Doktorandský den ’05

Ústav informatiky
Akademie věd České republiky

Hosty – Týn nad Vltavou

5. – 10. říjen 2005

vydavatelství Matematicko-fyzikální fakulty
University Karlovy v Praze

Publikaci ”Doktorandský den ’05” sestavil a připravil k tisku František Hakl
Ústav Informatiky AV ČR, Pod Vodárenskou věží 2, 182 07 Praha 8

ISBN *– not yet –*

Obsah

Odhadování struktury dat pomocí pravidlových systémů

doktorand:
ING. MARTIN ŘIMNÁČ
Ústav informatiky,
Pod Vodárenskou věží 2,
Praha 8
rimnacm@cs.cas.cz

školitel:
ING. JÚLIUS ŠTULLER, CSC
Ústav informatiky,
Pod Vodárenskou věží 2,
Praha 8
stuller@cs.cas.cz

obor studia:
Databázové systémy
číselné označení: II

Práce byla podpořena projektem 1ET100300419 programu Informační společnost "Inteligentní modely, algoritmy, metody a nástroje pro vytváření sémantického webu" a částečně i výzkumným záměrem AV0Z10300504 "Informatika pro informační společnost: Modely, algoritmy, aplikace".

Abstrakt

Metoda odhadování struktury dat spojuje vizi sémantického webu a dnešní webové datové zdroje, které převážně neobsahují žádnou doprovodnou sémantiku prezentovaných informací. Aby bylo možné tyto zdroje použít pokročilými nástroji sémantického webu, je potřeba sémantiku prezentovaných dat alespoň odhadnout. Příspěvek popisuje takovou metodu, ukazuje její použití pro úlohy induktivního logického programování a jmenuje výhody použití pravidlových systémů pro její implementaci.

Jednou z klíčových otázek dnešní doby je zpřístupnění informací nejen lidem, ale i výpočetním prostředkům, které by na dotaz uživatele získaly potřebné informace a provedly by na jejich základě potřebné úkony (integrace z více zdrojů, odvozování informací, zahrnutí předdefinovaných uživatelských profilů nebo preferencí) tak, aby uživatel získal informaci ve zkompletované, přehledné formě.

Možnou odpovědí na tyto otázky je vize sémantického webu [1], tedy rozšíření současných webových zdrojů o dokumenty navíc vhodné ke strojovému zpracování. Tyto dokumenty by vedle samotné informace obsahovaly i její popis - sémantiku, což by umožnilo strojové zacházení s těmito dokumenty. V dnešní době se jeví jako perspektivní formát RDF (Resource Description Framework) [2], popisující realitu pomocí binárních predikátů $vlastnost(objekt, subjekt)$ s návazností na deskripční logiky jako odvozovacího mechanizmu, nebo formát OWL (Web Ontology Language) [3], mapující realitu pomocí vztahů mezi množinami.

Avšak vybrané typy současných webových zdrojů (např. webová rozhraní pro databáze) nemusí být ochuzeny o možnosti sémantického webu, neboť i ony v jistém smyslu obsahují sémantiku, byť v nějaké implicitní formě. Takovou formou může být např. tabulka nebo poloha hodnoty v šabloně designu webové stránky (u XHTML stránek lze data extrahovat pomocí XPath dotazů) apod. Z takové formy je možné odhadnout strukturu dat (např. relační model známý z teorie databází) a z tohoto modelu následně i sémantiku. Informace ze zdrojů pak mohou být extrahovány a uloženy buď do databází nebo do formátů vhodnějších pro sémantický web. Začlenění dat takových dokumentů do portfolia sémantického webu je možné řešit integrací dokumentů sémantického webu.

Příspěvek popisuje jednu takovou metodu [4] odhadu, na jejímž vstupu je tabulka dat s označenými sloupci a výstupem je relační model dat. Metoda byla nejprve implementována pomocí uložených procedur v databázi Postgres [5], které ale většinou představovaly pravidla (Když-Pak). Z tohoto důvodu byla zvolena ještě implementace v pravidlovém systému Clips [6] popsaná v samostatném odstavci.

1. Odhadování struktury dat

Odhadování struktury dat vychází z metod dekomponujících databázový model [7, 8, 9]. Úlohou těchto metod je upravit vstupní model popsaný pomocí množiny funkčních závislostí na nový model tak, aby splňoval další požadavky, např. vyšší normální formu či automatické rozšíření modelu o další vlastnosti, např. modely označované jako multi-level secure [10]. Výstupem těchto metod je model popsaný množinou funkčních závislostí. Aby byl výčet dekompozičních metod kompletní, uveďme ještě metody označované jako vertical a horizontal partitioning [11, 12], sloužící k dekompozici modelu s ohledem na paralerizaci přístupu k datům.

Na rozdíl od výše uvedených přístupů, metoda odhadování struktury dat [4] získává model pouze z dat, vstupem metody je množina tabulka dat a výstupem je model, minimální množina funkčních závislostí platných na množině vstupních dat.

Metoda je primárně vyvíjena jako doplněk současných web-miningových metod [13]. Ty operují především nad metadaty webových stránek a zprostředkovávají o nich souhrnné informace. Navrhovaný doplněk rozšiřuje tyto metody o extrakci samotných prezentovaných dat a podrobuje je analýze a další agregaci.

1.1. Základní vlastnosti funkčních závislostí

V tomto odstavci zopakujeme některé základní vlastnosti známé z teorie relačních databází.

Pokud dva atributy jsou vzájemně funkčně závislé, mají shodnou velikost aktivních domén.

$$A_1 \rightarrow A_2 \wedge A_2 \rightarrow A_1 \Rightarrow \|\mathscr{D}_\alpha(A_1)\| = \|\mathscr{D}_\alpha(A_2)\| \tag{1}$$

Množina funkčních závislostí vykazuje transitivitu, tedy:

$$A_1 \rightarrow A_2 \wedge A_2 \rightarrow A_3 \Rightarrow A_1 \rightarrow A_3 \tag{2}$$

Funkční závislost mezi atributy může existovat pouze v případě, kdy velikost aktivní domény závislého atributu není větší nežli velikost aktivní domény atributu, na němž závisí.

$$A_1 \rightarrow A_2 \Rightarrow \|\mathscr{D}_\alpha(A_1)\| \geq \|\mathscr{D}_\alpha(A_2)\| \tag{3}$$

Triviální funkční závislosti jsou ty, které nepopisují vlastnosti modelu, platí nezávisle na něm. Mezi ně patří např.

$$\begin{aligned} A_i &\rightarrow A_i \\ A_i &\rightarrow \oslash \end{aligned} \tag{4}$$

Komplexní atribut slučuje několik atributů v jeden celek. Pokud (komplexní) atribut H funkčně závisí na (komplexním) atributu G, funkční závislost atributu H na atributu G rozšířeném o libovolný další atribut je triviální.

$$G \rightarrow H \Rightarrow G' \rightarrow H \quad \forall G' \supset G \tag{5}$$

1.2. Algoritmus odhadu struktury dat

Mějme množinu vstupní tabulku dat (relaci) o n sloupcích a hledejme minimální množinu funkčních závislostí, která daná data popisuje.

Algoritmus inicializujeme prvním prvkem množiny dat. V této chvíli můžeme hovořit o modelu obsahujícím n^2 funkčních závislostí $A_j \rightarrow A_i$, budeme-li uvažovat též komplexní atributy, pak model pokrývá $n!$ (i triviálních) funkčních závislostí.

Výstupem algoritmu je minimální množina funkčních závislostí, `kostra modelu`, reprezentující elementární vazby v modelu. Taková množina neobsahuje žádné triviální funkční závislosti (4, 5) a obsahuje pouze ty funkční závislosti, které tvoří jádro množiny všech netriviálních funkčních závislostí.

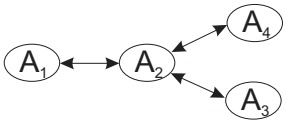
Hledání takového jádra vůči jejímu transitivnímu uzávěru je však NP úplná úloha [7, 14] a nemá jedinečné řešení. Proto navržený algoritmus vytvoří v prvním kroku triviálním způsobem kostru modelu a takto vytvořenou kostru v každém kroku aktualizuje, přičemž problém aktualizace je již polynomiálně řešitelný.

Povšimněme si, že po přidání prvního záznamu každý atribut je extensionálně funkčně závislý na každém jiném atributu (neexistuje žádný další záznam, který by takovou závislost porušoval), tyto závislosti jsou vzájemné. Diskutujme nyní, jaké jsou možné konfigurace kostry modelu a způsoby jejich odvození.

První způsob, `lineární kostra` na obrázku 1, spočívá v náhodném uspořádání atributů a umístěním všech orientovaných hran mezi sousedícími atributy do kostry modelu. Takových uspořádání je faktoriální počet, metoda však na něm dále nezávisí. Možnou nevýhodou je, že takováto kostra modelu obsahuje cykly délky až $2(n-1)$.

Obrázek 1: Kostra v lineární konfiguraci

Tuto potenciální nevýhodu odstraňuje druhý způsob, `star kostra` na obrázku 2, kdy je náhodně vybrán jeden z atributů a kostru modelu tvoří funkční závislosti mezi tímto atributem a ostatními. Je zřejmé, že každý cyklus nabývá buď délky 2 (vzájemná funkční závislost) nebo délky 4. Počet takových modelů je roven počtu atributů, tj. n. Možnou nevýhodou je větší počet změn v kostře při porušení funkční závislosti.



Obrázek 2: Kostra ve ”star” konfiguraci

Další možné konfigurace funkčních závislostí, které jsou kostrou, musejí být vytvořeny kombinací těchto dvou přístupů, vlastnosti takových koster se pohybují v rozmezí vlastností obou způsobů.

Přidejme nyní do úložiště další záznam. Předpokládejme, že některé atributy nabývají stejné hodnoty a některé hodnoty jiné. To podle (3) znamená, že bude porušena některá z funkčních závislostí.

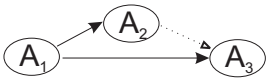
V okamžiku, kdy je do úložiště přidán další záznam, je nutné nejprve aktualizovat kostru modelu. Abychom zachovali daný požadavek ”elementárnosti vazeb v kostře”, definujme primární kritérium uspořádání atributů s ohledem na (3) tak, že

$$\|\mathscr{D}_\alpha(A_i)\| < \|\mathscr{D}_\alpha(A_j)\| \Rightarrow i < j \tag{6}$$

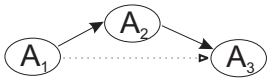
Během aktualizace kostry při změně pořadí atributů může dojít k situaci (7) nebo (8), kdy existuje kratší hrana (podle (7) obrázek 3), která je doplňkem hran tvořících kostru (délku hrany $\delta(A_i, A_j)$ reprezentuje rozdíl pozic v uspořádání atributů, S je množina funkčních závislostí tvořící kostru a $\overline{S}$ je transitivní uzávěr této množiny).

$$(A_1 \to A_3) \in S \wedge (A_1 \to A_2) \in S \wedge (A_2 \to A_3) \in \overline{S} \wedge \delta(A_1, A_3) > \delta(A_2, A_3) \tag{7}$$
$$(A_1 \to A_3) \in S \wedge (A_2 \to A_3) \in S \wedge (A_1 \to A_2) \in \overline{S} \wedge \delta(A_1, A_3) > \delta(A_1, A_2) \tag{8}$$

Jak ukazuje obrázek 4, tato hrana se pak stává hranou tvořící kostru na úkor delší z hran.



Obrázek 3: Kostra $\delta(A_1, A_3) > \delta(A_2, A_3)$

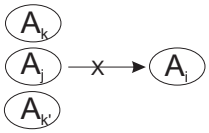


Obrázek 4: Kostra po aktualizaci

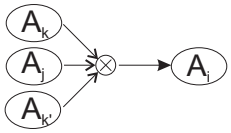
Předpokládejme, že kostra modelu je aktualizována. Otestujme nyní, zda-li některé funkční závislosti nejsou porušeny.

Protože kostra tvoří jádro množiny, pokud je porušena jakákoli funkční závislost v modelu, musí být porušena některá z funkčních závislostí $A_j \rightarrow A_i$ tvořících kostru. Díky tomuto faktu není nutné testovat při každém kroku všechny funkční závislosti, ale pouze ty, které tvoří kostru (takových je nejvýše $2n$). V případě, kdy je taková funkční závislost porušena, je nutné navíc testovat i funkční závislosti se stejnou stranou a aktualizovat kostru (najít jiné funkční závislosti s minimální délkou spojující atributy původně spojené přes A_i a A_j).

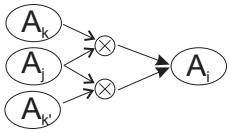
Přidejme další záznamy a předpokládejme, že při testování funkčních závislostí v aktuálním kroku je porušena závislost $A_j \rightarrow A_i$, přičemž v minulosti byly porušeny i funkční závislosti $A_k \rightarrow A_i$ a $A'_k \rightarrow A_i$ a mezi atributy A_k, A'_k a A_i neexistuje žádná funkční závislost. To může vést ke vzniku netriviální závislosti na komplexním atributu $\{A_j, A_k, A'_k\}$, viz obrázek 5. Vložme tedy tento `komplexní atribut` pomocí



Obrázek 5: Porušení $A_j \rightarrow A_i$



Obrázek 6: Virtuální atribut



Obrázek 7: Dekompozice

notace `virtuálního atributu` do modelu, virtuální atribut reprezentující kartézský součin je použit, aby nebylo nutné úlohu řešit v prostoru hypergrafů, a otestujme funkční závislost $\{A_j, A_k, A'_k\} \rightarrow A_i$. Pokud tato funkční závislost je splněna, virtuální atribut je v modelu ponechán - obrázek 6.

U komplexních atributů arity větší než 2 je navíc pomocí dekompozice komplexního atributu nutné testovat (obrázek 7), zda-li daná závislost není triviální (5). Pokud je, virtuální atribut je dekomponován. Všechny získané netriviální funkční závislosti z této operace jsou přidány do kostry modelu a model je rozšířen i o další neporušené funkční závislosti mezi novými virtuálními atributy a ostatními atributy v modelu.

Jak je patrné, vznik netriviální funkční závislosti na komplexním atributu o m atributech je možný za podmínky, že existuje $m \leq m'$ (alespoň m z celkových m') porušených funkčních závislostí na atributu A_i, který bude na hledaném komplexním atributu závislý a které nemají mezi sebou žádné jiné funkční závislosti. Operace hledání komplexního atributu je nepolynomiálně složitá, je potřeba otestovat celkem až k kombinací (dekompozice atributu), přičemž (při uvažování celočíselného dělení)

$$k = \max_{\forall i < m'} \left(\begin{array}{c} m' \\ i \end{array} \right) = \left(\begin{array}{c} m' \\ m'/2 \end{array} \right) = \frac{m'!}{(m'/2)!^2} \tag{9}$$

Jedinou možností, kdy bude hledání komplexního atributu polynomiálně řešitelné, je díky (3) případ, kdy velikost aktivní domény komplexního atributu roste rychleji než velikost aktivní domény atributu na komplexním atributu závislém.

$$(A \rightarrow B),\ A = \{A_0 \ldots A_m\} : \mathscr{D}_\alpha(A) \geq \prod_{i=0}^{m} \mathscr{D}_\alpha(A_i) > \mathscr{D}_\alpha(B) \tag{10}$$

To je ovšem velmi speciální případ, v praxi nastávající zřídkakdy. Příkladem může být tabulka popisující funkční závislost paritního bitu na vzoru. V případě, že je možné vstupní data předzpracovat, je vhodné je uspořádat podle takového kritéria, čímž se proces odhadování modelu urychlí (např. udržovat seznam vhodných trénovacích příkladů pro opakovaný odhad modelu).

Připomeňme, že ostatní operace v metodě jsou vždy polynomiálně složité, tedy hledání netriviálních funkčních závislostí komplexních atributů je jedinou operací, které z celé metody činí nepolynomiálně složitý algoritmus.

Obecně existují dva přístupy ke hledání netriviálních funkčních závislostí nových komplexních atributů. První vychází ze složení komplexního atributu s aritou m a jeho postupné dekompozici na jednodušší komplexní atributy (viz obrázek 6, 7). Naopak druhý přidává další atributy k atributu, na němž byla funkční závislost porušena, tak dlouho, dokud není dosaženo funkční závislosti nebo arity m'. Oba přístupy vedou ke shodnému výsledku, vhodnost použití může být dána heuristikou odvíjející se od hodnoty m'.

Algoritmus 1

```
A .. množina atributů
D .. množina dat
C .. matice pokrytí
S .. kostra modelu
C = {c_ij = 1, i ≠ j, ∀i, j ∈ 1..|A|}
S = {A_i → Aj : |i − j| = 1}
Pro ∀d ∈ D
{
    ulož d do úložiště
    aktualizuj velikost domén a podle změn i pořadí atributů v S a C
    dokud (zmena)
   {
      ∀(i, j, k), i < k < j : (A_i → A_j) ∈ S ∧ (A_k → A_j) ∈ S ∧ C_ik = 1
        S = S − {A_i → A_j} ∪ {A_i → A_k}
      ∀(i, j, k), i < k < j : (A_i → A_j) ∈ S ∧ (A_i → Ak) ∈ S ∧ C_kj = 1
        S = S − {A_i → A_j} ∪ {A_k → A_j}
    }
    testuj ∀(A_i → A_j) ∈ S
    {
      pokud A_i → A_j porušeno
      {
       testuj ∀(A_u → A_v), kde C_iv = 1 ∨ C_uj = 1
       pokud porušeno, C_uv = 0
      }
      S = S − (A_i → Aj)
      rozšiř kostru
      testuj komplexní atributy
    }
}
```

1.3. Vlastnosti modelu

Uspořádejme modely podle počtu všech funkčních závislostí pokrytých modelem. Označme n počet atributů, m počet záznamů, M_k pak model po k-tém záznamu a $|M_k|$ počet funkčních závislostí pokrytých modelem M_k. Podle algoritmu 1 počet hran monotónně klesá, tedy

$$|M_0| = n! \tag{11}$$

$$M_i < M_j \Rightarrow |M_i| > |M_j| \tag{12}$$

$$\forall M_k : M_0 \leq M_k \leq M_m \leq M_\infty \tag{13}$$

Označíme-li M_∞ nejpřesnější (logický) model reality, poslední nerovnost značí, že model vrácený algoritmem může oproti tomuto modelu obsahovat navíc některé funkční závislosti. Tento rozdíl může být způsoben nereprezentativními daty (jedná se o algoritmus strojového učení, kde reprezentativnost dat hraje marginální roli), jednak granularitou hodnot domén jednotlivých atributů (konečný počet záznamů versus nespočitatelné domény atributů v realitě) spojenou s uvažováním extensionálních funkčních závislostí. Poslední možností je možná závislost dat na zdroji (jak samotných hodnot, tak struktury dat). Z těchto důvodů hovoříme o odhadu struktury, nikoli o její rekonstrukci. První a poslední problém lze vyřešit integrací dat z více zdrojů, druhý pak představuje principiální limit metody.

Metoda je určena pro deterministická data neobsahující chybné příklady a předpokládá konzistenci dat v rámci každého zdroje. Pokud tyto podmínky nejsou splněny, mohou být některé funkční závislosti obsažené v logickém modelu z kostry vyjmuty (existují, byť chybné, záznamy porušující tyto závislosti). To může vést k situaci, kdy

$$M_\infty < M_m \tag{14}$$

Výhodou odhadování struktury dat je nezávislost výsledného modelu na pořadí vstupních dat.

1.4. Příklad

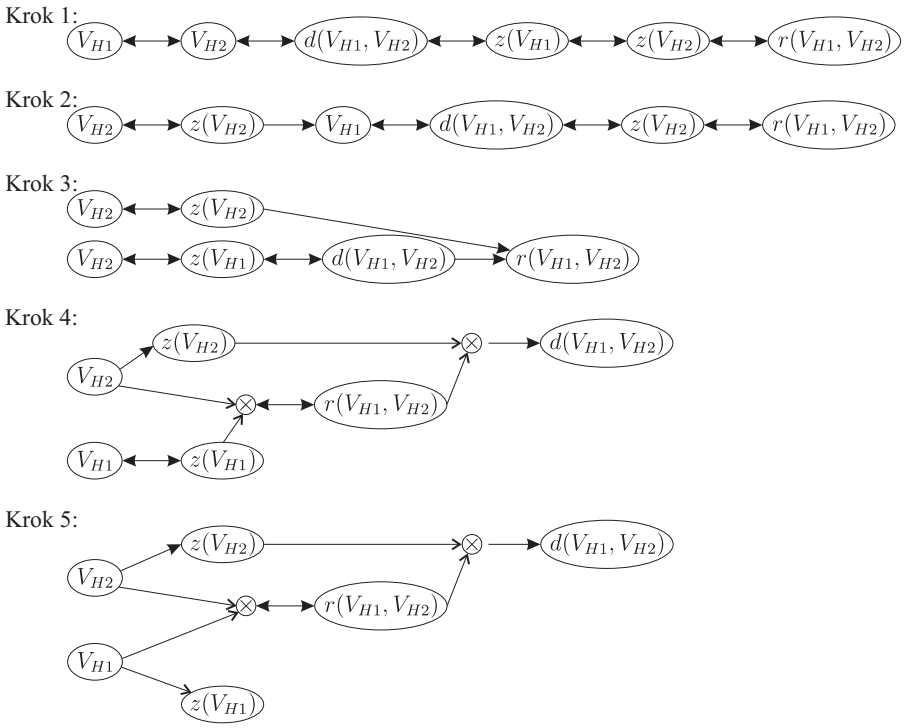
Některé jednoduché úlohy induktivního logického programování lze převést na úlohu odhadování struktury dat [15]. Úlohou induktivního logického programování [16, 17] je najít interpretaci predikátu na základě dat ve znalostní bázi. Příkladem takových algoritmů je GOLEM [18] (využívající postupnou generalizaci), FOIL [19, 20] (postupná specializace), inverzní metody (PROGOL, ALEPH) a další.

Ilustrativní příklad je právě z tohoto prostředí, hledá koncept predikátu $d(V_{H1}, V_{H2})$ popisující skutečnost, že objekt V_{H1} je dcerou objektu V_{H2}. Znalostní báze v tabulce (15) obsahuje predikát $r(V_{H1}, V_{H2})$ (objekt V_{H2} je rodičem objektu V_{H1}) a predikát $z(V_{H1})$ (objekt V_{H1} je ženského pohlaví).

n	V_{H1}	V_{H2}	$d(V_{H1}, V_{H2})$	$z(V_{H1})$	$z(V_{H2})$	$r(V_{H1}, V_{H2})$	$r(V_{H2}, V_{H1})$	$r(V_{H1}, V_{H1})$	$r(V_{H2}, V_{H2})$
1	Eva	Tomáš	$\oplus$	$\oplus$	$\ominus$	$\oplus$	$\ominus$	$\ominus$	$\ominus$
2	Eva	Kamila	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\ominus$	$\ominus$	$\ominus$
3	Milan	Tomáš	$\ominus$	$\ominus$	$\ominus$	$\oplus$	$\ominus$	$\ominus$	$\ominus$
4	Eva	Milan	$\ominus$	$\oplus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$
5	Karel	Milan	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$

(15)

Následující obrázek 8 ukazuje vývoj modelu po každém přidání řádku z tabulky (15) podle algo-



Obrázek 8: Ilustrativní příklad

ritmu 1. Je patrné, že koncept predikátu $d(V_{H1}, V_{H2})$ je odhadnut již ve 4. kroce. Model ve smyslu teorie relačních databází můžeme interpretovat tak, že pravdivostní ohodnocení predikátu $d(V_{H1}, V_{H2})$ je funkčně závislé na ohodnocení predikátu $z(V_{H2})$ a $r(V_{H1}, V_{H2})$. Z hlediska modelování struktury dat nezáleží na konkrétních hodnotách ohodnocení, hledáme globální obecný popis vlastností mezi býti-dcerou, býti-rodičem a býti-ženou.

Krok 5 pak ještě doladí koncept predikátu $z(V_{H1})$. Poznamenejme, že model v této fázi žádným způsobem nereflektuje fakt, že různé atributy $z(V_{H1})$ a $z(V_{H2})$ reprezentují tentýž predikát jen s jinou kombinací argumentů.

Použitá vstupní data byla volena tak, aby byla velmi reprezentativní, díky čemuž celý model byl přesně detekován v malém počtu kroků. To ale v praxi u obecných zdrojů nemusí platit.

2. Implementace pomocí pravidlového systému

Metoda byla v první fázi implementována jako soubor uložených procedur v databázovém systému Postgres [5] (verze 7.4). Základní funkčnost byla zajištěna pomocí triggerů. S rostoucím stupněm implementace se však tento způsob stával neschůdným, neboť u metody nejenže závisí na pořadí volání dílčích operací, ale některé z nich pracují již s aktualizovanými údaji a jiné, po nich následující, s údaji před začátkem aktualizace. Z těchto důvodů se stávalo použití databázového systému těžkopádné a SQL dotazy příliš složité. Většina z nich navíc implementovala vztahy "je-li splněna podmínka, změň data", tedy prakticky pravidla.

Druhým problémem byla nutnost uchovávat i dočasná data v tabulkách. To lze sice vyřešit použitím dočasných tabulek, avšak toto řešení výrazně zpomaluje start aplikace.

Díky těmto dvěma aspektům se začal rýsovat požadavek na hledání alternativních, lépe vyhovujících, způsobů implementace. Následně byl zvolen přechod na implementaci metody pomocí pravidlového systému.

Jako vhodný kandidát byl posléze zvolen Clips [6] (verze 6.23). Hlavním požadavkem byla možnost dynamicky měnit znalostní bázi (funkční závislosti se postupně odebírají, vznikají jejich nové instance, vzájemně se transformují). Další výhodou tohoto systému oproti jiným je přímá návaznost na souborový systém (funkce pro práci se soubory) a dynamické načítání znalostní báze, což v budoucnu umožní paralelní zpracování a uvažování pouze pro výpočet nutných funkčních závislostí a jejich instancí.

Další výhodou je multiplatformita a dostupnost zdrojových kódů tohoto systému, což umožňuje některé složitější nebo v základní sadě neobsažené operace doprogramovat a z ní plynoucí rozšiřitelnost a modularita (např. standardně dostupný modul pro fuzzy pravidla).

2.1. Transitivita, aktualizace kostry

Pro ilustraci použití takového systému pro implementaci metody uveďme několik příkladů vybraných pravidel. Mezi základní operace metody patří aktualizace kostry.

Abychom mohli kostru aktualizovat, je nejprve nutné podle (2) vytvořit koncept transitivity funkčních závislostí.

Pravidlo 1 *Transitivita*

```
(defrule transitivity
           (fd skeleton $?leftside "=>" $?centerside )
           (fd ? $?centerside "=>" $?rightside )
=>
       (assert
           (fd derived $?leftside "=>" $?rightside )))
```

Nadefinujme ve znalostní bázi fakt $(attribute\ order\ \{\ A_1\ \}\ \ldots\ \{\ A_n\ \})$ udávající pořadí atributů a fakt $(attribute\ domaincount\ \{\ A_i\ \}\ |\mathscr{D}_\alpha(A_i)|\ )$ udávající velikost aktivní domény daného atributu. Aktualizace pořadí atributů po přidání nového záznamu do úložiště podle (6) obsahuje pravidlo:

Pravidlo 2 *Aktualizace pořadí atributů*

```
(defrule attribute-order-update
           ?order<-(attribute order $?left { $?attr1 } { $?attr2 } $?right )
           (attribute domaincount { $?attr1 } ?domaincount1)
           (attribute domaincount { $?attr2 } ?domaincount2)
=>
       (if (> ?domaincount1 ?domaincount2)
        (
         (assert
            (attribute order $?left { $?attr2 } { $?attr1 } $?right))
         (retract ?order )
        )))
```

Nyní již máme celý aparát pro aktualizaci kostry. Pravidlo pro případ aktualizace kostry (7) je:

Pravidlo 3 *Aktualizace kostry*

```
(defrule skeleton-update
          (attribute order $? { $?leftside } $? { $?centerside } $? { $?rightside } $? )
          (fd skeleton { $?leftside } "=>" { $?centerside } )
          ?s<-(fd skeleton { $?leftside } "=>" { $?rightside } )
          ?d<-(fd derived { $?centerside } "=>" { $?rightside } )
=>
      (retract ?s ?d)
      (assert
          (fd derived { $?leftside } "=>" { $?rightside } )
          (fd skeleton { $?centerside } "=>" { $?rightside } )))
```

Výhodou tohoto přístupu je jednoduchost pravidel a jejich zápisu, uložené pl/pgsql procedury se stejnou funkčností byly podstatně složitější (kód byl dlouhý řádově v jednotkách kB), k výpočtu optimality kostry (analogicky k pravidlu 3) bylo potřeba spojit v základní verzi celkem 6 tabulek, 3 tabulky obsahující množinu funkčních závislostí a 3 tabulky atributů udávající jejich pořadí (pokud uvažujeme verzi pro komplexní atributy a neredundantní uložení dat, toto spojení je rozšířeno o další 3 tabulky popisující, které atributy tvoří daný komplexní atribut). Navíc na tento dotaz bylo možné provést pouze jednu změnu (dílčí změna kostry ovlivňuje i další závislosti v kostře).

U pravidlového systému se jedná o nalezení 4 faktů ve znalostní bázi, problematický může být jen počet možných kombinací, který je ale u těchto systémů dobře zvládnutý. Druhou možnou kombinací je interpretace negace výskytu faktu, které opět může vést k velkému počtu přípusntných kombinací.

2.2. Úložiště

Naopak některé operace se při použití pravidlového systému nepatrně komplikují. Jednou z nich je operace přidávání relace do úložiště, kterou nelze provést přímo, ale oklikou přes vzor takové instance. Mějme pro každý atribut nadefinován fakt ve tvaru $(attribute\ pattern \{\ A\ \}\ \{\ [\ A\ \_\ ]\ \}\ )$ nebo v případě komplexního atributu $(attribute\ pattern \{\ A\ B\ \}\ \{\ [\ A\ \_\ ]\ [\ B\ \_\ ]\ \}\ )$ a vkládaný záznam rozdělený po atributech $(tuple\ attribut\ hodnota\ )$.

Vložení záznamu do úložiště představuje nejprve sestavení vzoru instance funkční závislosti:

Pravidlo 4 *Příprava vzoru instance*

```
(defrule prepare-instance
          ?p<-(fd skeleton { $?leftside } "=>" { $?rightside } )
          (attribute pattern { $?leftside } { $?leftpattern } )
          (attribute pattern { $?rightside } { $?rightpattern } )
=>
      (assert
          (rel ?p { $?leftpattern } "=>" { $?rightpattern } )))
```

V druhém kroku je pak tento vzor naplněn daty podle vkládaného záznamu.

Pravidlo 5 *Plnění daty*

```
(defrule data-fill
          ?r<-(rel ?p $?left [ ?attr _ ] $?right )
          (tuple ?attr ?value] )
=>
      (retract ?r )
      (assert
          (rel ?p $?left [ ?attr ?value ] $?right )))
```

Výsledkem je uložený předpis pro asociační pravidlo.

3. Shrnutí

Příspěvek volně navazuje na práci publikovanou v minulém ročníku Doktorandského dne [21]. Základní myšlenka řešení problematiky zůstává stejná, dílčím způsobem se modifikují cíle práce.

Oproti předchozímu ročníku není hlavní cíl spatřován ve fuzzifikaci funkčních závislostí, neboť porušení klasické funkční závislosti v reálných datech může být problematické (data nemusí být natolik reprezentativní, aby došlo k porušení závislosti), tudíž další oslabení této vlastnosti není žádoucí. Spíše tedy dochází ke kladení požadavků na vstupní data, předpokládají se deterministické hodnoty, bezchybnost a konzistence zdrojů dat a hledání alternativních procesů, např. problémy chybovosti se dají převést na známý problém hledání vyjímek asociativních pravidel.

Hlavním cílem naopak zůstává orientace na sémantický web, konkrétně odhad sémantiky ze struktury a popis integrace modelů. Díky možnosti použít pravidlových systémů pro implementaci metody a s tím související implementací jednoduchých pravidel převádějící data na jiné reprezentace či různé granularity primitivních vztahů (např. pravidla pro získávání metadat nebo jednoduchá kombinace různých typů vztah; při prohledávání úložiště). Tyto nové možnosti otvírají cestu k jednoduššímu popisu vztahů mezi atributy, zjednoduší se popis integračního procesu (bude postačující přidat nový typ vztahu mezi atributy a příslušné odvozovací pravidlo).

Novým cílem se stává hledání generalizovaného modelu umožňující odhadování hodnot atributů na základě vlastností zintegrované kostry několika modelů, principy integrace či hledání výjimek. Poměrně zajímavým tématem se jeví konstrukce metadat, konverze mezi různými interpretacemi informace (různá granularita atributů majících tentýž význam) nebo hledání primitivních částí informace a vztahů mezi nimi.

Za poměrně úspěšné lze považovat hledání metodických isomorfismů. Podařilo se ukázat, že některé základní úlohy induktivního logického programování, jejichž zadání splňuje požadavky metody odhadování struktury dat, lze na tuto metodu převést [15]. Tento převod buď lze udělat triviálním způsobem transformováním znalostní báze do tabulky pokrývající příslušné kombinace konstant, jak ukazuje příklad (15), nebo (existuje-li taková informace - ohodnocení predikátu je závislé na jeho proměnných nebo na ohodnocení jiných predikátů) znalostní bázi převést přímo do relačního modelu a počítat pouze závislosti kolem atributu reprezentujícího hledaný predikát.

Další oblastí je rozbor vlastností kostry a její různé konfigurace (viz obrázek 1, 2) a definice uspořádání atributů (6) a vliv tohoto uspořádání na kostru (viz obrázek 3, 4). Zde se jeví jako perspektivní přístup postupného (proudového) on-line zpracování dat. On-line zpracování je vhodné jak z hlediska pravidlových systémů, tak z hlediska možné paralerizace problematiky. Postupné zpracování vychází z možnosti polynomiálně složité iterační aktualizace kostry oproti NP úplnému hledání jádra z transitivního uzávěru.

V neposlední řadě došlo oproti [21] k rozšíření modelu o komplexní atributy a analýze problému vytváření komplexních atributů v modelu (obrázky 5, 6, 7) a hledání netriviálních funkčních závislostí kolem těchto atributů [4]. To je jedinou částí vykazující až na speciální případy (10) nepolynomiální složitost (9).

Metoda byla implementována původně jako uložené pl/pgsql procedury databázového systému pro variantu, kdy data jsou uložena redundantním způsobem v předem dané struktuře (prakticky reprezentující jednu relaci mezi zadanými atributy). Tato varianta pokrývá celou problematiku odhadu struktury dat. Varianta s daty ukládanými podle odhadnutého modelu je rozpracována, některé partie se ukázaly jako obtížně zvládnutelné korektním způsobem. Proto došlo k přerušení implementačních prací a migraci z databázového systému na systém pravidlový, který se pro implementaci metody v současné době jeví jako velmi perspektivní. V současné době probíhá implementace metody právě v tomto systému (viz pravidla 1 až 5).

Jelikož znalostní báze generovaná metodou na základě odhadnutého modelu je ve formě asociačních pravidel, bude zajímavé do takové báze přidat znalosti pocházející z dokumentů sémantického webu a sledovat vliv struktury těchto dokumentů na globální model. V budoucnu by měla být data z této znalostní báze včetně metadat přístupná přes webové rozhraní a mělo by být implementováno webové rozhraní pro vyhledávání dat ve znalostní bázi.

Literatura

[1] Grigoris Antoniou, Frank van Harmelen. "A Semantic Web Primer". MIT Press, 2004. ISBN: 0-262-01210-3.

[2] Eric Miller, Ralph Swick, Dan Brickley. "Resource Description Framework". <http://www.w3.org/RDF/> [on-line]. 2004.

[3] Eric Miller, Jim Hendler. "Web Ontology Language". <http://www.w3.org/2004/OWL/> [on-line]. 2005.

[4] Martin Řimnáč "Web Information Integration Tool - Data Structure Modelling". In *Proceedings of 2005 International Conference on Data Mining*. CSRea, USA. pp 37-40. ISBN 1-934215-79-3. 2005.

[5] Postgres. <http://www.postgres.org/> [on-line]. 2005.

[6] Clips. <http://www.ghg.net/clips/CLIPS.html> [on-line]. 2005.

[7] G. Grahme, K. Räihä, "Database Decomposition into Fourth Normal Form". In *Conference on Very Large Databases*. pp. 186–196, 1983.

[8] G. Ausiello, A. D'Atri, M. Moscarini, "Chordality Properties on Graphs and Minimal Conceptual Connections in Semantic Data Models". In *Symposium on Principles of Database Systems*. pp. 164–170. 1985.

[9] Bruno T. Messmer, Horst Bunke "Efficient Subgraph Isomorphism Detection: A Decomposition Approach". In *IEEE Transactions on Knowledge and Data Engineering*. pp.: 307-323. 2000.

[10] F. Cuppens, K. Yazdanian, "A Natural Decomposition of Multi-level Relations". In *IEEE Symposium on Security and Privacy*, pp. 273-284. 1992.

[11] B.N. Shamkant, R. Minyoung, "Vertical Partitioning for Database Design – a Graphical Algorithm". In *SigMod*, pp. 440–450, 1989.

[12] L. Bellatreche, K. Karlapalem, A. Simonet, "Algorithms and Support for Horizontal Class Partitioning in Obect–Oriented Databases". In *Distributed and Parallel Databases, 8*, Kluwer Academic Publisher. pp. 115–179, 2000.

[13] A.A.Barfourosh, M.L. Anderson, H.R.M.Nezbad, D Perlis. "Information Retrieval on the World Wide Web and Active Logic: A Survey and Problem Definition". In <http://citeseer.ist.psu.edu/barfourosh02information.html> [online]. 2002.

[14] C. Beeri, P.A.Bernstein, "Computational Problems Related to the Design of Normal Form Relation Schemes". In *ACM Transactions on Database Systems*. 4,1. pp 30-59. 1979.

[15] Martin Řimnáč, "Odhadování struktury dat a induktivní logické programování". In *ITAT 2005* . (V tisku). 2005.

[16] Nada Lavrač, Sašo Džeroski, "Inductive Logic Programming - Techniques and Applications". Ellis Hordwood, Chichester. ISBN: 0-13-457870-8. 1994.

[17] Sašo Džeroski, Nada Lavrač, "Relational Data Mining". Springer-Verlag, Berlin. ISBN: 3-640-42289-7. 2001.

[18] Muggeton, S., Feng, C., "Efficient introduction of logic programs". In *Proceeding of the First Conference on Algorithmic Learning Theory*. pp 368-381. 1990.

[19] Quilan, J., "Learning logical definitions from relations". In *Machine Learning*, 5(3). pp 239-266. 1990.

[20] Quilan, J., "Knowledge acquisition from structured data - using determinate literal to assist search". In *IEEE Expert*, 6(6). pp 32-37. 1991.

[21] Martin Řimnáč, "Rekonstrukce databázového modelu na základě dat (studie proveditelnosti)". In *Doktorandský den '04, Ústav informatiky AV ČR*. pp. 113-120. ISBN 80-86732-30-4. 2004.

Ústav Informatiky AV ČR
DOKTORANDSKÝ DEN '05

Vydal
MATFYZPRESS
vydavatelství
Matematicko-fyzikální fakulty
University Karlovy
Sokolovská 83, 186 75 Praha 8
jako svou *– not yet –* . publikaci

Obálku navrhl František Hakl

Z předloh připravených v systému LaTeX
vytisklo Reprostředisko MFF UK
Sokolovská 83, 186 75 Praha 8

Vydání první

Praha 2005

ISBN *– not yet –*