Odhad struktury dat a induktivní logické programování

Martin Řimnáč

rimnacm@cs.cas.cz
Ústav informatiky, Akademie věd ČR,
Pod Vodárenskou věží 2, 182 07 Praha 8

Abstrakt Odhadování struktury dat je jednou z možností, jak automatizovaným způsobem interpretovat data. Ta mohou být popsána pomocí modelu funkčních závislostí, vytváření takového modelu lze srovnat s některými technikami strojového učení. Tento příspěvek shrnuje vybrané základní techniky induktivního logického programování a analyzuje je z pohledu metody odhadování struktury dat. Ukazuje se, že techniky induktivního logického programování lze v některých případech převést právě odhadování struktury dat.

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.

Druhou možností je využití vybraných typů současných webových zdrojů, extrakce informace z nich pomocí web-minnigových metod a uložení těchto informací do databází. K tomu je však potřeba znát strukturu dat (alespoň na pokrytí požadavků na normální formu databázového modelu), přičemž lze využít dále nastíněné metody odhadování struktury dat [4]. Mezi takové zdroje dat patří především webová rozhraní pro databázové pohledy, např. elektronické obchody.

1 Odhadování struktury dat

Odhadování struktury dat vychází z metod dekomponujících databázový model [5–7]. Úlohou těchto metod je upravit vstupní model popsaný pomocí množiny funkčních závislostí na nový model tak, aby splňoval další požadavky jako

vyšší normální forma či automatické rozšíření modelu o další vlastnosti, např. modely označované jako multi-level secure [8]. Výstupem těchto metod je model popsaný množinou funkčních závislostí. Aby byl výčet dekompozičních metod kompletní, uveďme ještě metody označované jako vertical a horizontal partitioning [9, 10], sloužící k dekompozici modelu s ohledem na paralelizaci 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 [11]. 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 atributy jsou vzájemně funkčně závislé, mají shodnou velikost aktivních domén.

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

Množina funkční závislostí vykazuje transitivitu, tedy:

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

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.

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

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

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

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.

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

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 `kostra modelu`, minimální množina funkčních závislostí 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 tranzitivnímu uzávěru je však NP úplná úloha [5] 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 extenzionálně funkčně závislý na každém jiném atributu (neexistuje žádný další záznam, který by takovou závislost porušoval), tyto závislosti jsou vzájemné.

Přidejme nyní 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, kdy existuje kratší hrana, která je doplňkem hran tvořících kostru (délku hrany $\delta(A_i, A_j)$ reprezentuje rozdíl pozic v uspořádání atributů). V tomto případě se tato hrana stává hranou tvořící kostru na úkor delší z hran.

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 $A_j \rightarrow A_i$ tvořící 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 $\{A_j, A_k, A'_k\}$.

Vložme nový `komplexní atribut` jako `virtuální atribut` do modelu. Virtuální atribut reprezentující kartézský součin je použit, aby nebylo nutné úlohu řešit v prostoru hypergrafů. Otestujme funkční závislost $\{A_j, A_k, A'_k\} \rightarrow A_i$. Pokud je splněna, virtuální atribut je v modelu ponechán. Pokud arita komplexního atributu je větší než 2, testujeme pomocí dekompozice komplexního atributu, zda-li daná závislost není triviální (5). Všechny získané netriviální funkční závislosti jsou přidány do kostry modelu a model je rozšířen funkční závislosti z nového uzávěru.

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)

$$k = \max_{\forall i < m'} \binom{m'}{i} = \binom{m'}{m'/2} = \frac{m'!}{(m'/2)!^2} \tag{7}$$

Jedinou možností, kdy bude hledání komplexního atributu polynomiálně řešitelné, je případ, kdy velikost aktivní domény komplexního atributu roste pomaleji než velikost aktivní domény atributu na komplexním atributu závislém.

Algoritmus 1

`A .. množina atributů`
`D .. množina dat`
`C .. matice pokrytí`
`S .. kostra modelu`
$C = \{c_{ij} = 1,\ i \neq j, \forall i, j \in 1..|A|\}$
$S = \{A_i \rightarrow Aj : |i - j| = 1\}$
`Pro` $\forall d \in D$
{
  `ulož` d `do úložiště`
  `aktualizuj velikost domén a podle změn i pořadí atributů v` S `a` C
  `while (zmena)`
  {
    $\forall(i, j, k),\ i < k < j : (A_i \rightarrow A_j) \in S \wedge (A_k \rightarrow A_j) \in S \wedge C_{ik} = 1$
      $S = S - \{A_i \rightarrow A_j\} \cup \{A_i \rightarrow A_k\}$
    $\forall(i, j, k),\ i < k < j : (A_i \rightarrow A_j) \in S \wedge (A_i \rightarrow Ak) \in S \wedge C_{kj} = 1$
      $S = S - \{A_i \rightarrow A_j\} \cup \{A_k \rightarrow A_j\}$
  }
  `testuj` $\forall(A_i \rightarrow A_j) \in S$
  {
    `pokud` $A_i \rightarrow A_j$ `porušeno`
    {
      `testuj` $\forall(A_u \rightarrow A_v)$, `kde` $C_{iv} = 1 \vee C_{uj} = 1$
      `pokud porušeno,` $C_{uv} = 0$
    }
    $S = S - (A_i \rightarrow Aj)$
    `rozšiř kostru`
    `testuj komplexní atributy`
  }
}

1.3 Vlastnosti modelu

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

$$|M_0| = n! \tag{8}$$

$$M_i < M_j \Rightarrow |M_i| > |M_j| \tag{9}$$

$$\forall M_k : M_0 \leq M_k \leq M_m \leq M_\infty \tag{10}$$

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). Z těchto důvodů hovoříme o odhadu struktury, nikoli o její rekonstrukci. První problém lze vyřešit integrací dat z více zdrojů, druhý pak představuje principiální limit metody.

Výhodou odhadování struktury dat je, že výsledný model není závislý na pořadí vstupních dat, což vyplývá z definice funkční závislosti.

2 Induktivní logické programování

Induktivní logické programování (ILP) [12, 13] je úloha strojového učení, jejíž úkolem je naučit se odvodit predikát P_H z literálů ve znalostní bázi B, přičemž tyto literály tvoří fakta nad množinou konstant K.

Vstupem těchto algoritmů je znalostní báze B a funkce $\epsilon \rightarrow \{\ominus, \oplus\}$, $\epsilon \subset K_P^a$ určující pravdivostní ohodnocení predikátu P_H arity a na trénovací množině. Výstupem je množina hypotéz H definujících predikát P_H.

Pro jednoduchost uvažujme pouze základní verze algoritmů, hledaná hypotéza se skládá z literálů, jejichž argumenty jsou pouze proměnné, báze dat obsahuje pouze literály s konstantami, neuvažujeme rekurzivní definice a všechny literály jsou deterministické.

2.1 GOLEM

Jedním ze základních algoritmů ILP je algoritmus GOLEM [12, 16]. Je založen na principu označovaném jako "relative least general generalization".

V jednoduchosti lze říci, že algoritmus převede literály na jejich modely a ty skládá podle konstant tak, aby všechny kombinace konstant v literálu nepokrývaly žádný negativní příklad. Tedy např.

$$p(K_1, K_2), p(K_3, K_4), p(K_1, K_4) \rightsquigarrow p(K_{1,3}, K_{2,4}) \tag{11}$$

Pak kombinuje tyto modely podle množin konstant, tedy např.

$$p(K_{1,3}, K_{2,4}), q(K_{1,3}) \rightsquigarrow p(K_{1,3}, K_{2,4}) \wedge q(K_{1,3}) \tag{12}$$

Pokud tedy převedeme hledaný predikát P_H a literály $b \in B$ na jejich modely, pomocí výše nastíněných operací lze odvodit generalizovaný model hypotézy o predikátu P_H a tento model zpětně převést na množinu literálů popisující hypotézu H.

2.2 FOIL

Dalším algoritmem ILP je algoritmus FOIL [12, 14, 15], který volí jiný způsob hledání klauzulí, oproti algoritmu GOLEM pracujícího na bázi generalizace, volí specializaci. Oproti uvedené notaci místo pravdivostní funkce do algoritmu vstupuje množina pozitivních ϵ^+ a negativních ϵ^- příkladů.

Algoritmus 2

ϵ^+ .. množina pozitivních příkladů
ϵ^- .. množina negativních příkladů
B .. množina literálů - znalostní báze
H .. množina hypotéz $H = \emptyset$
while $(\epsilon^+ \neq \emptyset)$
{
  Specialization algorithm $(\epsilon^+, \epsilon^-, B)$
  {
    $Q = \emptyset$
    $\epsilon_S^+ = \epsilon^+$, $\epsilon_S^- = \epsilon^-$, $B_S = B$
    while $(e_S^- \neq \emptyset \wedge B_S \neq \emptyset)$
    {
      Pro $\forall b \in B_S$:
        Hledej literál b nejlépe rozdělující ϵ^+ a ϵ^-
      $\epsilon_S^+ = \{e \in \epsilon_S^+ \text{ , } e \text{ splňuje } b\}$
      $\epsilon_S^- = \{e \in \epsilon_S^- \text{ , } e \text{ splňuje } b\}$
      $Q = Q \cup \{b\}$
      $B_S = B_S - \{b\}$
    }
  }
  Postprocess $(P \leftarrow Q)$
  $\epsilon^+ = \epsilon^+ - \{e \in \epsilon^+, e \text{ splňuje } Q\ \}$
  $H = H \cup (P \leftarrow Q)$
} until $(\epsilon^+ \neq \emptyset)$

Algoritmus 2 začíná s prázdnou množinou hypotéz.

Nejprve zavolá specializační algoritmus, jehož úkolem je postupně složit optimální tělo Q z literálů b. Tyto literály jsou hledány přes celou bázi znalostí B, mírou optimality se volí entropie. Pro hledání dalšího literálu se omezí množina pozitivních a negativních příkladů pouze na ty, které jsou předchozími literály těla Q splněny. Pokud literál nepokrývá žádný negativní příklad, není nutná další specializace a aktuální tělo je zvoleno jako optimální kandidát.

V opačném případě je nutné dále specializovat. Pokud již specializovat nelze (všechny literály ze znalostní báze byly již použity), algoritmus výběru končí. Pokud takovou kombinaci literálů zařadíme do množiny hypotéz H, nebude výsledná hypotéza konzistentní.

Specializační algoritmus vrací optimální tělo pro hypotézu. Toto tělo ještě můžeme upravit pomocí postprocesního algoritmu, který má eliminuje přebytečné literály v těle Q (takové literály, které sice mají velkou entropii, avšak na výsledné ohodnocení nemají sami o sobě vliv).

Všechny pozitivní příklady, které jsou pokryty tělem Q, jsou z množiny pozitivních příkladů odebrány a množina hypotéz je rozšířena o klausuli $P \leftarrow Q$.

Pokud existují stále hypotézou nepokryté pozitivní příklady, algoritmus se opakuje.

2.3 Jiné algoritmy

V přehledu nebyly ještě uvedeny algoritmy používající inverzních metod (inverzní rezoluce, inverzní dedukce - ALEPH, CIGOL, PROGOL).

Dalším přístupem je transformovat úlohy ILP na jiné algoritmy strojového učení, např. na algoritmy klasifikační [12, 13], kdy oddělující nadplocha je dána pomocí pravdivostní funkce $\epsilon \rightarrow \{\ominus, \oplus\}$ predikátu P_H a jako relevantní atributy klasifikační úlohy jsou hodnoty pravdivostního ohodnocení literálů b ve znalostní bázi B (úlohu je nutné převést do výrokové formy). Nejčastěji se pro tyto účely využívají algoritmy generující rozhodovací stromy. Zmiňme i data-miningové systémy LINUS a DINUS [12], kterých lze rovněž použít pro účely ILP.

3 Řešení problému induktivního logického programování pomocí odhadování struktury dat

Nejprve je nutné zadání úlohy ILP transformovat na zadání metody odhadu struktury dat. To je možné vytvořením tabulky výroků a spuštění algoritmu odhadu struktury dat na tuto tabulku. Výslednou množinu funkčních závislostí je pak nutné znovu transformovat do množiny klausulí tak, aby výsledek byl shodný s výsledky ILP metod.

3.1 Transformace úlohy

Výsledkem ILP je množina hypotéz nad literály, jakým způsobem závisí hodnoty ohodnocení literálů na ohodnocení hledaného predikátu. Takový výsledek metoda odhadu struktury neposkytuje, určuje jen, které predikáty jsou relevantní pro určení pravdivostního ohodnocení hledaného predikátu P_H, vrací množinu predikátů, na kterých ohodnocení hledaného predikátu funkčně závisí.

Pro potřeby metody je nutné ILP úlohu transformovat do výrokové formy [12] pomocí některé propozicionalizační metody, nejčastěji do tabulky pokrývající všechny výroky. Základem takové tabulky je pravdivostní ohodnocení hledaného predikátu P_H arity a podle hodnot jeho vstupních proměnných V_{H1} až V_{Ha} (atribut P_H udává, zda-li se jedná pozitivní $\oplus$ nebo negativní $\ominus$ trénovací příklad.

K této základní tabulce přidáme literály ze znalostní bázi B. Pro každý predikát s odpovídající kombinací proměnných bude vytvořen atribut A_i, který bude nabývat hodnoty podle uvedené notace, chybějící literál v bázi se ohodnotí

negativním symbolem. V některých případech bude nutné zavést další volné proměnné pomocí atributů V_{xi}. Poznamenejme, že tabulka musí obsahovat všechny možné kombinace proměnných (i volných) v predikátech, což v některých situacích může vést ke kombinatorické explozi.

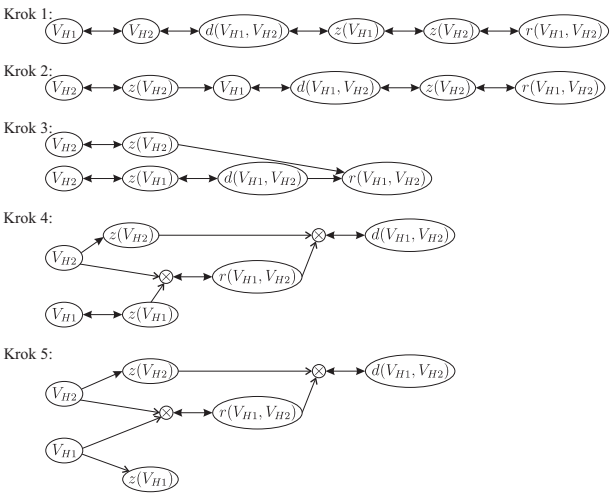
3.2 Příklad odhadu struktury

Pro ilustraci uveďme jednoduchý příklad takto vytvořené tabulky (13). Hledaným predikátem P_H je predikát $d(V_{H1}, V_{H2})$ popisující skutečnost, že objekt V_{H2} je dcerou objektu V_{H1}. Znalostní báze B obsahuje predikát $r(V_{H1}, V_{H2})$ (objekt V_{H1} je rodičem objektu V_{H2}) 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	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$

(13)

Následující obrázek (1) ukazuje vývoj modelu po každém přidání řádku z tabulky (13) podle Algoritmu 1.



Obrázek 1. Ilustrativní příklad

3.3 Transformace výsledku

Z konečného modelu vyplývá, které predikáty jsou relevantní pro určení ohodnocení hledaného predikátu P_H. Aby byl tento postup srovnatelný s algoritmy

ILP, je nutné vrátit výsledek ve shodném tvaru, tedy jako seznam klausulí. Z tabulky vybereme pouze atributy odpovídající těm predikátů, na kterých je ohodnocení hledaného literálu závislé a ty řádky, které obsahují příklady z množiny pozitivních příkladů. Pro každou jedinečnou kombinaci ohodnocení literálů budeme generovat klausule, je-li ohodnocení literálu v daném řádku negativní, bude literál v klausuli uveden s negací, je-li pozitivní, bude uveden bez negace.

Protože pro pravdivostní ohodnocení $d(V_{H1}, V_{H2}) = \oplus$ je podle (13) podmínkou, aby $r(V_{H1}, V_{H2}) = \oplus$ a zároveň $z(V_{H1}) = \oplus$, budou oba literály tvořící tělo podle výše uvedeného postupu klausule pozitivní, tedy

$$d(V_{H1}, V_{H2}) \leftarrow r(V_{H1}, V_{H2}), z(V_{H1}) \tag{14}$$

Takto vygenerované klausule nemusí být optimální, však je lze upravit pomocí libovolného z algoritmů minimalizující boolské funkce nebo postprocesní úpravu (jako FOIL [12, 14]) ověřující skutečný vliv literálu na výsledné přiřazení do kategorií trénovací množiny.

4 Srovnání metod

Výhodou nastíněné metody odhadování struktury dat je, že prostor možných řešení není prohledáván hladovým způsobem, jako např. u algoritmu FOIL, tedy nalézá všechna přípustná řešení. Představme si, že hledaný predikát lze odvodit více způsoby $P_H \leftarrow \{L_i^1\}$ a $P_H \leftarrow \{L_i^2\}$, přičemž $L^1 \cap L^2 = \oslash$. Algoritmus FOIL vrátí pouze množinu predikátů L^1 nebo L^2 (teoreticky náhodně vybere jednu z nich) [14, 15]. To sice vede k tomu, že vygenerovaná teorie není minimální, ale lépe vystihuje daný predikát.

Algoritmus odhadování struktury dat nejen že může generovat teorie pro více literálů zároveň stejně jako např. metody použité v systému DINUS [12], ale může uvažovat i vztahy mezi hledanými literály.

Podle (7) je rozšiřování modelu o komplexní atribut (v ILP ekvivalentní specializaci či generalizaci) je nepolynomiálně složité vyjma speciálních případů nepokrývající problematiku ILP, složitost je dána aritou funkční závislosti, která nebývá nikterak velká. U algoritmu FOIL je však tato složitost podmíněna počtem predikátů, který je vyšší nežli arita predikátů, u algoritmu GOLEM se tato složitost odvozuje od velikosti domény proměnných. Naopak nevýhodou metody odhadu struktury dat je nutnost výroky převést do tabulky, přičemž závislosti lze hledat pouze v rámci jednoho řádku, tedy neumožňuje rekurzivní definice. To sice lze vyřešit analogickým způsobem jako rozšířená verze FOIL [15], avšak podstatně se zvyšuje počet volných proměnných a jim odpovídajících literálů.

Odhadování struktury dat principem zaručuje nezávislost na pořadí vstupních dat, což u některých inverzních metod ILP nelze zaručit.

Srovnání vyjmenovaných klasických algoritmů ILP s metodou odhadu struktury tedy nelze obecně shrnout, záleží případ od případu, kdy je použití jedné z metod efektivnější. Slabým místem metody odhadu struktury dat je především nutnost mít k dispozici celý záznam, který reprezentuje všechny možné kombinace proměnných v literálech. Zde se však rýsuje možnost použít mechanizmů

podobných integraci dat a hledání globálního modelu dat, čímž by tato nepříjemná nutnost odpadla. Integrační mechanizmy při odhadování struktury dat jsou jedním z dalších předsevzatých úkolů a budou tématem na další práci.

Poděkování

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".

Reference

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 DMIN 2005, pp 37-40. ISBN 1-934215-79-3. 2005.
5. G. Grahme, K. Räihä, "Database Decomposition into Fourth Normal Form", in *Conference on Very Large Databases*, pp. 186–196, 1983.
6. 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.
7. Bruno T. Messmer, Horst Bunke "Efficient Subgraph Isomorphism Detection: A Decomposition Approach", in *IEEE Transactions on Knowledge and Data Engineering.* pp. 307-323. 2000.
8. F. Cuppens, K. Yazdanian "A Natural Decomposition of Multi-level Relations", in *IEEE Symposium on Security and Privacy*, pp. 273-284. 1992.
9. B.N. Shamkant, R. Minyoung, "Vertical Partitioning for Database Design – a Graphical Algorithm", in *SigMod*, pp. 440–450, 1989.
10. 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.
11. 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" http://citeseer.ist.psu.edu/barfourosh02information.html [online]. 2002.
12. Nada Lavrač, Sašo Džeroski "Inductive Logic Programming - Techniques and Applications", Ellis Hordwood, Chichester. ISBN: 0-13-457870-8. 1994.
13. Sašo Džeroski, Nada Lavrač "Relational Data Mining", Springer-Verlag, Berlin. ISBN: 3-640-42289-7. 2001.
14. Quilan, J. "Learning logical definitions from relations", in *Machine Learning*, 5(3). pp 239-266. 1990.
15. Quilan, J. "Knowledge acquisition from structured data - using deteminate literal to assist search", in *IEEE Expert*, 6(6). pp 32-37. 1991.
16. Muggeton, S., Feng, C. "Efficient introduction of logic programs", in *Proceding of the First Conference on Algorithmic Learning Theory.* pp 368-381. 1990.