pokud je relevance nově příchozích dat (atributu) větší nebo rovna relevanci stávajících dat (atributu), je tento atribut změněn. Pokud tomu tak není, hodnota atributu se nezmění, pouze se do záznamu o replikaci přidá informace o tomto odmítnutí změny dat. Uživatel provozního systému pak může novým potvrzením na straně lokální replikační aplikace zvýšit důležitost této změny a ta se pak projeví hlášením správci centrální databáze, který pak po ověření povolí změnu.

Pro určení relevance dat je zapotřebí brát v úvahu jak relevantnost zdroje tak relevantnost typu provozního systému.

Pro výpočet relevantnosti všech atributů entity z jednoho zdroje a jednoho provozního systému využijeme následující vzorec:

$$R_a = R_z \cdot R_{ps} \cdot R_e,$$

kde R_a je relevance atributu, R_z je relevance zdroje, R_{ps} je relevance provozního systému a R_e je relevance entity. Relevance entity vyjadřuje váhu dat poskytovaných danou entitou (např. informace získané z hlavního pracovního úvazku jsou důležitější).

Relevance zdroje a provozních systémů budou určeny přímo metatabulkami v části mapování aplikací a uživatelů. Relevance entity je pak určena následujícím předpisem

$$R_e = \prod_i R_{re}\,(i),$$

kde hodnota R_{re} (i) nabývá metatabulkami konfigurovatelného čísla při shodě hodnoty atributu entity s hodnotami uloženými v metatabulkách, při neshodě je rovna jedné. Tyto hodnoty mají za úkol identifikovat relevantnost jednotlivé entity podle druhu entity.

5 Závěr

Popsaná architektura informačních systémů představuje alternativu ke klasickým uspořádáním, která přináší řadu výhod, ovšem též určité nevýhody.

5.1 Výhody

Základními a všeobecně uplatnitelnými výhodami stohu je udržovatelnost, rozšiřitelnost a možnost uchovávání historie dat. K nim přibývá možnost určování míry relevantnosti dat a sledování období jejich platnosti. Pro některé aplikace pak může být užitečná i existence jednoznačné identifikace každého objektu (ENTID). Tyto výhody jsou aplikovatelné především v informačních systémech, které jsou vytvářeny s nejistotou ohledně směru jejich dalšího vývoje a tam, kde je v budoucnosti předpokládána analýza dat podle různých těžko předvídatelných kriterií. Vzhledem k oblibě různých manažerských systémů typu OLAP i k rostoucímu výzkumu v oblasti dolování dat lze očekávat, že takových informačních systémů bude stále přibývat.

Významnou výhodou popsaného přístupu je i skutečnost, že změna struktury dat je realizována změnou metadat uložených v řídících tabulkách, které jsou z pohledu databázového stroje uživatelskými tabulkami. To znamená, že úpravy struktury lze provádět jednak uživatelem, který nemá administrátorská privilegia, jednak softwarem, který má stejný charakter, jako uživatelské aplikace, a nevyžaduje spouštění DDL příkazů, které jsou speciální a špatně přenositelné.

5.2 Nevýhody

První otázkou při posuzování aplikovatelnosti architektury stohu je otázka nárůstu objemu dat oproti klasické horizontální architektuře. Konkrétní poměr je silně závislý na charakteru aplikace (na reálném vzorku dat aplikace personálního typu došlo k nárůstu o 130% při 200 000 položkách stohu), k nezanedbatelnému nárůstu ovšem dochází vždy. Vertikalizace dat má tedy smysl pouze tehdy, je-li tato nevýhoda vyvážena jinými výhodami, především možností ukládat historii dat.

Ačkoliv je základní princip vertikalizace dat používán již řadu let v některých nadstavbách nad SQL databázemi, v míře popsané v tomto článku je dosud používán naprosto ojediněle. Pro návrháře a programátory aplikací jde tedy o nezvyklý přístup a může jim činit obtíže. Tento přístup navíc není podporován nástroji, používanými při vývoji aplikací.

Tomuto problému je možno čelit zakrytím vnitřní struktury dat prostřednictvím pohledů nebo odvozených tabulek, které zpřístupují data v klasické horizontální podobě. Taková úprava má ovšem podstatné nevýhody: Dotazy prostřednictvím pohledů silně zatěžují databázový stroj, zatímco odvozené tabulky (či materializované pohledy) přinášejí otázku koherence, žádný z těchto postupů neřeší otázku změn dat prostřednictvím těchto aplikací. Navíc se touto úpravou ztrácí původní výhody vertikální architektury, především snadná rozšiřitelnost a uchovávání historie dat.

Toto řešení je tedy vhodné pouze v informačních systémech, kde je podstatná část aplikací určena pro jednodušší manipulace s daty s tím, že obecné jádro s vertikalizovanými daty slouží především jako základna pro slévání, dlouhodobé udržování a dolování dat. Pro základní aplikace je pak použita klasická technologie nad pohledy či cache, pro administrativní, statistické a analytické nástroje

jsou pak vytvořeny aplikace přímo nad vertikální architekturou. Takových informačních systémů však v poslední době přibývá a tím přibývá i prostor pro aplikaci architektury vertikálních dat a stohu.

5.3 Výhled

V poslední době se stále větší množství aplikací vytváří prostřednictvím paradigmat a nástrojů rodiny XML [3,4]. Efektivnímu nasazení těchto technik v oblasti informačních systémů však brání výrazná výkonová zaostalost za technikami klasických databázových strojů, absence transakčního zpracování a nedostatek návrhových nástrojů obvyklých u relačních databází. V tomto článku popsanou metodu je vzhledem k organizaci uložení dat a přístupu k nim možno chápat i jako určitý most mezi relačními databázemi a technologiemi XML. Výhodou tohoto přístupu je silné zázemí klasického databázového stroje poskytující vysoký výkon a transakční zpracování.

Stejně jako u XML zde chybí silné návrhové nástroje včetně teoretického zázemí známého z relačních technik. Budoucí vývoj popsané techniky by proto měl probíhat v následujících oblastech: Formálnější specifikace formátu a semantiky dat, uložených ve stohu; teoretické studie mechanismů synchronizace s důrazem na problém nekonzistence dat z různých zdrojů; návrh a implementace návrhových nástrojů a nasazení této techniky v reálných podmínkách včetně měření výkonu.

References

[1] Finger, M.: A Logical Reconstruction of Temporal Databases, IME, Univ. de Sao Paulo, Journal of Logic and Computation, 1997

[2] Jensen, C.S., Snodgrass R.T., Soo M.D.: Extending Normal Forms to Temporal Realtions, Technical Report TR 92-17, University of Arizona, 1992

[3] Mlýnková, I.: XML Schema a jeho implementace v prostředí relační databáze, MFF UK Praha 2003

[4] Toman, K.: XML data na disku jako databáze, MFF UK Praha 2003

Metody tvorby www prezentací – zkušenosti z Masarykovy univerzity v Brně

Šárka Ocelková, Jaromír Ocelka

Ústav výpočetní techniky, Masarykova univerzita v Brně
Botanická 68a, 602 00 Brno, Česká republika
{ocelkova, ocelka}@ics.muni.cz

Abstrakt Příspěvek se zabývá metodikou tvorby www prezentací, která je podložena několikaletými zkušenostmi s realizacemi různých www systémů a prezentací. Je popsán postup vytváření prezentace od počátečního zadání, přes rozdělení odpovědností mezi řešitele jednotlivých částí, až po konečnou realizaci. Převážně je popisována technická stránka možných řešení, jejich výhody a nevýhody pro konkrétní použití.

Klíčová slova: www prezentace, uložení dat, tvorba www stránek, navigace, interaktivita,bezpečnost, klasifikace www prezentací.

1 Úvod

Od svého vzniku se Internet postupně vyvíjí, v posledních letech si většina jeho uživatelů představuje pod pojmem Internet pouze www servery. Tento trend je způsoben hlavně tím, že téměř všechny síťové služby jsou uživatelům k dispozici přes jedinou aplikaci – www prohlížeč. Jedna z původních myšlenek zakladatelů služby *www* – integrace stávajících služeb – byla tímto více než splněna a v současné době může být pod www stránkami prezentace skryta i složitá vnitřní infrastruktura.

Na Ústavu výpočetní techniky Masarykovy univerzity v Brně (dále ÚVT MU v Brně) vzniklo v polovině devadesátých let specializované pracoviště, orientované právě na tvorbou www prezentací (dále webů), a později také na informační systémy, založené na www technologiích. Při realizaci různých webů bylo vždy potřeba dobře rozmyslet a zvolit konkrétní postupy a technologie, které jsou vhodné pro řešení dané problematiky. V následujících kapitolách jsou uvedeny jednotlivé kroky analýzy a realizace, které by se při tvorbě nového webu neměly opomenout.

2 Co obnáší vytváření www prezentací

Na www prezentaci lze pohlížet jako na tři samostatné části, u nichž se vyplatí dobrá analýza, a jež mohou z počátku řešit tři samostatné vývojové skupiny. Až posléze se tyto části spojí a vznikne úplný web. Již na počátku je však dobré si ujasnit, kdo bude za kterou z oněch částí odpovídat, určit tedy konkrétní *odpovědnou osobu*. Podívejme se na jednotlivé části blíže:

1. **Datový (informační) obsah**
   Odpovědná osoba se zpravidla nazývá *správce dat* nebo *správce informací*. Správcem může být odborník na informační technologie, ale často jím bývá i laik. S touto možností je dobré předem počítat, neboť úkolem správce je především trvale dohlížet na aktuálnost údajů, a proto mu musí být umožněn přístup k datům na odpovídající úrovni. Představa informačního obsahu a charakteru webu je nutným základem pro následující dvě části, proto musí správce dat hned na počátku sdělit alespoň základní představu o tom, jaké informace se na webu budou prezentovat, jaký je předpokládaný objem a jakého budou typu (textové dokumenty, obrázky nebo přehledové informace např. zaměstnanců, publikací apod.). Od toho se dále odvíjí rozsah celého webu (desítky, stovky nebo tisíce stránek). Dobré je také vědět předem, jak častý bude požadavek na aktualizaci dat a zda bude prováděn ručně nebo automaticky, např. přenosem z externího zdroje dat.
2. **Technická realizace**
   Odpovědná osoba – technický realizátor – se nazývá *správce systému*. Bývá jím téměř výhradně odborník na informační technologie. Jeho úkolem je na základě informací od správce dat navrhnout příslušnou technologii, celý web zrealizovat, nadále udržovat a odpovídat za jeho trvalý a bezchybný chod. Důležité pro volbu technologie je, zda prezentovaná data budou veřejná nebo přístupná autentizovaně. Před vlastní realizací webu je vhodné učinit základní odhad předpokládaného počtu návštěvníků a na jeho základě pak zvolit dostatečně výkonný hardware. Nezanedbatelným kritériem budou též finanční možnosti a nároky na web.
3. **Vzhled stránek (Grafika)**
   Odpovědnou osobou by měl být v ideálním případě zkušený *grafik*, který nejen navrhne vizuální podobu stránek, ale také ji převede do rozumně použitelné (elektronické) podoby. O takové odpovědné osoby bývá bohužel vždy nouze. Grafickou podobu stránek ale rozhodně není dobré podcenit. Grafika dává tvář celé www prezentaci a právě ta určuje, jak bude prezentace přehledná a jak s ní budou uživatelé rádi pracovat.

3 Možnosti uložení dat

Jak již bylo řečeno, data jsou základem každého webu. Vždy jsou ale nějak specifická a nemá smysl se v tomto příspěvku podrobněji zabývat jejich konkrétní strukturou. Vždy však musí být někde uložena, a možností jejich uložení již není mnoho. Nejpoužívanějšími úložišti jsou buď soubor (jak strukturovaný, tak nestrukturovaný) nebo databáze (vždy strukturovaná). Každé řešení má své výhody a nevýhody, na něž se nyní podívejme podrobněji:

1. **Nestrukturovaný soubor**
   Data jsou uložena v souboru společně s kódem, který formátuje stránku (např. HTML nebo SHTML), v tomto případě není dodržována žádná jednotná datová struktura. K vytvoření takového souboru postačí pouhá znalost jazyka HTML. Sloučení dat s kódem nese samozřejmě velké riziko chyb,

které mohou stránku (byť dočasně) znehodnotit. Tento způsob se však hodí pro malé (rozsahem stránek i objemem dat) weby (např. soukromá osobní stránka), kde by byla investice (vzhledem k počtu a povaze stránek) do větších řešení zbytečná a značně neekonomická.

2. **Strukturovaný soubor**
   Data jsou oddělena od formátovacího kódu stránky a uložena v předem definované struktuře do samostatného souboru. V dnešní době se nabízí moderní formát XML. K tomu je ovšem potřeba naprogramovat další software, v tomto případě XSL transformaci, jež bude data převádět do požadovaného výsledného HTML. Z nutnosti existence programu je tato technologie vhodná pro weby alespoň středního rozsahu (viz také dále). Velkou výhodou tohoto řešení je snadné zpřístupnění dat jejich správcům (např. zpřístupnění části disku s datovými zdroji) a snadná možnost editace v běžném textovém editoru nebo specializovaném XML editoru, obzvláště je-li správcem dat laik. Další výhodou je přehledný zápis, velmi vhodný pro strukturované texty (např. zápisy, výroční zprávy, články ve sbornících apod.). Toto řešení se stává nevýhodným v případě velkých objemů dat (tisíců, milionů záznamů), kdy je již editace souboru nepřehledná a manipulace s tak objemným souborem může být pro běžné textové editory komplikovaná a pomalá.
3. **Strukturovaná (relační) databáze**
   Data jsou uložena nejčastěji v relační databázi. Z povahy databáze je zřejmé, že data jsou zde uložena vždy strukturovaně a jsou oddělena od formátovacího kódu stránky. K vytvoření stránky je opět potřeba naprogramovat software – skript generující stránku (např. asp, php, jsp, ...). Velkou výhodou tohoto řešení je dynamičnost a možnost parametrizace stránek, snadná je hromadná manipulace s daty. Proto je databázové řešení vhodné pro weby velkého rozsahu (viz též dále). Komplikovanější bývá zpřístupňování dat v databázi správcům dat. Pokud jím není zrovna odborník znalý jazyka SQL, je nutné mít naprogramovaný software pro editaci dat. Použití databáze není příliš vhodné pro udržování a vkládání strukturovaných textů (zápisy, zprávy, ...).

4 Možnosti tvorby www stránek

Nyní se dostáváme k vlastní technické realizaci www stránek. Některé možnosti již byly lehce nastíněny v předchozí kapitole, věnujme se jim však nyní podrobněji. Z nejobecnějšího hlediska existují v podstatě dvě možnosti, jak vytvářet www stránky:

1. **„Ruční“ psaní stránek**
   Každá www stránka prezentace je napsána přímo v jazyce HTML a je statické povahy. Hrozí zde již zmíněné riziko chyb a znehodnocení stránky, proto je „ruční“ psaní vhodné pouze pro malé weby. Je-li navíc u prezentace požadován jednotný vzhled všech stránek, pak je zásadní nevýhodou tohoto řešení opisování téhož formátovacího kódu do každé stránky. Při požadavku na sebemenší drobnou změnu vzhledu je nutné projít všechny stránky a všude

provést příslušné úpravy. Tuto situaci lze částečně řešit použitím SHTML a vyčleněním společného formátovacího kódu do samostatného souboru, nelze tak již ale plnohodnotně a obecně řešit vnitřní formátovací prvky, jako např. vzhled tabulek, odkazů apod.

2. **Generování stránek**
   V tomto řešení stojí za každou stránkou webu skript/program, který ji na vyžádání vygeneruje (ať už ze souboru nebo databáze). Můžeme rozlišit dvě varianty generování stránek: on-line a off-line generování. Společnou výhodou obou možností je skutečnost, že veškeré formátovací prvky stránky (jednotný vzhled, navigační prvky, ...) mohou být uloženy na jediném místě, rovněž je možné ohlídat většinu chyb a překlepů. Existence skriptu přirozeně vyžaduje znalost programování, proto se generování stránek „vyplatí“ až u webů alespoň středního rozsahu nebo u webových aplikací (viz dále). Podívejme se nyní, v čem jsou tyto dva způsoby rozdílné.

   (a) **On-line generování**
   Každá stránka je vygenerována ihned na vyžádání. To přináší obrovskou výhodu možnosti variability a parametrizace stránek, jediný skript tak může na základě obměny parametru generovat tisíce stránek stejného typu a vzhledu, jen s jiným požadovaným obsahem. Vhodné použití může být např. pro stránky katalogu výrobků velkého podniku, kterých může být až několik tisíc.

   (b) **Off-line generování**
   Stránka je v tomto případě vygenerována a následně uložena do statického HTML souboru, díky tomu je zde podstatně menší možnost variability a parametrizace. Zpravidla se negeneruje jen jedna stránka, ale celá kolekce stránek, proto musí u tohoto řešení někde existovat (např. v konfiguračním souboru nebo v databázi) seznam všech skriptů, které se mají požadovat, seznam všech možných parametrů k těmto skriptům a seznam cílů, kam mají být výsledné vygenerované stránky uloženy (podrobněji popsáno také v [UNI-01]). Tento mechanismus pak může být automaticky načasován, a celý web se tak dávkově najednou zaktualizuje.

   Jelikož se při přegenerování může vyskytnout nějaká chyba (ať již ve skriptu generujícím stránku nebo nějaká jiná), je třeba zamezit situaci, kdy se vygeneruje jen část stránek a web je pak neúplný a chybný. Pro tento účel se osvědčuje řešení *transakčního přegenerování*, tj. celý web se vygeneruje do jiného adresáře, než je aktuální adresář webu, a v případě korektního vygenerování všech stránek se pak tyto adresáře „vymění“. Tento princip je podrobně popsán v [TSW-03j] a [TSW-03s].

   Vzhledem ke skutečnosti, že se generuje a ukládá tolik stránek, kolik je použitých hodnot parametrů, je tento způsob generování vhodný u menšího počtu (do několika set) stránek. Příkladem může být katalog výrobků nebo služeb menšího podniku, výroční či jiné zprávy apod.

5 Jednotný vzhled www stránek a navigační logika

Pro uživatele je většinou přitažlivější, je-li celá www prezentace v jednotném grafickém provedení. Samotná grafika však nestačí k tomu, aby se uživatel v celé stránkové struktuře dobře a správně orientoval, proto je vhodné také pouvažovat o rozvržení informací na stránkách a navigační logice.

Několikaletá zkušenost s vytvářením webů na ÚVT MU v Brně potvrdila, že na celý web lze pohlížet jako na kolekci www stránek, jež dohromady sice tvoří obecný graf, ale většinou vždy lze nalézt hlavní kostru, která vytváří stromovou strukturu. Tuto strukturu je možné procházet dvěma směry: vertikálně („shora dolů“) a horizontálně („vodorovně“). Vertikálním procházením se postupně zpřesňuje informace, horizontálním je pak možné získat tutéž informaci jen s jinými vstupními parametry nebo informaci na stejné úrovni, tj. ve stejném vztahu k nadřazené. Horizontální procházení webu je zpravidla reprezentováno ve formě svislé nabídky v levé části stránky, k vertikálnímu procházení dochází postupně volbou odkazů v hlavní části stránky. Aby byl umožněn návrat zpět (tj. procházení „zdola nahoru“), objevují se s každou další úrovní postupně v pravém horním rohu stránky ikony, odkazující na předchozí nadřazené úrovně (viz také [UVT-97] a [UNI-00]).

V závislosti na charakteru webu a předpokládané cílové skupině uživatelů je vhodné vytvářet zdrojový kód stránky s ohledem na případné zrakově handicapované uživatele. Lze definovat několik základních pravidel a doporučení, o nichž podrobněji pojednává [TSW-03r].

6 Interaktivita www stránek

Pro větší přehlednost, snadnější orientaci a také přitažlivost prezentace může být vhodné doplnit prezentaci o určitou míru interaktivity s uživatelem. Pro dosažení těchto cílů je možno použít různé technologie, ne všechny jsou však k dispozici ve všech verzích www prohlížečů a ve všech operačních systémech. Důležité je vždy si uvědomit, jaká je cílová skupina uživatelů, a podle toho zvolit příslušnou technologii. I při těch největších nárocích na funkčnost je možné dynamicky reagovat na typ uživatelova prohlížeče a operačního systému a podle toho mu nabídnout příslušnou variantu, která se z předem připravených vybere na www serveru.

Prakticky bezproblémové je generování stránky dynamicky skriptem, jež operativně reaguje dle parametrů uživatelova požadavku. Takovým uživatelským zpříjemněním může být už obyčejný seznam různě parametrizovaných odkazů nebo tzv. obrázek s klikou, tj. obrázek rozčleněný na části, kde klepnutí myši na určité části vyvolá příslušný požadavek na novou stránku (na každé části jinou). Standardní a pro uživatele srozumitelné je též použití formulářových prvků (například pro výběr kategorie výrobků apod.).

V poslední době se také hojně rozšířilo použití krátkých skriptů ve skriptovacím jazyce JavaScript, což odlehčí www serveru při interaktivním chování, neboť skript může logiku zpracovávat přímo prostřednictvím prohlížeče a částečně tak modifikovat obsah stránky jako odpověď na uživatelův požadavek.

Dalším typem je použití tzv. zásuvných modulů (pluginů), kdy je možné v okně prohlížeče spustit libovolný program. Tímto lze například ve www stránce spustit hru, ... Zde se však již dotýkáme hranice kompatibility, kdy uživatel nemusí mít příslušný plugin nainstalován, nebo dokonce výrobce pluginu nemusí vůbec podporovat daný prohlížeč či operační systém. V dnešní době jsou na tomto založeny nejčastěji reklamy spouštěné ve stránce.

7 Zabezpečení www prezentace

U některých webových prezentací může být důležité, aby určitá část informací nebyla veřejná. V tomto případě musí být zajištěno, aby nikdo nepovolaný nemohl zjistit, jaké informace uživatel serveru předal a jaký obsah si od serveru vyžádal. Ještě větší jsou nároky na zabezpečení v případě, kdy www server zná identitu uživatele (uživatel se vůči serveru autentizuje například loginem a heslem) a podle ní mu nabízí obsah. Proto www servery s citlivými informacemi používají jako zabezpečení nejzranitelnějšího místa, přenosové trasy mezi uživatelem a www serverem, šifrované komunikace – https. Uživatelům nezbývá než věřit, že je www server zabezpečen (je zamezen fyzický přístup nepovolaným osobám apod.), ale pro ochranu druhé strany komunikační cesty, tj. vlastního vstupního místa, mohou udělat mnohé, např. nepřistupovat z počítačů internetových kaváren apod.

Průměrný uživatel nemůže být znalý všech možných rizik, proto je možné na www serveru implementovat některé dodatečné prvky bezpečnosti orientované na konkrétní uživatele. V rámci informačních systémů MU v Brně byla například vyvinuta tzv. *IP bezpečnost* (podrobně viz [TSW-02]), kdy si může uživatel zvolit, které části systémů jsou dostupné z kterých počítačů. Jako příklad uveďme možnost nastavení dostupnosti elektronického výplatního lístku jen z vybraných počítačů, např. pouze ze sítě MU.

Další zabezpečení vyvinuté na MU v Brně bylo inspirováno skutečností, že univerzita má několik různě orientovaných informačních systémů a v každém systému měli uživatelé samostatná uživatelská jména a hesla. V takto komplikovaném heterogenním prostředí je nutné zabránit tomu, aby uživatelé měli do každého systému různá hesla (při větším počtu je větší pravděpodobnost, že si je poznamenají „na papírek“) a odstínit méně důvěryhodné servery (například studentský klub, ...) od přímého kontaktu s hesly. Z těchto důvodů bylo vyvinuto několik metod *poskytované autentizace*, které se snaží tyto problémy řešit. Podrobně o této problematice pojednává [DAT-03].

8 Návštěvnost a optimalizace www prezentace

Užitečné informace může přinášet sledování návštěvnosti webu a vyhodnocování statistik z logů požadavků www serveru, ve kterých se nachází například informace o typu prohlížeče. Další formou, jak získat ještě jiné informace, může být

malá anketa umístěna přímo na www stránkách, kde se uživatelé mohou vyjádřit k celé prezentaci. Na základě těchto údajů je pak možné přizpůsobovat web k větší spokojenosti návštěvníků.

Příkladem může být zjištění, že velké procento uživatelů je připojeno z domova pomocí modemu, a je tudíž vhodné, aby jednotlivé stránky včetně obrázků nebyly náročné na přenosovou kapacitu. Správce dat a správce systému by měli také vyhodnocovat, zda server není z přílišného počtu návštěvníků přetížen, a v takovém případně pak vhodně (v rámci možností) reagovat a zajistit výkonný hardware nebo pro velmi často navštěvovaný web raději použít cluster www serverů (viz [TSW-03j]). Provádění těchto změn nemá žádný vliv na funkčnost z pohledu uživatele, není tedy nezbytně nutné zabývat se tímto při prvotním návrhu prezentace a případné rozšíření hardwaru realizovat kdykoliv později.

9 Realizované www prezentace

Na Ústavu výpočetní techniky Masarykovy univerzity v Brně se od poloviny 90. let postupně zrealizovala a nadále provozuje řada www prezentací, ať již jako webové vrstvy informačních systémů, nebo jako samostatně stojící prezentace. Následuje chronologický přehled webů, které ÚVT vytvořilo a nadále provozuje:

1996: Zpravodaj ÚVT MU (`http://www.ics.muni.cz/bulletin/`)
elektronická verze informačního bulletinu o výpočetní technice na MU v Brně

1996: web ÚVT MU (`http://www.ics.muni.cz/`)
www prezentace Ústavu výpočetní techniky MU v Brně prezentující především informace o pracovištích, jejich zaměstnancích, službách ÚVT, ...

1997: web MU (`http://www.muni.cz/`)
institucionální www prezentace Masarykovy univerzity v Brně zpřístupňující přehledové informace o zaměstnancích, studentech, pracovištích, vědě a výzkumu, možnostech studia, kalendáři akcí apod.

1998: intranet webu MU (`http://wwwdata.muni.cz/`)
intranetový subsystém určený pro vkládání a editaci dat webu MU, především kontaktních údajů, vědy a výzkumu, kalendáře akcí apod.

1999: SIMS – Sdružené informace matrik studentů (`http://sims.ics.muni.cz/`)
první neuniverzitní web tvořený na zakázku pro Ministerstvo školství, mládeže a tělovýchovy ČR, jedná se o celostátní matriku studentů všech vysokých škol celé ČR.

2000: Studovna (`http://studovna.muni.cz/`)
www prezentace Celouniverzitní počítačové studovny, prezentuje informace o plánovaných přerušeních studovny, Radě studovny, provozním řádu, uživatelské návody apod.

2000: Inet MU (`http://inet.muni.cz/`)
celouniverzitní personálně-mzdový a ekonomický intranet zpřístupňující každému zaměstnanci jeho personálně-mzdové údaje, jako je např. výplatní lístek, evidence docházky, přehled čerpání grantů, ...

2001: MuniNet (`http://www.muninet.cz/`)
web prezentující informace o komerční činnosti ÚVT MU v Brně.

2002: ČKR – Česká konference rektorů (`http://crc.muni.cz/`)
další www prezentace na zakázku, tentokrát pro Českou konferenci rektorů, prezentuje informace o členech ČKR, usneseních, zápisech, historii, ...

10 Klasifikace www prezentací

Na základě několikaletých zkušeností s tvorbou www prezentací (uvedených v přechozí kapitole) se nyní můžeme pokusit obecně klasifikovat weby. Kritériem je rozsah stránek, objem dat a účel webu:

1. **Weby prezentující data**
   Jedná se o takové weby, jejichž účelem je pouze prezentovat data z datového zdroje. Většinou nemají autentizovanou část (tj. všechny údaje jsou prezentovány veřejně) a neumožňují data nijak modifikovat. Mohou být i částečně interaktivní, většinou prostřednictvím různých nabídek či formulářů (např. vyhledávací políčka apod.). Dle rozsahu je můžeme dále dělit na:
   (a) **Weby typu Homepage**
   – jsou weby jen o několika stránkách (v řádu jednotek nebo několik málo desítek), objem dat rovněž není nijak velký. Typickým příkladem takového webu jsou soukromé osobní stránky, web malé soukromé firmy, drobné cíleně zaměřené weby (např. informace o nějakém projektu) apod. Informace na takovém webu jsou zpravidla aktualizovány jen občasně dle potřeby, a co se týče vzhledu, většinou není nutné dodržet jej jednotný pro všechny stránky. Vzhledem k těmto skutečnostem se příliš nevyplatí investovat do nějakých větších technologií, zde plně postačuje uchovávat data nestrukturovaně přímo v HTML (eventuálně v SHTML, pokud by byl požadavek na jednotný vzhled). Nezanedbatelným aspektem je prakticky nulová finanční náročnost, neboť zdrojové soubory stránky lze editovat téměř v libovolném textovém editoru.
   Realizované weby na ÚVT: MuniNet (HTML), Studovna (SHTML).
   (b) **Weby středního rozsahu**
   – jsou weby o desítkách až stovkách stránek, za nimiž stojí už také větší datová základna. Typickým příkladem jsou elektronické verze časopisů, web středně velké firmy, weby konferencí apod. Požadavek na jednotný vzhled už by měl být žádoucí. Je také na místě uvažovat o oddělení dat od formátovacího kódu stránky a o tom, kam data uložit. Z výše popsaných možností se nabízí uložení do XML souboru nebo do malé databáze (např. MS Access, MySQL, ...). K oběma řešením je zapotřebí nějaký program, který data převede do výsledného HTML (XSL transformace, Perl skripty, ...). Správce dat pak přistupuje k datům buď přímo (editace souboru, XML editor, aktualizace databáze pomocí SQL) nebo prostřednictvím specializované aplikace (formuláře v MS Access).
   Realizované weby na ÚVT: web ÚVT MU (kombinace HTML a MS SQL, ASP, off-line generování), Zpravodaj ÚVT (kombinace HTML a MS SQL, ASP, off-line generování), ČKR (XML, XSL, off-line generování).

(c) **Weby velkého rozsahu**
– jsou rozsáhlé weby o stovkách, tisících a více stránkách, za nimiž stojí rozsáhlé množství dat. Příkladem jsou weby velkých firem, zpravodajské servery apod. Požadavek na jednotný vzhled stránek se zde stává (už kvůli orientaci v jejich struktuře) prakticky nutností, pro uložení dat je jedinou vhodnou možností výkonná databáze (MS SQL, Oracle, ...). Při takovém objemu stánek není možné celý web generovat do statických HTML stránek, proto je zde jedinou možností tvorby stránek dynamické on-line generování (ASP, ASP.NET, PHP, ...). Ke správě dat je již zapotřebí specializovaná aplikace (tlustý klient, Intranet, ...).
Realizované weby na ÚVT: web MU v Brně (MS SQL, ASP, kombinace on-line a off-line generování).

2. **Webové aplikace, portály**
Jedná se o svým způsobem zvláštní typ webů, které nejen prezentují data, ale dovolují s daty navíc pracovat, modifikovat je a dle požadavků operativně generovat. Takovým webům se zpravidla říká aplikace. Prakticky vždy mají autentizovanou část, která může být doplněna o definici pravidel oprávnění k jednotlivým specializovaným částem webu, v konečném výsledku pak každý uživatel vidí svou vlastní část. Příkladem takových webů jsou firemní Intranet (personalistika, ...), www e-mail, internetové obchody apod. Požadavek na jednotný vzhled aplikace je zde nutností už kvůli uživateli, aby se mohl v celém Intranetu dobře orientovat. Jediným možným úložištěm dat je výkonná databáze (MS SQL, Oracle, ...), stejně tak z povahy webu musí být aplikace tvořeny dynamickými on-line stránkami (.NET, J2EE, ...) a jsou převážně založeny na třívrstvé architektuře.
Realizované weby na ÚVT: intranet webu MU (MS SQL, ASP, on-line generování), SIMS (MS SQL, ASP, on-line generování), Inet MU (Informix, XML, XSL, J2EE WebLogic, on-line generování).

11 Závěr

Popsané metody a postupy, ověřené provozem na ÚVT MU v Brně, se v průběhu času vyvíjely a jistě se budou vyvíjet i nadále, neboť prostředí Internetu patří v oblasti informačních technologií mezi nejdynamičtěji se rozvíjející. V rámci budování a podpory realizovaných systémů je nutné neustále sledovat nové vývojové trendy.

Odkazy

[UVT-97] Kohoutková, J.: *Masarykova univerzita na internetových WWW stránkách.* Zpravodaj ÚVT MU: bulletin pro zájemce o výpočetní techniku na Masarykově univerzitě. ISSN 1212-0901, 1997, roč.7, č.3, s.10–13. http://www.ics.muni.cz/bulletin/issues/vol07num03/kohoutkova/.

[UNI-00] Procházková, Š., Ocelka, J.: *Internetová prezentace MU v Brně.* In UNINFOS 2000. Zborník príspevkov. Nitra: SPU v Nitre, 2000. ISBN 80-7137-713-9, s.186–189. http://www.ics.muni.cz/depts/wwwm/publikace/UNINFOS2000.doc.

[UNI-01] Procházková, Š., Ocelka, J.: *Automatizace univerzitní prezentace.* In UNINFOS 2001. Zborník príspevkov. Zvolen : Vydavateľstvo TU vo Zvolene, 2001. ISBN 80-228-1062-2, s.124–128. http://www.ics.muni.cz/depts/wwwm/publikace/UNINFOS2001.doc.

[TSW-02] Ocelka, J., Měcháček, J.: *Uživatelská bezpečnost informačních systémů.* In Tvorba softwaru 2002. Vyd. první 2002. Ostrava: TANGER, s.r.o., 2002. ISBN 80-85988-74-7, s. 160–164. http://www.ics.muni.cz/depts/wwwm/publikace/TS2002_JO_JM.doc.

[TSW-03j] Ocelka, J.: *WWW Server v přílivu uživatelů Internetu.* In Tvorba softwaru 2003. Vyd. první. Ostrava: Tanger s.r.o Ostrava, MARQ, 2003. ISBN 80-85988-83-6, s.154–160. http://www.ics.muni.cz/depts/wwwm/publikace/TS2003_JO.doc.

[TSW-03s] Ocelková, Š.: *Návrh a realizace WWW prezentace ČKR.* In Tvorba softwaru 2003. Ostrava: Tanger, s.r.o., 2003. ISBN 80-85988-83-6, s.161–169. http://www.ics.muni.cz/depts/wwwm/publikace/TS2003_SO.doc.

[TSW-03r] Pěnka, P., Ráček, J.: *Bezbarierový web.* In Tvorba softwaru 2003. Ostrava: Tanger s.r.o., 2003. ISBN 80-85988-83-6, s.180–189. http://honor.fi.muni.cz/tsw/2003/180.pdf.

[DAT-03] Ocelka, J.: *Poskytnutí autentizace v informačních systémech.* In DATAKON 2003. Brno: Masarykova univerzita v Brně, 2003. ISBN 80-210-3215-4, s.259–264. http://www.ics.muni.cz/depts/wwwm/publikace/DATAKON2003_JO.doc.

Neurónové siete pre komprimáciu zvukových dát

Ľudovít Hvizdoš, Miroslav Levický

Ústav informatiky
Prírodovedecká fakulta
Univerzita Pavla Jozefa Šafárika
Jesenná 5, 040 01 Košice, Slovensko

email: {hvizdos, levicky}@science.upjs.sk

Abstrakt

Tento príspevok sa zaoberá možnosťami použitia neurónových sietí pri komprimácii rečového signálu. Neurónové siete sú celkom dobrou alternatívou algoritmov určených pre túto problematiku. Doposiaľ sú totiž známe výsledky z komprimácie obrázkov [1], pričom oblasť komprimácie zvukových dát je málo skúmaná.

Úvod

V súčasnej dobe neustále narastá množstvo informácií, čo s postupom času kladie vyššie nároky na prenosové a záznamové kapacity médií. Preto je pre ich optimálne využitie vhodné použiť komprimáciu prenášaných a zaznamenávaných dát. Pri prenášaní a uchovávaní reči je možné využiť štandardné metódy, ktoré však neponúkajú dostatočné možnosti. Ak požadujeme zachovanie len určitej úrovne kvality, namiesto dokonalej reprodukcie, umožní nám to uvažovať aj o nových metódach, ako sú napríklad neurónové siete (ďalej NS), ktorými je možné dosiahnut oveľa lepšie výsledky.

V súčasnosti je známych niekoľko algoritmov pre komprimáciu dát pomocou NS. Tie sú založené na využití viacvrstvových perceptrónových sietí, hebbovskom učení či prediktívnom kódovaní [2], [3], [4].

Pri komprimácii dát pomocou viacvrstvovej perceptrónovej siete sa využíva model siete, ktorý sa skladá zo vstupnej, výstupnej a jednej skrytej vrstvy. Počet vstupných a výstupných neurónov je rovnaký. Počet neurónovej v skrytej vrstve je menší ako je počet vstupných neurónov. Tento počet predstavuje mieru komprimácie. Komprimácia spočíva v tom, že sa vstupy transformujú na výstupy skrytej vrstvy. Prvá časť siete takto vlastne predstavuje komprimáciu. Dekomprimácia je potom transformácia skrytej vrstvy na výstupy siete. Pri učení sa využíva to, že vstupy aj výstupy siete sa rovnajú. Inak povedané, sieť sa učí funkciu identity.

Možnými rozšíreniami predošlej metódy je využitie hierarchických viacvrstvových perceptrónových sietí alebo adaptívnych viacvrstvových perceptrónových sietí [7].

Ako ďalšia metóda pre komprimáciu dát môže byť použitá metóda hlavných komponentov (Principal Component Analysis). Pri tejto metóde sa redukujú tie vstupné príznaky, ktoré nie sú významné (duplicitné alebo nadbytočné informácie). Táto metóda môže byť realizovaná pomocou lineárnej asociatívnej neurónovej siete [7].

Komprimácia dát môže byť taktiež realizovaná pomocou neurónovej vektorovej kvantizácie. Cieľom tejto metódy je aproximovať hustotu pravdepodobnostného rozdelenia vstupných dát pomocou určeného počtu reprezentantov [8]. Tendenciou je vyberať taký počet reprezentantov, ktorý by zodpovedal počtu výrazných zhlukov dát v priestore. Reprezentati sa potom určujú ako stredy jednotlivých zhlukov. Títo reprezentanti sa môžu určiť klasickou zhlukovou analýzou alebo Kohonenovými NS.

Komprimácia dát pomocou prediktívneho kódovania je technika, ktorá sa v súčasnosti úspešne aplikuje pri komprimácii reči a obrazu, tam kde je vysoká závislosť medzi susednými vzormi [2], [4], [7].

Štruktúra zvukových dát

Cieľom nášho výskumu bolo zmenšiť veľkosť zvukových záznamov, ktoré predstavujú signál zaznamenaný mikrofónom, ktorý je digitalizovaný A/D prevodníkom. Kvalita záznamu je daná kvalitou zariadenia a pre jej uchovanie potrebujeme zvoliť vhodné kódovanie. Zvukový signál ktorý dokáže vo všeobecnosti ľudské ucho rozlíšiť predstavuje vlnenie 20Hz-20kHz. Ostatné frekvencie nie je schopný ľudský sluchový aparát rozlíšiť. Pri zázname nemusíme tieto zložky uchovávať, čím môžeme zmenšiť veľkosť uchovávanej informácie bez zmeny kvality. Toto je príklad ako je možné zefektívniť spôsob záznamu. My sme sa zaoberali komprimáciou záznamov hlasov, čo je špecifický zvuk generovaný hlasovým traktom. Tieto zvuky vykazujú vysokú úroveň korelácie čo umožňuje pri použití vhodných algoritmov dosiahnuť vysoký komprimačný pomer [4]. Na základe týchto poznatkov a vlastnosti rečových nahrávok sme sa pokúsili vytvoriť systém s čo najvyššou komprimáciou na základe NS, a preto sme skúmali aj vplyv komprimácie na kvalitu reprodukovaného záznamu podľa normovaných postupov.

Komprimácia rečového signálu

Reč obsahuje veľké množstvo redudatných informácii a poslucháč je schopný rozumieť aj skreslenému signálu. Ak je preň prípustné zníženie kvality, tak pri prenose potom môžeme znížiť množstvo prenášaných dát. Použitie stratovej komprimácie rečového signálu prináša nasledujúce výhody:

- redukcia miery počas prenosu dát z vysielača do prijímača
- úspora miesta potrebného k uloženiu dát

Jej najväčšou nevýhodou je strata kvality. Pre hodnotenie kvality sme použili štandardný postup hodnotenia Mean Opinion Score (MOS) (viď tab.1). V tomto

hodnotení máme stupnicu od 1 do 5, pričom známkou 1 hodnotíme nepoužiteľnú kvalitu a známkou 5 kvalitu výbornú.

Doposiaľ používané štandardy používajú pevne danú rýchlosť prenosu. Tá je daná vzhľadom na požadovanú aplikáciu využitia. Pre predstavu PCM 64kbps MOS=4.3, ADPCM 40kbps MOS=2-4.3, RPE-LTP 13kbps MOS=3.71. Prínosom nových metód je využitie v efektívnejšom využití prenosového pásma.

Známka	Slovné hodnotenie
1	Nepoužiteľná kvalita
2	Je rozpoznateľné, že ide o reč
3	Reč nie je vždy dostatočne zrozumiteľná
4	Kvalita reči je dostatočná
5	Kvalita reči je výborná

Tabuľka 1. Prehľad hodnotenia MOS

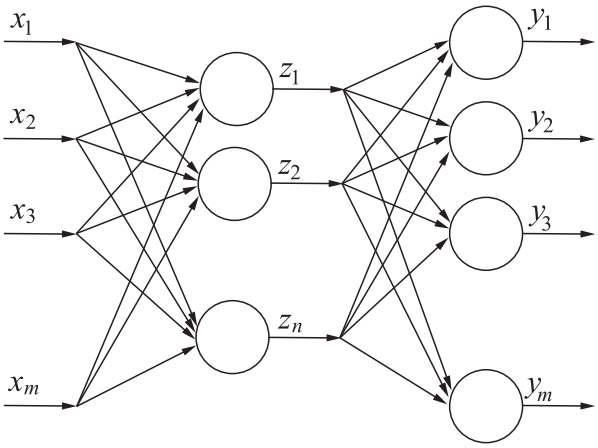
Modely sietí pre komprimáciu rečového signálu

Navrhovaná metóda je založená na schopnosti neurónových sietí aproximovať funkcie [5], [6]. Ak nám totiž postačuje signál reprodukovať s určitou povolenou chybou, tak môžeme použiť stratovú komprimáciu. Pri stratovej komprimácii nie je reprodukovaný signál identický so vstupným. Zmeny sa však prejavia len v detailoch, ktoré pri reprodukcii nepostrehneme alebo sme ochotní zmeny tolerovať.

Pri experimentoch bola použitá architektúra siete „úzkeho hrdla“, ktorá nám vytvára kóder a dekóder signálu. Ide o doprednú neurónovú sieť (obr. 1) s tromi vrstvami (vstupná, skrytá, výstupná). Vstupná a výstupná vrstva majú rovnaký počet neurónov m. Skrytá vrstva má n neurónov, pričom platí $n < m$. Táto sieť je trénovaná funkcii identity. Takto natrénovaná sieť je po častiach využitá na komprimáciu a následnú dekomprimáciu dát. Dosiahnutý komprimačný pomer je určený ako $m : n$.

Samotný postup na komprimáciu a dekomprimáciu je nasledovný. Najprv je potrebné natrénovať sieť funkcii identity ($X \rightarrow Y$). Komprimácia potom prebieha nasledovným spôsobom. Pre vzorku, ktorá má byť komprimovaná treba vypočítať výstupné hodnoty skrytých neurónov z_i (viď obr. 2). Tieto vypočítané hodnoty už predstavujú komprimované dáta.

Pri dekomprimácii potom použijeme vypočítané hodnoty z_i ako vstupy do neurónov výstupnej vrstvy (viď obr. 3). Výstup neurónovej siete nakoniec predstavuje dekomprimované dáta.



Obr. 1. Model doprednej neurónovej siete

Experimentálne výsledky

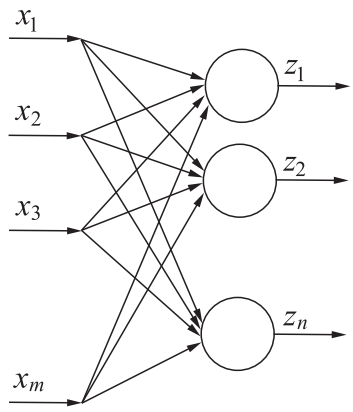
Postup vyššie popísanej metódy sme aplikovali na voľne dostupné dáta z verejných zvukových databáz rádia BBC [10] a vlastnej vytváranej databázy. Vzorky hlasov, ktoré sme použili boli zaznamenané v rôznych prostrediach. Pri pokusoch sme použili 3 sekundové vzorky, na ktoré sme aplikovali navrhnutú metódu. Jednotlivé vzorky mali nasledovné parametre:

- 1 kanál (mono)
- vzorkovacia frekvencia 22 kHz (kvalita rádia)
- veľkosť vzorky 16 bit
- štúdiové prostredie (bez alebo zanedbateľné rušivé vplyvy)

Experimenty sme robili s rôznou architektúrou skrytej vrstvy. Použili sme architektúru 16-8-16, 16-9-16 až po 16-15-16. Počet vstupných neurónov bol zvolený vzhľadom na spôsob kódovania zvuku. Jednotlivé výsledky sú uvedené v tab. 2.

Architektúry 16-7-16, 16-6-16 a ďalšie, sa ukázali ako nepoužiteľné, pretože dekomprimované vzorky vykazovali veľkú chybu. Sieť sme trénovali na 3000 vzorkách. Pre učenie siete sme použili klasický back-propagation algoritmus. Počet opakovaní trénovania mal efekt na zlepšenie kvality komprimovaných dát po ich následnej dekomprimácii (viď tab. 3).

Ďalšie zvyšovanie počtu opakovaní pri učení už kvalitu nezlepšovalo. Podobne sa chovali aj ostatné architektúry sietí. To nám dáva dobrú predstavu o schopnostiach komprimácie. Použitie ďalších metód ako napr. quick-propagation zmenšuje počet opakovaní, nedochádza však k zlepšeniu kvality pri reprodukcii dekomprimovaných dát.



Obr. 2. Časť siete použitá pre komprimáciu.

Architektúra	Počet iterácií	Kvalita
16-8-16	30 000	1
16-9-16	30 000	1
16-10-16	30 000	2
16-11-16	30 000	3
16-12-16	30 000	3
16-13-16	30 000	4
16-14-16	30 000	4
16-15-16	30 000	5

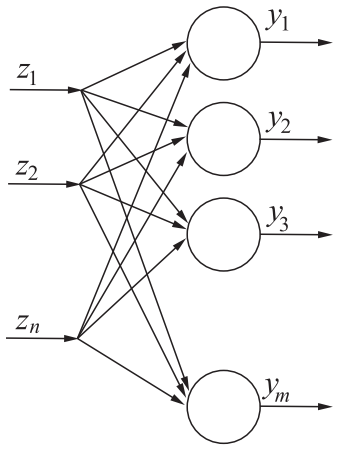
Tabuľka 2. Výsledky experimentov s rôznym počtov neurónov v skrytej vrstve.

Architektúra	Počet iterácií	Kvalita
16:13:16	10 000	1
16:13:16	20 000	2
16:13:16	30 000	4

Tabuľka 3. Výsledky experimentov s rôznym počtom opakovaní trénovania.

Záver

Dosiahnuté výsledky preukazujú možnosť použitia neurónových sietí na komprimáciu rečových nahrávok. Hlavnou výhodou je možnosť zvoliť si kvalitu komprimácie podľa potreby. Možným sa ukazuje využitie pri prenose napríklad konferenčných hovorov, kde je možné ospravedlniť určitú stratovosť. V porovnaní s ostatnými už štandardnými metódami sme nedosiahli vo všeobecnosti lepšie výsledky, prínosom je však nový prístup k problému, ktorý je možný ďalej zlepšovať. Použitie dopredných sietí využitím periodických vlastností rečového sig-



Obr. 3. Časť siete použitá pre dekomprimáciu.

nálu by umožnilo dosiahnuť lepšie kvality pri zachovaní zvolenej komprimácie. Naša ďalšia práca smeruje k skúmaniu použiteľnosti ďalších architektúr NS využiteľných pri komprimácii. Výsledky výskumu nás inšpirujú k hľadaniu ďalších smerov aplikácie NS.

Tento výskum bol podporený grantom VVGS/043/2003, ktorý bol pridelený vnútorným vedeckým grantovým systémom Prírodovedeckej fakulty UPJŠ.

Literatúra

1. Wang, Weiqiang (1999). *Improving the SFSN Neural Model for Colored Image Compression.* M.S.Thesis, Computer Science Department, New Mexico Tech.
2. Gas, B., Zarader, J.L. and Chavy C. (2000). *A New Approach To Speech Coding: The Neural Predictive Coding.* International Journal of Advanced Computational Intelligence, Vol 3, n6, Novembre 2000 pp 19-28.
3. Verma, B., Muthukkumarasamy V. (2002). *Speech Compression for VOIP: Neural Networks vs. G723.1.* Workshop on Signal Processing, WoSPA 2002, pp. 101-104, Australia.
4. Zarader, J.L., Gas, B., Chavy, C., Neslon, Charles Elie Nelson, D. (2001). *New Compression and Decompression of Speech Signals by a Neural Predictive Coding.* WSEAS conference proceedings.
5. Šíma, J., Neruda, R. (1996). *Teoretické otázky neurónových sítí.* MATFYZPRESS, Praha.
6. Sinčák, P., Andrejková, G. (1996). *Neurónové siete II (Inžiniersky prístup).* ELFA Press, Košice.
7. Mařík, V., Štěpánková, O., Lažanský, J. a kol. (2003). *Umělá inteligence 4.* Academia, Praha.

8. Rojas, R. (1996). *Neural Networks: A Systematic Introduction*. Springer-Verlag, Berlín, Heidelberg, New York.
9. http://lumumba.luc.ac.be/jori
10. http://www.bbc.co.uk

Formálna analýza bezpečnosti protokolov a paralelizmus

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 Na ochranu elektronických informácií prenášaných prostredníctvom celosvetovej siete internet sa používajú šifrované prenosy informácií. Na začiatku, kryptografickým protokolom sa komunikujúci účastníci dohodnú na kľúči používanom na šifrovanie. V literatúre môžeme nájsť mnoho protokolov, ktoré sa ukázali ako nie bezpečné z nejakého dôvodu. Formálna analýza sa zaoberá kontrolou, či daný protokol je bezpečný, pričom šifrovanie sa považuje za neprelomiteľné. Pomocou odchytenia komunikácie a manipuláciou posielaných správ je možné dosiahnuť, že "oklameme" účastníka (jedného alebo aj viacerých) protokolu. V článku spomenieme viaceré spôsoby formálnej analýzy, ako aj techniky slúžiace na automatizáciu a urýchlenie takejto analýzy.

1 Úvod

Viacero protokolov sa ukázalo byť nepoužiteľnými na zabezpečenie ochrany informácii počas prenosu, nie kvôli slabému šifrovaniu, ale manipuláciou posielaných správ. Asi najznámejším prípadom je Needham-Schroeder Public Key Protocol [5], kde takáto bezpečnostná diera bola odhalená až 17 rokov po dokázaní jeho bezpečnosti pomocou BAN logiky [1] (prvý spôsob formálnej analýzy z roku 1989).

Prvým cieľom formálnej analýzy bolo dokázať bezpečnosť pri kryptografických protokoloch, t.j. protokoloch ktoré slúžia na dohodnutie spôsobu šifrovania a kľúča pre ďalšiu komunikáciu. Pomocou logiky viery (BAN, GNY [6], ...) sa odvádzalo čomu postupne každý účastník protokolu verí, t.j. na začiatku verí nejakím kľúčom (dohodnutým predtým, zvyčajne symetrickým) alebo verí v nejakú autoritu (server, ktorý prideľuje kľúče). Každým odoslaním správy sa rozširuje množina vier pomocou odvodzovacích pravidiel.

Neskôr sa prešlo k modelovaniu nielen účastníkov protokolu, ale aj jedného špeciálneho "účastníka" - útočníka. Zisťovalo sa či pomocou získaných vier (odchytených správ) nedokáže vystupovať rovnako ako pôvodný účastník protokolu.

My sa budeme venovať ďalšiemu postupu, a to reprezentáciou účastníkov ako stavových automatov, kde každý účastník môže byť v jednom z viacerých stavov, pred prijatím alebo po odoslaní nejakej správy.

2 Formalizácia protokolu

V tejto časti si popíšeme ako treba predspracovať protokol. Toto predspracovanie, formalizácia, slúži na neskoršiu automatizovanú analýzu. Táto časť analýzy stále nie je automatizovaná. Súvisí to hlavne s dôležitosťou správnosti - ak zle zapíšeme môže sa stať, že nejaké chyby nám potom analýza neodhalí, alebo nájde chyby, ktoré predtým neexistovali (falošné útoky).
Protokol sa štandardne zapisuje v Alice-Bob notácii. Príkladom jednej správy (jedného kroku protokolu) môže byť $A \rightarrow B : K$, kde prvý z účastníkov A posiela kľúč K druhému účastníkovi. Takýto zápis ale nie je vhodný pre automatizovanú analýzu. Vhodnejšie by bolo kľúč zapísať K_{AB}, ale pre formálnu analýzu sa zaužíval zápis $A \overset{K}{\longleftrightarrow} B$, ktorý umožňuje ľahšie rozlíšenie, ak takýchto kľúčov medzi dvoma účastníkmi je viacero.
Ďalším problémom sú dohadované kľúče, kedy sa neposiela priamo kľúč, ale len jeho časti. Samotný kľúč vznikne použitím funkcie na informácie, ktoré účastník mal a prijal. V tomto prípade funkcia musí spĺnať vlastnosť, že obaja účastníci, každý použitím svojej tajnej časti a prijatej verejnej časti kľúča dostanú ten istý výsledný kľúč. Pre formalizáciu treba popísať všetky funkcie, ktoré sú použité, pričom treba určiť či dané funkcie sú jednosmerné alebo existuje inverzná funkcia vzhľadom na niektorú premennú.
Ciele protokolov sú rôzne. V niektorých sa dôraz kladie na overenie doručenia ku konkrétnemu účastníkovi, v iných zase k nemožnosti prečítania počas posielania. Pri formalizácii treba stanoviť aké ciele má dosiahnuť protokol. Pri bezpečnostných protokoloch sú obvyklými cieľmi sú ciele typu $A \models A \overset{K}{\longleftrightarrow} B$, tzn. účastník A verí, že kľúč K je symetrickým kľúčom medzi účastníkmi A a B. Zosilnením požiadavky môže byť cieľ $B \models A \models A \overset{K}{\longleftrightarrow} B$, teda účastníci navzájom veria, že symetrický kľúč je použiteľný pri komunikácii medzi nimi.

3 Analýza protokolu

Počas behu jedného protokolu sa posielajú správy, teda vymieňajú informácie. Analýza spočíva v zisťovaní aké informácie mám v okamihu prijatia správy a aká správa prišla. Na základe týchto údajov sa snažíme odvodiť čo môže ďalej pokračovať. Najprv rozoberieme analýzu jedného behu protokolu, a potom sa budeme zaoberať paralelizmom protokolov. Odvodzovanie sa najčastejšie uskutočňuje zhora nadol alebo zdola nahor, ale k zisteniu slabých miest v protokole, ktoré by mohli viesť k útoku je možneé použiť aj kombinovanú metódu DLA [7].

3.1 Samotný protokol

Uvažujme protokol Needham-Schroeder Public Key (použitie asymetrického šifrovania):

1. $A \rightarrow B : \{A, N_A\}_{PK(B)}$
2. $B \rightarrow A : \{N_A, N_B, B\}_{PK(A)}$

3. $A \rightarrow B : \{N_B\}_{PK(B)}$

Z príkladu vidno, že úlohy účastníkov nie sú rovnaké (A je iniciátorom). V praxi sa ale častokrát v komunikácii stáva, že chvíľu je iniciátorom jeden účastník, chvíľu druhý. Pre model protokolu preto požijeme zápis $NSPK(Alice, Bob)$, presnejšie budeme mať dve triedy účastníkov $Initiator(Alice)$ a $Responder(Bob)$. Oproti analýze logikou viery [4] takto budeme vedieť odhaliť ďalšie možné útoky. Triedu $Initiator$ budeme reprezentovať stavom:

$$\begin{aligned} Initiator(A) \triangleq\ & \Box B : Agent \bullet env.A.B \rightarrow \\ & \Box N_A : Nonce_F \bullet send.A.B.\ \{A, N_A\}_{PK(A)} \rightarrow \\ & \Box N_B : Nonce \bullet recieve.B.A.\ \{N_A, N_B, B\}_{PK(A)} \rightarrow \\ & \quad send.A.B.\ \{N_B\}_{PK(B)} \rightarrow \\ & \quad close.A.Initiator \rightarrow Initiator(A) \end{aligned}$$

Znakom $\Box$ oddeľujeme jednotlivé stavy, v ktorých sa môže účastník nachádzať, sú to vlastne správy prijaté alebo odoslané daným účastníkom. Pri samotnej analýze treba rozlišovať čo je čerstvé (práve vytvorené), v príklade N_A. Akonáhle účastník prijíma správu, ktorá obsahuje jemu známy údaj, prebieha kontrola či nedošlo k zmene. V uvedenom príklade sa pri *recieve* overuje, či prijaté N_A je totožné s tým, čo bolo vytvorené v predchádzajúcom stave (v predchádzajúcej správe).

3.2 Paralelizmus protokolov

Pod pojmom paralelizmus protokolov sa rozumie vykonávanie viacerých inštancií jedného protokolu súbežne, prípadne komunikácia vo viacerých protokoloch naraz. Poradie správ v rámci jedného protokolu sa zachováva, ale správy dvoch inštancií sa môžu vyskytovať v ľubovoľnom poradí.
Príkladom protokolu, ktorý sa ukázal ako nie bezpečný, je protokol Yahalom, ktorého cieľom je dohodnúť kľúč K_{AB} na komunikáciu medzi dvoma účastníkmi (pričom kľúč generuje dôveryhodný server):

1. $A \rightarrow B : N_A$
2. $B \rightarrow S : N_B, \{A, N_A\}_{K_{BS}}$
3. $S \rightarrow A : N_B, \{B, K_{AB}, N_A\}_{K_{AS}}$
4. $S \rightarrow B : N_A, \{A, K_{AB}, N_B\}_{K_{BS}}$
5. $A \rightarrow B : \{N_B\}_{K_{AB}}$

Na prvý pohľad vyzerá v poriadku, komunikuje sa s dôveryhodným serverom S. Ale, čo sa stane, ak príde útočník I sťaby muž v strede (man-in-the-middle attack)? Čo ak namiesto protokolu Yahalom(A,B) sa vykoná Yahalom(A,I) a súčasne útočník inicializuje paralelne inštanciu protokolu Yahalom(I, A)? Zľava budeme písať správy prvej inštancie, odsadené budú správy druhej inštancie (sú aj odlíšené indexom), pri utočníkovi I v indexe je uvedené za ktorého účastníka sa vydáva:

$$1_I.A \rightarrow I_B : N_A$$

$$1_{II}.I_B \rightarrow A : N_A$$

$$2_{II}.A \rightarrow I_S : N_B, \{B, N_A\}_{K_{AS}}$$
$$2_I.I_A \rightarrow S : N_A, \{B, N_A\}_{K_{AS}}$$
$$3_I.S \rightarrow A : N_A, \{A, K_{AB}, N_A\}_{K_{BS}}$$
$$4_I.S \rightarrow I_B : N_A, \{B, K_{AB}, N_A\}_{K_{AS}}$$
$$3_{II}.I_S \rightarrow A : N_B, \{B, K_{AB}, N_A\}_{K_{AS}}$$
$$5_I.A \rightarrow B : \{N_B\}_{K_{AB}}$$

Po skončení, si účastník A myslí, že komunikoval s účastníkom B, ale ten sa v skutočnosti ani raz nezapojil do komunikácie. Útočník môze v ďalšej komunikácii vystupovať ako účastník B.

Ciele protokolov sa ešte dopĺňajú (tak ako pri bezpečnostných protokoloch sú to vzájomné viery) tak, aby zaručovali, že ak jeden z účastníkov dokončil protokol až do konca, tak aj všetci ostatní. Inak by sa ľahko mohlo stať, že raz zaplatíte v internetovom obchode za tovar a z účtu vám to neodrátajú, ale takisto by sa mohlo stať, že vám to odrátajú viackrát. To všetko, ale závisí od druhu protokolu, teda dôležitá je správna formalizácia (ako už bolo spomínané).

4 Veľkosť analyzovaného priestoru

Samotná analýza pozostáva z otestovania všetkých možností, či útočník nevie dosiahnuť cieľ, ktorý by mali dosiahnuť len ozajstný účastníci protokolu. Pri takejto analýze sa veľkým problémom stáva veľkosť analyzovaného priestoru. Na redukciu tohoto priestoru sa používa niekoľko rôznych techník.

4.1 Redukcia a falošné útoky

Snahou redukcie je zmenšenie stavového priestoru pri zachovaní tých istých vlastností. Na dôkaz bezpečnosti musíme prehľadať všetky možnosti. Počet týctho možností rastie exponenciálne vzhľadom k počtu stavov. Uvažujme protokol P a jeho redukci H. Redukciou je snaha zoskupiť informácie rovnakého typu (zobrazenie do jedného reprezentatívneho prvku). V analýze potom znalosť jednej informácie znamená znalosť všetkých informácií tohto typu. Znížime tak prehľadávaný stavový priestor, so zachovaním funkčnosti, ale redukcia môže vnášať falošné útoky.

Predstavme si, že účastník vlastní kľúč K_1. Majme správu m zašifrovanú kľúčom K_2, ktorý účastník nepozná. To znamená, že účastník nie je schopný rozlúštiť z $\{m\}_{K_2}$ pôvodnú správu. Zoberme si zobrazenie H (redukcia je zobrazenie) také, že $H(K_1) = H(K_2) = K_3$. Zašifrovaná správa po zobrazení je $H(\{m\}_{K_2}) = \{m\}_{K_3}$, účastník pozná $H(K_1) = K_3$, teda správu vie rozlúštiť. Takže pri každej redukcii je treba dbať na to, aby nevnášala žiadne falošné útoky.

4.2 Funkcia asymetrického kľúča

Pri používaní dôveryhodného centra sa môžu použiť dva zápisy pre kľúče. Buď každý účastník si pamätá svoje heslo s dôveryhodným centrom a samotné centrum si pamätá všetky heslá (tak ako je to pri symetrických kľúčoch, kde si

každý z dvojice musí pamätať kľúč), alebo sa vytvorí funkcia $f : A \Rightarrow K$ ako zobrazenie z účastníkov ku kľúčom (kľuč slúžiaci na komunikáciu s centrom). Potom si každý účastník pamätá jednu konkrétnu hodnotu tejto funkcie a centrum pozná funkciu (t.j. všetky jej hodnoty). V analýze ale takéto poznanie funkcie je výhodnejšie hlavne, ak je účastníkov viacero, a tí si môžu meniť svoje úlohy navzájom.

4.3 Predpočítanie predchádzajúcich inštancií protokolu

Ďalším spôsobom redukcie je predpočítanie údajov, niečo ako dynamické programovanie. Útočník môže použiť všetky informácie, ktoré sa vyskytli počas predchádzjúcej inštancie protokolu, tak mu ich na začiatku stanovíme ako informácie, ktoré pozná už hneď na začiatku. Takto nebudeme musieť analýzu pustiť dva krát, prvý krát pre získanie informácií pre útočníka, druhý krát pre ich využitie k útoku. Takisto v tomto kroku môžeme zozbierať informácie, ktoré vzniknú výmenov úloh medzi účastníkmi. Ale ani tak nám to nebude stačiť, aby sme vždy dokázali potom spracovať len jedným prechodom. Treba analýzu doplniť aj o ďalšieho (ďalších) účastníka. Predstavme si, že máme protokol medzi účastníkmi A a B. Nie je možné získať informácie, ak budem poznať komunikáciu medzi A a C a medzi B a C? Pri tejto redukcii nevznikajú falošné útoky, ale môže sa stať že nejaké útoky sa stratia. Táto redukcia je dokazateľná len do určiteho stupňa paralelizmu (počet úačstníkov, počet súbežne bežiacich inštancií).

4.4 Jednotné generovanie čerstvých údajov

V skutočnosti každý účastník si pamätá čo je čerstvé (teraz platné) a čo nie. Lenže pri väčšom počte účastníkov, počte rôznych rolí, je takýto spôsob náročný. Vytvorením centra, ktoré bude zodpovedné za všetky tieto údaje (nonce), tiež dokážeme redukovať stavový priestor. Toto centrum pri žiadosti o nový čerstvý údaj, ho vytvorí, a pri poslednom použití automaticky vyradí zo systému.

Použitím týchto redukcií pri protokole Yahalom je možné dosiahnuť nasledujúce veľkosti stavového priestoru:

External Server, bez predpočítania	222.104
Internal Server, bez predpočítania	12.977
External Server, s predpočítaním	1.024
Internal Server, s predpočítaním	249

5 Záver

Ukázali sme, ako je možné formálne analyzovať protokol. Využitím viacerých redukčných techník sa nám podarilo znížiť stavový priestor o niekoľko rádov. Pokračovať by sme chceli tak, aby bolo možné zachytiť aj ďalšie typy útokov - útoky na dostupnosť, type flaws (pretypovanie).

Literatúra

1. M.Burrows, M.Abadi, R.Needham, *A logic of authentication*, ACM Transaction on Computer Systems 8, February 1990
2. Elton Saul, *Facilitating the Modelling and automated Analysis of cryptografic Protocols*, Data Network Architectures Laboratory, Department of Computer Science, University of Cape Town, South Africa, 2001
3. Philippa J. Broadfoot, *Data Independence in the Model Checking of Security Protocols*, University of Oxford, 2001
4. Rastislav Krivoš-Belluš, *Formálna analýza bezpečnosti protokolu IKE*, ITAT 2002
5. Catherine A. Meadows, *Analyzing the Needham-Schroeder Public Key Protocol: A Comparison of Two Approaches*, Fourth European Symposium on Research in Computer Security, ESORICS '96
6. Li Gong, Roger Needham, Raphael Yahalom. *Reasoning about Belief in Cryptographic Protocols*, Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy
7. Eva Jenčušová, Jozef Jirásek, *Formal Methods of Analysis of Security Protocols*, Tatra Mountains Math. Publications, 2002
8. Rastislav Krivoš-Belluš, *Parallelism in Formal Analysis*, Tatracrypt '03