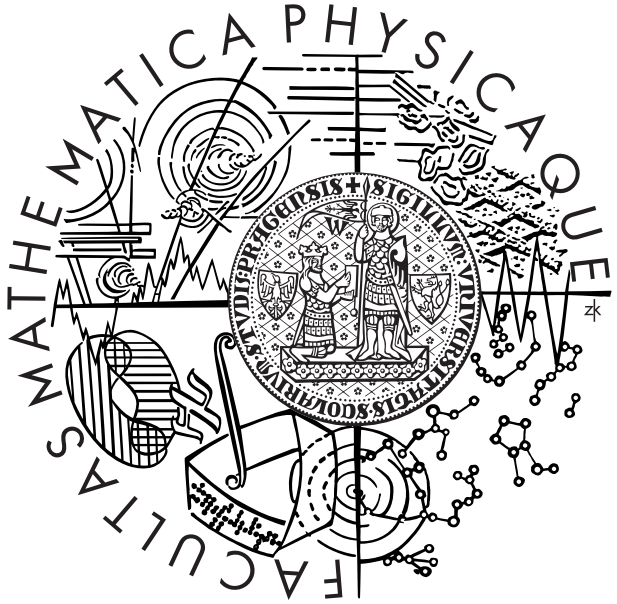
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

Diplomová práce



Alexander Kuzmin

Relační GUHA procedury

Katedra softwarového inženýrství
Vedoucí diplomové práce: Doc. RNDr. Jan Rauch, CSc.
Studijní program: Informatika

Poděkování

Chtěl bych poděkovat zejména vedoucímu práce Doc. RNDr. Janu Rauchovi, CSc. za vedení diplomové práce, odborné rady a cenné připomínky. Chtěl bych také poděkovat zejména Tomášovi Kuchařovi za jeho pomoc a všem členův vývojářského týmu Ferda, kteří mi svými radami a pomocí umožnili tuto práci dokončit.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne: Alexander Kuzmin

Obsah

Abstrakt

Název práce: Relační GUHA procedury
Autor: Alexander Kuzmin
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí diplomové práce: Doc. RNDr. Jan Rauch, CSc.
e-mail vedoucího: rauch@vse.cz

Abstrakt: Jedná se o diplomovou práci kategorie "implementace" Jejím cílem je navrhnout a implementovat relační rozšíření vybraných procedur z GUHA procedur implementovaných v systému LISp-Miner. Pro implementaci rozšíření byly zvoleny procedury 4ft-Miner a SD4ft-Miner. Rovněž byly implementovány některé prostředky pro transformaci atributu. Procedury byly implementovány v rámci prostředí Ferda [8].
Klíčová slova: Relační datamining, relační rozšíření, virtuální atribut, hypotézový atribut, agregační atribut

Title: Relational GUHA procedures
Author: Alexander Kuzmin
Department: Department of Software Engineering
Supervisor: Doc. RNDr. Jan Rauch, CSc.
Supervisor's e-mail address: rauch@vse.cz

Abstract: The thesis belongs to implementation category. The goal of this thesis is to design and implement relational extensions of the selected GUHA procedures that are implemented in LISp-Miner system. 4ft-Miner and SD4ft-Miner have been selected for the implementation of relational extensions. There have also been impemented several methods for attribute transformation. Procedures have been implemented in the Ferda environment [8].
Keywords: Relational datamining, relational extensions, virtual attribute, hypotheses attribute, aggregation attribute

Kapitola 1

Cíle práce

V této kapitole jsou popsány cíle diplomové práce, její zadání a jeho upřesnění. Dále je popsána struktura textu dané diplomové práce. V textu budou se budou používat pojmy metod dobývání znalostí a prostředí Ferda DataMiner (například GUHA, kategorie, atribut a jiné), jejichž definice bude uvedena v kapitole 2.

1.1 Původní zadání práce

Níže je uvedeno zadání diplomové práce dle Studijního informačního systému:

Východiskem pro diplomovou práci jsou zkušenosti s GUHA procedurami implementovanými v systému LISp-Miner [3] a s relačním rozšířením procedury 4ft-Miner, které je implementováno v rámci disertační práce T. Karbana [5], viz též [6, 7]. Zkušenosti ukazují, že má smysl rozšířit obdobným způsobem i další ze zmíněných GUHA procedur. Práce s relačním rozšířením procedur však bude vyžadovat zadávání poměrně rozsáhlé sady vstupních parametrů, k čemuž bude zapotřebí různých netriviálních znalostí. Je proto třeba zajistit možnost využití metod znalostního inženýrství při určování hodnot vstupních parametrů relačních GUHA procedur. Příkladem je využití zabudovaného expertního systému pro definici množiny přípustných literálů. Předpokládá se přitom, že uživatel bude mít možnost modifikovat bázi znalostí tohoto expertního systému.

Jedná se o diplomovou práci kategorie "implementace" Jejím cílem je navrhnout a implementovat relační rozšíření vybraných procedur z GUHA procedur implementovaných v systému LISp-Miner. Implementované procedury budou zahrnovat možnosti využití metod znalostního inženýrství při zadávání jejich vstupních parametrů. Procedury budou implementovány v

rámci prostředí Ferda [8]. Výběr procedur k rozšíření a jejich specifikace budou podrobně konzultovány s vedoucím práce.

1.2 Upřesnění a rozšíření zadání práce

V průběhu diplomové práce se však ukázalo, že bude vhodné zadání práce upravit, upřesnit a rozšířit. Toto se stalo po konzultacích s vedoucím diplomové práce. V zadání došlo tedy k následujícím upřesněním a změnám.

Byly upřesněny procedury, pro které bude implementováno relační rozšíření. Rovněž bylo rozhodnuto nahradit implementaci a využití expertního systému za implementací dalších prostředků pro transformaci atributu.

1.2.1 Výběr procedur pro relační datamining

Systém Ferda DataMiner je používán primárně v akademickém prostředí, jak pro výuku studentů, tak i pro zpracování zajímavých úloh obecně. Nejčastěji se používají procedury 4FT a SD4FT z důvodů relativní jednoduchosti zadání úlohy a interpretace výsledků. Proto se autor spolu s vedoucím práce rozhodli pro implementaci relačních rozšíření procedur 4FT a SD4FT.

1.2.2 Využití metod znalostního inženýrství

Od požadavků na využití metod znalostního inženýrství, konkrétně využití zabudovaného expertního systému pro určování hodnot vstupních parametrů GUHA procedur bylo po konzultaci s vedoucím diplomové práce upuštěno. Zjistilo se, že náročnost této úlohy výrazně přesahuje náplň diplomové práce a proto by nebylo možné ji v rámci dané práce implementovat.

1.2.3 Atributy

Na druhou stranu během práce s dosavadní implementaci procedur GUHA v prostředí Ferda DataMiner se ukázalo jako velmi vhodné implementovat další prostředky pro transformaci atributu, konkrétně pro vytváření ekvidistančních a ekvifrekvenčních intervalů a modulu pro ruční úpravu. Tyto nebyly z časových důvodů v rámci diplomové práce T.Kuchaře [10] implementovány, ačkoliv je nutné se zmínit, že většina příslušných základních algoritmů již byla připravena.

Implementace těchto metod přispěla nejen ke zkvalitnění systému Ferda obecně, ale rovněž umožnila autorovi této práce podstatně rozšířit okruh úloh, které mohou řešit nově implementovaná relační rozšíření původních dataminingových procedur. Proto bylo zadání diplomové práce rozšířeno o metody pro transformaci atributu. Následuje jejich krátký popis, podrobné informace lze nalézt v kapitole 4.

Ekvidistanční intervaly

Tato metoda umožní uživateli definovat atribut skládající se ze zadaného počtu tzv. ekvidistančních intervalů – ty mají stejnou velikost a pokrývají množinu hodnot atributu. Příkladem je rozdělení klientů banky dle výši příjmu například vytvořením následujících intervalů: $\langle 5000, 8000)$, $\langle 8000, 12000)$ a $\langle 12000, 15000)$.

Ekvifrekvenční intervaly

Tato metoda umožní uživateli definovat atribut skládající se ze zadaného počtu tzv. ekvifrekvenčních intervalů – ty mají takovou velikost, aby počet hodnot vstupní množiny atributu, který do každého z nich patří, byl přibližně stejný. Tato metoda pro transformaci atributu má využití pro zajištění rovnoměrného rozdělení množin hodnot, které nejsou původně takto rozděleny. Příkladem opět poslouží rozdělení klientů na zhruba stejně velké skupiny dle výše jejich příjmu například vytvořením následujících intervalů: $\langle 5000, 10000)$, $\langle 10000, 18000)$ a $\langle 18000, 25000)$. Znamená to, že klienty banky rozdělíme do skupin, které znamenají, že klient má nízký, střední či vyšší příjem, přičemž parametry skupin (hranice intervalů) jsou zvoleny tak, že každá skupina obsahuje zhruba stejný počet klientů.

Statický atribut

Tato metoda umožní uživateli buď upravovat již vytvořený atribut (například za pomocí metody ekvidistančních intervalů) anebo ručně vytvořit zcela nový atribut. Umožňuje upravovat kategorie v atributu, výčty a intervaly patřící ke každé kategorii a výsledek se samozřejmě může dále použít pro definici dataminingové úlohy v prostředí Ferda DataMiner. Statický atribut je podrobněji popsán v článku [8].

1.3 Východiska řešení

K naplnění cílů dané diplomové práce bylo nezbytné vycházet z existující implementace dataminingových procedur, způsobů zadávání úloh pro práci s těmito procedurami a celkově z prostředí Ferda DataMiner, na kterém proběhla implementace relačních rozšíření. Připomeňme zde, že prostředí Ferda DataMiner vzniklo v rámci studentského softwarového projektu pod vedením Doc. RNDr. Jana Raucha, CSc. – podrobněji viz 2.2.3 anebo [8] a je nadále vyvíjeno jak účastníky původního projektu v rámci diplomových a disertačních prací, tak i jinými zájemci.

Klíčovou diplomovou práci pro prostředí Ferda DataMiner se stala práce "Experimentální GUHA procedury" Tomáše Kuchaře, která je uvedena v seznamu použité literatury jako [10]. Tomáš implementoval nejdůležitější prvky pro práci s úlohami dobývání znalostí (mj. 6 dataminingových procedur) a připravil programátorské rozhraní pro další implementaci konkrétně metod GUHA.

Tato informace je zde uvedena proto, že autor dané diplomové práce se musel pro úspěšné řešení zevrubně seznámit s diplomovou práci [10], zejména s její implementační části a nejen implementovat vlastní řešení, ale zajistit jeho škálovatelnost a připravit prostředky pro budoucí implementace relačních rozšíření v rámci prostředí Ferda DataMiner. Toto se stalo dalším, byť explicitně neuvedeným cílem diplomové práce.

1.4 Struktura diplomové práce

S ohledem na výše stanovené cíle byla zvolena následující struktura diplomové práce.

V dané kapitole byly uvedeny a popsány zadání, jeho upřesnění a cíle dané diplomové práce.

V kapitole 2 budou popsány pojmy z oblasti dobývání znalostí a prostředí Ferda DataMiner, které jsou v dané diplomové práce používány. Také bude podrobně rozebrán pojem relačního dobývání znalostí, jehož některé procedury daná diplomová práce implementuje.

V kapitole 3 bude popsán vztah dané diplomové práce k diplomové práci Tomáše Kuchaře "Experimentální GUHA procedury" uvedené v seznamu použité literatury jako [10]. Budou popsány důsledky této návaznosti ve vztahu k implementaci a celkové architektuře programátorského řešení. V této kapitole je rovněž uvedená diskuze nad způsobem implementace.

V kapitole 4 jsou podrobně popsány všechny implementované prvky z uživatelského hlediska. Rovněž jsou uvedeny nejdůležitější detaily implementace.

V kapitole 5 obsahuje výsledky testů zvoleného řešení pro různá zadání úlohy a datové matice spolu s diskuzi nad nimi.

V kapitole 6 je uveden podrobný seznam vytvořených a změněných souborů se zdrojovým kódem, rovněž je uveden stručný návod pro vytvoření relačních rozšíření pro další procedury.

V kapitole 7 jsou shrnuty dosažené výsledky, zhodnoceno splnění cílů dané diplomové práce a rovněž je doporučen směr dalšího vývoje.

Kapitola 2

Úvod

V této kapitole jsou uvedeny základní pojmy dobývání znalostí. Je popsána metoda GUHA a jiné metody a jejich implementace. Podrobněji je uveden relační datamining. Také je zde představeno prostředí Ferda DataMiner, pro které proběhla implementace relačních rozšíření.

2.1 Úvod do dobývání znalostí z dat

2.1.1 Přehled

Data mining (DM) je analytická metodologie získávání netriviálních skrytých a potenciálně užitečných informací z dat. Někdy se chápe jako analytická součást dobývání znalostí z databází (Knowledge Discovery in Databases, KDD), jindy se tato dvě označení chápou jako souznačná.[1] V češtině se používá pojem *dobývání*, *získávání*, *dolování* nebo *vytěžování dat*. Zde se budeme zabývat nalezením vzorů nebo vztahů (z anglického slova *patterns*) v podobě asociačních pravidel a asociačních pravidel SDS (podrobněji viz 2.1.3).

Dobývání znalostí stojí na pomezí výpočetních statistických metod, databázových technologii a strojového učení. Metodami dobývání znalostí se dají řešit úlohy deskripce dat, sumarizace, segmentace, deskripce konceptů, klasifikace, predikce a analýzy závislostí v celé řadě aplikačních oblastí.

[1] Definice je převzata z http://cs.wikipedia.org/wiki/Data_mining

2.1.2 Metody dobývání znalostí

Při dobývání znalostí z dat lze použít více metod. Dobrých výsledků lze dosáhnout při jejich kombinaci. Jako příklad můžeme uvést statistické procedury, rozhodovací stromy, neuronové sítě nebo asociační pravidla, se kterými pracuje metoda GUHA a kterými se zde budeme podrobněji zabývat.

2.1.3 Asociační pravidla – analýza nákupních košíků

"Klasické" asociační pravidlo je definováno Aggravalem ([11]) v souvislosti s analýzou nákupního košíku. Zde použijeme jeho popis, jak je uveden J.Rauchem v [4].

Klasické asociační pravidlo je ve tvaru $X \rightarrow Y$, kde X a Y značí dvě množiny prvků. Toto pravidlo nám říká, že transakce obsahující prvky z X, mají tendenci obsahovat prvky z Y.

Analogie s nákupními košíky říká, že když nákupní koš obsahuje množinu předmětů X, pravděpodobně bude obsahovat množinu předmětů Y. Například klienti, kteří nakoupí víno, často nakupují také sýry. Každý z nákupních košíku $b_1, \ldots, b_n$ tedy obsahuje některé z položek $P_1, P_2, \ldots, P_K$.

Nákupní košík	Položky
b_1	P_1, P_2, P_3, P_4, P_5
b_2	P_2, P_3, P_4, P_5, P_K
b_3	P_1, P_3, P_{12}
b_4	P_1, P_3
$\vdots$	$\vdots$
b_{n-1}	$P_2, P_4, P_7, P_7, P_{107}$
b_n	P_2, P_4, P_K

Tabulka 2.1: Příklad množiny transakcí

Výraz $P_2, P_5 \rightarrow P_1, P_3$ je příklad asociačního pravidla, které říká, že nákupní košíky obsahující položky P_2 a P_5, často obsahují i položky P_1 a P_3.

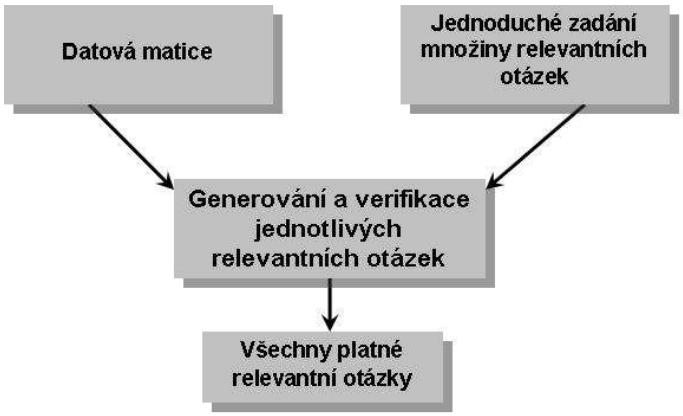
Používají se dvě míry intenzity asociačního pravidla – *podpora (support)* a *konfidence (confidence)*. *Podpora* asociačního pravidla $X \rightarrow Y$ je podíl

počtu transakcí obsahujících jak X, tak Y k počtu všech transakcí. *Konfidence* je podíl počtu transakcí obsahujících jak X, tak Y k počtu transakcí obsahujících X.

Analýzou transakcí supermarkety mohou optimalizovat rozmístění zboží v prodejně a zajistit rychlejší a výnosnější prodej. Pro dobývání znalostí za využití takových pravidel byl navržen algoritmus Apriori, viz [11].

2.1.4 GUHA

GUHA je zkratka pro General Unary Hypotheses Automaton. Teoretický základ byl formulován Hájkem v roce 1966 (podrobněji viz [1]), detailněji pak v [2]. Pro ilustraci práce metody GUHA uvádíme obrázek 2.1.



Obrázek 2.1: Práce metody GUHA

Metoda GUHA je realizována procedurami GUHA. Procedura je program, který na základě vstupu, který je tvořen analyzovanými daty a jednoduchým zadáním relevantních otázek, generuje zadané otázky a ověřuje je oproti vstupním datům. Výstupem jsou pak všechny relevantní otázky (také hypotézy), které jsou prosté. Hypotéza je prostá, jestliže je pravdivá v analyzované datové matici a zároveň nevyplývá z jiných, jednodušších hypotéz.

Jednou z nejpoužívanějších GUHA procedur je procedura ASSOC (podrobněji v [2]), která hledá vztahy, jenž jsou zobecněním "klasických" asociačních pravidel definovaných Aggravalem v [11]. Nejnovější implementací ASSOC je procedura 4FT, která je popsána dále v tomto textu.

2.1.5 Relační dobývání znalostí

Následující úvaha o relačním dobývání znalostí byla převzata z disertační práce T.Karbana uvedené v seznamu použité literatury jako [5].

Existující metody dobývání znalostí byly převážně navrženy pro data uložena v jedné datové matici (tzv. *plochá reprezentace*). Při takové organizaci dat řádky datové matice reprezentují pozorování (objekty studia, osoby, záznamy, předměty apod.) a sloupce datové matice reprezentují proměnné (vlastnosti, atributy nebo charakteristiky). Ačkoliv takový pohled na zkoumaná data zjednoduší metody používané pro dobývání znalostí, v praxi se velmi často vyskytují data ve složitější reprezentaci, například, ve vícero datových maticích, které jsou mezi sebou v relaci.

Je samozřejmě možné požadovaná data spojit do jedné datové matice, avšak takový úkon je časově velmi náročný a často se musí provést manuálně pro každou úlohu, chceme-li zkoumat například jiné atributy objektů z matice.

Některé úlohy pro dobývání znalostí navíc nejdou jednoduše provést nad "plochou" reprezentaci, obzvláště v případě, kde uvažujeme mnoho různých objektů s jejich vlastnostmi a vzájemnými vztahy. Cílem zavedení relačního dataminingu je odstranění omezení na jednu tabulku dat a umožnění dobývání znalostí z více relačně propojených datových tabulek. Navíc, jak uvádí T.Karban v [12], ponechání dat v relačním tvaru může zachovat informace, které mohou být vyjádřeny ve formě relačních vztahů.

2.2 Implementace metody GUHA

2.2.1 LISp-Miner

Od roku 1996 se na pražské VŠE vyvíjí systém LISp-Miner. Jak se uvádí na stránkách `http://lispminer.vse.cz/`,

> "LISp-Miner je akademický projekt pro podporu výzkumu a výuky dobývání znalostí z databází. Je vhodný zejména pro výuku studentů, pilotní a středně velké projekty KDD."

Výuka probíhá zejména na Fakultě informatiky a statistiky Vysoké školy ekonomické v Praze a Matematicko-fyzikální fakultě Univerzity Karlovy v Praze. LISp-Miner obsahuje implementaci několika procedur pro dobývání znalostí spolu s moduly pro zadávání úlohy a prezentace výsledků. Vzhledem k dlouhé době vývoje prostředí (již od roku 1996) značně vzrostla komplexnost celého prostředí z hlediska uživatelské práce – každá dataminingová procedura se například skládá ze tří modulů: pro zadání úlohy, pro samotné generování a pro prohlížení výsledků. Za jeden z největších nedostatků systému LISp-Miner lze považovat i fakt, že zadání úloh pro jednotlivé procedury lze sdílet mezi sebou pouze minimálně – v tom autor spatřuje jednu z velkých výhod prostředí Ferda DataMiner popsaného dále, z uživatelského hlediska. Na druhou stranu je třeba říct, že vzhledem k nejdelší historii systému LISp-Miner procedury v něm implementované jsou relativně stabilní a odladěné a při jejich nových implementacích v jiných prostředích pro dobývání znalostí lze výsledky ze systému LISp-Miner považovat za vzorové. Podrobné informace o systému LISp-Miner lze nalézt v [3].

2.2.2 Rel-Miner

Systém Rel-Miner je softwarový projekt vznikající v rámci disertační práce (viz. [5]) Mgr. Tomáše Karbana na Matematicko-fyzikální fakultě Univerzity Karlovy. T.Karban uvádí jako cílech své práce následující:

- rozšířit jazyk asociačních pravidel pro relační dobývání znalostí,
- implementovat systém Rel-Miner za použití bitových řetízků,
- experimentovat s relačním dataminingem, zkoumat efektivitu a použitelnost,
- vytyčit možnosti dalšího vývoje.

Stávající implementace dataminingových procedur a parametrů zadávání úlohy, které vznikly v rámci diplomové práce T.Kuchaře, jsou v některých ohledech inspirovány právě systémem Rel-Miner. Rovněž tato diplomová práce má některé podobné cíle jako Rel-Miner.

Rozdíly jsou zejména ve větším teoretickém přínosu projektu Rel-Miner, také v zaměření T.Karbana hlavně na možnost distribuovanosti běhu relačních i obecně nerelačních dataminingových procedur. Podstatným rozdílem je způsob implementace: Rel-Miner byl od počátku vyvíjen jako samostatné softwarové dílo, které je zaměřeno na teoretické aspekty relačního dobývání dat a implementace je spíše vzorová a obsahuje relační rozšíření pouze procedury 4ft-Miner, kdežto implementace relačních rozšíření v rámci této diplomové práce je zaměřená na práci v prostředí Ferda DataMiner, na využití veškerých jeho uživatelských i programátorských výhod a již implementovaných prostředků. Měla za cíl kromě dvou implementovaných procedur relační těžby dat položit základy pro snazší přidávání relačních rozšíření pro zbývající procedury v rámci systému Ferda DataMiner.

Rel-Miner měl být prvotní implementací v oblasti relačního dataminingu a principu bitových řetězců. Momentálně je projekt ve stádiu dokončení, nicméně ještě není hotov a autorovi této práce není známo, zda již existuje ve spustitelné podobě umožňující běh relačních dataminingových úloh. Podobně jako systém LISp-Miner je i systém Rel-Miner vytvářen především pro výzkumné účely a bude tedy vhodný pro malé a středně velké projekty, především pak pro výuku studentů.

2.2.3 Ferda DataMiner

Následuje popis prostředí Ferda DataMiner, jak je uveden v článku [8].

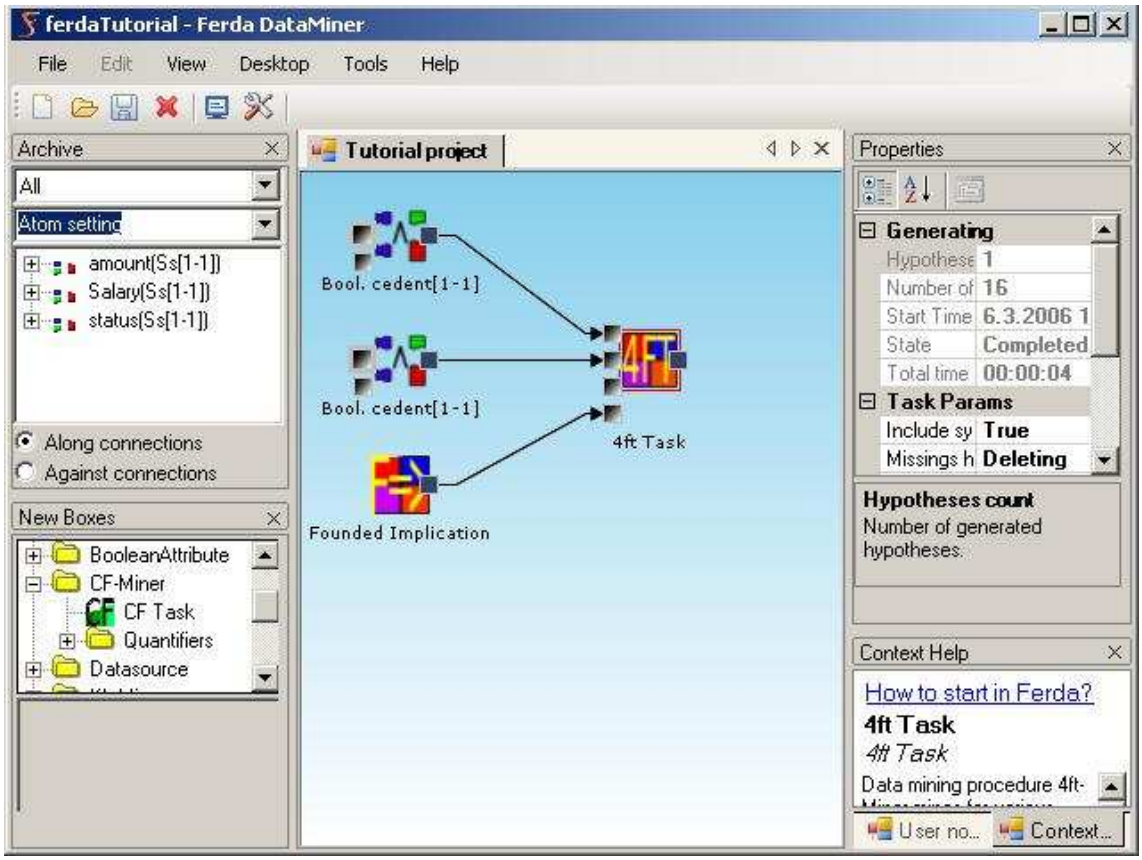
Ferda DataMiner (`http://sourceforge.net/projects/ferda`) vznikl jako softwarový projekt na Matematicko-fyzikální fakultě Univerzity Karlovy. Měl odstranit největší nedostatky systému LISp-Miner zejména v oblasti modularity a jednoduchosti uživatelské práce. Představuje společné vizuální prostředí pro šestici výše uvedených procedur systému LISp-Miner, přináší vizuální model postupu dobývání znalostí podobný modelům v komerčních systémech SPSS Clementine, SAS Enterprise Miner apod. Srovnání s těmito systémy je rovněž uvedeno v [8].

Krabičky

Ferda přináší pohled na zadání úlohy dobývání znalostí jako na zapsání funkce pomocí jistých vizuálních objektu – tzv. krabiček. Každá krabička zastává také roli funkce nebo několika funkcí. Krabičky mají parametry a vyhodnocují své funkce nad hodnotami těchto parametrů. Parametrům se u krabiček říká zásuvky. Do zásuvek se zapojují jiné krabičky. Krabičky jsou

rozděleny podle typu a do každé zásuvky lze zapojit pouze některé typy krabiček.

Pomocí řady jednoduchých krabiček je uživateli umožněno sestavovat vlastní algoritmy. Ferda tedy rovněž představuje něco jako uživatelský programovací jazyk. Na krabičky se lze však také dívat jako na konstrukce logiky či sémantiky, ke které existuje lambda-kategoriální gramatika. Pro ilustraci prostředí uvedeme obrazovku z prostředí Ferda jako obrázek 2.2.



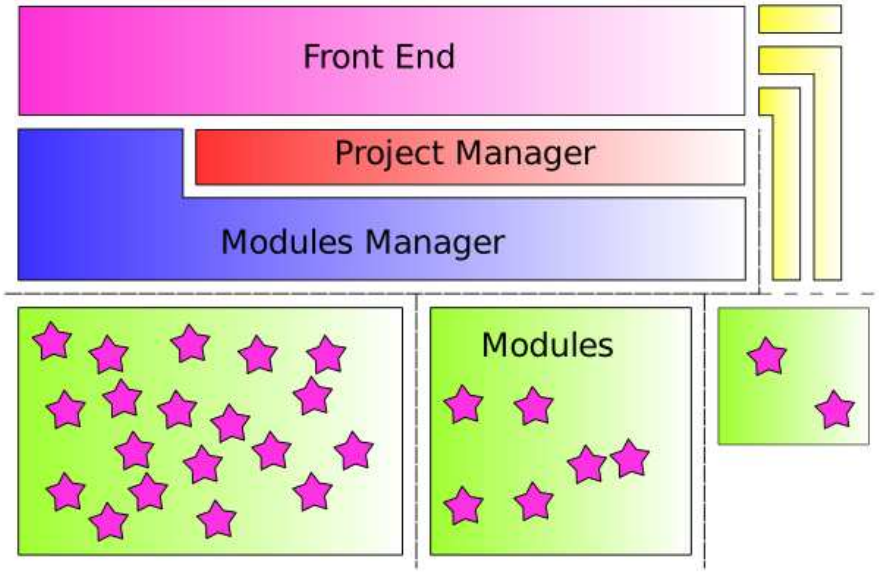
Obrázek 2.2: Hlavní obrazovka prostředí Ferda DataMiner

Krabičkový model je považován za největší přínos prostředí Ferda DataMiner. Nová funkcionalita se do prostředí může přidat ve formě krabičky zapojitelné mezi stávající krabičky. V této diplomové práci autor implementuje metody uvedené v kapitole 1 ve formě krabiček, které se zapojují do stávajícího zadání úlohy.

Základní programátorské rozdělení prostředí Ferda DataMiner

Zde shrneme programátorské rozdělení prostředí Ferda DataMiner na jednotlivé části, jak je uvedeno v dokumentu "Implementace Ferda", který je součástí dokumentace studentského projektu Ferda DataMiner.

Pro ilustraci uvedeme obrázek 2.3 celkového programátorského návrhu prostředí Ferda DataMiner.



Obrázek 2.3: Programátorský design prostředí Ferda DataMiner

Uživatelské rozhraní (FrontEnd) má za úkol zprostředkovávat funkce manažeru projektů uživateli a postarat se o pohodlnou a přívětivou práci s moduly.

Manažer projektů má za úkol zprostředkovávat funkce manažeru modulů, přidat funkce nad moduly důležité pro projekt a zajistit funkce archivu, pohledů pracovní plochy a ukládání a načítání stavu pracovní plochy do XML formátu.

Manažer modulů má za úkol načítání seznamu modulů z IceGridu, udržování seznamu modulů, se kterými může pracovat a přitom nejsou v IceGridu, vytváření nových instancí krabiček za pomocí modulů krabiček,

zprostředkování funkcí jednotlivých modulů a nabízení jistých funkcí modulům.

Moduly mohou být několika typů: *moduly krabiček* — moduly, které vytvářejí instance, *moduly pro nastavování nestandardních vlastností krabiček* a jiné moduly pro komunikaci s uživatelem (tzv. *moduly pro interakci*). Jedná se především o různé pomocné dialogy pracující vždy nad určitými typy krabiček.

Add Ins jsou přídavné části FrontEndu, které s ním komunikují přímo přes CLI. Nejčastěji obsahuje moduly pro interakci či moduly pro nastavování nestandardních vlastností. Není však nutné, aby například modul pro interakci byl součástí nějakého Add Inu.

Metoda pro statickou úpravu atributu v této práci byla naprogramována jako modul pro interakci s uživatelem.

2.2.4 Diplomová práce "Experimentální GUHA procedury"

Zásadním přínosem pro prostředí Ferda DataMiner se stala diplomová práce T.Kuchaře "Experimentální GUHA procedury", která přinesla novou implementaci krabiček šesti procedur pro dobývání znalostí a zobecnění některých prvků zadání úlohy oproti jejich stávající implementaci v systému LISp-Miner.

Při implementaci relačních rozšíření procedur autor vycházel z jejich nové implementace v rámci výše uvedené práce. Popis provázanosti se systémem Ferda DataMiner a konkrétně s novou implementaci dataminingových procedur od T.Kuchaře v rámci jeho diplomové práce [10] je součástí popisu implementace a patří do kapitoly 3.

2.3 Důležité rysy metody GUHA

Zde bude následovat stručný přehled principů dobývání znalostí dle metody GUHA, které jsou nezbytné k pochopení popisu samotné problematiky a implementace zadání dané diplomové práce. Tyto principy jsou obecné a byly využity při implementaci v systémech pro dobývání znalostí pracujících na principu GUHA. Pro zevrubnější popis autor doporučuje prostudování zejména zdrojů [3] a [10].

2.3.1 Booleovské atributy

Metoda GUHA pracuje s diskrétními daty. Data, která tuto podmínku nesplňují, musí být diskretizována. Diskretizovaný sloupec, který může nabývat konečně mnoha hodnot, se označuje jako *atribut* a jeho hodnoty jsou *kategorie*. Na základě sloupců analyzované matice dat vytváříme *booleovské atributy*. Následuje definice booleovského atributu.

Definice booleovského atributu 1 *Booleovský atribut je formule splňující následující podmínky.*

Pokud A *je atribut a* α *je vlastní neprázdná podmnožina množiny všech* kategorií *atributu* A*, pak* $A(\alpha)$ *je* základní booleovský atribut*. Každý základní booleovský atribut je booleovský atribut. Pokud* ϕ *a* ψ *jsou booleovské atributy, pak* $\phi \wedge \psi$*,* $\phi \vee \psi$ *a* $\neg\phi$ *jsou odvozené booleovské atributy. Každý* odvozený booleovský atribut *je booleovský atribut.*

Dále GUHA pracuje s pojmem *dílčí booleovský atribut*, kterým označuje booleovské atributy, které vystupují jako operandy v konjunkcích a disjunkcích booleovských atributů.

Základní booleovský atribut $A(\alpha)$ nabývá hodnoty *true* na řádku o matice M, pokud $a \in \alpha$, kde a je hodnota atributu a na řádku o – dle odstavce 2 v [3]. Hodnota odvozených booleovských atributů se definuje obvyklým způsobem dle následující tabulky:

ϕ	ψ	$\phi \wedge \psi$	$\phi \vee \psi$	$\neg\phi$
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Tabulka 2.2: Hodnoty odvozených booleovských atributů

Hodnota booleovského atributu ϕ v řádku o matice M se značí $\phi(o, M)$. Platí, že $\phi(o, M) = true$, pokud ϕ je pravdivý na řádku o matice M, jinak $\phi(o, M) = false$.

2.3.2 Relevantní otázky

Procedury metody GUHA hledají zajímavé vztahy na množině dat z datové matice. Tyto vztahy (také relevantní otázky) jsou generovány ve formě zobecněných asociačních pravidel. Zadání množiny relevantních otázek se pro procedury 4FT a SD4FT provádí pomocí booleovských atributů. Tvary asociačních pravidel pro procedury 4FT a SDFT jsou uvedeny v 2.4.

2.3.3 Zadání množiny relevantních otázek

Podrobné možnosti nastavení jednotlivých složek zadání booleovského atributu jsou popsány T.Karbanem v [5] a implementovány do prostředí Ferda DataMiner T.Kuchařem v [10]. V prostředí Ferda DataMiner byly v rámci diplomové práce [10] implementovány krabičky, pomocí kterých lze zadávat booleovské atributy. Podrobný popis zadávání množiny relevantních otázek je uveden v práci T.Kuchaře [10]. Zde pouze uvedeme, že možnosti zadávání množiny relevantních otázek v prostředí Ferda byly rozšířeny ve srovnání s možnostmi v LISp-Mineru.

2.4 Dataminingové procedury

Zde popíšeme dataminingové procedury 4FT a SD4FT, protože v dané diplomové práci byly implementována relační rozšíření právě těchto dvou procedur. Další implementované procedury jsou KL, CF, SD4CF, SD4KL, a procedura strojového učení KEX. Tyto procedury jsou implementovány v prostředí LISp-Miner a taktéž v prostředí Ferda DataMiner (kromě procedury KEX). Dále jsou zde vysvětleny pojmy *kvantifikátor* a *kontingenční tabulka* pro proceduru 4FT. Přesná znění jejich definic jsou převzaty z práce [5] T.Karbana.

2.4.1 4FT

4FT je dataminingová procedura, která hledá právě vztahy tvaru $Ant \approx Succ$, případně $Ant \approx Succ/Cond$. Ant, $Succ$, a $Cond$ jsou zde booleovské atributy odvozené ze sloupců vstupní datové matice. Výrazu $Ant \approx Succ$ se říká *asociační pravidlo*, výrazu $Ant \approx Succ/Cond$ se říká *podmíněné asociační pravidlo*. Intuitivní význam vztahu $Ant \approx Succ$ a $Ant \approx Succ/Cond$ je ten, že atributy Ant a $Succ$ jsou ve vztahu daném $\approx$, případně je-li splněna podmínka zadaná booleovským atributem $Cond$ na analyzované datové matici.

2.4.2 SD4FT

SDFT je dataminingová procedura, která hledá právě vztahy tvaru $Ant \approx Succ/(\alpha, \beta)$, případně $Ant \approx Succ/(\alpha, \beta, Cond)$. Ant, $Succ$, α, β a $Cond$ jsou zde booleovské atributy odvozené ze sloupců vstupní datové matice. Výrazu $Ant \approx Succ/(\alpha, \beta)$ se říká *SDS asociační pravidlo*, výrazu $Ant \approx Succ/(\alpha, \beta, Cond)$ se říká *podmíněné SDS asociační pravidlo*. Intuitivní význam vztahu $Ant \approx Succ/(\alpha, \beta)$ a $Ant \approx Succ/(\alpha, \beta, Cond)$ je ten, že vztah daný $\approx$ atributů Ant a $Succ$ na množině α se liší od vztahu Ant a $Succ$ na množině β, případně je-li splněna podmínka zadaná booleovským atributem $Cond$ na analyzované datové matici.

2.4.3 Kontingenční tabulka

Pro verifikaci (podmíněných) asociačních pravidel úloh procedury 4FT je využívaná tzv. čtyřpolní kontingenční tabulka (four fold contingency table, zkráceně 4ft – odtud název procedury 4FT). Ta zachycuje počty (frekvence) objektů, které splňují antecedent i sukcedent, splňují pouze jeden, případně ani jeden z nich. Pro pevně zvolený antecedent ϕ, sukcedent ψ a podmínku χ můžeme spočítat frekvence objektů v matici dat M/χ (matice dat splňujících podmínku χ) splňujících antecedent, sukcedent, antecedent i sukcedent nebo ani antecedent ani sukcedent. Frekvence těchto objektů můžeme reprezentovat čtyřpolní kontingenční tabulkou.

M/χ	ψ	$\neg\psi$	$\sum$
ϕ	a	b	r
$\neg\phi$	c	d	s
$\sum$	k	l	n

Tabulka 2.3: Čtyřpolní kontingenční tabulka pro antecedent ϕ, sukcedent ψ a podmínku χ na matici dat M

V tabulce 2.3 je a počet objektů (řádků) analyzované matice dat (splňujících podmínku χ) splňujících jak antecedent, tak sukcedent, b je počet objektů splňujících antecedent a nesplňujících sukcedent, c je počet objektů nesplňujících antecedent a splňujících sukcedent a d je počet objektů nesplňujících ani antecedent ani sukcedent. Poznamenejme, že $r = a + b$ značí počet objektů splňujících antecedent, $s = c + d$ je počet objektů, které nesplňují antecedent. Součty sloupců $a + c = k$ resp. $b + d = l$ odpovídají počtu objektů, které splňují resp. nesplňují sukcedent. Navíc $n = r + s = k + l = a + b + c + d$ je počet všech objektů v matici dat M/χ.

2.4.4 Kvantifikátory

Symbol $\approx$ se nazývá 4ft-kvantifikátor. 4ft-kvantifikátory odpovídají podmínce na čtyřpolní tabulku. Na základě čtyřpolní tabulky se určuje, zda relevantní otázka je či není relevantním tvrzením tj. hypotézou. 4ft-kvantifikátor přiřazuje čtyřpolní tabulce $\langle a, b, c, d\rangle$ číslo z množiny $\{0, 1\}$. Relevantní otázka $\phi \approx \psi/\chi$ je relevantním tvrzením právě tehdy, když $\approx (a, b, c, d) = 1$. 4ft-kvantifikátory mohou vyjadřovat různé vztahy, například implikační, ekvivalenční, mohou mít formu statistického testu atd. Jako příklad uvedeme 4ft kvantifikátor *fundované implikace*.

Fundovaná implikace 4ft-kvantifikátor $\Rightarrow_{p,Base}$ *fundované implikace* pro $0 < p \leq 1$ a $Base > 0$: platí, že $\Rightarrow_{p,Base} (a, b, c, d) = 1$ právě když $\frac{a}{a+b} \geq p$ a $a \geq Base$.

Asociační pravidlo $\phi \Rightarrow_{p,Base} \psi$ je pravdivé právě tehdy, pokud je splněna podmínka výše. Fundovaná implikace může být interpretována například takto: "100% zkoumaných objektů splňujících ϕ splňuje také ψ" nebo méně formálně "ϕ implikuje ψ na 100%". Podmínka na a zajišťuje, že v datové matici M (resp. M/χ) je dostatečný počet objektů splňujících ϕ a ψ (tedy že pravidlo má dostatečnou podporu v datech matice M).

2.5 Základy implementace metody GUHA

2.5.1 Princip bitových řetízků

Typická efektivní realizace GUHA procedury používá reprezentaci databáze, kde každá hodnota každého atributu je reprezentována bitovým řetízkem (viz. kapitola 2.5.1). Každý bit v řetězci popisuje jeden objekt. Bit 1 reprezentuje "objekt má tuto hodnotu", zatímco bit 0 znamená "objekt nemá tuto hodnotu".

Bitový řetízek můžeme chápat jako bitmapový index databáze. Při klasické implementaci atributů se bitové řetízky vytvářejí při prvním průchodu daty a poté není třeba k těmto datům přistupovat. Zde vyvstává otázka, jak efektivně uchovávat potřebné bitové řetízky v paměti. Toto bylo vyřešeno v rámci diplomové práce T.Kuchaře [10] pomocí cache bitových řetízků.

Při implementaci relačního rozšíření pro existující dataminingové procedury se tento princip modifikoval z důvodů toho, že virtuální atributy nejsou

vytvářeny přímo ze sloupců datové matice, ale za běhu – podrobněji pojem virtuálního atributu je rozebrán v kapitole 2.6.2.

Jako ukázku použijeme příklad uvedený v práci T.Karbana [5].

ID	A	B
1	1	1
2	2	2
3	1	3
4	1	1
5	2	3
6	2	3

Tabulka 2.4: Vzorová datová matice

Bitový řetězec $A(\alpha)$ má hodnotu 1 na pozici i právě tehdy, když objekt s ID i má hodnotu α ve sloupci A, jinak má bitový řetězec na pozici i hodnotu 0. Bitové řetězce základních booleovských a booleovských atributů mohou vypadat následovně.

	Atributy		Základní booleovské atributy		Booleovské atributy	
ID	A	B	$A(1)$	$B(1,2)$	$A(1) \wedge B(1,2)$	$\neg B(1,2)$
1	1	1	1	1	1	0
2	2	2	0	1	0	0
3	1	3	1	0	0	1
4	1	1	1	1	1	0
5	2	3	0	0	0	1
6	2	3	0	0	0	1

Tabulka 2.5: Bitové řetězce atributů

2.5.2 Zadání úlohy

Bude následovat popis práce s některými parametry zadání úlohy v prostředí Ferda DataMiner. Většina následujícího popisu se vztahuje i na systém LISp-Miner.

Datový zdroj

GUHA pracuje s daty z datových matic. Ty jsou získávány převážně jako databázové tabulky či výsledky operací nad nimi z databází. Pro tento účel je nutné při zadávaní úlohy definovat datový zdroj (momentálně nejčastěji ve formě přípojení k databázi přes rozhraní ODBC) a alespoň jednu datovou matici. V relačním rozšíření dataminingových procedur GUHA se pracuje nad více tabulkami, které jsou mezi sebou ve vztahu 1:N. Pro pohodlí se o nich zmiňujeme jako o hlavní (master) a vedlejších (detail) tabulkách.

Dále se z datových matic vybírají datové sloupce, ze kterých se budou vytvářet atributy obsahující kategorie pro potřebné rozdělení vstupních dat. Data mohou patřit do různých datových typů. Tyto dále ovlivní možnosti kategorizace. Stojí za zmínku, že zatím nejkomplexnější implementace práce s jednotlivými sémantickými typy byla realizována v práci [10] – "Experimentální GUHA procedury".

Datová sémantika

Z pohledu datové sémantiky se v prostředí Ferda DataMiner rozlišují tyto čtyři druhy dat:

Nominální data Hodnoty nejsou vzájemně porovnatelné, například pohlaví nebo oblíbená příchuť zmrzliny.

Ordinální data Hodnoty jsou vzájemně porovnatelné, například výše dosaženého vzdělání.

Cyklická ordinální data U tohoto sémantického typu se vychází ze zkušenosti, že při zadávaní koeficientu typu *cyklické intervaly* si často nevystačíme pouze s ordinálními daty. Potřebujeme další vlastnost a to, aby první kategorie přirozeně následovala po poslední. Jako příklad mohou posloužit dny v týdnu nebo měsíce v roce.

Kardinální data Tato data jsou reprezentována číselnými typy, například, reálnými čísly. Jako příklad lze zvolit výši příjmu, vzdálenost od místa bydliště, výši krevního tlaku a podobně.

Kategorie

Možnosti práce s kategoriemi se postupně vyvíjely v implementacích metody GUHA. Kategorie může být dvojího typu: buď intervalová – obsahuje konečný počet intervalů, který pokrývá zkoumanou množinu dat, anebo

výčtová, kde výčet opět pokrývá zkoumanou množinu dat. V systémech LISp-Miner a Rel-Miner kategorie může obsahovat buď jenom intervaly anebo jenom výčet. Až v rámci diplomové práce [10] možnosti zadávání kategorie byly přirozeně rozšířeny a implementovány do systému Ferda DataMiner tak, aby mohla obsahovat jak intervaly, tak i výčet. Kategorie tedy obsahuje hodnoty, které jsou pokryty sjednocením výčtu a intervalů v kategorii. Podmínkou pro možnost vytváření intervalů v kategorii je práce nad daty, které mají sémantický typ ordinální, cyklický ordinální nebo kardinální (viz 2.5.2).

V rámci diplomové práce T.Kuchaře [10] byly vytvořeny prostředky pro práci s atributy a kategorii, avšak pro uživatelskou práci byl z důvodu pracnosti implementována pouze krabička pro transformaci atributu "Each value one category", která, jak plyne z názvu, umí vytvořit ze zkoumaného datového sloupce atribut obsahující výčtové kategorie pro každou hodnotu ve sloupci.

Krabičky pro atributy "Ekvidistanční intervaly" a "Ekvifrekvenční intervaly" ve formě krabiček pro uživatelskou práci v systému Ferda DataMiner byly implementovány autorem této diplomové práce, jedná se o přínos autora. Taktéž byla autorem implentována krabička "Statický atribut", která umožňuje přímou úpravu atributu, včetně editace výčtu a intervalů u jednotlivých kategorii.

Výše uvedené krabičky pro transformaci atributu byly přítomny v první verzi Ferdy, avšak diplomová práce T.Kuchaře "Experimentální procedury" podstatně změnila práci se zadáváním úlohy pro procedury GUHA, proto se krabičky musely implementovat znovu.

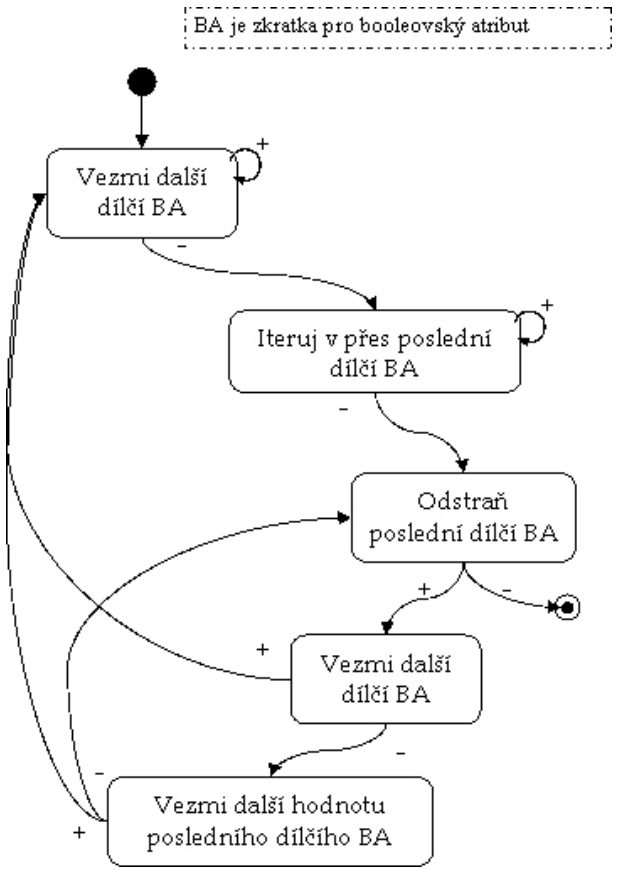
2.5.3 Generování booleovských atributů

Zde pouze stručně naznačíme způsob generování booleovských atributů a poznamenejme, že tento způsob užívaný metodou GUHA se podstatně liší od způsobu, který používá algoritmus APRIORI popsaného v [11]. Způsob generování naznačuje obrázek 2.4.

Podrobný popis generování booleovských atributů v prostředí Ferda DataMiner je uveden v [10]

2.5.4 Ověřování relevantních otázek

Dataminingové procedury GUHA generují relevantní otázky na základě vstupní datové matice a zadaných parametrů úlohy, které poté ověřují na datech z



Obrázek 2.4: Generování booleovských attributů

datové matice. Relevantní otázky, které jsou při ověření na vstupních datech pravdivé, se nazývají hypotézy a posílají se na výstup. Podmínka platnosti relevantních otázek je dána kvantifikátory. Kvantifikátor odpovídá podmínce na kontingenční tabulku.

2.5.5 Generování relevantních otázek

Následující odstavec naznačuje způsob generování a verifikace relevantních otázek. Podrobněji se s principem generování lze seznámit v disertační práci T.Karbana [5], pro konkrétní implementaci v prostředí Ferda DataMiner v diplomové práci T.Kuchaře [10] a taktéž v [3].

Na následujícím pseudokódu lze nahlédnout na způsob generování relevantních otázek u dataminingové procedury 4ft-Miner.

```
foreach(BitString condition in Conditions)
{
    foreach(BitString antecedent in Antecedents)
    {
        foreach(BitString succedent in Succedents)
        {
            ContingencyTable table =
                ComputeTable(succedent, antecedent, condition);
            if (VerifyQuantifiers(table))
                result.Add(succedent, antecedent, condition);
        }
    }
}
```

Objekty `Antecedents`, `Succedents` a `Conditions` jsou iterátory bitových řetězců umožňující průchod přes všechny možné hodnoty zadání booleovských atributů. Ústřední implementační úlohou autora dané diplomové práce byla právě implementace iterátoru bitových řetězců, který by poskytoval bitové řetízky odpovídající virtuálním atributům.

2.6 Relační asociační pravidla a virtuální atributy

2.6.1 Úvod

Zde bude popsáno rozšíření asociačních pravidel pro práci s vícero datovými tabulkami takové, aby byly zachovány všechny vlastnosti metody GUHA a zároveň aby se umožnil popis zajímavých vztahů nalezených na více datových tabulkách. Vlastnosti metody GUHA, zejména zadávání úlohy a implementace pomocí práce s bitovými řetízky zůstanou zachovány. Pro podrobnější popis teorie rozšíření metody GUHA o pravidla pro relační dobývání dat autor doporučuje prostudovat [5]. Protože cílem dané diplomové práce je konkrétní implementace postupu relačního dataminingu, zde se zaměříme pouze na některé nejdůležitější aspekty kritické pro pochopení implementace v rámci prostředí Ferda.

2.6.2 Virtuální atributy

Asociační pravidla se rozšíří o možnost práce s více datovými tabulkami. Datová matice, která byla zkoumána pomocí původních asociačních pravidel,

bude hlavní. Další datové tabulky budou rozšířením hlavní tabulky o nějaký atribut a mohou být obecně ve vztahu 1:N ke hlavní datové matici.

Jako příklad může posloužit databáze klientů banky, kde v hlavní tabulce máme uloženy osobní údaje klientů a například v tabulce transakce jsou uloženy transakce pro účty klientů. Dalším příkladem je databáze pacientů, kde v rozšířující tabulce jsou uložená měření krevního tlaku pro jednotlivé pacienty.

Základním pojmem, o který se rozšíří původní asociační pravidla, je *virtuální atribut*. Virtuální atributy jsou nově přidané sloupce k analyzované datové matici. Tyto sloupce se do původní datové matice zpětně neukládají, avšak dolování dat probíhá, jakoby tyto nově vytvořené sloupce v původní matici byly – vytvářejí se (nebo v případě konkrétní implementace v rámci dané diplomové práce se vytváří pouze jejich reprezentace za pomocí bitových řetízků) za běhu.

Pro názornost uveďme několik příkladů virtuálních atributů. Použijeme databázi klientů fiktivní banky Barbora, kde hlavní tabulka obsahuje základní data o klientech a tabulka transakcí obsahuje pohyb na jejich účtech (použité tabulky jsou podrobně popsány v kapitole 5). Předpokládejme, že nás zajímá typ operace v závislosti na výši transakce. Zkoumejme následující asociační pravidlo:

$$(TYPE = \text{“}VKLAD\text{”})\&(AVGAMOUNT > 5000) \\ \Rightarrow_{0,8;20} OPERATION = \text{“}PREVODNAUCET\text{”} \tag{2.1}$$

Typ transakce je VKLAD, jehož průměrná výše je větší než 5000. $\Rightarrow_{0,8;20}$ značí kvantifikátor fundované implikace, jak je uveden v 2.4.4. Zde jsme použili jeden virtuální atribut $AVGAMOUNT$, který je odvozený jako:

$$AVG(amount) \tag{2.2}$$

kde AVG je agregační funkce SQL. Hypotéza může znamenat zjednodušeně řečeno předpoklad o faktu, že vyšší přísuny peněz na účet probíhají spíše bezhotovostně. Tato hypotéza může a nemusí mít smysl – autor neaspiruje na odborníka v oblasti bankovnictví.

Dále můžeme zkoumat status úvěru klientů v závislosti na nějaké charakteristice plateb. Za platby můžeme považovat transakce, které mají zaplatit nějaké služby – například inkaso apod. Lze předpokládat, že na status úvěru bude mít vliv, zda je klient například "šetrný" nebo naopak "rozhazovačný". Takové charakteristiky zkusíme zjistit na základě dat v matici transakcí. Zde si můžeme všimnout, že požadovaných výsledků lze velmi těžko dosáhnout za použití jedné matice dat – ztratíme tím možnost zkoumat zajímavé vztahy, které platí na vedlejší datové matici. Zkoumané charakteristiky nelze jednoduše přidat do hlavní datové matice. Uvažme další hypotézu:

$$(VYSOKEINKASO)\&(SALARY(>15000))\& \\ (BYDLISTE = \text{“}Praha\text{”}) \Rightarrow_{0,8;10} STATUSUVERU = \text{“}Dobry\text{”} \quad (2.3)$$

Hypotéza tvrdí, že klienti, kteří mají vyšší plat a platí vysoké inkaso (to je příklad výše zmíněné charakteristiky plateb) v okrese Praha, mají z 80% dobré úvěry. Zde jsme použili jeden virtuální atribut $VYSOKEINKASO$, který může vypadat následovně:

$$TYPE = \text{“}INKASO\text{”} \Rightarrow_{0,9;10} AMOUNT > 5000 \quad (2.4)$$

Jinými slovy, pokud typ transakce je INKASO, je z 90% vysoké.

Zde data s informacemi o klientech a data s informacemi o jejich transakcích jsou dvě samostatné datové matice.

Výše uvedené příklady uvádí oba druhy virtuálních atributů, jak jsou chápány v [5]. Oba druhy atributů jsou v rámci dané práce implementovány do prostředí Ferda.

První skupinou jsou sloupce vzniklé jako výsledek předzpracování dat z databáze. Jmenují se *agregační atributy*. Druhou skupinou jsou virtuální atributy vzniklé jako výsledek běhu některé z dataminingových úloh – konkrétně například můžeme použít míru platnosti hypotéz pro jednotlivé záznamy v hlavní tabulce jako nový datový sloupec. Těm říkáme *hypotézové atributy*. Navenek oba druhy atributu pracují s daty z vedlejší datové matice, provádí na ní nějaké transformace a jako výsledek vznikají nové "virtuální" sloupce skládající se obecně z reálných čísel, které jsou přidány do hlavní datové matice.

Agregační atributy

Agregační atribut se reprezentuje novými virtuálními sloupci (které se z definice virtuálního atributu neukládá do databáze), Tyto sloupce vznikají jako výsledek předzpracování dat z více datových tabulek. Pro každý objekt z hlavní datové matice se přiřadí reálné číslo vypočítané na základě hodnot v odpovídajících sloupcích z vedlejších datových matic. Formálněji lze agregační atribut definovat jako funkci nad relevantními sloupci vedlejší datové matice, která vrací reálné číslo.

Přínosem agregačního atributu je zjednodušení zadávání dataminingové úlohy. Výsledku, který přináší agregační atribut, lze dosáhnout i jinak – například předzpracováním dat před samotnou těžbou dat, přidáním požadovaných sloupců do hlavní datové matice apod. Agregační atribut umožňuje pružně měnit zadání úlohy – zjistíme-li například z výsledků po doběhnutí úlohy, že bychom potřebovali změnit zadání a zkoumat jiná data, nemusíme ručně vytvářet potřebné sloupce v hlavní datové matici (které navíc budeme potřebovat pouze pro právě zpracovávanou úlohu a které se budou muset po doběhnutí úlohy opět smazat) a místo toho změníme pouze zadání agregačního atributu.

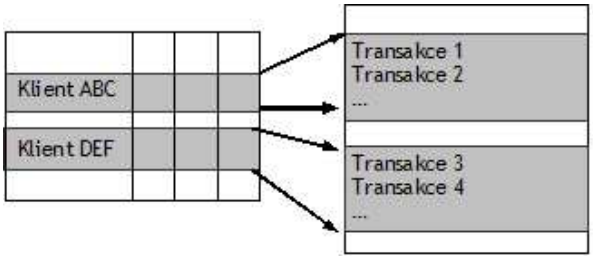
Jako příklady takto uvažovaných funkcí může posloužit mimo jiné množina SQL agregačních funkcí, která vyhovuje výše uvedené definici. Dále můžeme uvažovat množinu obecnějších funkci zpracovávajících data z vedlejších datových tabulek. Příklad implementace takových funkcí můžeme naleznout v [9] – jedná se o implementaci ve formě skriptu napsaného v jazyce C# interpretovaného přímo za běhu. Takto lze implementovat i obecnější agregační atributy než pouze za použití SQL agregačních funkcí.

Hypotézové atributy

Na rozdíl od agregačních atributů, *hypotézové atributy* nelze získat za pomocí předzpracování dat z hlavní a vedlejších datových tabulek. Tyto atributy vznikají jako výsledek běhu dataminingových úloh nad vedlejšími datovými tabulkami a dále se používají při běhu dataminingových úloh nad hlavní datovou tabulkou. Ústřední myšlenkou je zavést definici atributu se sémantikou hypotézy (ve smyslu metody GUHA), jehož hodnotou by byla konkrétně síla nebo platnost hypotézy nad vedlejšími datovými tabulkami a takto nadefinovaný sloupec by se použil "o úroveň výše" – pro generaci hypotéz při běhu další dataminingové úlohy.

Uvedli jsme příklad 2.3. Na základě platnosti hypotézy $VYSOKEINKASO$ můžeme rozdělit množinu klientů a zkoumat další souvislosti pouze na té podmnožině klientů, kde platí hypotéza $VYSOKEINKASO$. Uvažujeme-li platnost hypotézy $VYSOKEINKASO$ jako (virtuální) atribut, můžeme ho zapojit do zadání nové úlohy například pro 4FT jako antecedent nebo sukcedent.

Z příkladu 2.3 vyplývá, že zadání virtuálního atributu je vlastně výsledkem běhu dataminingové úlohy nad vedlejší datovou matici. Tato úloha nad vedlejší datovou matici (budeme jí říkat podúloha) musí proběhnout vícekrát, vždy pouze nad těmi řádky z vedlejší matice, které se vztahují ke zkoumanému objektu z hlavní matice. Toto lze ilustrovat obrázkem 2.5.



Obrázek 2.5: Master-detail tabulka

Zde podúloha proběhne nad transakcemi 1 a 2, výsledek (reálné číslo) přidá do nového virtuálního sloupce na pozici pro klienta ABC. Dále podúloha proběhne znovu nad transakcemi 3 a 4, přidá na pozici pro klienta DEF další reálné číslo a obdobným způsobem pokračuje pro zbytek matice.

Uvažujeme-li o podúloze jako například o proceduře 4FT, víme, že ta generuje relevantní otázky na základě zadání a ověřuje na nich platnost zadaných kvantifikátorů oproti zkoumané datové matici. Procedura 4FT vrací jako výstup pouze relevantní otázky, na kterých platí takto zadané kvantifikátory.

Spustíme-li proceduru 4FT nad relevantními řádky vedlejší datové matice, dostaneme jako výstup platné i neplatné relevantní otázky (uvažujme na chvíli, že 4FT je takto jednoduše upraven). V případě procedury 4FT můžeme

získat booleovský vektor platností jednotlivých relevantních otázek, obecně procedury mohou vracet sílu platnosti jako reálné číslo, dostaneme tedy vektor reálných čísel. Po prozkoumání celé vedlejší datové matice získáme počet vektorů platnosti relevantních otázek rovný počtu řádků v hlavní datové matici – pro každého klienta prozkoumáme relevantní otázky generované podúlohou, které platí na právě jeho transakcích. Počet virtuálních sloupců vygenerovaných virtuálním atributem (podúlohou 4FT) je rovný délce vektoru platnosti relevantních otázek. Chceme-li získat i-tý virtuální sloupec, vezmeme z každého vektoru hodnotu na i-té pozici. Počet vygenerovaných virtuálních sloupců virtuálním atributem se bude rovnat počtu vygenerovaných hypotéz podúlohou. Množina nových virtuálních sloupců se přidá za běhu k hlavní datové matici a hlavní dataminingová úloha s ní pracuje jako s ostatními sloupci.

2.6.3 Nárůst složitosti výpočtu

Z dosavadních zkušeností s prostředím LISp-Miner a Ferda DataMiner víme, že zkoumané úlohy často generují zajímavé vztahy v řádu tisíců. Z výše uvedeného popisu získávání virtuálních sloupců plyne, že pro každý zajímavý vztah vygenerovaný podúlohou se do hlavní datové matice přidá virtuální sloupec. Je vidět, že při používání hypotézového atributu v zadání úlohy pro dataminingovou proceduru lze velmi lehce dosáhnout výrazného zesložitění původní úlohy a rychlému nárůstu velikosti zkoumané datové matice. Tato skutečnost vede ke zpomalení běhu úlohy a ke generování velkého počtu hypotéz, které říkají skoro totéž, protože mají velmi podobné hodnoty hypotézového atributu. Výhodou je větší granularita úlohy a možnost nalezení nejpřesnějšího zadání virtuálního atributu tak, aby platil na co největší podmnožině zkoumaných dat.

2.6.4 Omezení na hvězdicové schéma

V dané diplomové práci se autor rozhodl pro omezení akceptovaného databázového schématu na hvězdicový, kde máme pouze jednu hlavní datovou matici (databázovou tabulku) a vedlejší datové matice (tabulky), které jsou v relaci 1:N s hlavní databázovou tabulkou. Rovněž je zakázána rekurze virtuálních atributů – tj. nelze hypotézový atribut použít v zadání pro další hypotézový atribut.

Teoretické důvody daných omezení jsou podrobněji rozebrány v [5]. Zde se uvede krátké shrnutí z důvodu lepšího porozumění způsobu implementace, který je popsán dále v dané diplomové práci, konkrétně v kapitole 3.

Význam hypotéz generovaných pomocí rekurzivních hypotézových atributů lze jen těžko popsat v přirozeném jazyce. Obtížně se interpretují závislosti nalezené na zkoumaných datech. Autor dané práce se proto domnívá, že hypotézy vzniklé z úlohy obsahující rekurzivní virtuální atributy sice mohou mít dobrý teoretický smysl, avšak vzhledem k celkovému zaměření prostředí Ferda DataMiner na výuku dataminingu pro studenty se nehodí do celkové koncepce tohoto prostředí, ač jsou v tomto prostředí implementovatelné a jsou pro ně autorem vytvořeny programovací prostředky.

Zobrazení výsledků úloh obsahujících rekurzivní virtuální atributy se stává samo o sobě obtížným. Pro každou výslednou hypotézu se musí zobrazit její atributy, které mohou být tvořeny opět hypotézami, které mají své atributy atd. Tento důvod souvisí s předchozím – nejasný význam výsledků vede k potížim jak při interpretaci, tak i při pouhém zobrazení.

Rychlost zpracování úlohy s rekurzivně zapojenými relačními atributy bude velmi rychle klesat. Podrobněji se o tomto pojednává v [5]. Z uvedeného způsobu generování virtuálních sloupců je vidět, že jejich počet může být velký a proto úloha nad hlavní matici se zpomalí. Představíme-li si, že použijeme v zadání hypotézového atributu další hypotézový atribut, pak první podúloha bude zkoumat velký počet sloupců vygenerovaných druhou podúlohou, a opět vygeneruje velký počet relevantních otázek pro nadúlohu. Když uvážíme možnost použití v zadání vícero virtuálních atributu, je vidět, že složitost a doba výpočtu značně stoupnou. Tento problém se nazývá "Hypothesis space explosion", jak o něm pojednává T.Karban v [5].

Z výše uvedených důvodů a dosavadních pokusů s funkční implementaci relačního rozšíření dvou minerů bylo rozhodnuto ponechat možnost rekurzivního zapojení virtuálních atributů pro budoucí vývoj.

2.6.5 Jiné metody relačního dataminingu

Zde se pouze krátce zmíníme o existujících přístupech k dataminingem nad více tabulkami dat.

Nejrozšířenějším metodou relačního dobývání znalostí je metoda *induktivního logického programování (ILP)*. Princip ILP může být popsán následovně: z databáze faktů a očekávaných výsledků, které jsou rozděleny na

kladné a záporné příklady, metody pracující na základě ILP se snaží odvodit logický program, který dokáže všechny kladné a žádný záporný příklad.

Významným zástupcem metod ILP, které pracují s metodami relačního dataminingu je algoritmus WARMR. Je rozšířením algoritmu APRIORI pro hledání relačních asociačních pravidel. Detailní porovnání relačního rozšíření metody GUHA a metody WARMR provádí T.Karban jako součást své disertační práce [5].

Kapitola 3

Řešení

V této kapitole budou popsané obecné charakteristiky programátorské implementace v rámci této diplomové práce.

3.1 Vztah k práci "Experimentální GUHA procedury"

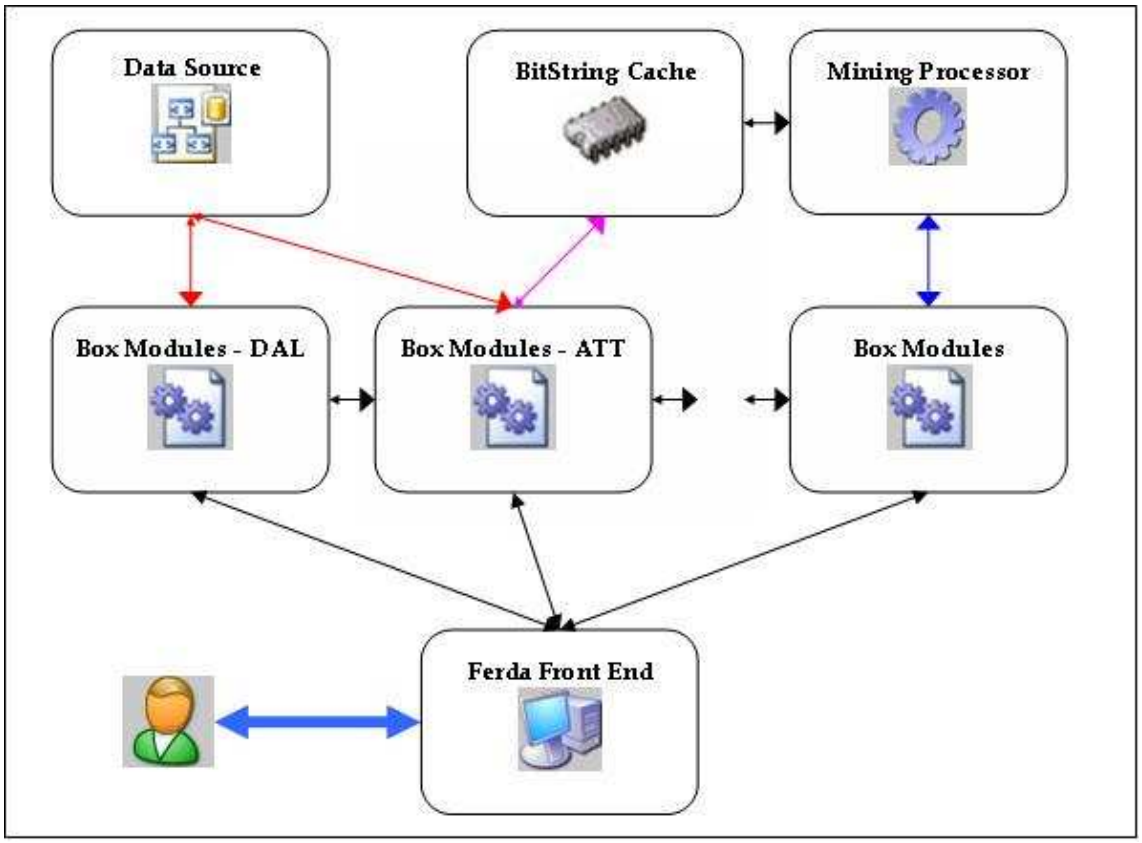
Zde se podrobně uvede vztah diplomové práce T.Kuchaře "Experimentální GUHA procedury" uvedené v seznamu použité literatury jako [10].

Jak bylo podrobně popsáno v kapitole 2.2.3, Ferda DataMiner vznikl jako studentský projekt s cílem implementace obecnější platformy pro dobývání znalostí se základním prvkem pojmenovaným krabička, viz [8] pro podrobnější informace.

V době jeho vzniku již byly získány rozsáhlé uživatelské zkušenosti se systémem LISp-Miner a Ferda DataMiner byl navržen s ohledem na tyto zkušenosti. V rámci studentského projektu byly implementovány procedury metody GUHA s použitím existujících modulů prostředí LISp-Miner. To přineslo řadu potíži podrobně popsaných T.Kuchařem v [10] a proto bylo evidentní, že procedury GUHA se musí do prostředí Ferda DataMiner přidat bez využití LISp-Mineru.

Implementaci šesti GUHA procedur a modulů pro zadání úlohy implementoval T.Kuchař ve své diplomové práci "Experimentální GUHA procedury", jak už zde bylo několikrát zmíněno. Pro implementaci relačních rozšíření existujících procedur byly konkrétně použity nebo upraveny následující prvky

implementované T.Kuchařem. Pro ilustraci uvádíme obrázek 3.1 ilustrující architekturu systému Ferda DataMiner, jak ho uvádí T.Kuchař v [10].



Obrázek 3.1: Architektura prostředí Ferda DataMiner

Společná databázová vrstva byla použita pro implementaci agregačního a hypotézového atributem. Některé metody této vrstvy musely být upraveny pro použití s relačními rozšířeními. Byly mj. přidány prostředky pro práci s tzv. *agregačním virtuálním sloupcem* ve smyslu sloupce generovaného při použití agregačních funkcí a také prostředky pro generování tzv. *CountVectoru*, jehož význam je objasněn později v kapitole 4.

Prostředky "Mining Processor" byly využity pro implementaci hypotézového atributu. Relační rozšíření využívají zadání úlohy stejně jako

nerelační procedury 4FT a SD4FT. Byla upravena vrstva generování výsledků, aby místo pravdivých hypotéz poskytovala všechna asociační pravidla, jak je popsáno v úvodu v kapitole 2.6.2 a dále byla upravena mezivrstva poskytování bitových řetízků, aby zajistila práci s vlastní cache bitových řetízků pro virtuální atributy a aby zajistila získávání bitových řetízků na odlišném principu, než byl implementován v [10]. Dále byly provedeny úpravy pro snazší implementaci relačních rozšíření zbývajících procedur.

BitString Cache nebyl používán pro hypotézové atributy, místo toho se pro ně implementoval odlišným způsobem, jak bude objasněno v kapitole 4.

Krabičky DAL byly rozšířeny o krabičku "Agregační virtuální sloupec", která implementuje agregační atribut ve smyslu relačního dobývání znalostí. Krabička má stejné vlastnosti a zásuvky jako krabička "Sloupec" implementována v rámci [10] a přidává některé nové prvky, které jsou podrobněji ukázány v kapitole 4.

Krabičky ATT byly rozšířeny o krabičky pro transformaci atributu a krabičky hypotézových atributů. Krabičky pro transformaci atributu využívají algoritmů pro generování ekvidistačních a ekvifrekvenčních intervalů implementovaných v rámci [10].

Podrobný seznam vytvořených a změněných souborů se zdrojovým kódem je uveden v kapitole 6.

Možnost snadné integrace a využití existujících prostředků byla od začátku jedním z hlavních cílů projektu Ferda DataMiner. Z tohoto a dalších důvodů uvedených výše se odvíjely zásady implementace.

3.2 Zásady implementace

Zde bude popsán obecný programátorský přístup k řešení implementace.

3.2.1 Licence

Protože prostředí Ferda DataMiner je vyvíjeno pod licenci GNU Public License (GPL), implementace v rámci dané diplomové práce je rovněž vyvíjeno pod licenci GPL. Daná licence umožňuje nejen volné použití programu, ale také zaručuje volný přístup ke zdrojovému kódu programu a jeho rozšiřování

pod podmínkou, že výsledný kód bude rovněž pod GPL licenci. Plný text GNU Public License lze nalézt na `http://www.gnu.org/licenses/gpl.txt`.

3.2.2 Platforma

Implementace v rámci dané diplomové práce byla napsána v jazyce C# stejně jako stávající implementace prostředí Ferda DataMiner. Jedním z cílů (už od doby studentského projektu) je možnost práce s prostředím Ferda DataMiner na více platformách. Pod OS Windows lze využít platformu Microsoft .NET Framework (`http://www.microsoft.com/cze/net/`), na jiných operačních systémech (například Linux nebo MacOS) pak prostředí Mono (`http://www.mono-project.com`) .

Prostředí Ferda DataMiner bylo navrženo modulárně. Jeho jednotlivé části mezi sebou komunikují přes middleware Internet Communications Engine ICE (`http://www.zeroc.com`), což je moderní alternativa podobnému frameworku CORBA a podporuje několik rozšířených programovacích jazyků, mezi které patří také C#.

3.2.3 Výkonnostní testy

Součástí dané diplomové práce jsou testy relačních rozšíření dataminingových procedur. Testy se zaměří na hranice únosnosti zadání úlohy ve smyslu doby běhu, na vliv velikosti zpracovávané datové matice na běh relačních rozšíření a také na důležitost využití cache pro generované bitové řetízky virtuálním atributem. V úplně původním návrhu implementace se s cache vůbec nepočítalo z důvodů uvedených dále, proto testy zahrnují i běh úlohy s vypnutou cache pro ilustraci zdůvodnění její zavedení navzdory původnímu návrhu. Testy jsou popsány v kapitole 5.

3.2.4 Dokumentace

Uživatelská dokumentace

Uživatelská dokumentace je zahrnutá v textu práce jako popis implementovaných krabiček v kapitole 4 .

Programátorská dokumentace

Návrh řešení implementace je součástí textu této diplomové práce. Zásahy do architektury prostředí Ferda DataMiner byly popsány výše, diskuze nad

implementaci konkretních metod je popsána dále v tomto textu. Poznámky přímo k detailům implementace jsou součástí zdrojového kódu. Tato dokumentace může být kdykoliv vygenerována ve zvláštním souboru.

3.3 Agregační atribut

3.3.1 Diskuze nad implementaci

Při rozhodování o rozsahu implementované funkčnosti agregačního virtuálního atributu se bral v potaz fakt, že daleko zajímavějším atributem pro využití v relačním dobývání znalostí je atribut hypotézový. Proto po dohodě s vedoucím práce bylo rozhodnuto implementovat pouze základní funkčnost agregačního virtuálního atributu. Základní funkčnost zde znamená možnost projekce vybraného sloupce z vedlejší datové matice na základě zadaných klíčů pro spojení hlavní a vedlejší datové tabulky, přičemž sloupec může být specifikován nejen názvem, ale i vlastním výrazem SQL, který může obsahovat například použití agregačních funkcí.

Autor dané diplomové práce si uvědomuje fakt, že tento atribut může být obecnější než pouze spojení dvou databázových tabulek dle klíče a použití funkcí SQL. V diplomové práci M.Ducháčka uvedené v seznamu použité literatury jako [9] je implementován obecnější přístup k agregačnímu virtuálnímu atributu. Ve zmíněné práci je umožněno na tomto atributu spouštět nejen agregační funkce jazyka SQL, ale obecně lze provádět jakoukoliv úpravu dat pomocí procedur napsaných v jazyce C#, jejichž kód je interpretován za běhu.

Musíme si však také uvědomit, že výše uvedená funkčnost tvoří velmi podstatnou část diplomové práce M.Ducháčka. Vzhledem k faktu, že autor této diplomové práce měl před sebou implementaci jak hypotézového atributu, tak i tří dalších modulů pro dynamickou transformaci atributu a jednoho modulu pro statickou úpravu atributu, bylo rozhodnuto naprogramovat v rámci agregačního hypotézového atributu pouze výše uvedené vlastnosti.

3.4 Hypotézový atribut

3.4.1 Diskuze nad implementaci

Při implementaci muselo být vyřešeno několik problému. Jejich řešení bylo podrobně diskutováno s vedoucím diplomové práce přibližně v následujících

bodech.

Pro začátek si ještě jednou připomeneme základní algoritmus generování asociačních pravidel a ověřování hypotéz pro dataminingovou proceduru 4FT.

```
foreach(BitString condition in Conditions)
{
    foreach(BitString antecedent in Antecedents)
    {
        foreach(BitString succedent in Succedents)
        {
            ContingencyTable table =
                ComputeTable(succedent, antecedent, condition);
            if (VerifyQuantifiers(table))
                result.Add(succedent, antecedent, condition);
        }
    }
}
```

Objekty `Antecedents`, `Succedents` a `Conditions` jsou v prostředí Ferda DataMiner naprogramovány jako iterátory poskytující bitové řetízky a tím umožňují průchod přes všechny možné hodnoty zadání booleovských atributů. Při implementaci relačních rozšíření stávajících procedur bylo nutné zajistit poskytování bitových řetízků pro tyto iterátory (zde uvedený pseudokód lze chápat jako běh nadúlohy). Podúloha generuje virtuální sloupce a je potřeba je napojit na výše zmíněné iterátory.

Způsob běhu podúlohy

Během diskuze se diskutovaly následující tři přístupy, jak může podúloha běžet.

Předběžné generování Podúloha by se spouštěla předem na vedlejší datové matici. Výsledky ve formě buď virtuálních sloupců a(nebo) bitových řetízků by se přechodně skladovaly v paměti či ve formě serializovaných objektů a při následném běhu nadúlohy by se této předložily. Podobným způsobem jsou implementovány například stávající procedury pro transformaci atributu – "Each value one category", "Ekvidistanční intervaly" a "Ekvifrekvenční intervaly".

Výhodou takového přístupu by byla možnost využít stávající výhody implementace metody GUHA v prostředí Ferda DataMiner a to zejména cache bitových řetízků. Dále by se mohly bez větších změn použít dataminingové procedury jako podúlohy, v podstatě by se jednalo o jejich spuštění na podmnožině datové matice místo na celé množině řádků. Také takto implementovaný virtuální atribut by šel využit pro zadání pro všechny dosud implementované GUHA dataminingové procedury v prostředí Ferda DataMiner.

Hlavním důvodem napsání předchozího odstavce v podmiňovacím způsobu se stala nemožnost škálovatelnosti takového řešení. Už i pro existující vzorová data virtuální banky Barbora mohl nastat problém se zabíranou paměti pro vygenerované bitové řetízky. Počet asociačních pravidel vygenerovaných v zadané dataminingové úloze v řádu tisíců či desítek tisíc je běžnou záležitostí. Protože každé asociační pravidlo v podúloze znamená nový virtuální sloupec v hlavní tabulce, nad kterou běží hlavní dataminingová úloha, při úvaze o velikosti požadované operační paměti (protože bitové řetízky krabičky atributů ve Ferda DataMiner uchovávají právě tam) se musí počet sloupců vynásobit počtem řádků hlavní datové matice. V relačních rozšířeních procedur v této diplomové práci se generují virtuální sloupce s hodnotami *true* nebo *false*, proto výše spočítané číslo by znamenalo počet bitů potřebných pro uchování virtuálních sloupců, ale obecně virtuální atributy mohou generovat sloupce s reálnými hodnotami a velikost potřebné paměti se tak zvýší.

Při úvaze s matici banky Barbora o velikosti řádově kolem 5000 řádku a počtem asociačních pravidel v podúloze kolem 10000 nám vzniká ještě vcelku přijatelná velikost zabírané RAM, nicméně je třeba si uvědomit, že velikost zkoumané datové tabulky pro úlohy blížící se reálné situaci je spíše v řádech statisíců až miliónů záznamů. Toto je zásadní omezení pro budoucí aplikace prostředí Ferda DataMiner a takto navržený způsob implementace by omezil nejen relační rozšíření procedur implementovaných v rámci dané diplomové práce, ale i případných dalších implementací relačních rozšíření v budoucnu.

Z výše uvedených důvodů bylo tedy od tohoto způsobu implementace upuštěno.

Čekající procedury Po zavržení předchozího přístupu bylo jasné, že řešení předem generovat bitové řetízky do paměti nepřipadá v úvahu. Začal se hledat způsob, jak virtuální sloupce a tedy i bitové řetízky, generovat průběžně, vždy na pokyn nadúlohy.

Vznikla následující úvaha: protože nadúloha potřebovala ke zpracování celý sloupec hlavní datové matice najednou, cyklus podúlohy poskytující bitové řetízky by mohl běžet v samostatném vlákně. Podúloha by jako vstupní matici obdržela podmnožinu řádků vedlejší datové matice pro jeden objekt z hlavní datové matice. Tedy pro každý objekt z hlavní datové matice by běželo jedno vlákno podúlohy, které by postupně vracelo hodnoty *true* nebo *false* (nebo obecně reálná) čísla, ze kterých by se skládaly virtuální sloupce.

Na první pohled je však jasné, že by musel běžet počet vláken stejný jako počet řádků v hlavní datové matici. Idea řádově tisíce vláken pro výpočet byť jednoduché úlohy (i nad tabulkou virtuální banky Barbory – viz kapitola 5) se autorovi zdála jako velmi špatný návrh z důvodu příliš velké režie operačního systému na přepínaní takového počtu vláken. Z dosavadních zkušeností bylo vypozorováno, že na běžně dostupném osobním počítači řádově i stovka vláken většinou představuje velké vytížení komponent systému.

Z výše uvedených důvodů bylo tedy od tohoto způsobu implementace upuštěno.

Generování po krocích Předchozí úvaha však inspirovala autora práce k jinému přístupu k řešení. Jak je vidět z algoritmu generování asociačních pravidel, který je uveden na počátku této sekce, dataminingová procedura (v tomto případě nadúloha) potřebuje v jednotlivých krocích svých cyklů pouze *jeden bitový řetízek*. Z tohoto vznikla idea navrhnout dataminingovou proceduru pro podúlohu tak, aby uměla poskytnout na požádání ne nějaký konkrétní sloupec nebo bitový řetízek, nýbrž *další* sloupec a *další* bitový řetízek anebo nic v případě vyčerpání možných asociačních pravidel (a tím i virtuálních sloupců).

Zásadní rozdíl oproti dosud existujícím (nevirtuálním) atributům byl ten, že tyto měly celou množinu svých bitových řetízků v okamžiku

před začátkem generování známou. Bitové řetízky jsou v těchto atributech identifikovatelné dle klíče skládajícího se z identifikátoru atributu a názvu kategorie. Je znám celkový počet bitových řetízků, tudíž lze stanovit počet celkově vygenerovaných asociačních pravidel. Je možné přistoupit k jakémukoliv bitovému řetízku na základě klíče.

Oproti tomu navrhovaný způsob implementace hypotézového atributu zaručoval pouze to, že takový atribut v okamžiku zavolání příslušné metody buď vydá další bitový řetízek nebo oznámí, že bitové řetízky už vydávat nebude. Není tedy známá a snadno přístupná celá množina bitových řetízku před začátkem generování (nadúlohou) asociačních pravidel. Tudíž nelze spočítat ani předpokládaný počet vygenerovaných asociačních pravidel, protože hypotézový atribut jednoduše reaguje buď tím, že dá další bitový řetízek nebo oznámí, že řetízky došly.

Výhody tohoto řešení ve smyslu škálovatelnosti jsou zřejmé. Podúloha běží synchronně s nadúlohou a v paměti se uchovává a předává pouze jeden virtuální sloupec a jeho bitový řetízek v jednom okamžiku. Poté, co nadúloha přejde na další krok generování a požádá virtuální atribut o další bitový řetízek, tento zavolá svoji metodu *MoveNext* a vydá další řetízek. Předchozí vygenerované řetízky se nikde neukládají a nezabírají paměť.

Nevýhody tohoto řešení jsou již naznačeny výše a budou podrobněji probrány dále. Po diskuzi s vedoucím diplomové práce se autor rozhodl implementovat relační rozšíření dataminingových procedur právě tímto způsobem, protože případné nevýhody a jiný přístup k řešení znamenaly spíše nepohodlí pro programátora, avšak nijak nenarušovaly princip metody GUHA jako takové. Navíc poskytovaly základní výhodu škálovatelnosti – z principu návrhu lze zaručit, že úloha skončí i na velkých tabulkách, nevyčerpá RAM nebo počet povolených běžících vláken v systému. Může se stát, že poběží delší dobu, ale skončí korektně.

3.4.2 Důsledky zvoleného způsobu

Zde budou popsány důsledky, které mělo zvolené řešení implementaci a spolupráci s již existujícími moduly metody GUHA v prostředí Ferda DataMiner.

Vypnutí cache bitových řetízku

Nejpodstatnějším důsledkem bylo neumožnění používat stávající cache bitových řetízků. Ta totiž funguje tak, že uchovává bitové řetízky pro atributy (pro všechny atributy je společná) a v případě potřeby vyřazuje z cache málo používané bitové řetízky na základě nějaké strategie. V případě, že v cache potřebný bitový řetízek neexistuje, požádá o něj přímo potřebný atribut na základě jednoznačného identifikátoru.

Jak už bylo zmíněno výše, takový přístup by se zvoleným způsobem implementace nefungoval – předpokládá totiž možnost přístupu ke všem bitovým řetízkům atributu. Návrh hypotézového virtuálního atributu s tímto nepočítá – umí dát pouze "další" bitový řetízek bez možnosti zjistit, který další to bude, jaký bude mít identifikátor, ke které kategorii bude patřit atd. Z těchto důvodů nešlo stávající cache pro hypotézový virtuální atribut použít. Použití cache bitových řetízků u ostatních buď existujících nebo nově implementovaných krabiček atributů zůstalo beze změny, protože se pro tyto krabičky nic nezměnilo.

Rozdělení na krabičky

Další otázkou bylo, zda implementovat hypotézový virtuální atribut jako dvě krabičky nebo jednu společnou. Stávající nevirtuální atributy se v případě potřeby generování booleovských atributů zapojují do krabičky "AtomSetting". Ta zajistí generování koeficientů pro booleovský atribut.

V případě hypotézového virtuálního atributu by nebylo možné použít stávající krabičku pro generování koeficientů "AtomSetting", protože pracuje s požadavkem "poskytni přesně tento bitový řetízek", kdežto hypotézový atribut by uměl zpracovat pouze požadavek "poskytni další bitový řetízek". Tudíž krabička pro generování koeficientů využitelná pouze pro hypotézový virtuální atribut by se musela také naprogramovat.

Po úvaze se autor rozhodl neimplementovat zvláštní krabičku pro generování koeficientů pro hypotézový virtuální atribut z následujících důvodů.

Potřeba pouze jednoho koeficientu Bylo rozhodnuto, že při generování koeficientů pro booleovský atribut bude potřeba implementovat pouze podmnožiny délky jedna. Krabička "AtomSetting" pro hypotézový virtuální atribut by byla nadbytečná, protože by tam chybělo uživatelské nastavení, vše se odehrávalo "pod kapotou". Nebyl tedy důvod tuto krabičku vytvářet.

Zbytečná komunikace krabiček přes rozhraní ICE Navíc by tato krabička, která by se zapojila za hypotézový virtuální atribut, znamenala komunikaci přes rozhraní ICE. Protože idea implementace spočívá v synchronním běhu nad- a podúlohy, kde nadúloha uvnitř svých cyklů (například pro sukcedent) volá podúlohu, která jí předává bitové řetízky, je potřeba zajistit toto předávání plynule, aby se hlavní úloha příliš nezpomalovala čekáním na bitový řetízek od podúlohy.

Protože ze zkušeností získaných při programování prostředí Ferda DataMiner ještě v rámci studentského projektu autor věděl, že komunikace krabiček přes rozhraní ICE je relativně pomalá (s ohledem na skutečnost, že na komunikaci přes ICE by bylo třeba čekat při každém kroku `foreach` cyklu), bylo rozhodnuto zajistit generování atributu a koeficientu v jedné krabičce a tím zrychlit běh hypotézového virtuálního atributu.

Buffer bitových řetízků

Při ladění prvotní implementace hypotézového virtuálního atributu se zjistilo, že ačkoliv zvolený návrh zaručí škálovatelnost a jistotu toho, že úloha byť za delší dobu doběhne a nezabere zbytečnou paměť, doba běhu celkové úlohy je neúměrně dlouhá.

Důvod dlouhého běhu úlohy s virtuálním atributem celkem logicky spočíval v neexistenci jakéhokoliv cacheování bitových řetízků poskytovaných virtuálním hypotézovým atributem. Při prvotním návrhu autor a vedoucí diplomové práce na tento fakt nebrali příliš ohled, protože spíše diskutovali o dlouhodobějších dopadech zvoleného způsobu implementace. Velký rozdíl v rychlostech běhu úlohy byl ještě umocněn faktem, že všechny dosavadní GUHA procedury v prostředí Ferda DataMiner byly implementovány s použitím cache bitových řetízků a generování bitových řetízků při běhu úlohy proběhlo pouze poprvé a poté se načítaly z cache. Proto hypotézový atribut, který žádnou cache nepoužíval, způsoboval opravdu velké zpomalení běhu úlohy, byl-li do ni zapojen.

Podíváme-li se na pseudokód algoritmu procedury 4ft-Miner uvedený výše a představíme-li si ho jako nadúlohu, do které bude zapojen minimálně jeden virtuální hypotézový atribut (například jako sukcedent), tak bude jasné, že tento algoritmus poběží opravdu pomalu. Zapojení virtuálního atributu do sukcedentu je pro tento miner "nejhorší případ", protože bitstringy jim

generované jsou používány ve vnitřním `foreach` cyklu, a to v případě nepřítomnosti cache znamená, že uvnitř tohoto cyklu se *pokaždé* spustí podúloha (vnitřní procedura), aby mohla nadúloze poskytnout bitový řetízek.

Při vší snaze autora dané práce jakkoliv urychlit běh úlohy se zapojeným hypotézovým atributem vycházelo najevo, že od původně plánované implementace prosté jakéhokoliv cacheování bude nutné upustit. Při krátkém otestování a optimalizaci dosud implementovaného kódu krabičky autor byl schopen dosáhnout rozumné doby běhu úlohy (v řádu desítek minut) při omezení maximálního počtu virtuálních sloupců a potažmo i bitových řetízků vygenerovaných krabičkou virtuálního atributu na 15-20. Při zvýšení maximálního počtu virtuálních sloupců na 50 doba běhu jednoduché úlohy se pohybovala v řádu hodin.

Protože běžné počty vygenerovaných asociačních pravidel v procedurách GUHA se pohybují v řádech tisíců až desetitisíců při potřebné době běhu spíše v řádu minut, bylo zřejmé, že zpomalujícím faktorem byla právě absence cacheování bitových řetízků pro virtuální atribut. Bylo tedy rozhodnuto upustit od kanonického řešení a autor naimplementoval cache i pro virtuální atributy.

3.5 Atributy

Zde bude popsána podstata úlohy, kterou bylo nutné vyřešit pro implementaci atributů v prostředí Ferda DataMiner. Podrobnější popis principů fungování daných atributů je uveden v dané diplomové práci v sekci 1.2.3.

3.5.1 Ekvidistanční a ekvifrekvenční intervaly

Při implementaci tohoto modulu pro transformaci atributu byly využity algoritmy pro vytvoření ekvifrekvenčních intervalů připravené T.Kuchařem v rámci jeho diplomové práce uvedené jako [10]. Tyto algoritmy byly adaptovány z prostředí projektu Rel-Miner T.Karbana a připraveny pro využití v prostředí Ferda DataMiner. Autor této diplomové práce vytvořil pro dané moduly pro transformaci atributu krabičku v prostředí Ferda DataMiner, jejíž funkčnost bude detailně popsána v kapitole 4.

Krabička pro vytváření ekvidistančních intervalů pracuje mírně odlišným způsobem než v systému LISp-Miner. Intervaly se vygenerují na základě zadání požadovaného *počtu* intervalů a počátečního a koncového bodů. V

systému LISp-Miner se zadá požadovaná *délka* intervalu a počáteční a koncový body. Krabička implementující transformaci atributu vytvářením ekvidistančních intervalů stejným způsobem jako v systému LISp-Miner byla rovněž přidána do prostředí Ferda DataMiner v rámci dané diplomové práce.

3.5.2 Statický atribut

Při implementaci tohoto modulu pro transformaci atributu byla vytvořena krabička, která umožňovala spouštění modulu pro ruční úpravu atributu. Atribut (přesněji kategorie v atributu) mohly být předem vytvořeny například výše uvedenými krabičkami pro tvorbu ekvidistančních nebo ekvifrekvenčních intervalů.

3.5.3 Modul pro úpravu atributu

Tento modul je spouštěn nad krabičkou "Statický atribut" a umožňuje ruční úpravu kategorií v atributu. Při ní uživatel může jak upravovat již předem připravené kategorie, tak i vytvářet kategorie nové. Upravovaný atribut může být i zcela prázdný a kategorie mohou být vytvořeny uživatelem kompletně od začátku.

Modul zaručuje korektnost ručně přidaných kategorií (zaručí například disjunktní intervaly v rámci kategorie, jak je požadováno při tvorbě zadání úlohy pro dataminingovou proceduru). Využívá algoritmů již implementovaných v rámci diplomové práce T.Kuchaře uvedené jako [10]. Funkčnost tohoto modulu bude detailně popsána v kapitole 4.

Kapitola 4

Implementace

V této kapitole budou uvedeny a popsány implementované procedury a krabičky včetně uvedení některých nejdůležitějších detailů implementace.

4.1 Agregační atribut

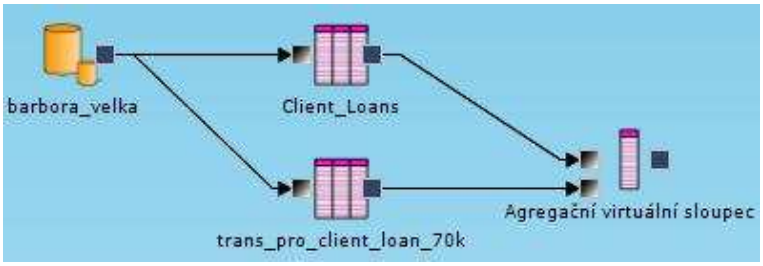
Podrobnější popis účelu virtuálního agregačního atributu byl uveden v předchozích kapitolách. Zde podrobně probereme implementovanou funkčnost včetně popisu vytvořené krabičky.

4.1.1 Detaily implementace

Agregační virtuální atribut poskytuje možnost spojit dvě datové matice (databázové tabulky) na základě zadaného klíče a volby výsledného sloupce z vedlejší datové matice. Výsledný sloupec lze zapojit do dataminingové úlohy GUHA stejně jako výsledný sloupec, který poskytuje krabička "Sloupec". Dále daný atribut může využívat možnosti práce s SQL funkcemi díky použití stejného postupu jako při nadefinování odvozeného sloupce. Sloupec lze získat i jako výsledek SQL funkce, kterou lze zadat při volbě názvu sloupce.

4.1.2 Krabička

Daná krabička reprezentuje výše popsaný agregační virtuální atribut z uživatelského hlediska (obrázek 4.1).



Obrázek 4.1: Agregační virtuální sloupec

Seznam zásuvek

Hlavní datová matice Do této zásuvky se zapojuje krabička reprezentující hlavní datovou matici.

Vedlejší datová matice Do této zásuvky se zapojuje krabička reprezentující vedlejší datovou matici.

Seznam vlastností

Sémantika Umožní vybrat sémantiku sloupce. Sémantika v prostředí Ferda DataMiner byla podrobněji popsána v 2.5.2. Uživatel může tedy vybrat z možností *Nominální, Ordinální, Cyklický ordinální, Kardinální.*

Id sloupec hlavní datové tabulky Zde se vybere sloupec ze seznamu nabízených sloupců hlavní datové tabulky. Záznamy z daného sloupce budou použity jako klíče pro záznamy z hlavní datové tabulky při spojení hlavní a vedlejší datové tabulky.

Id sloupec vedlejší datové tabulky Zde se vybere sloupec ze seznamu nabízených sloupců hlavní datové tabulky. Záznamy z daného sloupce budou použity jako klíče pro záznamy z vedlejší datové tabulky při spojení hlavní a vedlejší datové tabulky.

Seznam vlastností pouze pro čtení

Vlastnosti pouze pro čtení obecně v prostředí Ferda DataMiner slouží k informování uživatele o vlastnostech zkoumané krabičky.

Vlastnosti pouze pro čtení pro danou krabičku jsou shodné s vlastnostmi pro čtení krabičky "Sloupec".

Datový typ Zobrazí datový typ hodnot záznamů ve sloupci.

Průměr Zobrazí průměr hodnot záznamů ve sloupci.

Navzájem různé hodnoty Zobrazí počet záznamů s navzájem různými hodnotami ve sloupci.

Maximum Zobrazí maximální hodnotu ze všech hodnot záznamů ve sloupci.

Minimum Zobrazí minimální hodnotu ze všech hodnot záznamů ve sloupci.

Standardní odchylka Zobrazí hodnotu standardní odchylky ze všech hodnot záznamů ve sloupci.

Rozptyl Zobrazí hodnotu variability ze všech hodnot záznamů ve sloupci.

4.2 Hypotézový atribut

Podrobnější popis účelu virtuálního hypotézového atributu byl uveden v předchozích kapitolách. Zde rozebereme implementovanou funkčnost, objasníme použité postupy a dále popíšeme vytvořené krabičky.

4.2.1 Detaily implementace

Hypotézový virtuální atribut generuje virtuální sloupce (přesněji přímo bitové řetízky) na základě zadání podúlohy. Tato se zadává obdobně jako běžná dataminingová úloha pro již existující procedury. Rozdílem je především poskytovaný výsledek. Hypotézový virtuální atribut neposkytuje výsledek pro prohlížení uživatelem (na rozdíl od běžných nerelačních procedur), ale pro nerelační dataminingovou proceduru, do zadání které je zahrnut jako booleovský atribut.

Jako součást implementace hypotézového virtuálního atributu v prostředí Ferda DataMiner byly vytvořeny krabičky pro 4ft virtuální hypotézový atribut a SD4ft virtuální hypotézový atribut. Rovněž byly provedeny úpravy některých modulů, které používají krabičky dataminigových procedur.

Jak už bylo zmíněno v 3.4.1, krabičky 4FT a SD4FT virtuálního atributu generují rovnou bitové řetízky pro druh koeficientu "jednoprvková podmnožina". V prostředí Ferda DataMiner proto vystupují na místě krabičky jak "AtomSetting", tak krabičky atributu.

Generování kolekce

Při implementování dle návrhu z 3.4.1 se autor rozhodl využít nových vlastností, které nabízel jazyk C# ve verzi 2.0 a to konkrétně zavedení klíčového slova `yield` pro velmi snadné implementování iterátoru (více viz například na `http://msdn2.microsoft.com/en-us/library/9k7k7cf0.aspx`).

Tato vlastnost nového C# usnadnila implementování přístupu "poskytni další bitový řetízek", protože velmi zjednodušeně řečeno umožňuje vrátit kolekci prvků, kterou lze procházet pomocí cyklu `foreach`, aniž by byly dopředu známé prvky v dané kolekci. Ta se totiž generuje dynamicky, což se nám hodí právě při implementaci kolekce bitových řetízků, které jsou získávány metodou "poskytni další bitový řetízek".

Count vector

Pro generování bitového řetízku správné délky (tzn. stejné délky jako bitové řetízky v nadúloze) je nutné vědět, kolik řádků z vedlejší datové matice se váže ke každému objektu z hlavní datové matice. Potom bude možné generovat virtuální sloupec správné délky – viz 3.4.1.

Tato informace je získávána pomocí tzv. *CountVectoru*, který na i-té pozici obsahuje počet řádků vedlejší datové matice patřících i-tému objektu z hlavní datové matice (to samozřejmě předpokládá setřídění obou matic dle stejného klíče).

Za použití *CountVectoru* lze generovat celý virtuální sloupec, potažmo odpovídající bitový řetízek, již poměrně snadno. Podúloha generující virtuální sloupce jako výsledek platnosti relevantních otázek, bude počítat na podmnožině řádků zjištěné z informace dané *CountVectorem*. Před spuštěním generování relevantních otázek se pro každý objekt z hlavní datové matice vytvoří tzv. *masky*, což jsou bitové řetězce s délkou rovnající se počtu záznamů ve vedlejší datové matici a jejich počet odpovídá počtu objektů v hlavní datové matici. Každý objekt z hlavní datové matice má vytvořenou svoji masku. Bitový řetízek masky pro i-tý řádek hlavní datové matice obsahuje jedničky na pozicích odpovídajících umístění záznamů z vedlejší datové matice vztahujících se k i-tému řádku hlavní datové matice. Například, máme-li v tabulce klientů dva klienty a v tabulce transakcí 10 transakcí, 4 pro prvního klienta a 6 pro druhého, pak vzniknou dvě masky o délce 10 bitů. Maska pro prvního klienta bude obsahovat jedničky na pozicích 0-3 a jinak nuly a maska pro druhého klienta bude obsahovat jedničky na pozicích 4-9 a jinak nuly.

Takto vygenerované masky se aplikují na bitové řetízky generované atributy v zadání podúlohy operací AND, čímž se vynulují hodnoty na pozicích bitového řetízku, které neodpovídají právě zkoumanému objektu z hlavní matice. Podúloha tedy vždycky proběhne pouze pro požadované řádky vedlejší datové matice, jak je požadováno.

Cache bitových řetízků

Z důvodů zmíněných v 3.4.1 byla naimplementována jiná cache pro bitové řetízky generované hypotézovými atributy. Dle principu fungování je to spíše buffer než cache – uchová prvních N bitových řetízků vygenerovaných hypotézovým atributem, které mohou být poté použity pro běh nadúlohy. Velikost N byla stanovena konstantně, protože, jak bylo zjištěno při hlubším zkoumání způsobu implementace stávajících procedur při programování hypotézových atributů, ani původní cache pro bitové řetízky neobsahuje žádnou logiku pro stanovení své velikosti.

Úprava existujících dataminingových procedur

Existující dataminingové procedury musely být rozšířeny o metody umožňující generování a verifikaci relevantních otázek, bez generování hypotéz a s implementaci výše zmíněného enumerátoru. Rovněž bylo nutné rozšířit dataminingové procedury o možnost práce s jiným druhem cache pro bitové řetízky.

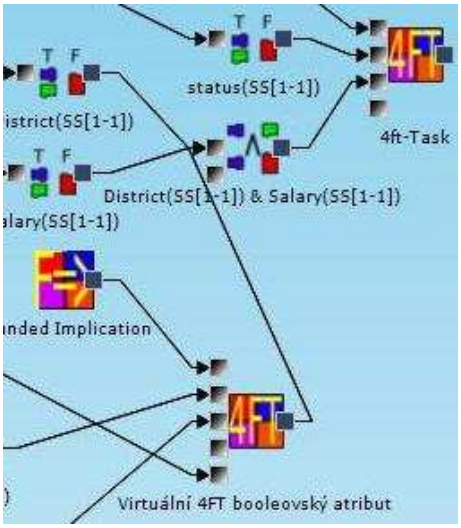
Přínos pro budoucí implementace

Výše uvedené postupy jsou navrženy a implementovány co nejobecněji. Autorům implementace relačních rozšíření dalších datamingových procedur se tímto může jejich práce velmi usnadnit. Mohou používat cache pro bitové řetízky virtuálních atributů, CountVector, masky a další navržené postupy.

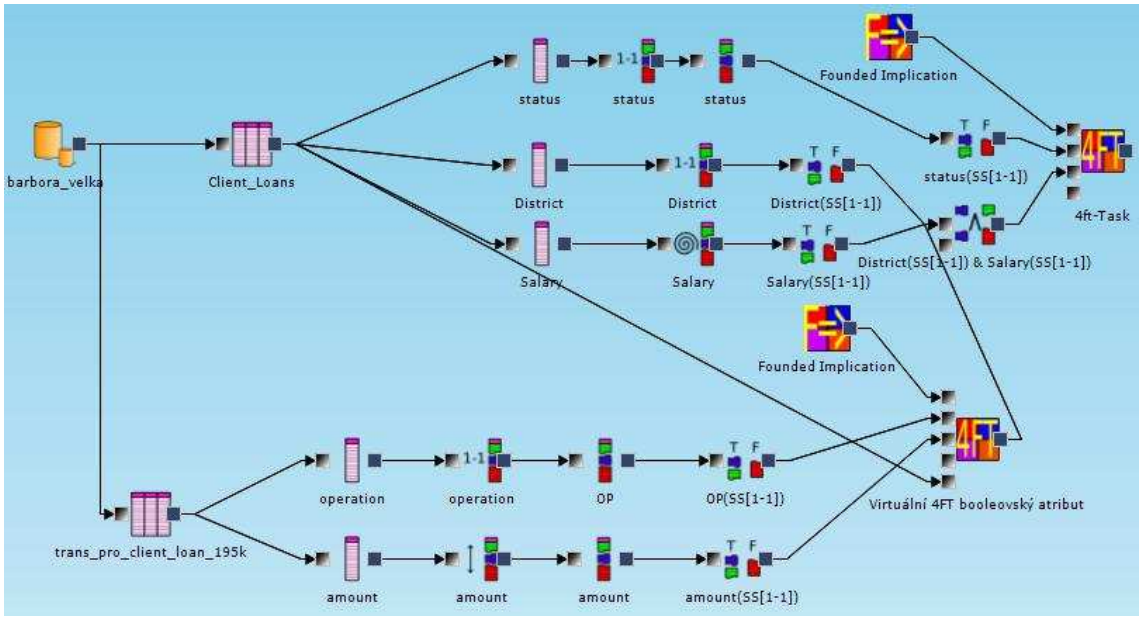
4.2.2 Krabičky

Krabička 4ft virtuální booleovský atribut

Daná krabička ilustrována obrázkem 4.2 reprezentuje výše popsaný hypotézový virtuální atribut z uživatelského hlediska. Celková úloha se zapojeným 4FT virtuálním booleovským atributem je uvedena na obrázku 4.3



Obrázek 4.2: 4FT virtuální booleovský atribut – detail



Obrázek 4.3: Úloha s 4FT virtuálním booleovským atributem

Seznam zásuvek

Zásuvky jsou shodné s krabičkou "4FT" až na výjimky, jejichž popis následuje.

Hlavní datová matice Do této zásuvky se zapojuje krabička reprezentující hlavní datovou matici.

Seznam vlastností

Vlastnosti jsou shodné s krabičkou "4FT" až na výjimky, jejichž popis následuje.

Maximální počet virtuálních sloupců Zde se zadává maximální počet virtuálních sloupců, které může virtuální atribut vygenerovat.

Pro úspěšné spuštění úlohy, do níž je zapojena tato relační procedura, je nutné zvolit klíč, podle kterého se budou identifikovat záznamy ve vedlejší datové matici oproti záznamům v hlavní datové matici. Pro hlavní datovou matici, která je zapojena do zásuvky se stejným jménem, se zvolí její primární klíč pomocí vlastností krabičky "Tabulka" se jménem "Sloupce primárního klíče". Pro vedlejší datovou matici, ze které se vytvářejí atributy pro zapojení do krabičky relační procedury, se zvolí sloupec obsahující záznamy již vybraného v předchozím kroku klíče z hlavní datové matice.

Úloha počítá s integritou dat: existuje-li ve vedlejší datové matici záznam, kterému neodpovídá žádný záznam hlavní datové matice, úloha skonči chybou. Význam uvedeného omezení můžeme chápat následujícím způsobem: máme-li databázi banky a v ní tabulky pro klienty a transakce identifikované například dle id klienta, lze počítat s tím, že pro každou transakci bude existovat klient, který ji provedl. Případ existence klienta, který žádné transakce neprovedl, může být naopak běžný a při takové situaci úloha proběhne korektně.

Krabička SD4FT virtuální booleovský atribut

Daná krabička reprezentuje výše popsaný hypotézový virtuální atribut z uživatelského hlediska. Způsob zapojení do úlohy je obdobný jako u krabičky "4FT virtuální booleovský atribut".

Seznam zásuvek

Zásuvky jsou shodné s krabičkou "SD4FT" až na výjimky, jejichž popis následuje.

Hlavní datová tabulka Do této zásuvky se zapojuje krabička reprezentující hlavní datovou matici.

Seznam vlastností

Vlastnosti jsou shodné s krabičkou "SD4FT" až na výjimky, jejichž popis následuje.

Maximální počet virtuálních sloupců Zde se zadává maximální počet virtuálních sloupců, které může virtuální atribut vygenerovat.

Pro úspěšné spuštění úlohy je požadováno správná volba klíčů hlavní a vedlejší datové matice stejně jako v případě 4FT virtuálního boolovského atributu - viz výše.

4.3 Ekvidistanční intervaly

Z důvodů popsaných v 1.2 byla implementovány krabičky atributu pro generování ekvidistančních intervalů. Krabičky fungují skoro stejně až na odlišnost zadaného parametru, podle kterého se mají intervaly generovat. Verze krabičky s názvem "Ekvidistanční intervaly LISp" vyžaduje zadání délky generovaného intervalu místo celkového požadovaného počtu intervalů.

4.3.1 Detaily implementace

Implementace krabičky využívá algoritmů, které byly připraveny T.Kucharem v rámci diplomové práce [10] a původně pocházejí od T.Karbana – viz [5]. Autor se proto soustředil na implementování samotné krabičky.

4.3.2 Krabička

Seznam zásuvek

Sloupec Do této zásuvky se zapojuje krabička reprezentující sloupec dat, nad kterým budou vytvořeny ekvidistanční intervaly.

Seznam vlastností

Počet intervalů Zde se zadává počet intervalů, které má krabička vygenerovat.

Délka intervalu Vlastnost se zadává místo vlastnosti "Počet intervalů" ve verzi krabičky "Ekvidistanční intervaly LISp". Zde se zadává délka intervalu, které má krabička vygenerovat.

X kategorie Zde se zadává kategorie chybějící informace.

Doména Zde se zadává typ vybrané domény sloupce transformované do kategorií. Uživatel může vybrat z možnosti "Celá aktuální doména" a "Uživatelem definována doména". První možnost nastaví vytváření kategorie pro každou hodnotu z aktuální domény sloupce, druhá možnost nastaví omezení pro použitou doménu na základě hodnot vlastností "Od" a "Do".

Od Při zvolení uživatelem definované domény kategorie budou vytvořeny pouze pro hodnoty z aktuální domény sloupce, které jsou větší nebo rovny hodnotě dané vlastnosti.

Do Při zvolení uživatelem definované domény kategorie budou vytvořeny pouze pro hodnoty z aktuální domény sloupce, které jsou menší nebo rovny hodnotě dané vlastnosti.

Název ve výsledcích Atribut je uveden v hypotézách pod daným názvem.

Sémantika Umožní vybrat sémantiku sloupce. Sémantika v prostředí Ferda DataMiner byla podrobněji popsána v 2.5.2. Uživatel může tedy vybrat z možností *Nominální, Ordinální, Cyklický ordinální, Kardinální.*

4.4 Ekvifrekvenční intervaly

Z důvodů popsaných v 1.2 byla implementována krabička atributu pro generování ekvifrekvenčních intervalů.

4.4.1 Detaily implementace

Implementace krabičky využívá algoritmů, které byly připraveny T.Kuchařem v rámci diplomové práce [10] a původně pocházejí od T.Karbana – viz [5]. Autor se proto soustředil na implementování samotné krabičky.

4.4.2 Krabička

Seznam zásuvek

Sloupec Do této zásuvky se zapojuje krabička reprezentující sloupec dat, nad kterým budou vytvořeny ekvifrekvenční intervaly.

Seznam vlastností

Počet kategorií Zde se zadává počet intervalů, které má krabička vygenerovat.

X kategorie Zde se zadává kategorie chybějící informace.

Doména Zde se zadává typ vybrané domény sloupce transformované do kategorií. Uživatel může vybrat z možnosti "Celá aktuální doména" a "Uživatelem definována doména". První možnost nastaví vytváření kategorie pro každou hodnotu z aktuální domény sloupce, druhá možnost nastaví omezení pro použitou doménu na základě hodnot vlastností "Od" a "Do".

Od Při zvolení uživatelem definované domény kategorie budou vytvořeny pouze pro hodnoty z aktuální domény sloupce, které jsou větší nebo rovny hodnotě dané vlastnosti.

Do Při zvolení uživatelem definované domény kategorie budou vytvořeny pouze pro hodnoty z aktuální domény sloupce, které jsou menší nebo rovny hodnotě dané vlastnosti.

Název ve výsledcích Atribut je uveden v hypotézách pod daným názvem.

Sémantika Umožní vybrat sémantiku sloupce. Sémantika v prostředí Ferda DataMiner byla podrobněji popsána v 2.5.2. Uživatel může tedy vybrat z možností *Nominální, Ordinální, Cyklický ordinální, Kardinální.*

4.5 Statický atribut

Z důvodů popsaných v 1.2 byla implementována krabička statického atributu, která umožní ruční úpravu kategorií v atributu.

4.5.1 Detaily implementace

Daná krabička v podstatě zajišťuje spouštění modulu pro úpravu atributu a předává upravený atribut pro další využití v prostředí Ferda DataMiner. Podstatná funkčnost je soustředěná v modulu pro úpravu atributu.

4.5.2 Krabička

Seznam zásuvek

Sloupec nebo atribut Do této zásuvky se zapojuje buď krabička reprezentující sloupec dat nebo krabička některého z atributů. Při zapojení krabičky atributu budou pro modul pro úpravu kategorií poskytnuty již vytvořené kategorie ze zapojené krabičky.

Při zapojení krabičky reprezentující sloupec není předem vytvořena žádná kategorie, tudíž uživatel musí použít modul pro úpravu kategorií pro vytvoření kategorií. Krabička reprezentující sloupec v tomto případě poskytuje informace o datech ve sloupci (důležité jsou například všechny navzájem různé hodnoty ze zapojeného sloupce, ale i jiné informace a vlastnosti).

Seznam vlastností

Úprava kategorií Při kliknutí na tlačítko spustí modul pro úpravu kategorií.

X kategorie Zde se zadává kategorie chybějící informace.

Doména Zde se zadává typ vybrané domény sloupce transformované do kategorií. Uživatel může vybrat z možnosti "Celá aktuální doména" a "Uživatelem definována doména". První možnost nastaví vytváření kategorie pro každou hodnotu z aktuální domény sloupce, druhá možnost nastaví omezení pro použitou doménu na základě hodnot vlastností "Od" a "Do".

Od Při zvolení uživatelem definované domény kategorie budou vytvořeny pouze pro hodnoty z aktuální domény sloupce, které jsou větší nebo rovny hodnotě dané vlastnosti.

Do Při zvolení uživatelem definované domény kategorie budou vytvořeny pouze pro hodnoty z aktuální domény sloupce, které jsou menší nebo rovny hodnotě dané vlastnosti.

Název ve výsledcích Atribut je uveden v hypotézách pod daným názvem.

Sémantika Umožní vybrat sémantiku sloupce. Sémantika v prostředí Ferda DataMiner byla podrobněji popsána v 2.5.2. Uživatel může tedy vybrat z možností *Nominální, Ordinální, Cyklický ordinální, Kardinální.*

Seznam vlastností pouze pro čtení

Počet kategorií Zobrazí aktuální počet kategorií v daném atributu.

4.6 Modul pro úpravu atributu

Tento modul se spouští nad krabičkou "Statický atribut". Ta mu poskytuje již existující kategorie v atributu pro úpravu a po skončení práce modulu opět uloží upravené kategorie pro další využití v prostředí Ferda DataMiner.

4.6.1 Funkčnost

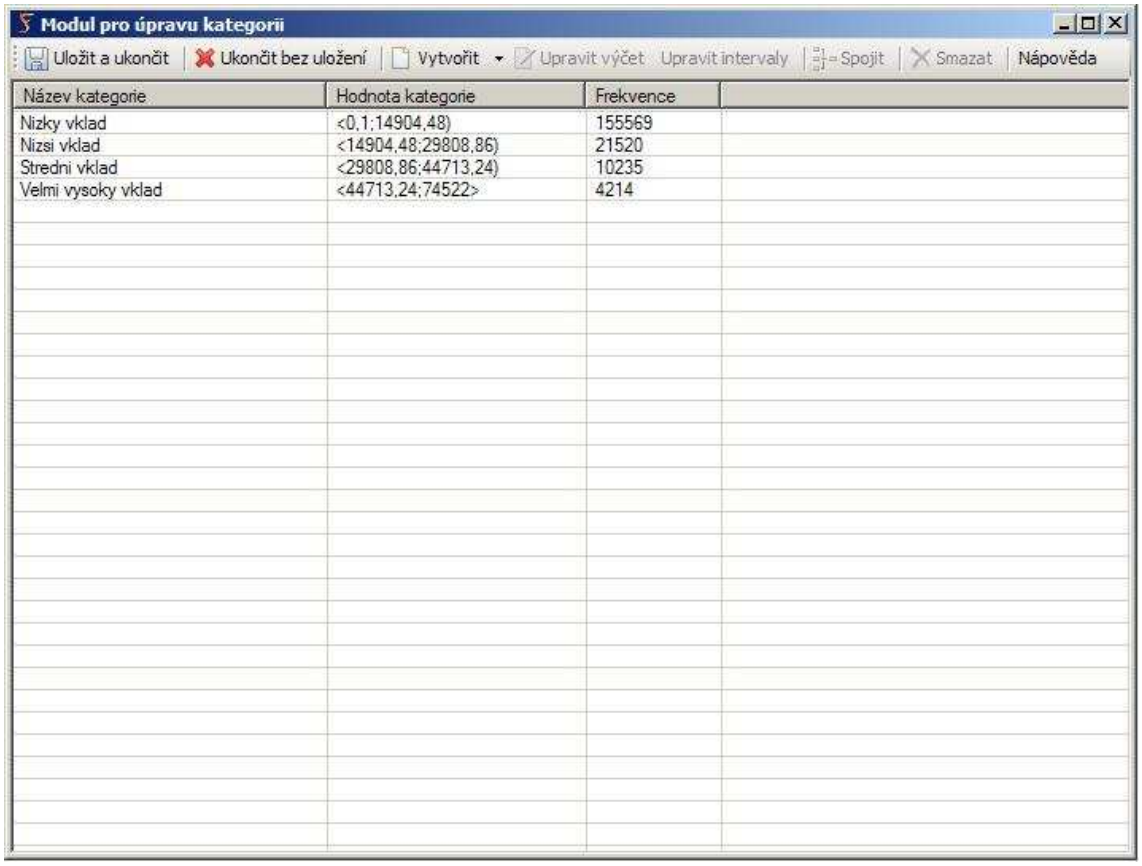
Hlavní obrazovka

Na obrázku 4.4 je zobrazen seznam existujících kategorií. Kategorie může být přejmenována standardním postupem pro přejmenování položky v seznamu GUI MS Windows, tj. dvojím pomalým kliknutím na její název, případně kliknutím na název a stisknutím klavesy F2 a poté zadáním nového názvu – ilustruje obrázek 4.5. Protože kategorie musí mít navzájem různé názvy, po zadání nesprávného (již existujícího názvu) bude oznámena chyba a kategorie se nepřejmenuje.

Více kategorií může být spojeno do jedné tak, že s podržením tlačítka Control případně Shift a kliknutím na požadované kategorie se vybere více kategorií a pak kliknutím na tlačítko "Spojit" se spojí do jedné kategorie.

Kategorie může být odstraněna z atributu vybraním a následným kliknutím na tlačítko "Odstranit".

Provedené změny v seznamu kategorie se uloží kliknutím na tlačítko "Uložit a zavřít". Pokud je potřeba ukončit práci modulu bez uložení změn, použije se tlačítko "Zavřít bez uložení". Uživatel může přidat novou kategorií do seznamu kategorií v atributu. Z definice kategorie v metodě GUHA a její následného rozšíření T.Kuchařem v [10] plyne, že kategorie může obsahovat intervaly a výčet hodnot.

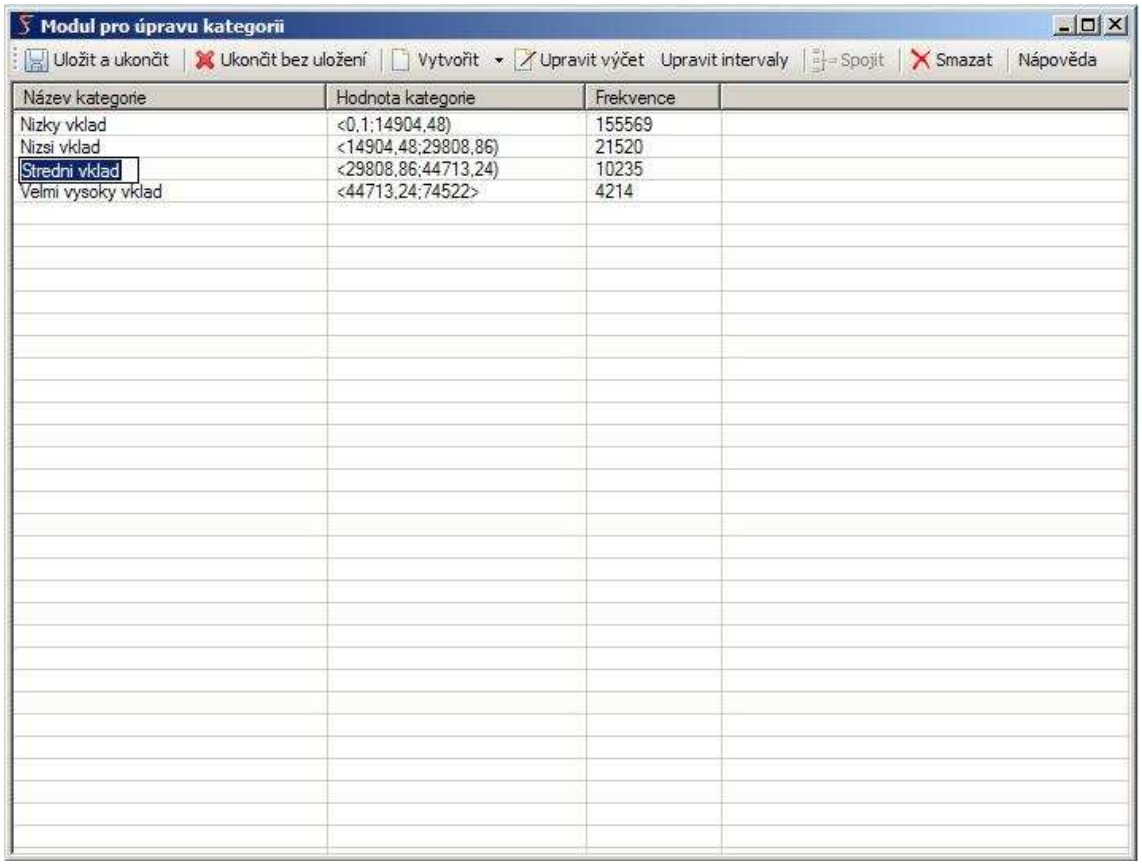


Obrázek 4.4: Úprava kategorií – hlavní obrazovka

Obrazovka pro přidání a úpravu intervalů

Ve formuláři pro přidání intervalů (na obrázku 4.6) uživatel může přidávat intervaly do nově vytvořené kategorie. Ve formuláři lze vybrat typy hranic intervalu a hodnoty krajních bodů, poté kliknutím na tlačítko "Přidat do kategorie" přidat interval do kategorie. Tento se poté zobrazí v seznamu intervalů dané kategorie a může být odebrán zvolením a následným kliknutím na tlačítko pro odebrání intervalu. Dále se vybraný interval může upravovat přepnutím do režimu úpravy.

Při přidávání nebo změně se kontroluje validita kategorie v atributu. Pro přidávaný interval to tedy znamená, že se zkontroluje jeho disjunktnost se stávajícími intervaly v kategoriích a také s hodnotami ve výčtech kategorií. Pokud by přidávaný interval nebyl disjunktní s prvky existujících kategorií,



Obrázek 4.5: Úprava kategorií – přejmenování kategorie

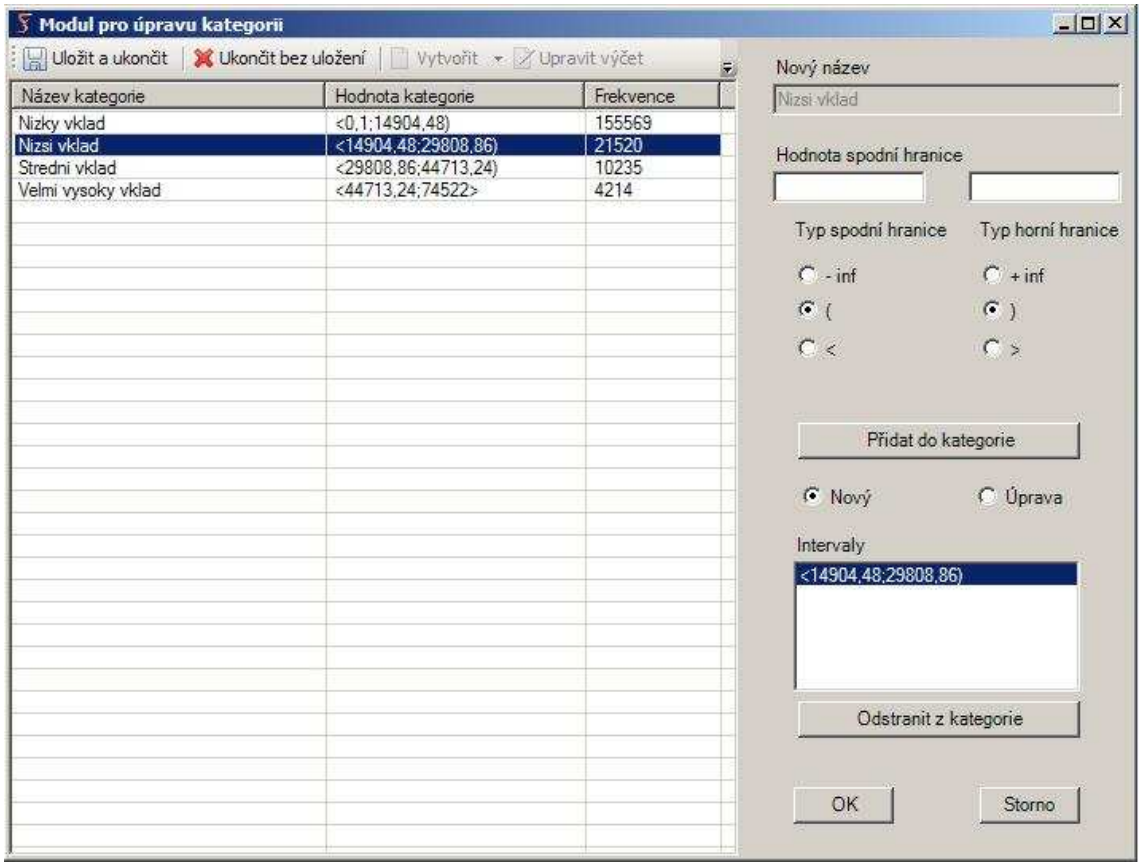
modul oznámí chybu a nedovolí takový interval do kategorie přidat.

Po přidání intervalů do nově vytvářené kategorie lze potvrdit přidání kategorie tlačítkem "OK". Pro zrušení přidání kategorie se použije tlačítko "Storno".

Obrazovka pro úpravu intervalů v již existující kategorii se liší tím, že nelze změnit název kategorie (ten lze změnit v hlavní obrazovce). Logika při úpravě intervalů v kategorii je stejná jako je popsáno výše.

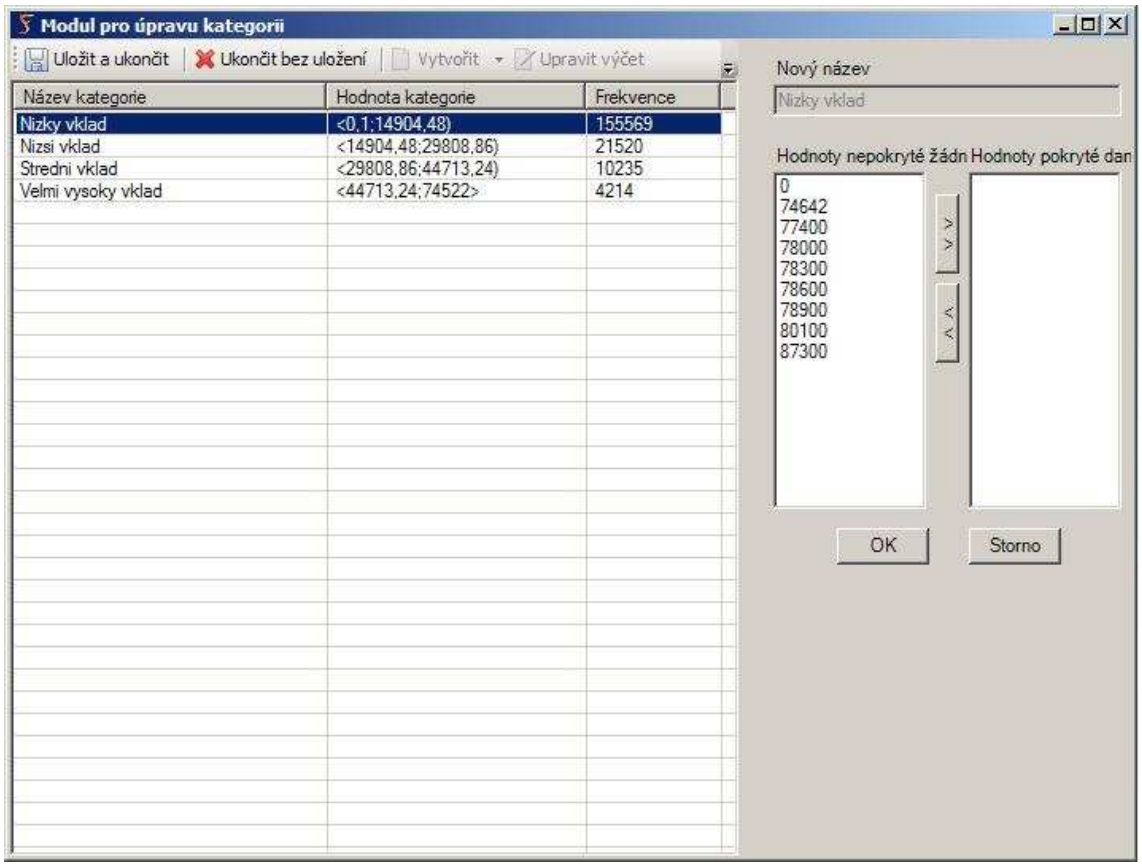
Obrazovka pro přidání a úpravu výčtu

Ve formuláři pro přidání výčtu (na obrázku 4.7) uživatel může přidávat hodnoty do výčtu nově vytvořené kategorie. Ve formuláři lze vybrat zatím nepo-



Obrázek 4.6: Úprava kategorií – práce s intervaly

kryté hodnoty v odpovídajícím sloupci a přidat je do výčtu nově vytvářené kategorie kliknutím na příslušné tlačítko. Hodnota se poté objeví ve výčtu hodnot dané kategorie. Odtud může být odstraněna zvolením a kliknutím na tlačítko pro odebrání hodnoty z výčtu. Po přidání hodnot do výčtu nově vytvářené kategorie lze potvrdit přidání kategorie tlačítkem "OK". Pro zrušení přidání kategorie se použije tlačítko "Storno".



Obrázek 4.7: Úprava kategorií – práce s výčtem

Kapitola 5

Testy

V této kapitole bude popsán vliv různých faktorů na běh úlohy s použitím relačního rozšíření dataminingové procedury. Autor bude zkoumat vliv zejména velikosti dat a počtu vygenerovaných virtuálních sloupců. Také se ukáže, že použití bufferu pro bitové řetízky vygenerované virtuálním atributem je z hlediska výkonu nezbytné.

5.1 Zkoumané datové matice

Zkoumaná data vychází z databáze fiktivní banky "Barbora" používané původně v projektu LISp-Miner a dále Ferda DataMiner. Databáze obsahuje data o fiktivních klientech, jejich pohlaví, místa bydliště, transakcích, půjčkách atd.

5.1.1 Hlavní datová matice

Hlavní datová matice s názvem "Client_Loans" obsahuje údaje o jednotlivých klientech. Velikost této datové matice je kolem 700 řádků. Následující tabulka obsahuje popis sloupců hlavní datové matice.

Název sloupce	Typ sloupce	Popis sloupce
client_id	Dlouhé celé číslo	Identifikátor klienta
loan_id	Dlouhé celé číslo	Identifikátor půjčky klienta (modelována situace, kde každý klient má jednu půjčku)
account_id	Dlouhé celé číslo	Identifikátor účtu klienta (modelována situace, kde každý klient má jeden účet)
date	Dlouhé celé číslo	Datum založení účtu
amount	Dlouhé celé číslo	Velikost půjčky
duration	Dlouhé celé číslo	Doba půjčky
payments	Dlouhé celé číslo	Výše splátky
status	text	Kvalita dluhu (subjektivní hodnocení)
birth_number	Dlouhé celé číslo	Rodné číslo
DistrictID	Dlouhé celé číslo	Identifikátor bydliště
District	text	Bydliště
Region	text	Region
Inhabitants	Dlouhé celé číslo	Počet obyvatel
Muni499	Dlouhé celé číslo	Počet obcí v regionů s maximálně 499 obyvateli
Muni1999	Dlouhé celé číslo	Počet obcí v regionů s maximálně 1999 obyvateli
Muni9999	Dlouhé celé číslo	Počet obcí v regionů s maximálně 9999 obyvateli
MuniOver	Dlouhé celé číslo	Počet obcí v regionů s více než 9999 obyvateli
Cities	Dlouhé celé číslo	Počet měst v regionu
PopulationRation	Dlouhé celé číslo	
Salary	Dlouhé celé číslo	Průměrný plat v regionu
Unemployment95	Dlouhé celé číslo	Nezaměstnanost v regionu v roce 1995
Unemployment96	Dlouhé celé číslo	Nezaměstnanost v regionu v roce 1996
EnterpreneursRation	Dlouhé celé číslo	Podnikatelé v regionu
Crimes95	Dlouhé celé číslo	Kriminalita v regionu v roce 1995
Crimes96	Dlouhé celé číslo	Kriminalita v regionu v roce 1996

Tabulka 5.1: Sloupce hlavní datové matice "Client_Loans"

5.1.2 Vedlejší datová matice

Vedlejší datová matice "trans_pro_client_loan" obsahuje transakce pro klienty z tabulky "Client_Loans". Pro účely testování je tato tabulka obsažena v databázi ve třech verzích: malá, střední a velká, kde počet řádků pro jednotlivé verze je přibližně 10000, 70000 a 200000.

Následující tabulka obsahuje popis sloupců vedlejší datové matice.

Název sloupce	Typ sloupce	Popis sloupce
trans_id	Dlouhé celé číslo	Identifikátor transakce
client_id	Dlouhé celé číslo	Identifikátor klienta
account_id	Dlouhé celé číslo	Identifikátor účtu
date	Dlouhé celé číslo	Datum transakce
type	text	Typ transakce (příjem, výdej nebo výběr)
operation	text	Typ operace (převod z / na účet, výběr, vklad nebo výběr kartou)
Amount	Dlouhé celé číslo	Velikost transakce
Balance	Dlouhé celé číslo	Zůstatek
k_symbol	Dlouhé celé číslo	Konstantní symbol (účel transakce, např. inkaso)
bank	text	Zdrojová nebo cílová banka transakce
account	Dlouhé celé číslo	Zdrojový nebo cílový účet transakce

Tabulka 5.2: Sloupce vedlejší datové matice "trans_pro_client_loan"

5.2 Východiska zadání a zadaná úloha

Základní inspirací pro úlohu pro relační rozšíření procedur 4FT a SD4FT bylo zjistit nějaké údaje o objektech z hlavní datové matice na základě údajů z vedlejší datové matice. V našem případě hledáme nějaké vztahy v matici klientů na základě údajů jak v hlavní matici klientů, tak i ve vedlejší matici jejich transakcí. Pomocí procedury SD4FT zkoumáme rozdíly mezi dvěma množinami.

Na základě údajů obsažených ve zkoumaných datových maticích a po diskuzi s vedoucím práce bylo rozhodnuto začít od jednodušších úloh pro relační proceduru a zapojovat je do úlohy pro (nerelační) 4FT. V úlohách se zkoumá

status úvěru v závislosti na různých parametrech. Zde předpokládáme a chceme najít nějaké zajímavé závislosti statusu půjčky na charakteristice plateb klienta, jak bylo podrobněji uvedeno v kapitole 2.6.2.

5.2.1 4FT virtuální booleovský atribut

Úloha 1

Úloha je zadaná pro proceduru 4FT následujícím způsobem:

$$District\&Salary\&TypeImplAmount \Rightarrow_{0,9;50} LoanStatus$$

$\Rightarrow_{0,9;50}$ je v tomto případě kvantifikátor fundované implikace *Foundedimplication* a *TypeImplAmount* je virtuální 4FT atribut s následujícím zadáním:

$$Type \Rightarrow_{0,9;5} Amount$$

$\Rightarrow_{0,9;5}$ je v tomto případě rovněž kvantifikátor fundované implikace *Foundedimplication*.

5.2.2 Výsledky testů

V následujících tabulkách jsou uvedeny doby běhu úloh s různými velikostmi vedlejší datové matice a počtem relevantních otázek vygenerovaných relačním 4ft booleovským atributem.

Každý test se pouštěl třikrát a výsledný čas se zprůměroval. Mezi každým spuštěním Ferda DataMiner byl restartován, aby byly zachovány stejné podmínky pro jednotlivé běhy (bez tohoto by se při druhém a dalším spuštění načítaly informace pro krabičky z cache). V uvedené tabulce 5.3 *Transakce* znamená běh úlohy na vedlejší matici transakcí o různých velikostech a *# vir.sloupců* je počet virtuálních sloupců vygenerovaných krabičkou virtuálního atributu.

Úloha 1

Úloha 1 – bez cache

Zde byla zkoumána rychlost práce bez cache. Porovnáním s předchozí tabulkou lze zjistit časové rozdíly běhu úlohy.

Transakce / # vir.sloupců	malé	střední	velké
16	14s	25s	1m 10s
50	15s	25s	1m 32s
80	9m 2s	11m 11s	12m 2s
100	12m	24m	1h 3m

Tabulka 5.3: Test relační procedury 4FT: úloha 1

Transakce / # vir.sloupců	malé	střední	velké
16	21s	1m8s	6m 45s
50	45s	3m 5s	26m 33s
100	50m 30s	9h 52m	–[1]

Tabulka 5.4: Test relační procedury 4FT: úloha 1 – bez cache

5.2.3 SD4ft virtuální booleovský atribut

Východiska zadání a zadaná úloha

Úloha je zadaná pro 4FT následujícím způsobem:

$$District\&Salary\&TypeImplAmountBank \Rightarrow_{0,9;50} LoanStatus$$

$\Rightarrow_{0,9;50}$ je v tomto případě kvantifikátor fundované implikace *Founded-implication* a *TypeImplAmountBank* je virtuální SD4FT booleovský atribut s následujícím zadáním:

$$Type \Rightarrow_{0,9;5} Amount(Bank)$$

$\Rightarrow_{0,9;5}$ je v tomto případě rovněž kvantifikátor fundované implikace $Founded-implication$.

5.2.4 Výsledky testů

V následujících tabulkách jsou uvedeny doby běhu úloh s různými velikostmi vedlejší datové matice a počtem virtuálních sloupců vygenerovaných relačním SD4FT booleovským atributem.

[1] Vzhledem k velmi dlouhé předpokládané době běhu test nebyl spouštěn

Transakce / # vir.sloupců	malé	střední	velké
16	8m01s	13m05s	21m30s
40	16m12s	28m57s	29m30s
80	25m40s	35m20s	58m05s
100	30m34s	55m50s	1h13m50s

Tabulka 5.5: Test relační procedury SD4FT: úloha 1

5.3 Závěr

Z provedených testů plyne, že doba běhu pro obě úlohy záleží hlavně na počtu vygenerovaných virtuálních sloupců. Velikost vedlejší datové matice sice má na úlohu vliv, rozhodujícím faktorem však zůstává právě počet vygenerovaných sloupců. To vyplývá ze způsobu, jakým virtuální atribut funguje: při prvním generování spočítá bitové řetízky a uloží si je do bufferu, při dalších voláních se pracuje s bufferem. Proto při různých velikostech vedlejší datové matice se úměrně mění pouze doba prvního generování.

Dále se názorně ukazuje potřeba bufferu pro bitové řetízky generované virtuálním atributem. Lze to zjistit jak z časů v tabulce, tak i z faktu, že při testování s vypnutou cache byl testovací počítač při běhu úlohy vytížen o poznání více.

Také je potřeba uvést, že implementace relačních rozšíření v rámci dané práce je pilotní a chování úloh s hypotézovým virtuálním atributem vyžaduje další zkoušení a zkoumání, stejně jako tomu bylo v případě jíž implementovaných dataminingových úloh.

Kapitola 6

Zdrojové kódy

6.1 Seznam změněných a vytvořených souborů

Jako součást dané diplomové práce byly vytvořeny nebo upraveny soubory se zdrojovými kódy. Na projektu Ferda DataMiner pracuje v současné době více lidí. Zdrojové kódy jsou proto spravovány systémem Subversion a nacházejí se na `https://svn.sourceforge.net/svnroot/ferda`. K diplomové práci autora se vztahují změny provedené autorem *kuzmos* od čísla revize 292.

- `ferda/slice/Modules/Guha.MiningProcessor.ice`
- `ferda/slice/Modules/Boxes/DataPreparation/DataPreparation.ice`
- `ferda/slice/Modules/Common.ice`
- `ferda/src/FrontEnd/AddIns/ResultBrowser/`
- `ferda/src/FrontEnd/AddIns/EditCategories/`
- `ferda/src/Modules/BoxModulesServices/Boxes/*.xml`
  Drobné změny pro práci s novými atributy.
- `ferda/src/Modules/BoxModulesServices/DataPreparation/Categorization/Common.cs`

- `ferda/src/Modules/BoxModulesServices/DataPreparation/Categorization/EachValueOneCategory/`
  Drobné změny existujícího kódu pro práci s virtuálním agregačním atributem.
- `ferda/src/Modules/BoxModulesServices/DataPreparation/Categorization/EquidistantIntervals`
- `ferda/src/Modules/BoxModulesServices/DataPreparation/Categorization/EquifrequencyIntervals`
- `ferda/src/Modules/BoxModulesServices/DataPreparation/Categorization/StaticAttribute`
- `ferda/src/Modules/BoxModulesServices/DataPreparation/DataSource/VirtualColumn`
- `ferda/src/Modules/BoxModulesServices/GuhaMining/VirtualAttributes/VirtualFFTBooleanAttribute`
- `ferda/src/Modules/BoxModulesServices/GuhaMining/VirtualAttributes/VirtualSDFFTBooleanAttribute`
- `ferda/src/Modules/Core/MiningProcessor/Miners/`
  Úpravy v souborech `FourFoldMiningProcessor.cs` a `SDFourFoldMiningProcessor.cs`, kde byly vytvořeny samotné procedury pro relační mining. Úpravy v souborech `MinningProcessor.cs` a `MiningProcessorFunctionsI.cs` pro zajištění funkčnosti práve vytvořených a v budoucnu vytvářených relačních rozšíření.
- `ferda/src/Modules/Core/MiningProcessor/QuantifierEvaluator/FirstNNoResult.cs`
- `ferda/src/Modules/Core/MiningProcessor/Generation/LeafEntities.cs`
  Přidána procedura pro generování bitových řetízků pro relační rozšíření.

6.2 Postup pro přidání nových relačních rozšíření

6.2.1 Vytvoření krabičky

Pro uživatelské rozhraní se musí vytvořit krabička pro systém Ferda DataMiner. Obecný návod pro tento postup je popsán v dokumentu "Implementace

krabiček", který se nachází v repozitáři zdrojových kódu prostředí Ferda DataMiner jako `ferda/docsrc/draft/implementaceKrabicek.xdb`. Jako vzorovou krabičku pro implementaci relačních rozšíření se může použít jedna z krabiček `VirtualFFTBooleanAttribute` nebo `VirtualSDFFTBooleanAttribute`. Nesmí se zapomenout na úpravu souborů `ferda/src/Modules/BoxModulesServices/Boxes/*.xml`

6.2.2 Úprava procedur

Zde se v podstatě jedná o implementaci procedury `TraceBoolean` vracející enumerátor bitových řetízků v jednotlivých procedurách, které se nacházejí v `ferda/src/Modules/Core/MiningProcessor/Miners`. Jako vzor implementace může opět posloužit dotyčná procedura implementována pro `FourFoldMiningProcessor.cs` nebo `SDFourFoldMiningProcessor.cs`. Zbývající úpravy pro zajištění funkčnosti relačních rozšíření byly implementovány autorem dané diplomové práce pro ulehčení přidávání rozšíření dalších procedur.

Kapitola 7

Zhodnocení práce

7.1 Splnění cílů

V této kapitole proběhne shrnutí výsledků dané diplomové práce.

7.1.1 Pilotní implementace

Implementace relačních rozšíření dataminingových procedur byla zcela novým úkolem pro prostředí Ferda DataMiner. Předtím tato rozšíření nebyla realizována ani v první verzi prostředí Ferda DataMiner, ani v rámci systému LISp-Miner (ačkoliv existovala nějaká pilotní implementace, se kterou autor nepřišel do styku a od jejího rozvoje a užívaní se upustilo).

Teoretická příprava pro implementaci relačních rozšíření byla vykonána T.Karbanem ve své dizertační práci "DATA MINING IN RELATIONAL DATABASES" uvedené v seznamu použité literatury jako [5]. Ačkoliv tato práce je rovněž implementační, k okamžiku zahájení i dokončení dané diplomové práce dle informací přímo od vedoucího dizertační práce J.Raucha nebyl k dispozici funkční spustitelný program. Proto implementaci v dané diplomové práci lze brát jako pilotní, v čemž spočívá jak její přínos, tak i zjištěná omezení již zmíněna v tomto textu.

7.1.2 Splněné cíle ze zadání

Jak již bylo zmíněno, byly implementovány relační rozšíření stávajících dataminingových procedur 4FT a SD4FT. Odpovídající krabičky v prostředí Ferda se jmenují "Virtuální booleovský 4FT atribut" a "Virtuální booleovský SD4FT atribut". Implementují hypotézový virtuální atribut, který je

relačním rozšířením pro procedury 4FT a SD4FT a tím tedy přímo splňují zadání práce.

Dále byl implementován agregační virtuální atribut. Odpovídající krabička v prostředí Ferda se jmenuje "Agregační virtuální sloupec". Tímto se splňuje zadání práce, protože patří do relačních rozšíření dataminingových procedur.

Také byly implementovány krabičky pro transformaci atributů pro vytvoření ekvidistančních intervalů, ekvifrekvenčních intervalů a krabička pro přímou úpravu atributu. Tyto úlohy původně nepatřily do zadání, avšak v odevzdané diplomové práci Tomáše Kuchaře [10] nebyly rovněž z různých důvodů přítomny a ukázalo se, že by bylo velmi žádoucí je mít implementované co nejdříve. Umožňují podstatné rozšíření způsobů zadávaní dataminingových úloh a tímto výrazně přispěly k lepším výsledkům i této diplomové práce. Zadání práce bylo rozšířeno o požadavek implementace těchto metod, který byl splněn. Jedná se tedy o naplnění cílů práce.

7.1.3 Příprava pro budoucí implementace

V průběhu práce byly navrženy takové změny stávajících jednotlivých prvků systému Ferda DataMiner, aby relační rozšíření mohla být relativně snadno implementována i pro jiné dataminingové procedury, než pro ty, které implementoval autor této práce. Jinými slovy, cílem a přínosem dané diplomové práce byla i příprava prostředí Ferda DataMiner pro snadnější další vývoj. Autor práce tedy tvrdí, že tento cíl, který (byť ne explicitně) vychází ze zadání, byl v dané diplomové práci splněn.

7.1.4 Závěr

Na základě výše uvedeného autor prohlašuje cíle své diplomové práce za splněné.

7.2 Doporučení dalšího směru práce

V této kapitole bude naznačen možný směr dalšího vývoje relačních rozšíření pro dataminingové procedury v prostředí Ferda DataMiner.

7.2.1 Rozšíření pro zbývající procedury

Jak jíž bylo zmíněno, lze poměrně snadno implementovat relační rozšíření pro zbývající dataminingové procedury z prostředí Ferda DataMiner. K okamžiku odevzdání diplomové práce je v prostředí implementováno 6 procedur, lze tedy uvažovat o relačních rozšířeních pro KL a SDKL, dále pro CF a SDCF. Tyto autorem nebyly implementovány z důvodů toho, že cílem práce bylo kromě samotného naprogramování také vytvoření základu pro relační rozšíření v prostředí Ferda DataMiner, vyřešit obecné problémy, které při implementaci vznikly a otestovat chování vytvořených rozšíření. Navíc způsob implementace je stejný jako pro jíž vytvořená rozšíření, nejednalo by se tedy o podstatný přínos k práci.

7.2.2 Reálný hypotézový atribut

Logickým směrem dalšího rozšiřování se jeví implementace obecnějšího hypotézového atributu, který produkuje virtuální sloupce skládající se z reálných čísel (a ne pouze z hodnot *true* a *false*). Takový atribut může dávat na výstup tyto sloupce buď rovnou jako fuzzy bitové řetězce anebo je bude předávat dalším modulům ke zpracování. Fuzzy bitové řetězce sice zatím nejsou v prostředí Ferda DataMiner implementovány, avšak dle T.Kuchare ([10]) systém generování může být o ně rozšířen. Totéž platí i pro relační rozšíření – systém generování virtuálních sloupců může být poměrně snadno upraven pro sloupce reálných čísel.

7.2.3 Nový modul pro prohlížení výsledků

Pro lepší práci s hypotézami generovanými z úloh, které používají nově vytvořená relační rozšíření procedur, je nutné rozšířit možnosti současného modulu pro prohlížení výsledků. Ten zobrazuje atributy v hypotézách ve formě prostého textu, což dostačuje pro prohlížení výsledků s použitím nerelačních atributů, avšak pro relační atributy by bylo vhodné mít možnost pracovat s atributy ze zadání podúlohy zvlášť – teď se zobrazí vcelku jako jeden textový řetězec.

Literatura

[1] Hájek P., Havel I., Chytil M. (1966): The GUHA method of automatic hypotheses determination, Computing 1 293-308.

[2] Hájek P., Havránek T.: Mechanizing Hypothesis Formation – Mathematical Foundations for a General Theory. Springer-Verlag, 1978, ISBN: 3-540-08738-9, available for download at `http://www.cs.cas.cz/~hajek/guhabook/`

[3] Rauch J., Šimůnek M.: GUHA Method and Granular Computing. In: Hu X., Liu Q., Skowron A., Lin T. Y., Yager R., Zang B. (ed.). Proceedings of IEEE konference Granular Computing 2005. 2005, s. 630–635.

[4] Rauch J.: Systém LISp-Miner Stručný popis určený pro posluchače kurzu Metod zpracování informací

[5] Karban T.: První praktické zkušenosti s RelMinerem, viz `http://146.102.64.29/%7Esvatek/keg/seminar/keg-sem.html`

[6] Rauch J.: Interesting Association Rules and Multirelational Association Rules. Taiwan 06.05.2002. In: LEE, H. C., LAI, F. (ed.). Communications of IICM. Taipei : IICM, 2002, s. 77–82

[7] Rauch J., Šimůnek M.: An Alternative Approach to Mining Association Rules. In: Lin, T. Y. – Ohsuga, S. – Liau, C. J. – Hu, X. – Tsumot, S. (ed.): Foundations of Data Mining and Knowledge Discovery. Berlin: Springer, 2005, pp. 211 – 232

[8] Kováč M., Kuchař T., Kuzmin A., Ralbovský M.: Ferda, nové vizuální prostředí pro dobývání znalostí. Přijato k prezentaci na konferenci ZNALOSTI 2006, viz `http://fim.uhk.cz/znalosti/index.php?p=prispevky`

[9] Ducháček M.: Nástroj pro správu databází s využitím pro multirelační datamining, diplomová práce na Katedře softwarového inženýrství, 2006

[10] Kuchař T.: Experimentální GUHA procedury, diplomová práce na Katedře softwarového inženýrství, 2006

[11] Aggraval, R. et al.: Fast Discovery of Association Rules. In Fayyad, U.M. et al.: Advances in Knowledge Discovery and Data Mining, pp. 307-328, AAAI Press / MIT Press, 1996

[12] Databases, Texts Specifications, and Objects 2005. April 13 . 15, 2005 Desná – Cerná Říčka.

©K. Richta, V. Snášel, J. Pokorný, editors. Relational Data Mining and GUHA Tomáš Karban

Seznam tabulek

Seznam obrázků