VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE DOLOVACÍCH MODULŮ SYSTÉMU PRO DOLOVÁNÍ Z DAT NA PLATFORMĚ NETBEANS

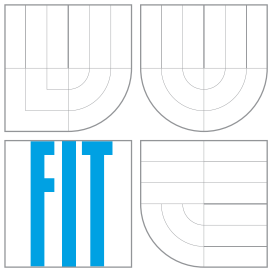
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ROSTISLAV STRÍŽ

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE DOLOVACÍCH MODULŮ SYSTÉMU PRO DOLOVÁNÍ Z DAT NA PLATFORMĚ NETBEANS

IMPLEMENTATION OF MINING MODULES OF DATA MINING SYSTEM

ON NETBEANS PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ROSTISLAV STRÍŽ
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL ŠEBEK
SUPERVISOR

BRNO 2010

Abstrakt

Sběr a ukládání dat hraje významnou roli v mnoha aspektech dnešního podnikání a kvalitní informace se stávají klíčem k úspěchu. Proces *získávání znalostí z databází* umožňuje z uložených dat získat skryté informace, které lze využít k dalšímu rozvoji. Tato práce se zabývá rozšířením nástroje, který slouží právě k dolování informací.
Cílem bylo vytvořit modul pro dolovací program, pracující na platformě NetBeans a vyvíjený na FIT ke studijním účelům. Nový modul bude umožňovat dolování z databázového systému Oracle pomocí netradičního použití genetického algoritmu. Obsahem práce je postup tvorby tohoto modulu – od teoretických základů až po podrobnosti implementace, testování a zhodnocení.

Abstract

Data collecting plays an important role in many aspects of today's businesses and quality information is the key to success. Process called Knowledge Discovery in Databases makes possible to extract hidden information that can be used further in our efforts. Main goal of this thesis is to describe an addition to such Data Mining System.
Main objective is to create data mining module for NetBeans application, developed for demonstrational purposes by Faculty of Information Technology. New module is going to be able to mine information from Oracle database server via unusual use of Genetic Algorithm. This thesis describes the whole process of module implementation, begining with theoretical basics through coding details to final testing and summary.

Klíčová slova

získávání znalostí z databází, dolování z dat, platforma NetBeans, modul, klasifikace, genetický algoritmus, databázový systém Oracle

Keywords

knowledge discovery in databases, data mining, NetBeans Platform, module, classification, genetic algorithm, Oracle database system

Citace

Rostislav Stríž: Implementace dolovacích modulů systému pro dolování z dat na platformě NetBeans, bakalářská práce, Brno, FIT VUT v Brně, 2010

Implementace dolovacích modulů systému pro dolování z dat na platformě NetBeans

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Šebka.

.....
Rostislav Stríž
17. května 2010

Poděkování

Na tomto místě bych rád poděkoval Ing. Michalu Šebkovi za jeho odbornou pomoc, rady, konzultace a především velkou ochotu.

Obsah

Kapitola 1

Úvod

Díky postupnému rozvoji a rozšiřování informačních technologií je dnes běžnou praxí, že se většina získaných dat ukládá v elektronické podobě, nejčastěji do různých databází či datových skladů. S narůstajícím množstvím obdržených dat roste také problém, jak se v nich smysluplně orientovat, případně jaké teoreticky zajímavé závěry z nich můžeme odvodit. Jedním východiskem mohou být analytické nástroje OLAP (z anglického *Online Analytical Processing*), které slouží především k zobrazování zesumarizovaných dat z multidimenzionálních datových skladů na základě jednotlivých dimenzí, např. data za určitý časový úsek (měsíc, čtvrtletí, ...), rozdělení dat podle regionální příslušnosti atp., více v [11]. Tyto nástroje však vyžadují poměrně značnou míru komunikace s uživatelem, což není ideální. Proto byl kladen důraz na vývoj metod, které mohou probíhat pokud možno autonomně. Jako reakce na tento požadavek vznikl nový směr v oblasti informačních technologií s názvem získávání znalostí z databází (z anglického *Knowledge Discovery in Databases*) [4]. Obor čerpá z celé řady již známých oblastí počítačových technologií, avšak vhodnou kombinací a implementací těchto znalostí ve výsledku umožňuje popisovat obrovská kvanta dat z mnoha různých pohledů.

Tato práce byla vypracována jako technická zpráva k bakalářskému projektu, jehož tématem je seznámení se se školním programem, který slouží k získávání znalostí z databázového systému Oracle, a následným vytvořením modulu s použitím genetického algoritmu pro tento systém. Program je vypracován na platformě NetBeans v programovacím jazyce Java. V kapitole 2 je popsán samotný proces získávání znalostí z databází, kapitola 3 popisuje podrobněji návrh algoritmu a kapitola 4 obsahuje stručnou charakteristiku školního dolovacího systému. Kapitolou 5 se dostaneme do fáze implementace a popíšeme proces, kterým je možné vytvořit obecně nový dolovací modul pro fakultní systém, v kapitole 6 je uveden způsob konkrétní implementace – popis vytvořených tříd, problémy při implementaci a další. Obsahem stručné kapitoly 7 je rozbor změn, které bylo nutné vytvořit v DMSL dokumentu, v kapitole 8 je poté uveden příklad použití hotového modulu v praxi. Celá práce je ukončena kapitolou 9, kde jsou popsány a diskutovány výsledky testování a celkově shrnuta úspěšnost projektu.

Co se týče **motivace**, hlavním cílem projektu bylo otestovat, zda lze vůbec smysluplně využít genetického algoritmu k dolování z dat, případně na jaká úskalí a problémy narazíme v části implementační i testovací. Předem je potřeba zdůraznit, že k danému tématu chybějí kvalitní a hlavně **konkrétní** zdroje informací, tudíž nebylo zprvu jasné, zda bude projekt úspěšný a zda nebude třeba velkého množství komplikovaných optimalizací, aby bylo možné dosáhnout alespoň nějakých relevantních výsledků. Tyto otázky budou zodpovězeny v závěrečné kapitole. Stručnější nástin celého projektu lze nalézt v [10].

Kapitola 2

Získávání znalostí z databází

Pod pojmem získávání znalostí z databází se skrývá proces, který nám umožní dozvědět se *zajímavé* a *netriviální* informace z uložených dat. Pojmem netriviální označujeme informace, které nejsou v databázi přímo uloženy nebo které se nedají zjistit pomocí jednoduchých dotazů (např. SQL). Takto získané informace by měly být zároveň uživatelsky přínosné a mít komerční využití. Samotný pojem bývá dnes často zkracován a většinou používáme označení **dolování z dat** (z anglického *data mining*), přesto že je dolování pouze jednu z částí celého procesu, jak se dozvíme dále v této kapitole. Jako příklad pro využití proces získávání znalostí z databází může posloužit představa obchodního domu, kde je v transakční databázi uložen veškerý seznam zboží, které jednotliví zákazníci nakupovali. Pomocí dolování zjistíme, že nezanedbatelný počet zákazníků nakoupil produkt A a současně také produkt B, prakticky tuto informaci můžeme využít např. umístěním produktů v nákupním středisku blíže k sobě či vytvořením speciálních akcí pro tyto produkty dohromady. Další příklady lze nalézt v [11].

2.1 Získávání znalostí z databází jako proces

Proces získávání znalostí z databází je poměrně komplikovaný a je rozdělen na několik základních kroků (obrázek 2.1):

1. **Čištění dat** – výsledkem by měly být konzistentní data bez odlehlých či chybějících hodnot.

2. **Integrace dat** – v některých případech mohou být potřebná data uložena ve více zdrojích (databázích, souborech, ...), úkolem integrace je potom taková data logicky zpracovat a spojit. Je výhodné mít data uložená v datovém skladu.

3. **Výběr dat** – ne všechna data mají pro konkrétní proces dolování smysl – výběr relevantních dat (tabulky, atributy, v případě datového skladu dimenze).

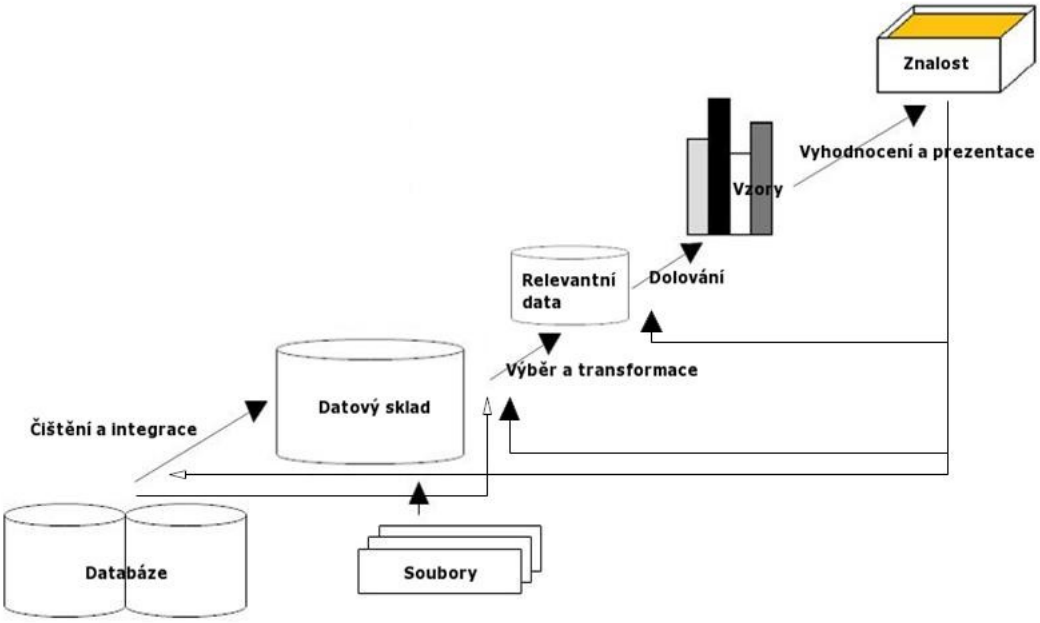
4. **Transformace dat** – zahrnuje agregaci atributů nebo úpravu jejich rozsahu podle potřeb dolovací úlohy.

5. **Dolování z dat** – jádro celého procesu, pomocí specifických algoritmů se provádí operace nad daty za vzniku modelů, vzorů.

6. **Vyhodnocení modelů a vzorů** – z velkého množství výsledků vybíráme jen ty pro náš účel nejzajímavější.

7. **Prezentace výsledků** – výstupem tohoto kroku by měly být výsledky celého procesu ve formě pochopitelné i pro běžné uživatele, hojně se využívá např. vizualizačních nástrojů.

Souhrnně můžeme první čtyři body nazvat jako tzv. fáze **předzpracování** dat, tedy fáze kdy vezmeme surová data a upravíme je pro potřeby samotného dolování. Význam samotného předzpracování je velký, jelikož od jeho kvality se přímo odvíjí kvalita vydolovaných výsledků, nicméně tato práce se nebude předzpracováním zabývat, zaměřuje se na samotnou fázi dolování s již relevantními daty v použitelném formátu.



Obrázek 2.1: Schéma procesu *získávání znalostí z databází* – zdroj [7]

2.2 Druhy dat pro dolování

Dolovat můžeme v podstatě z libovolných dat, která jsou buď persistentně uložená nebo se s časem mění (datové proudy). Mezi nejčastější persistentní zdroje dat patří **relační databáze**, kterou používá i školní program, jenž bude předmětem našeho zájmu. Relační databáze se skládá z tabulek s atomickými hodnotami atributů a je jednou z nejrozšířenějších forem uchovávání dat, manipulace s daty se provádí typicky pomocí jazyka SQL.
Dalším možným úložištěm dat může být *datový sklad*, který je reprezentován multidimenzionální kostkou. V datovém skladu jsou atributy reprezentovány dimenzemi a obsahují agregovaná data. Dolovat lze i z *transakčních databází* (nenormalizované tabulky s např. obchodními transakcemi, uloženy v pokladnách), *objektově-relačních databází*, *webu*, diskrétních či spojitých *proudů dat* a z mnoha dalších. Více o zdrojích dat pro dolování lze nalézt v [4].

2.3 Typy dolovacích úloh

Podle druhu modelu, který se snažíme z dat získat, můžeme dolovací úlohy rozdělit do dvou základních kategorií:

- **deskriptivní** – slouží pouze k charakteristice již uložených dat,
- **prediktivní** – vytváří model, podle něhož lze následně předpovídat určité vlastnosti nově získaných dat, příkladem může být *klasifikace*, kterou podrobněji popisuje kapitola 2.3.3 a kterou budeme pomocí specifického algoritmu implementovat.

Nástroje pro dolování však musí být i přes toto rozdělení dostatečně flexibilní a univerzální vzhledem k tomu, že používají různé typy algoritmů a protože často ani samotní uživatelé nemají zcela jasno, jaké informace je budou zajímat.
Jednotlivé **základní typy dolovacích úloh** jsou popsány dále. Informace k následujícím kapitolám byly čerpány především z [11] a z [4].

2.3.1 Popis konceptu/třídy

Cílem je popsat data (asociovaná s určitou třídou či konceptem) souhrnným, stručným a přesným způsobem. Používají se dva přístupy:

- **Charakterizace dat** – popis cílové třídy pomocí sumarizace jejích obecných vlastností.
- **Diskriminace dat** – popis cílové třídy na základě srovnání s ostatními třídami.

2.3.2 Dolování frekventovaných vzorů a asociačních pravidel

Pojem **frekventované vzory** logicky označuje vzory, které jsou často obsaženy v analyzovaných datech. Příklad dolovací úlohy spadající do této kategorie byl uveden již v kapitole 2. Ze získaných vzorů se následně generují asociační pravidla, která jsou popsána *minimální podporou* a *minimální spolehlivostí*. Tyto dva parametry specifikují významnost nalezeného pravidla. Mějme pravidlo:

$$kupuje(X,\text{'produkt A'}) \Rightarrow kupuje(X,\text{'produkt B'})[podpora = 10\%, spolehlivost = 50\%].$$

Asociační pravidlo nám říká, že v datech existují záznamy, kdy si kupující koupil produkt A a současně s ním i produkt B. V tomto případě značí *podpora*, že se tato asociace vyskytovala v 10% ze všech testovaných záznamů. Hodnota *spolehlivosti* potom vyjadřuje relativní míru výskytu položky na pravé straně ve všech transakcích obsahujících položku levé strany – konkrétně pokud si zákazník koupil produkt A, v kolika procentech případů k němu přikoupil i produkt B.
Ukázkové pravidlo patří do skupiny tzv. *jednodimenzionálních asociačních pravidel* – obsahuje pouze jeden predikát. Kdybychom ještě např. zjistili, že k produktu A dokupují produkt B především ženy, pravidlo bychom upravili:

$$kupuje(X,\text{'produkt A'}) \wedge pohlav(X,\text{'žena'}) \Rightarrow kupuje(X,\text{'produkt B'})$$
$$[podpora = 10\%, spolehlivost = 50\%],$$

čímž by z něj vzniklo *multidimenzionální asociační pravidlo*.

2.3.3 Klasifikace a predikce

Výsledkem dolování pomocí klasifikace je model dat, který jsme schopni aplikovat na nově příchozí data a rozdělit je pomocí něj do tříd. Zatímco pomocí **klasifikace** získáváme diskrétní hodnoty, **predikce** slouží k získání přesných hodnot ze spojitých intervalů. Proces klasifikace probíhá v několika krocích:

1. **Trénování** – na základě dat z tzv. *trénovací množiny* vytvoříme klasifikační model.
2. **Testování** – vzniklý model otestujeme na testovacích datech a ohodnotíme.
3. **Aplikace** – model aplikujeme na objekt neznámé třídy.

Je nutné podotknout, že u trénovacích i testovacích dat známe příslušné třídy – při trénování tuto znalost používáme pro vytvoření samotného modelu, v případě testování k porovnávání výsledků a hodnocení modelu. Jako příklad si představme banku poskytující půjčky. Na základě dolování uložených dat rozdělíme žadatele podle rizikovosti (tedy jestli půjčku včas a pravidelně spláceli bez problémů, či nikoli) do tříd a vygenerujeme model pomocí klasifikační dolovací metody. Při příchodu nového žadatele vytvoříme objekt s příslušnými vlastnostmi a aplikujeme vydolovaný klasifikační model, který nám umožní předpovědět s určitou pravděpodobností rizikovost nově příchozího.
Klasifikačních metod je celá řada, jako příklad můžeme uvést nejběžněji využívané *rozhodovací stromy* či *neuronové sítě*. Námětem této práce je implementace klasifikační metody pomocí *genetického algoritmu*, více se tomuto tématu bude věnovat kapitola 3.3.

2.3.4 Shluková analýza

Shluková analýza rozděluje data do tříd ne podle specifických pravidel, ale podle vzájemné vzdálenosti objektů v prostoru atributů – shlukuje co nejpodobnější objekty. Lze ji s úspěchem použít např. v předzpracování dat k identifikaci odlehlých hodnot. Více o předzpracování dat v [11].

2.3.5 Analýza odlehlých hodnot

V některých situacích je naopak potřeba analyzovat odlehlá data – objekty, které se odlišují od ostatních, např. při zjišťování nestandardního chování (podvodné použití kreditní karty a další).

2.3.6 Analýza evoluce

Cílem je analyzovat měnící a vyvíjející se data. Pro tento účel lze zvolit i některé standardní výše popsané metody, nicméně existují i metody specializované.

Kapitola 3

Genetické algoritmy

3.1 Obecné genetické algoritmy

Genetický algoritmus je stochastická optimalizační metoda založená na principech evoluční biologie [6]. Jednotlivé datové záznamy jsou chápány jako jedinci s vlastním genetickým kódem. Pro správnou orientaci v problematice genetických algoritmů je třeba definovat několik základních pojmů, které se týkají struktury datových jedinců:

- **gen** – označení jedné určité vlastnosti,
- **chromozóm** (**genom**) – označuje celého jedince (soubor vlastností),
- **populace** – pod tímto pojmem rozumíme skupinu jedinců.

Naším prvotním cílem je vytvořit čitelné genomy pro jednotlivé datové záznamy – zpravidla pomocí vhodného algoritmu *zakódujeme* hodnoty genů (příhodná může být např. binární reprezentace, která je nejrozšířenější, případně je možné použít méně obvyklé způsoby s pomocí matic, stromů, neuronových sítí, ...). Následně vytvoříme náhodnou *inicializační* populaci, kam umístíme startovní jedince. Stěžejní pro práci algoritmu je tzv. **fitness funkce**, která dokáže číselně hodnotit jednotlivé genomy. Podle její hodnoty (angl. *fitness value*) se následně celá populace dále vyvíjí – jedinci s vysokou fitness hodnotou přežívají, s nízkou naopak zanikají. Mezi operace prováděné nad populací jedinců patří:

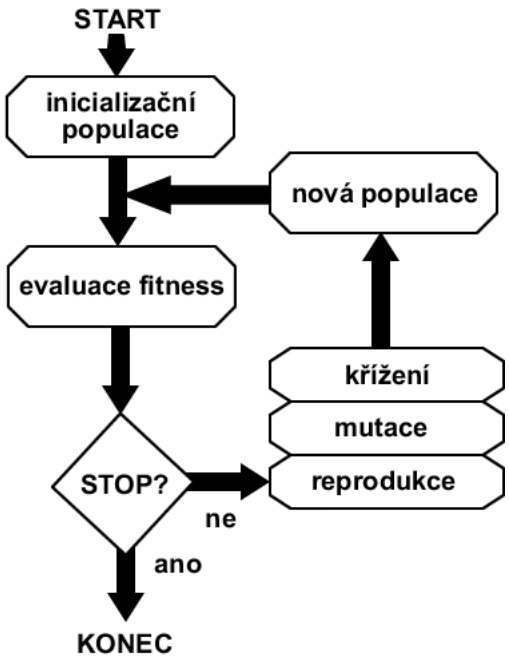
- **výběr** (*selekce*) – příslušným způsobem vybereme jedince z populace, aplikace např. při **reprodukci** (umístění nezměněného jedince do nové populace),
- **křížení** – ve většině případů probíhá mezi dvěma jedinci, kdy si navzájem vymění části genomu podle binární masky (**uniformní křížení**) – nejčastěji dělíme jedince na dvě části (*křížení jednobodové*),
- **mutace** – změníme náhodný gen na libovolnou hodnotu (z jeho domény).

Operace křížení a mutace je třeba používat v rozumné míře – pro každou implementaci algoritmu musíme určit pravděpodobnosti křížení (cross-probability p_c) a mutace (mutation probability p_m). Pokud jsou pravděpodobnosti špatně nastaveny, ovlivníme tím negativně chování algoritmu (příliš dlouhé vyhledávání při vysokém křížení, při málo časté mutaci zase hrozí neobjevení důležitého genu atd.). Vliv pravděpodobností se liší i na základě typu použitých optimalizací v konkrétních implementacích (obecně se používá vysoká pravděpodobnost křížení $p_c > 70\%$ a nízká pravděpodobnost mutace $p_m < 5\%$). Na základě fitness

ohodnocení jednotlivých nových genomů a pomocí popsaných operací se snažíme o vytváření „silnějších“ populací (s jedinci s velkou fitness hodnotou) – naším cílem je pak najít nejlepšího jedince či populaci. Ukončení algoritmu je specifické pro konkrétní implementace (např. maximální počet vytvořených populací, minimální míra „vylepšení“ populace v porovnání s předchozí atd.).
Obecný genetický algoritmus může probíhat například takto:

1. Vytvoříme náhodně inicializační populaci jedinců.
2. Vypočítáme fitness hodnotu jednotlivých genomů v populaci.
3. Vybereme skupinu nejzdatnějších a pomocí reprodukce, křížení a mutace z nich vytvoříme novou populaci.
4. Pokud nebyla splněna podmínka pro skončení algoritmu, vrátíme se ke kroku č. 2, jinak ukončíme algoritmus a získáváme cílovou populaci.

Toto schéma je znázorněno na obrázku 3.1. Více informací o obecném genetickém algoritmu nalezneme v [6] a [5].



Obrázek 3.1: Možný průběh genetického algoritmu

3.2 Problémy genetických algoritmů

Genetické algoritmy (dále jen GA) jsou mladé a populární odvětví, proto můžeme být svědky jejich častého použití na místě, kde by bylo vhodnější použít algoritmus jiný, případně narážíme na jistá omezení a nedokonalosti dosud známých řešení. Další problémy vyplývají ze samotného návrhu konkrétní implementace – úspěšnost algoritmu je závislá především na správnosti fitness funkce, ovšem i přes kvalitní fitness funkci nemusí GA

dokonvergovat ke správnému výsledku (např. tzv. *klamné problémy* – z angl. deceptive problems, kdy k ideálnímu řešení nelze dojít postupným vylepšováním – stojí stranou ostatních jedinců).
Je též nutné podotknout, že se jedná o velmi složitý algoritmus, kde pomocí jednotlivých operací manipulujeme se samotnými datovými záznamy a při práci s velkým objemem dat pracuje pomalu, proto se vyvíjí metody pro *paralelní zpracování* (angl. parallel processing) a vzorkování (angl. sampling). Zejména v obchodním sektoru se pak můžeme setkat s problémy ohledně přílišné různorodosti dat, kde se dnes začínají prosazovat spíše fuzzy systémy – podrobnosti v [6].

3.3 Použití genetického algoritmu při dolování z dat

Následující sekce vychází převážně z [3].

3.3.1 Základní náhled a pojmy, reprezentace

Každý datový záznam z relační databáze můžeme reprezentovat jako víceprvkovou relaci hodnot jednotlivých atributů. Použijeme-li názvosloví evolučních algoritmů, hodnoty atributů budou jednotlivé geny, genomem poté označíme řetězec (relaci) takovýchto genů. Při označení atributů att_1, att_2, ..., att_n jsou relace popisující datové záznamy podmnožinou kartézského součinu $\mathbf{dom}(att_1)$ x $\mathbf{dom}(att_2)$ x ... x $\mathbf{dom}(att_n)$, kde dom(att_i) označuje všechny hodnoty, kterých může att_i nabývat (*doména atributu*) – datový záznam můžeme tedy chápat jako identifikovatelný seřazený seznam hodnot atributů.
Naším cílem je vytvořit populaci jedinců, která dokáže určitým způsobem popsat a shrnout jedince (datové záznamy) z databáze. Aby toto bylo možné, je třeba umožnit algoritmu nějakým způsobem zobecňovat jedince v populaci – k tomuto účelu poslouží *prázdná hodnota* genu, kterou uměle přidáme do všech atributových domén. Tímto způsobem bude algoritmus schopen označit gen, který nemá vliv na jeho další vývoj. Jako výstupní data tedy očekáváme populaci **obecnějších jedinců**, kteří popisují uložená data vzhledem k námi označenému třídnímu atributu.
Dolování můžeme tedy v našem případě charakterizovat jako hledání užitečného a dříve neznámého výrazu pro popis třídy bez nutnosti procházet celý prostor zmíněného kartézského součinu.
Užitečnost jednotlivých výrazů je individuální a musí být řešena obecně, nicméně existuje několik skupin, které můžeme ihned označit za zbytečné:

- výraz který neodpovídá žádnému záznamu – např. $pohlav(X,\text{'muž'}) \wedge pohlav(X,\text{'žena'})$, je logicky jasné že tomuto pravidlu neodpovídají žádné záznamy,
- výraz který popisuje skupinu, jejíž jedinci nemají žádné společné vlastnosti – tohoto stavu můžeme lehce dosáhnout pomocí **disjunkce**,
- výraz o jediné hodnotě – lze vyčíst jednoduše ze samotných dat.

Na základě těchto poznatků můžeme stanovit základní pravidla pro hledané výrazy:

1. každý výraz může obsahovat pouze unikátní atributy,
2. výrazy nebudou obsahovat disjunkci,
3. každý výraz by měl obsahovat alespoň jednu konjunkci.

V našem případě bude algoritmus navržen tak, aby první dva body splňoval, bod třetí vynutíme pomocí samotné fitness funkce (jedinec bez jediné konjunkce nebude mít oproti ostatním šanci uspět).

3.3.2 Klasifikace pomocí GA

Jak již bylo řečeno v kapitole 2.3.3, klasifikace umožňuje rozdělení dat do určitých tříd. Z pohledu genetického algoritmu je ovšem zbytečné rozdělovat vstupní data na *trénovací* a *testovací* množiny, jelikož testování kvality pravidel bude úkolem fitness funkce, která bude hodnotit pravidla za běhu ve fázi trénování. Testování vydolovaných pravidel lze provést následně ručně při aplikaci pravidel na data, u kterých cílovou třídu známe, nicméně v samotném algoritmu práce s daty, která bychom explicitně označili jako „testovací“, probíhat nebude – tolik k odklonu od standardního modelu klasifikačních dolovacích úloh.

3.3.3 Fitness funkce

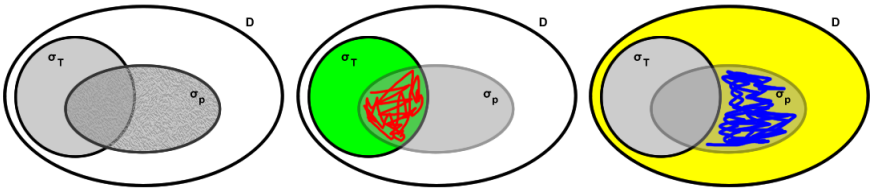
Aby mohl algoritmus smysluplně fungovat, je naprosto zásadní definovat vhodnou fitness funkci (viz obecný úvod o GA, kapitola 3). V našem případě by měla fitness hodnota jedince vyjadřovat, jak kvalitně jsme schopni zařadit nový datový záznam do označených tříd pomocí tohoto jedince. Musíme pamatovat na to, že každé vygenerované pravidlo může pokrývat množinu uvnitř i vně třídy, případně nemusí pokrývat nic.
Pro formální definici fitness funkce zavedeme pojmy:

- D – množina všech jedinců v databázi,
- $\sigma_T(D)$ – množina jedinců třídy,
- $\sigma_p(D)$ – množina jedinců pokryta pravidlem p.

Nyní můžeme definovat fitness funkci pro pravidlo p vzorcem 3.1.

$$F(p) = \frac{|\sigma_p(D) \cap \sigma_T(D)|}{|\sigma_T(D)|} - \frac{|\sigma_p(D) \backslash \sigma_T(D)|}{|D \backslash \sigma_T(D)|}, F(p) \in < -1; 1 > \tag{3.1}$$

Funkce v podstatě vyjadřuje rozdíl relativních četností správně a špatně klasifikovaných datových záznamů, kdy správně klasifikovaní jsou jedinci uvnitř množiny třídy, kterou se pravidlo snaží popsat. Funkce může logicky nabývat hodnot od -1 do 1, kdy hodnotou 1 bude klasifikováno ideální pravidlo, které samo o sobě popisuje všechny jedince pouze v dané třídě.
Obrázek 3.2 poslouží jako grafická pomůcka pro lepší představu o principu fitness funkce.



Obrázek 3.2: Grafické vyjádření fitness funkce

Vzorec 3.1 je vlastně rozdílem dvou poměrů – první zlomek vyjadřuje, kolik procent jedinců ze třídy pravidlo popisuje (na obrázku $\frac{cervena}{zelena}$), druhý zlomek zase kolik procent jedinců popisuje pravidlo mimo třídu (na obrázku $\frac{modra}{zluta}$).

3.3.4 Další optimalizace

Vzhledem k netradičnímu použití algoritmu byly vybrány některé optimalizace popsané v [5].

1. **Selekce** – cílem selekce je, jak již bylo zmíněno, vybrat náhodně jedince z počáteční populace. Jedná se o nejčastěji prováděnou operaci nad populací jedinců – pomocí selekce vybíráme jedince pro křížení, pro mutaci či pro prosté přenesení do populace nové. Z tohoto pohledu je tedy tato operace velice významnou součástí algoritmu. V genetických algoritmech existuje mnoho přístupů, jak selekci řešit – od čistě náhodného výběru, přes tzv. *turnaj*, kdy vybereme náhodnou množinu jedinců a podle fitness hodnoty vybereme nejsilnějšího, až po rozdělení pravděpodobnosti výběru v přímé souvislosti s fitness hodnotou. Při realizaci projektu bude použit výběr pomocí metody *rank selection*, který funguje na principu rozložení pravděpodobnosti podle pořadí fitness hodnoty v populaci. Seřadíme tedy jedince podle jejich fitness hodnoty od nejvyšší po nejnižší a přiřadíme jim pravděpodobnosti, že budou vybráni selekčním mechanismem, přímo úměrné jejich pořadí. Absolutní fitness hodnota tedy nebude hrát při selekci přímou roli.

2. **Křížení** – algoritmus bude používat zmíněné jednobodové křížení, kdy genom jedinců rozdělíme na dvě části v náhodném bodě a tyto pak mezi nimi vyměníme. Bylo upuštěno od uniformního křížení s náhodnou binární maskou zejména kvůli jeho rozkladnému charakteru.

3. **Přidání jedince do populace** – cílem této optimalizace je neztrácet v průběhu evoluce již získaná kvalitní řešení bez nutnosti zavádění složitých heuristik (např. speciální porovnávání rodičů a potomků křížení, jak je popsáno v [3]). Novou populaci budeme tvořit na základě předešlé a sice tak, že vždy při vytvoření nového jedince (ať už jakýmkoli způsobem) porovnáme jeho fitness hodnotu s **nejnižší** fitness hodnotou v aktuální populaci. Pokud bude nový jedinec silnější, odstraníme dosavadního nejslabšího jedince z populace a nově vytvořeného vložíme, v opačném případě se populace nemění a nově vytvořeného jedince ignorujeme.

4. **Jedinci v populaci** – populace jedinců musí být v každém stavu vždy **unikátní**. Tímto se zabrání umělému rozmnožení velmi silného jedince na úkor slabších, což je vzhledem k typu úlohy silně nežádoucí.

Komentář k úspěšnosti jednotlivých optimalizací je uveden v sekci 9.

Kapitola 4

Dolovací systém FIT

Následující podkapitoly obsahují popis školního dolovacího systému fakulty FIT VUT v Brně.

4.1 Použité technologie

Dolovací systém můžeme označit jako aplikaci s architekturou *klient - server*, kde roli klienta zastává program `Dataminer` implementovaný v jazyce Java, serverovou část potom databázový server Oracle. Následující podkapitoly vycházejí především z [12] a [9].

4.1.1 Platforma NetBeans

Vývojová platforma NetBeans je součástí open source projektu společnosti *Sun Microsystems*, která poskytuje modulárně rozšiřitelný základ pro vytváření rozsáhlých aplikací [1]. Za zmínku stojí také grafické API *NetBeans Visual Library*, která je součástí zmíněné *NetBeans Platform*. Knihovna obsahuje vizualizační nástroje, jejíchž záměrem je sjednocení zobrazení aplikací postavených na této platformě. Více informací o projektu lze nalézt [2].

4.1.2 Oracle Data Mining

V původní podobě umožňoval systém pracovat pouze nad databází MySQL. Ta však samotné dolování nijak nepodporuje, tudíž byla nahrazena databází Oracle, která nabízí API pro samotné dolování. Původní verze systému byla přizpůsobena pro verzi *Oracle 10g Release 2* a veškeré dosavadní moduly používaly technologii *Oracle Data Mining (ODM)*, případně pod názvem *Data Mining Engine (DME)*. V současné době byl systém upraven a server běží na verzi 11, nicméně ani v této verzi nenabízí Oracle podporu pro genetické algoritmy, tudíž se ODM dále zabývat nebudeme a k implementaci bude použito standardních prostředků pro komunikaci s databázovým serverem.

4.1.3 DMSL

Jazyk DMSL (*Data Mining Specification Language*) slouží k popisu celého dolovacího procesu a jeho uložení, je odvozen od XML. Díky tomu je bez problému přenositelný a jednoduše čitelný. Umožňuje definovat vstupní data, jejich transformace, znalosti z nich získané, nastavení jednotlivých dolovacích modulů a podobně. Jazykem DMSL se zabývá [8]. Podstatný z hlediska implementace je poznatek, že pro každý projekt je vytvořen DMSL soubor, do kterého se v průběhu práce s dolovacím procesem zaznamenávají informace – pomocí

datových struktur odvozených od tohoto formátu jsou spolu schopny komunikovat jednotlivé komponenty (popsány v kapitole 4.3.1) a díky tomuto lze i celý dolovací proces kdykoli uložit a následně znovu načíst.

4.2 Vývoj systému

Dolovací systém je na naší fakultě vyvíjen již několik let v rámci diplomových, v menší míře pak i bakalářských prací. Vývoj započal v roce 2006 Ing. Doležal, který vytvořil samotné jádro systému, ke kterému bylo možné připojit dolovací moduly. O rok později potom Ing. Galét rozšířil systém o grafické rozhraní postavené na NetBeans Visual Library. Původní jádro však bylo poměrně omezené – tuto skutečnost změnil Ing. Krásný v roce 2008, kdy jej přepracoval (již nedochází k nekonzistentním situacím, kvalitní struktury, všechny změny lze nalézt v [9]). Na jeho snažení pak navázal v loňském roce Ing. Šebek, který zaměřil svoje úsilí zejména na fázi předzpracování dat a mj. vyladění komunikace s databází (podrobnosti v [12]). Moje práce bude zaměřená na vývoj dolovacího modulu pro tento systém.

4.3 Grafické uživatelské rozhraní systému

Grafické rozhraní využívá již zmíněnou knihovnu *NetBeans Visual Library*, AWT a *Swing* a je inspirováno profesionálním rozhraním aplikace *SAS Enterprise Miner*. Pracuje na principu rozdělení jednotlivých dolovacích úloh do projektů, při jejichž vytváření je nutné zadat kontaktní údaje pro pracovní databázi. Následnou dolovací úlohu lze velmi pohodlně sestavit pomocí „přetažení“ jednotlivých komponent na pracovní plochu (využití Drag&Drop) a jejich následnému spojení. Takto vytvořená úloha je nejenom velmi přehledná, ale lze ji jednoduše měnit a rozšiřovat. Grafická metadata (např. informace o poloze jednotlivých komponent na pracovním plánu) se ukládají ve formě komentářů do souboru DMSL, což opět dopomáhá k jednoduchému uložení celého projektu.

4.3.1 Komponenty dolovacího procesu

Seznam všech komponent s popisem je k dispozici v práci [9] a informace o jejich nejaktuálnějších rozšířeních a změnách potom v [12]. Zde je uveden pouze krátký přehled a informace o komponentách, které mohou být užitečné při dolovací úloze pomocí genetického algoritmu.

- **Select data** – komponenta pro výběr vstupních dat. Komponenta nabízí možnost vybrat vstupní data z databáze, nebo CSV souboru. V případě výběru dat z databáze je možné zvolit konkrétní sloupce ze zdrojové tabulky, případně spojit několik tabulek dohromady. Tato komponenta je nutnou součástí každé dolovací úlohy.
- **Transformations** – komponenta pro transformace vstupních dat (*normalizace*, *diskretizace* a další).
- **Reduce/Partition** – komponenta umožňující redukci a rozdělení vstupních dat. Pomocí specifických algoritmů dokáže komponenta zredukovat objem vstupních dat, případně rozdělit data na *trénovací* a *testovací*, což mohou vyžadovat některé klasifikační algoritmy.
- **Insight** – komponenta pro analýzu vstupních dat. Pomocí této komponenty jsme schopni zobrazovat informace o vstupních datech – zobrazení v tabulkách, nejrůznější

formy grafů a pohledů na data. Umožňuje export vybraných dat do CSV souboru, což je velmi užitečné ve fázích testování dolovacích modulů.

- **VIMEO** – komponenta fungující jako datový filtr. VIMEO nám umožňuje definovat funkce nad daty, jejichž výsledky můžeme následně zobrazit pomocí komponenty Insight a použít dále v procesu dolování.
- **Report** – zobrazení výsledků dolovacího procesu. Tato komponenta je implementována jako součást každého dolovacího modulu zvlášť.

Tyto komponenty jsou v uživatelském rozhraní seskupeny pod hlavičkou *Core*, neboť jsou součástí jádra systému. Pod hlavičkou *Mining Modules* se nacházejí jednotlivé dolovací moduly, které se tvoří nezávisle. Na toto místo bude umístěn i modul pro dolování pomocí GA.

Kapitola 5

Vytvoření dolovacího modulu

Systém můžeme rozdělit, jak již bylo zmíněno, na dvě části - centrální **jádro** a připojené dolovací **moduly**. Tato kapitola popisuje jakým způsobem je možné připojit k jádru nový modul.

5.1 Jádro systému

Za základní část jádra můžeme označit třídu *Kernel*. Přes tuto třídu dochází ke spolupráci ostatních součástí jádra – třída *DMSL* poskytuje nástroje pro práci s DMSL souborem, třídy *ConnectionInfo* zajišťují připojení k databázovému serveru, třída *Caller* volání zkompilovaných interních funkcí a třídy *Clipping*, *Normalization* a *Discretization* umožňují transformaci vstupních dat (použití v komponentě Transformations z kapitoly 4.3.1). Další informace o jádře systému najdeme v [9]. Z [12] víme, že pro jednodušší připojení nového modulu byl vytvořen virtuální systém souborů. Tento systém je implementován souborem *layer.xml* a umožňuje připojit nebo změnit modul bez nutnosti rekompilace jádra.

5.2 Vytvoření nového modulu v prostředí NetBeans IDE

Pro vytvoření nového modulu je nutné využít vývojový software NetBeans IDE. V současné době systém funguje na verzi NetBeans IDE 6.5. Při tvorbě modulu jsem se držel informací z [9] a z [7].
V hlavním menu zvolíme tvorbu nového projektu, následně v kategorii *NetBeans Modules* vybereme možnost *Module*, zadáme jméno, cestu a podle toho, zda máme k dispozici zdrojové kódy projektu, nebo ne, zvolíme možnost *Add Module to Suite*, respektive *Standalone Module*. Jako *Code Name Base* zadáme `cz.vutbr.fit.dataminer`. Po vytvoření modulu je pro jeho funkčnost nutné naimportovat knihovny, potřebné pro jeho běh. Pravým tlačítkem klikneme na podsložku projektu s názvem *Libraries* a zvolíme *Add Module Dependancy*. Přes otevřený dialog vybereme minimálně *Dialogs API, dm-api, dm-core-wrapper, UI Utilities API* a *Utilities API*. V případě, že bychom si při implementaci nevystačili s těmito závislostmi, obdobným způsobem je možné navázat další.

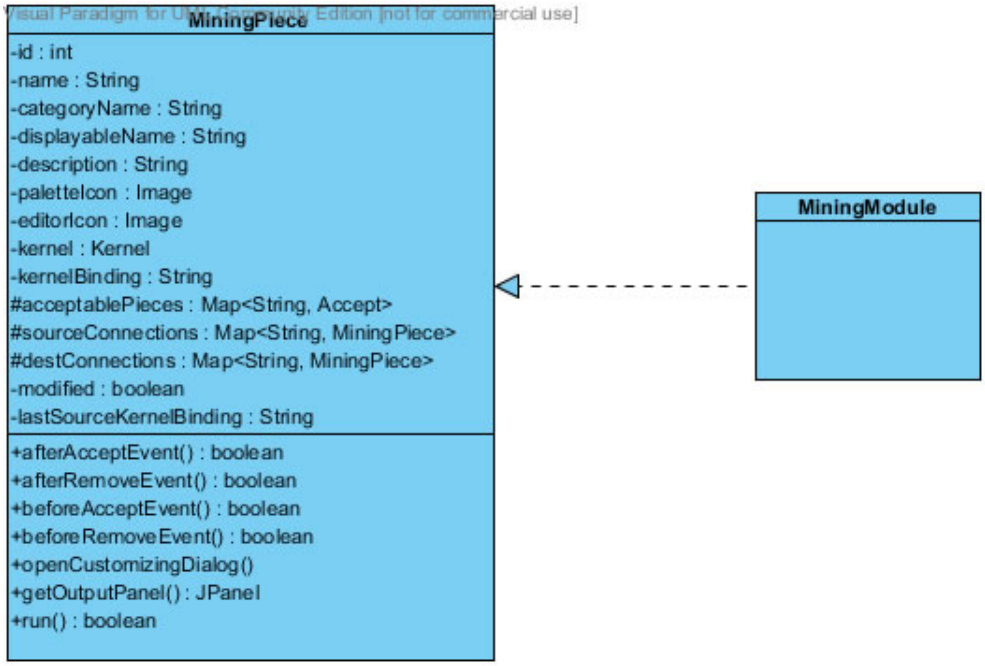
5.2.1 Implementace třídy *Mining Piece*

Aby byl modul přístupný v aplikaci, je nutné implementovat abstraktní třídu *Mining Piece* z API vrstvy. Třídy dědící od *Mining Piece* představují prvky, které je možné vložit do

dolovacího procesu. Třída, kterou takto naimplementujeme se poté stává základem celého modulu.
Pomocí lokalizačního bundle nastavíme v konstruktoru parametry nově vytvářeného modulu:

```
setDisplayableName(NbBundle.getMessage(ModuleName.class, "NAME_ModuleName"));
setDescription(NbBundle.getMessage(ModuleName.class, "HINT_ModuleName"));
setPaletteIcon("cz/vutbr/fit/dataminer/module/resources/Module16.png");
setEditorIcon("cz/vutbr/fit/dataminer/module/resources/Module48.png");
```

První dva řádky zápisu odkazují právě na lokalizační bundle a říkají nám, že název a popis modulu budou uloženy v souboru *Bundle.properties* pod vlastnostmi *NAME_ModuleName*, resp. *HINT_ModuleName*. Další dva řádky udávají cestu k ikonám modulu pro paletu komponent a pro editor.
Strukturu abstraktní třídy *MiningPiece* můžeme vidět na obrázku 5.1.



Obrázek 5.1: Abstraktní třída *MiningPiece*

Dále si uvedeme několik důležitých metod, na které při implementaci *MiningPiece* narazíme:

- **openCustomizingDialog()** – povinná metoda, zajišťuje otevření konfiguračního dialogu komponenty (v našem případě dolovacího modulu). Dialog by měl být vytvořen pomocí prostředků platformy NetBeans, které se postarají o korektní chování a o zachování jednotného vzhledu.
- **afterAcceptEvent()** – metoda, kterou systém provede po vložení komponenty na pracovní plochu dolovacího procesu. Jejím úkolem je většinou zapsat výchozí konfiguraci komponenty do DMSL souboru.
- **afterRemoveEvent()** – metoda, kterou systém provede po vyjmutí komponenty z pracovní plochy.

- **run()** – spustí hlavní proces komponenty.
- **getOutputPanel()** – metodu volá komponenta *Report* (viz. kapitola 4.3.1), zajišťuje vytvoření panelu pro zobrazení a případnou aplikací výsledků práce implementované komponenty.

Při implementaci je doporučeno prozkoumat způsob realizace již vytvořených modulů pro jednodušší pochopení všech principů.

5.2.2 Integrace do aplikace

Jak již bylo zmíněno na začátku této kapitoly, propojení modulů se zbytkem systému zabezpečuje soubor `layer.xml`. Důležité je připomenout, že jednotlivé moduly o sobě navzájem nevědí a není možné mezi nimi sdílet třídy ani jiná vnitřní data.
Aby se náš nový modul objevil v paletě komponent (třída *MiningPieceRegistry*), je třeba provést následující úpravy v souboru `layer.xml` [9]:

```
<filesystem>
  <folder name="MiningPieceRegistry">
    <folder name="MiningModules">
      <file name="dataminer-module-testmodule-TestModule.instance" />
    </folder>
  </folder>
```

Název *TestModule* samozřejmě zaměníme za název námi implementovaného modulu.
Následujícím kódem charakterizujeme, které komponenty bude náš modul akceptovat na vstupu a na výstupu:

```
  <folder name="MiningPieceConfig">
    <folder name="dataminer-module-testmodule-TestModule">
      <folder name="accept">
        <file name="cz-vutbr-fit-dataminer-editor-palette-items-Select"/>
      </folder>
    </folder>
    <folder name="cz-vutbr-fit-dataminer-editor-palette-items-Report">
      <folder name="accept">
        <file name="dataminer-module-testmodule-TestModule"/>
      </folder>
    </folder>
  </folder>
</filesystem>
```

Tímto zápisem říkáme, že na vstupu bude náš modul akceptovat připojení od komponenty *Select Data* a na výstupu komponentu *Report* (viz opět 4.3.1).

Kapitola 6

Implementace modulu pro klasifikaci pomocí genetického algoritmu

V této kapitole bude shrnut vývoj modulu z hlediska implementace, tedy samotného programování. Popíšeme vytvořené třídy a jejich některé významné metody a představíme vytvořené uživatelské rozhraní modulu.
Základní implementace probíhala přesně podle principu popsaného v kapitole 5.2. Navíc nebylo nutné (jako v některých dřívějších případech) zasahovat do kódu jádra – modul je zcela samostatný a lze v současné době bez problémů připojit k aktuální verzi aplikace.

6.1 Připojitelné komponenty

Modul bude reprezentován jako komponenta s názvem *Genetic Algorihm* s ikonkou šroubovice DNA. Bude umožňovat vstup z komponent *Select Data*, *Transformation* a *Reduce/Partition*, na výstupu bude očekávána komponenta *Report*. Tyto závislosti byly nastaveny v souboru `layer.xml` podle kapitoly 5.2.2.

6.2 Objektový model

Na obrázku 6.3 můžeme vidět uspořádání stěžejních tříd použitých při implementaci. Následující podkapitoly se budou zabývat jejich jednotlivými významy a důležitými metodami.

6.2.1 GAConfig

Na úvod je třeba zmínit tuto třídu, jejímž posláním je sdílení konfiguračních parametrů algoritmu mezi hlavními třídami, použitými v implementaci. Je tvořena sadou privátních vlastností s příslušnými metodami pro *get* a *set* (metody umožňující přístup k těmto vlastnostem). Jako příklad konfiguračních vlastností, kterou třída obsahuje, můžeme uvést třídní atribut, velikost inicializační populace, tabulka zdrojových dat, mapa domén všech vstupních atributů, pravděpodobnosti křížení, mutace a další. Tato třída je v průběhu algoritmu nainicializována a následně předávána mezi jednotlivými objekty.

6.2.2 GAModule

Tato třída je implementací v kapitole 5.2 popsané třídy *MiningPiece* a samotný proces její realizace probíhal přesně podle popisu ve zmíněné kapitole.
Před vložením komponenty pro genetické dolování je vydolováno vnitřní ID modulu, aby jej bylo možné odlišit od ostatních komponent v projektu. Po přidání modulu do dolovacího procesu pomocí funkce **afterAcceptEvent()** vložíme do DMSL souboru následující záznam:

```
<DataMiningTask name="ZDROJ_DOLOVANI" language="XML">
  <GAClassification>
    <DataInfo>
      <targetAtt>null</targetAtt>
    </DataInfo>
    <Params>
      <PSize>100</PSize>
      <CProb>0.8</CProb>
      <MProb>0.8</MProb>
      <PCount>200</PCount>
      <PRes>20.0</PRes>
    </Params>
  </GAClassification>
</DataMiningTask>
```

Inicializační hodnoty jednotlivých parametrů byly nastaveny na základě testování. Pro lepší představu uvedeme významy těchto parametrů:

- **targetAtt** – atribut, jehož hodnoty reprezentují jednotlivé třídy dat,
- **PSize** – velikost inicializační populace,
- **CProb** – pravděpodobnost křížení,
- **MProb** – pravděpodobnost mutace,
- **PCount** – počet iterací algoritmu (počet vytvořených populací),
- **PRes** – procento jedinců z výsledné populace, které označíme za výsledek.

Modul nelze spustit, pokud nebyl nastaven třídní atribut *targetAtt*.
Při poklepání na obrázek modulu na pracovní ploše, otevře se pomocí funkce **openCustomizingDialog()** dialog třídy *GAModuleParamPanel* (viz sekce 6.2.7). Prostřednictvím tohoto dialogového okna lze změnit, resp. nastavit hodnoty zde zmíněných atributů algoritmu.
Dalším krokem je samotné spuštění algoritmu, což z implementačního hlediska znamená vyvolání funkce **run()**, která spustí metodu **initMineTask()**. Tato metoda inicializuje ze souboru DMSL objekt třídy *GAConfig* (sekce 6.2.1) a předá jej třídě *GAMineTask*, která následně převezme řízení celého algoritmu.
Po dokončení algoritmu obdrží třída výslednou populaci pravidel společně s daty pro vytvoření výsledného grafu ve formě objektu *GAResult*. Tento objekt je předán funkcí **getOutputPanel()** výstupnímu panelu *GAModuleOutputPanel* (sekce 6.2.8), který zabezpečuje zobrazení výsledků dolování.
Na závěr, po odstranění modulu z dolovacího procesu je metodou **afterRemoveEvent()** odstraněn příslušný DMSL záznam.

6.2.3 GAMineTask

První důležitou metodou třídy *GAMineTask* je **initData()**, která je volána v průběhu inicializace konfigurační třídy (metoda *initMineTask()* hlavní třídy modulu *GAModule*). Tato metoda nastaví a zařídí připojení k databázi a následně vytvoří mapu domén jednotlivých atributů.
Po dokončení inicializace je opět z hlavního modulu vyvolána funkce **run()**, která reprezentuje samotný genetický algoritmus. Začíná vytvořením inicializační populace a následně spustí cyklus vylepšování populace (využití třídy *GAPopulation* – viz sekce 6.2.6). Po dosažení ukončovací podmínky (v našem případě pokud počet iterací dosáhne zadaného limitu) algoritmus ukončí a vytvoří množinu s požadovanou částí výsledných pravidel. Během samotných iterací dochází také ke sběru informací o vývoji fitness, které jsou přidány po ukončení do výsledkového objektu *GAResult*.

6.2.4 GAResult

Třída obsahující výsledky dolování – množinu vygenerovaných pravidel a data potřebná pro vytvoření grafů pro vizualizaci průběhu algoritmu.

6.2.5 GAIndividuum

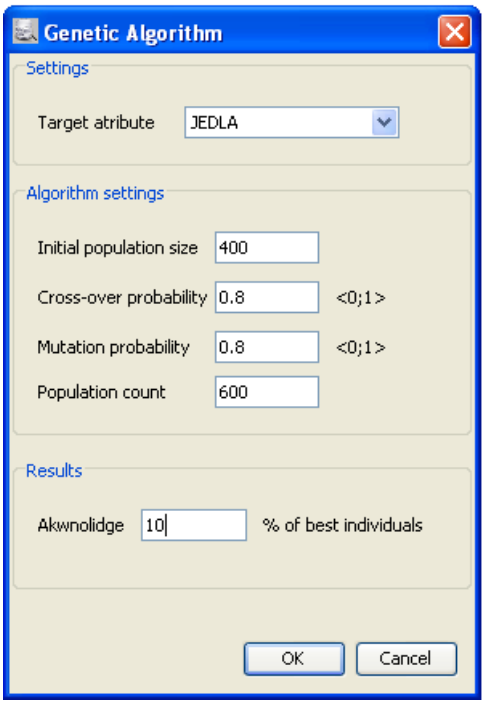
Tato třída reprezentuje jedince (pravidlo) v algoritmu (více o reprezentaci dat v GA kapitola 3.3.1). Jak víme, jedinec je popsán *genomem* a *fitness hodnotou*, což jsou dva hlavní atributy každého objektu této třídy. Ostatní vlastnosti třídy se pouze snaží o zjednodušení práce se třídou.
Třída disponuje třemi typy *konstruktoru* – jedinec může být vytvořen náhodně (při inicializaci), na základě křížení a nebo pouhou kopií již existujícího jedince. Metoda **mutate()** poté umožňuje zmutovat genom jedince.
Za stěžejní můžeme označit metodu **computeFitness()**, která je schopna na základě principů, uvedených v kapitole 3.3.3, a pomocí agregačních dotazů nad vstupním zdrojem dat vypočítat a nastavit fitness hodnotu jedince.

6.2.6 GAPopulation

Třída *GAPopulation* je abstrakcí populace v kontextu genetického algoritmu – využívá objekty typu *GAIndividuum*.
Při konstrukci vytvoří inicializační populaci náhodných individuí. Metoda **select()** je implementací *rank selection* a slouží k výběru jedince z populace (jedna z optimalizací z kapitoly 3.3.4). Pomocí funkcí **createHalfMask()** a **createRndMask()** jsme schopni vytvořit masku pro křížení – jednobodové, resp. uniformní (které nakonec nebylo použito).
Nejvýznamnější metodou je pak metoda **improve()**, jež implementuje jednu iteraci genetického algoritmu. Dochází v ní ke tvoření nové populace na základě předcházející s použitím klasických genetických operací (viz kapitola 3).

6.2.7 GAModuleParamPanel

Tato třída rozšiřuje třídu *JPanel* z grafického balíku *swing* a implementuje dialog pro nastavení atributů algoritmu pro dolování a je znázorněn na obrázku 6.1. Za zmínku stojí metody **resetUIDueDMSL()** a **fillUIComponents()**, které načtou potřebná data z DMSL



Obrázek 6.1: Dialog s nastavením dolovacího modulu

a z databáze a inicializují jednotlivá pole formuláře. Po stisknutí tlačítka OK se zadané hodnoty uloží do DMSL pomocí funkce **updateDMSL()**.

6.2.8 GAModuleOutputPanel

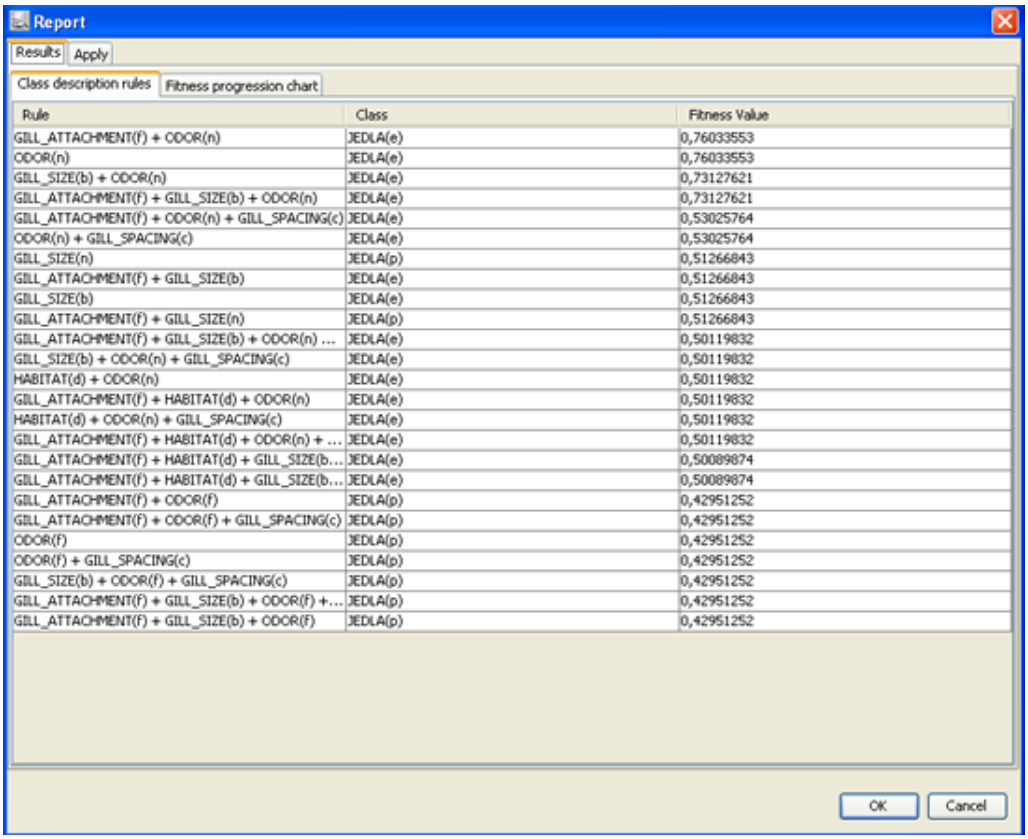
Podobně jako *GAModuleParamPanel* i třída *GAModuleOutputPanel* rozšiřuje třídu *JPanel*. Pomocí této třídy jsme schopni na závěr dolovací úlohy skrz komponentu *Report* zobrazit výsledky dolování a nabídnout uživateli možnost vydolovaná pravidla aplikovat na nová data. Náhled dialogu je k dispozici na obrázku 6.2.
Dialog je rozdělen na několik záložek:

- **Result – Class description rules** – prostý seznam vydolovaných pravidel. Pole *Rule* zobrazuje tvar pravidla, pole *Class* potom třídu, kterou pravidlo popisuje. V poli *Fitness Value* nalezneme fitness hodnotu daného pravidla (pro připomenutí – může nabývat hodnot od -1 po 1, kdy jedničkou je označeno ideální pravidlo).

- **Result – Fitness progression chart** – graf vývoje fitness v průběhu iterací algoritmu. Graf zobrazuje jak vývoj sumární fitness, tak i její hodnoty v závislosti na jednotlivých třídách.
  Grafická komponenta pro zobrazení grafu je objektem třídy *FitnessChartPanel*, která dokáže z nasbíraných dat vytvořit námi požadovaný graf.

- **Apply** – rozhraní pro aplikaci vydolovaných pravidel na vložený CSV soubor. Po aplikaci pravidel je možné upravený soubor znovu vyexportovat do CSV, formát se zachová, pouze na místo třídy program doplní dopočítané hodnoty.

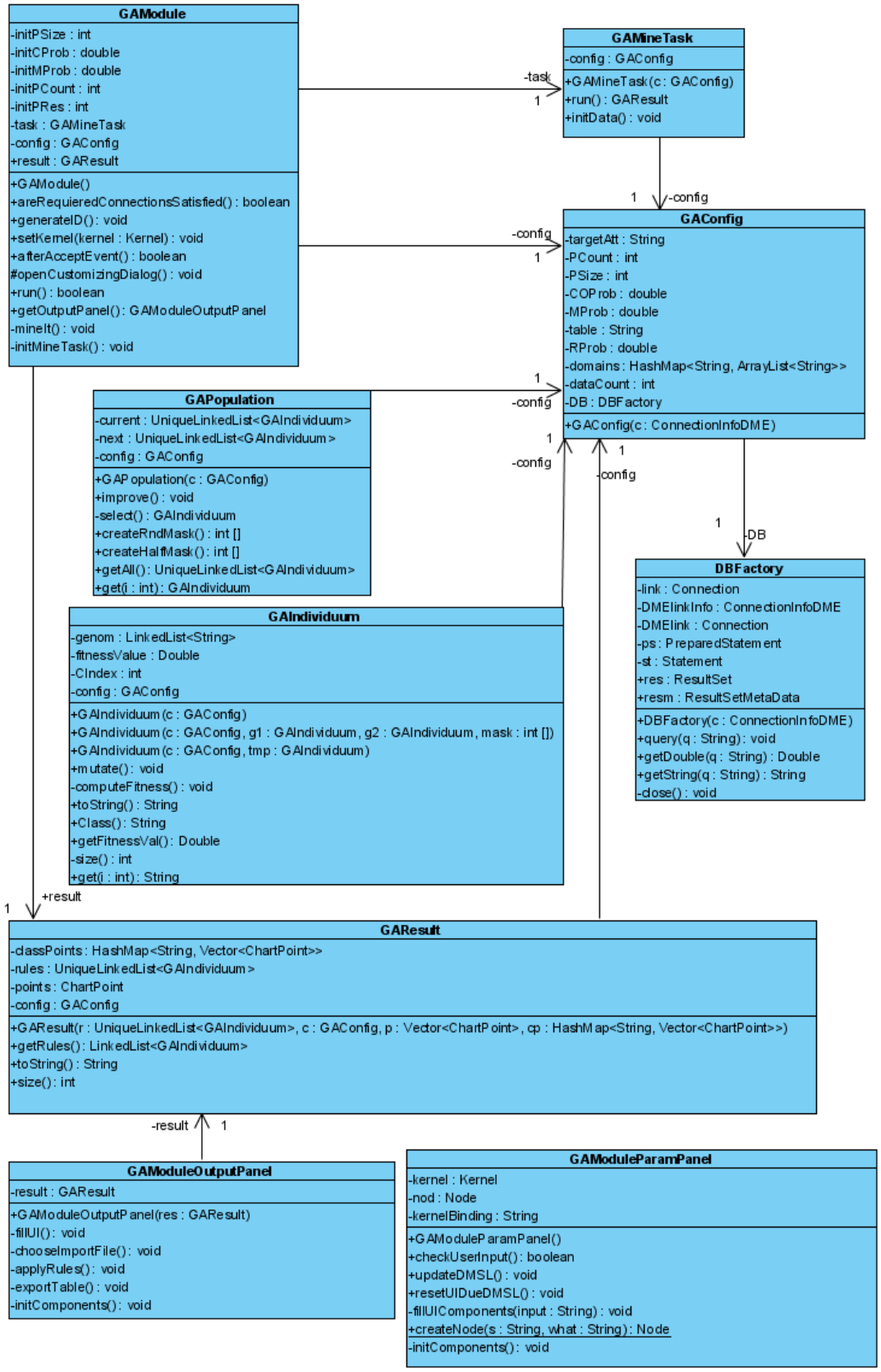
Pomocí metod **chooseImportFile()** a **fillUI()** načteme vybraný soubor, rozparsujeme a naplníme tabulku příslušnými hodnotami. Stisknutí tlačítka Apply následně spustí metoda **applyRules()**, která porovná jednotlivé řádky souboru s vydolovanými pravidly a při nálezu shody do řádku doplní chybějící hodnotu příslušné třídy. Takto upravenou tabulku je poté možno pomocí funkce **exportTable()** vyexportovat do souboru.



Obrázek 6.2: Dialog pro zobrazení výsledků dolovací úlohy

6.2.9 Ostatní pomocné třídy

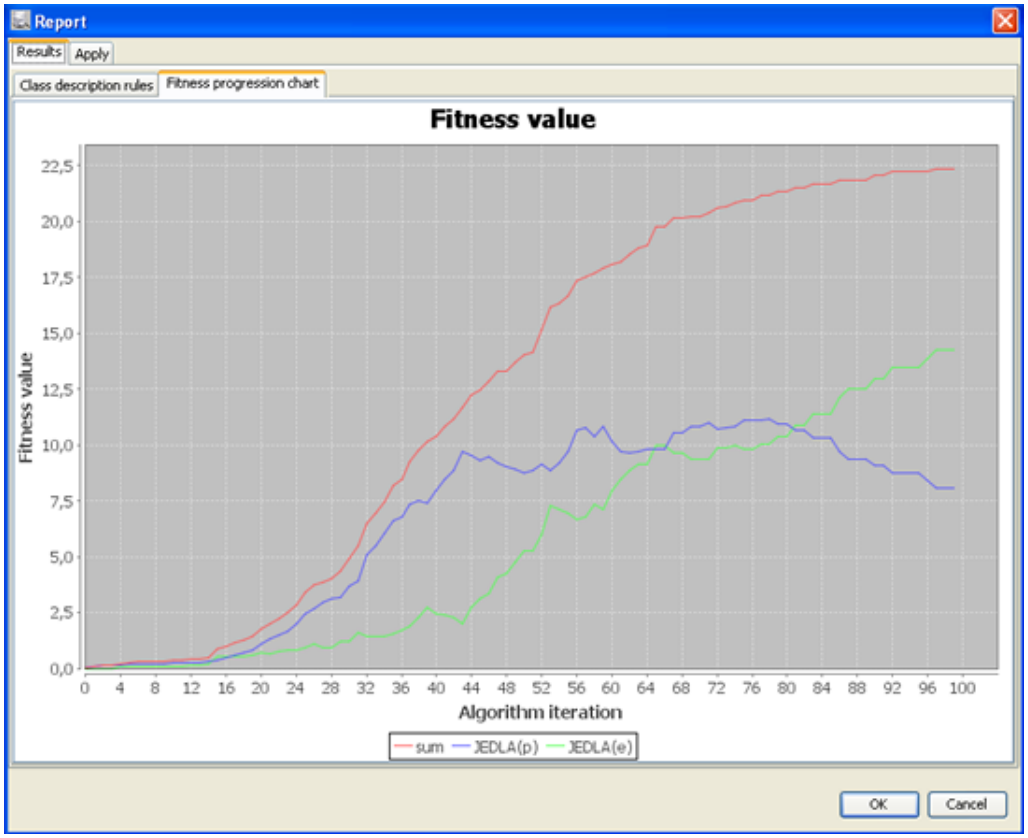
Při implementaci bylo použito i několik výše nepopsaných pomocných tříd. Jde především o stručně zmíněnou třídu *FitnessChartPanel*, která pracuje s objekty typu *ChartPoint* a slouží k implementaci grafu progressu fitness pro výsledkový dialog. Další nezmíněnou třídou je *DBFactory*, která zastřešuje jako součást objektu *GAConfig* spolupráci s databází. Pomocí třídy *GAModuleParamPanelActionListener* potom program čeká na potvrzení dialogu se vstupními parametry. Na závěr třída *UniqueLinkedList* je použita pro implementaci populace jedinců, jejíž podmínkou je unikátnost.



Obrázek 6.3: Stěžejní třídy implementace (zjednodušené)

6.3 Použité externí knihovny

Podobně jako v [7] byla i v tomto projektu využita knihovna *JFreeChart*, která je šířena pod LGPL licencí (tedy zdarma). Díky této knihovně jsme schopni vytvářet jednoduchým způsobem efektní 2D grafy. Knihovna byla v implementaci využita k vytváření grafů průběhu fitness (viz obrázek 6.4).



Obrázek 6.4: Dialog pro zobrazení výsledků dolovací úlohy – průběh fitness

6.4 Problémy spojené s implementací

Při implementaci nebylo nutné řešit přílišné množství problémů, po loňské úpravě jádra Ing. Šebkem ([12]) nabízí opravdu kvalitní rozhraní a zabudovanou podporu pro velké množství operací, jako je např. parsování CSV souborů. Vzhledem ke zkvalitnění a upravení dosavadních komponent jádra je možné vytvořit velmi efektivní rozbor vybraných vstupních dat , což velmi zjednodušuje a zpříjemňuje samotné testování práce modulů. Přesto můžeme v modulu pro genetické dolování narazit na problém spojený s velikosti heapu aplikace. Samotné počítání fitness funkce každého jedince je spojeno se sadou agregačních dotazů nad databází Oracle, jež zařizuje JDBC (Java rozhraní pro připojení k databázi), který pro velký počet dotazů (v řádu desetitisíců) vyžaduje velké množství paměti (v řádu GB). Jediné momentálně známé řešení tohoto problému je v rozšíření heapu aplikace. Při testování menších populací s jedinci s krátkými genomy na tento problém nenarazíme.

Kapitola 7

Návrh elementů DMSL dokumentu

V následující kapitole se seznámíme s formálním návrhem syntaxe konfiguračních elementů v DMSL dokumentu pro vytvořený modul. Pro každý modul je tento popis nutné vytvořit, jelikož není naimplementován v jádře z důvodu velkých odlišností mezi moduly – každý pracuje na jiném principu a potřebuje ukládat individuální nastavení, případně vyzískané znalosti (na konci procesu dolování). Současná implementace modulu nepodporuje ukládání výsledků do DMSL, omezíme se tedy pouze na definici vstupní konfigurace.

7.1 Element *DataminingTask*

Jako součást kapitoly 6.2.2 je uveden příklad popisované konfigurace, která slouží k uložení vstupních hodnot algoritmu. Struktura elementu *DataminingTask* není pevně daná a vyžaduje pouze dva povinné atributy – *name* k označení typu modulu a *language*, který určuje, v jakém jazyce je element popsán. Takto vypadá současná struktura elementu *DataminingTask*:

```
<!ELEMENT DataMiningTask ANY>
<!ATTLIST DataMiningTask name CDATA #REQUIRED
                         language CDATA #REQUIRED
                         type CDATA #IMPLIED
                         languageVersion CDATA #IMPLIED>
```

Do tohoto elementu je vložen element *GAClassification*, obsahující konfiguraci GA:

```
<!ELEMENT GAClassification (DataInfo, Params)>

<!ELEMENT DataInfo (targetAtt)>
<!ELEMENT targetAtt (#CDATA;)>

<!ELEMENT Params (PSize, CProb, MProb, PCount, PRes)>

<!ELEMENT PSize (%INT-NUMBER;)>
<!ELEMENT CProb (%REAL-NUMBER;)>
<!ELEMENT MProb (%REAL-NUMBER;)>
<!ELEMENT PCount (%INT-NUMBER;)>
<!ELEMENT PRes (%REAL-NUMBER;)>
```

Kapitola 8

Možné použití modulu

Cílem této kapitoly je představit stručný postup, jak lze v programu *Dataminer* vytvořit a provést dolovací proces s použitím genetického algoritmu.
Jako zdroj dat nám bude sloužit tabulka *LEPIOTA*, ve které jsou uložena data o houbách – fyzické parametry, vůně atd. – na základě kterých se budeme snažit rozdělit houby do tříd podle jejich poživatelnosti.

8.1 Vytvoření nového projektu

Po spuštění aplikace vytvoříme nový projekt (v menu File → New Project, klávesová zkratka Ctrl+Shift+N). Zvolíme možnost *Data Mining Project*, vyplníme Project Name a umístění projektu. V následujícím dialogu *Oracle Connection Settings* zvolíme možnost *New Oracle database connection* a zadáme buď svoje přihlašovací informace do systému Oracle, nebo následující:

- **Login:** xstriz03
- **Password:** jouda2
- **Connection url:** jdbc:oracle:thin:@minerva2.fit.vutbr.cz:1521:ORACLE

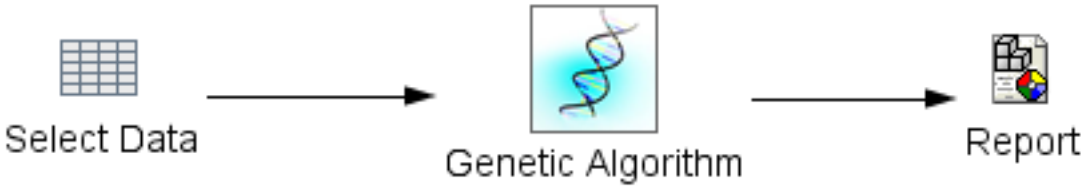
Vytvoření projektu potvrdíme stiskem tlačítka Finish.

8.2 Vytvoření dolovacího procesu

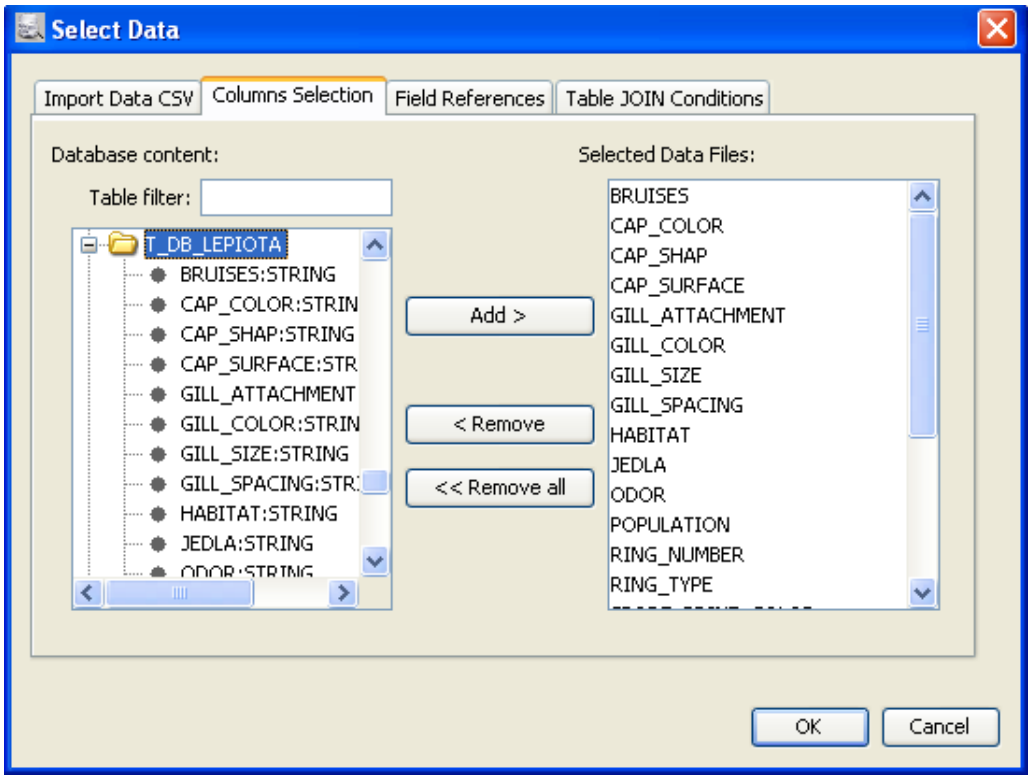
Dolovací proces je možné tvořit z komponent popsaných v kapitole 4.3.1. V našem případě zvolíme nejjednodušší možnou variantu s co nejmenším počtem zainteresovaných komponent. Přetažením z palety napravo umístíme na pracovní plochu projektu komponenty *Select Data*, *Genetic Algorithm* a *Report*. Následně je nutné tyto komponenty funkčně propojit, toho dosáhneme pomocí vytažení šipky z libovolné komponenty myší za stisknutého tlačítka Shift a jejího připojení na komponentu jinou. Jak by měl vytvořený graf vypadat vidíme na obrázku 8.1.

8.3 Nastavení komponenty *Select Data*

Poklepáním na ikonu komponenty *Select Data* vyvoláme její konfigurační dialog (obrázek 8.2). Vybereme záložku *Column Selection*, která čerpá informace z připojené databáze.



Obrázek 8.1: Graf dolovacího procesu



Obrázek 8.2: Dialog komponenty *Select Data*

V levém sloupci vidíme obsah databáze – vybereme příslušnou tabulku (v tomto případě *LEPIOTA*) a pomocí tlačítka Add vybereme atributy, které chceme při dolování zohlednit. Počet atributů je pro dolování s pomocí GA kritický – pro kratší genom bude trvat dolování kratší dobu a bude pravděpodobně úspěšnější. Stiskem tlačítka OK potvrdíme nastavení komponenty a dialogové okno se zavře.

8.4 Nastavení dolovacího modulu

Dialog pro nastavení dolovacího modulu je znázorněn na obrázku 6.1 v jedné z předcházejících kapitol. Vyvoláme jej opět poklepáním na příslušnou komponentu.
V prvé řadě je nutné vybrat atribut, který označíme za třídní (podle jeho hodnoty rozdělíme data na jednotlivé třídy) – v případě hub je tímto atributem *JEDLA*. Následuje nastavení

velikosti inicializační populace, pravděpodobnosti křížení, pravděpodobnosti mutace, počtu iterací algoritmu a na závěr v poli *Result* nastavíme, kolik procent vydolovaných pravidel označíme za výsledek. Tyto hodnoty mají zásadní vliv na průběh algoritmu, jejich role byla popsána v kapitole 3 a bude nadále diskutována v závěru.
Po potvrzení nastavení dolovacího modulu je možné spustit proces dolování. Pravým tlačítkem klikneme na modul a zvolíme možnost Run. Při úspěšném spuštění procesu se objeví dialog s popisem *Mining...*, který zmizí, jakmile dolování skončí.

8.5 Zobrazení výsledků dolování, jejich aplikace

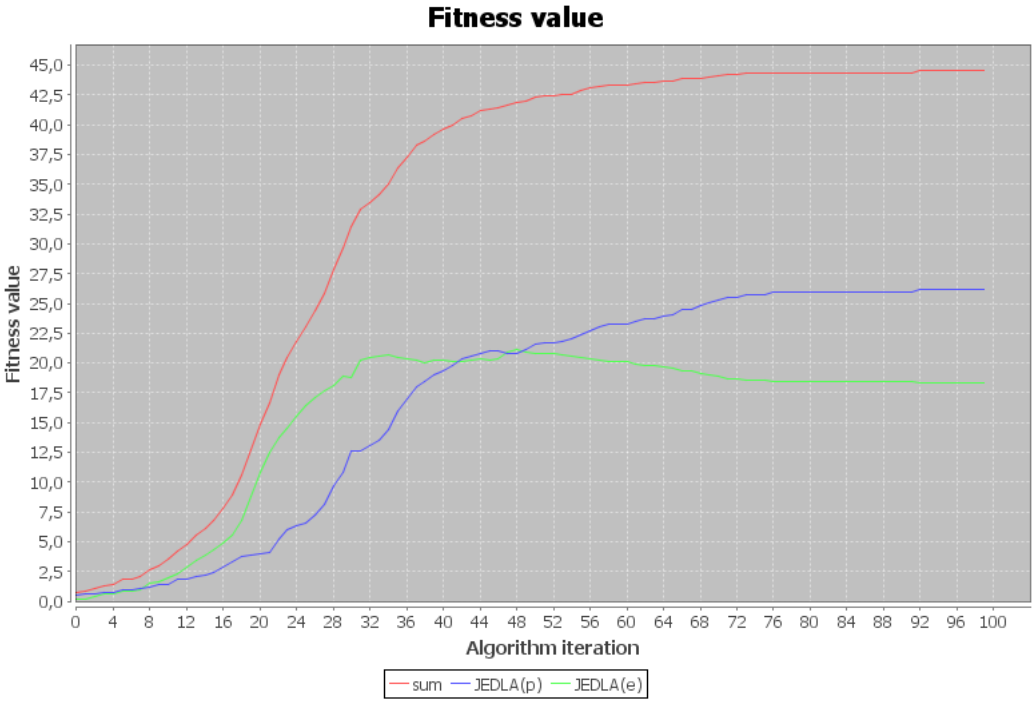
Komponenta pro zobrazení výsledků byla kompletně popsána v kapitole 6.2.8. Umožňuje zobrazit výsledná pravidla, průběh vývoje fitness hodnot v populaci v průběhu iterací a aplikovat vydolovaná pravidla na CSV soubor, které je následně možno vyexportovat s doplněnými hodnotami předpovězených tříd.

Kapitola 9

Výsledky testování

9.1 Testování

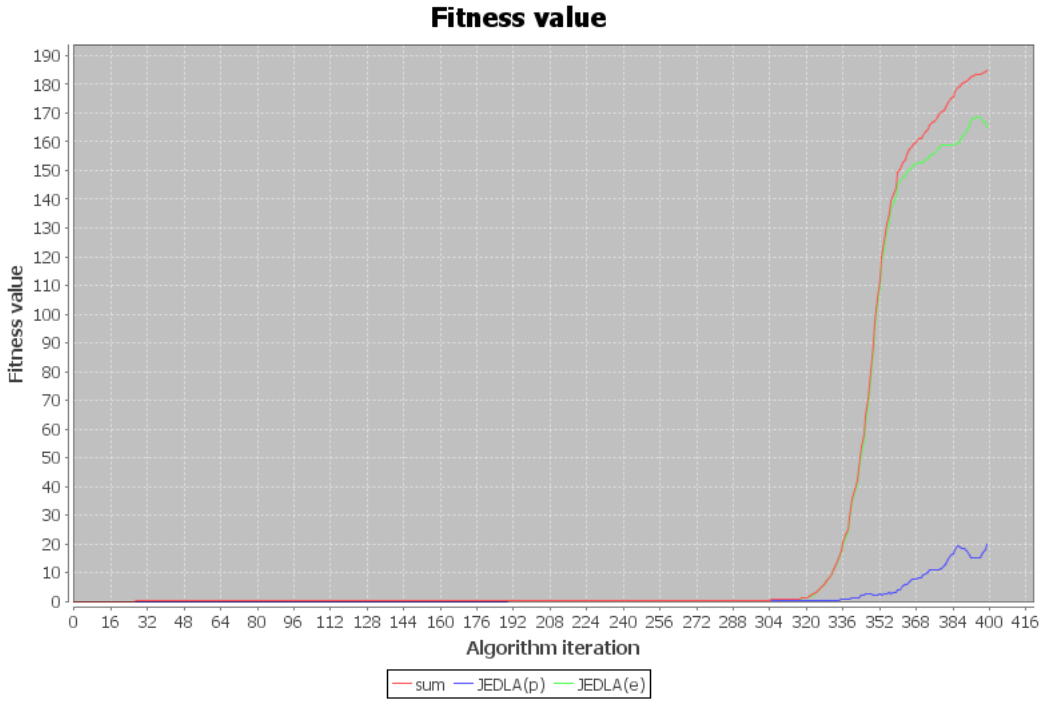
V závěrečné fázi vývoje modulu bylo zároveň prováděno intenzivní testování na vzorové databázi. Uvedeme zde dva odlišné příklady, které vykazují některé zajímavé charakteristiky algoritmu.
Na obrázku 9.1 je znázorněn vývoj fitness funkce pro populaci s krátkým genomem – z databáze bylo vybráno pouze 8 atributů, nad kterými byl algoritmus prováděn. Z grafu vidíme,



Obrázek 9.1: Vývoj fitness pro krátký genom (export z dolovacího systému)

jak sumární fitness (červená linka) konverguje, což je dobré znamení – síla populace stoupá až po určitou hranici, kdy jsou sumární změny fitness hodnot již minimální. V této hodnotě pravděpodobně došlo k vyčerpání potenciálu algoritmu a bylo nalezeno nejlepší možné řešení (pokud se tato hranice dlouho nemění). Sumární fitness tedy roste, fitness hodnoty

jednotlivých tříd navzájem kolísají – jev, který se očekával, ne všechny třídy budou ve výsledku zastoupeny stejnou silou, resp. stejným počtem jedinců.
Co se týče vydolovaných pravidel, nejvyšší fitness hodnoty se pohybovaly kolem $0,8$, což můžeme označit za hodnotu vysokou, čili pravidlo je vysoce kvalitní. Při aplikaci pravidel na testovací množinu dat jsme dosáhli úspěšnosti 95 % na cca 4000 vzorků.
Co se týče doby zpracování experimentu, vzhledem k poměrně nízkému počtu iterací nebyl časově příliš náročný. Můžeme tedy konstatovat, že experiment provedený na menší populaci s kratším genomem dopadl úspěšně.
Na obrázku 9.2 vidíme vývoj fitness pro velkou populaci s relativně delším genomem (více než 20 atributů). Opět můžeme vidět konvergenci sumární fitness, ovšem v tomto případě až



Obrázek 9.2: Vývoj fitness pro krátký genom (export z dolovacího systému)

po velkém počtu iterací. Dále můžeme konstatovat, že algoritmus pravděpodobně nedosáhl svého maximálního potenciálu na určeném množství iterací, i když se k němu jistým krokem blížil. Dalším, na první pohled zřetelným, poznatkem je výrazná přesila jedinců jedné třídy nad druhou (zelená vs. modrá linka), nedá se však předpokládat, že by se s dalšími iteracemi tyto rozdíly srovnaly, vzhledem k tomu, že se jedná o algoritmus pracující náhodně. Je dokonce možné, že by časem jedna třída úplně vymizela, pakliže je její zastoupení malé nebo pravidla k jejímu popisu složitá a slabá.
Maximální fitness hodnoty vydolovaných pravidel se opět pohybovaly v rozmezí kolem $0,8$, tedy opět velmi dobré. I úspěšnost byla odpovídající, opět se blížící hodnotě 95 %.
Doba zpracování se v tomto případě dostala do řádu hodin, což již není příliš příjemné – přesto můžeme experiment s drobnými připomínkami označit za úspěšný.
Byly provedeny i další experimenty s jejichž obecnými závěry se seznámíme v následující kapitole.

9.2 Shrnutí fáze testování

Po provedených analýzách a testech můžeme s jistotou říci, že genetický algoritmus je využitelný v oblasti dolování z dat. Zavedené optimalizace z kapitoly 3.3.4 se ukázaly jako velmi vhodné (testování bez nich vykazovalo bídné výsledky) a bylo by jistě možné zavést i některé další, zaměřené především na zrychlení celého procesu (např. bitová reprezentace, paralelní zpracování, ...).
Obecně můžeme říct, že pro problémy s velkým vyhledávacím prostorem (velmi dlouhé genomy, velké populace) bude algoritmu trvat delší dobu, než se dopracuje ke kvalitnímu výsledku – u příliš velkých prostorů je možné, že k výsledku ani nedospěje. Dalším problémem, pro klasifikaci kritickým, může být vymizení pravidel určité třídy, resp. tříd z výsledné populace – toto hrozí pro data s velkým počtem různých tříd a pro malé inicializační populace. Tomuto jevu však nejde bez dalších optimalizací (např. zavedení určitých heuristik) nijak zabránit, riziko snížíme tvorbou dostatečně velkých inicializačních populací.

Kapitola 10

Závěr

Po důkladné teoretické přípravě byl úspěšně implementován modul v jazyce Java na platformě NetBeans, který rozšiřuje stávající aplikaci *Dataminer*, vyvíjenou na FIT pro studijní účely.
Teoretická příprava se skládala z několika částí – nejdříve bylo nutné seznámit se s procesem získávání znalostí z databází, následně proniknout do tajů genetických algoritmů a konečně na závěr nastudovat technologie použité při implementaci dolovacího systému. Diplomové práce z minulých let posloužily jako kvalitní zdroj informací o konceptu celého systému. Stěžejní pak bylo samotné uzpůsobení genetického algoritmu pro dolování a analýza jednotlivých modifikací – tolik k teoretické přípravě.
Následovala fáze realizace. Byl vytvořen modul, který úspěšně implementuje netradiční dolovací techniku v podobě genetického algoritmu a i přes svá omezení je cenným příspěvkem do současného repertoáru dolovacích modulů aplikace. Algoritmus se podařilo zoptimalizovat a implementovat tak, aby sloužil smysluplně pro účely dolování, konkrétně klasifikace. Výzvou bylo především stanovení správné fitness funkce a výběr kvalitních optimalizací, které výrazně ovlivnily kvalitu obdržených výsledků.
Ve fázi testování jsme dospěli k závěru, že genetického algoritmu lze úspěšně využít pro dolování z dat a přes některé problémy, jakými jsou zejména časová a paměťová náročnost, je současná implementace prakticky použitelná. Úspěšnost klasifikace se při správném nastavení vstupních parametrů pohybuje nad hranicí 90 % a v tomto směru implementace předčila počáteční očekávání. Vytvořený modul je také první, který implementuje dolovací algoritmus přímo bez použití ODM.
Co se týče dalšího možného vývoje, jak již bylo uvedeno v sekci 9.2, modul by bylo možné rozšířit o další optimalizace – ideální by bylo vytvořit nabídku těchto optimalizací, které by spravoval uživatel (bylo by možné je zapínat a vypínat pro jednotlivé dolovací úlohy), čímž by vznikl komplexní nástroj pro detailní testování vlivu těchto optimalizací na proces dolování. Dále by bylo možné implementovat ukládání vydolovaných pravidel v rozumném formátu do struktury DMSL, aby byl proces uložitelný společně s vydolovanými výsledky.
Ohledně dalších prací na fakultním dolovacím systému v tomto roce, v rámci bakalářských prací je vytvářen další modul, který by měl implementovat klasifikaci pomocí neuronových sítí. V rámci diplomových prací bude přírůstek modulů větší, rozšíření se dočká i samotné jádro.

Literatura

[1] NetBeans: Co je NetBeans? [online], 2008, [cit. 2010-01-24].
URL http://netbeans.org/index_cs.html

[2] NetBeans Visual Library in NetBeans Platform. [online], 2008, [cit. 2010-01-24].
URL http://platform.netbeans.org/graph/

[3] Ghosh, A.; Tsutsui, S.: *Advances in Evolutionary Computing: Theory and Applications*. Springer-Verlag New York, Inc., 2003, ISBN 3-540-43330-9, 1004 s.

[4] Han, J.; Kamber, M.: *Data Mining: Concepts and Techniques*. Elsevier Inc., 2006, ISBN 1-55860-901-6, 770 s.

[5] Hynek, J.: *Genetické algoritmy a genetické programování*. GRADA, 2008, ISBN 978-80-247-2695-3, 200 s.

[6] Jun-shan, T.; Wei, H.; Yan, Q.: Application of Genetic Algorithm in Data Mining. In *2009 First International Workshop on Education Technology and Computer Science*, Ieee, 2009, ISBN 1424435811.

[7] Kmošťák, O.: *Vytvoření nových klasifikačních modulů v systému pro dolování z dat na platformě NetBeans*. Diplomová práce, FIT VUT v Brně, 2009.

[8] Kotásek, P.: *DMSL: The Data Mining Specification Language*. Dizertační práce, FIT VUT v Brně, 2003.

[9] Krásný, M.: *Systém pro dolování z dat v prostředí Oracle*. Diplomová práce, FIT VUT v Brně, 2008.

[10] Stríž, R.: Implementation of Genetic Algorithm for Data Mining System. In *Proceedings of the 16th Conference Student EEICT 2010: Volume 1*, Vysoké učení technické v Brně, 2010, ISBN 978-80-214-4076-0.

[11] Zendulka, J.; Bartík, V.; Lukáš, R.; aj.: *Získávání znalostí z databází*. Studijní opora, FIT VUT v Brně, 2009.

[12] Šebek, M.: *Rozšíření funkcionality systému pro dolování z dat na platformě NetBeans*. Diplomová práce, FIT VUT v Brně, 2009.

Seznam příloh

Dodatek A – Obsah přiloženého CD

Příloha A

Obsah CD

Adresářová struktura přiloženého datového média:

- `apps` – aplikace doporučené pro další vývoj/běh projektu
- `dataminer` – hlavní složka programové realizace
  - `dist` – distribuční verze systému pro dolování
  - `examp-data` – ukázková data do databáze (CSV formát)
  - `sources` – **zdrojové kódy systému pro dolování**
- `docs` – dokumentace k projektu
  - `javadoc` – programová dokumentace Javadoc
- `thesis` – text diplomové práce
  - `src` – zdrojový kód práce pro systém LaTeX