Příprava umělých dat pro výuku a testování pomocí evolučního algoritmu

Milan Šimůnek
Fakulta informatiky a statistiky, VŠE Praha
nám. W. Churchilla 4, 130 67 Praha 3 Česká republika
simunek@vse.cz

Abstrakt: *Pro výuku data miningu a také pro testování nově vyvinutých SW nástrojů je třeba velké množství datových sad, které jsou zároveň pro analýzu tou či jinou technikou vhodné. Kvůli celé řadě problémů s reálnými daty mohou být řešením realisticky vypadající data umělá. Jejich přípravu chápeme jako reverzní proces ke klasickému data miningu, při kterém máme data a chceme nalézt skryté vztahy. Místo toho nejprve definujeme požadované vztahy, jejich typ i sílu, a potom se snažíme vygenerovat data, kde jsou přesně takové vztahy „skryty". Při generování dat využíváme evoluční algoritmus a systém LISp-Miner. Evoluce je řízena množstvím a sílou požadovaných vztahů už přítomných v datech. Vygenerovaná data mohou být následně analyzována běžnými postupy a v jakémkoliv nástroji pro data mining.*

Klíčová slova: uměle generovaná data, výuka, testování, data mining, evoluční algoritmus, LISp-Miner

Abstract: *Various kinds of data with well-known characteristics are necessary both for providing students with suitable material for learning of methods and techniques of data mining and for development of new SW tools. Due to several problems with real data, synthetic data offers a solution. We treat this as a reverse process to the classical data mining analysis where a given data is mined for hidden patterns. Instead, we define desired patterns first and we are trying then to prepare data where such patterns are "hidden". Our approach utilizes an evolutionary algorithm and the LISp-Miner system. Measure of compliance with the desired patterns in (initially randomly) generated data is used to guide the evolution. Generated data could be analyzed afterwards in a common way.*

Key words: synthetic data, teaching, testing, data mining, evolutionary algorithm, LISp-Miner

1. Úvod

Při přípravě výukových dat pro studenty i vhodných dat pro testování nově vyvíjených nástrojů se setkáváme s několika praktickými problémy. Data musí vypadat přirozeně, přitom musí být dostatečně jednoduchá na pochopení během výukové hodiny. Zároveň musí být pečlivě připravená (nebo dostatečně dobře dopředu zanalyzovaná), aby obsahovala pouze učiteli (nebo testovacími týmu) známé vztahy. Tyto vztahy musí odpovídat výukovému záměru, vyučované metodě nebo testované vlastnosti vyvíjeného nástroje. V datech použité atributy a vyskytující se hodnoty i jejich rozsahy musí být dostatečně podrobně popsány, aby umožňovaly jak formulaci zajímavých analytických otázek a zadání úloh, tak i srozumitelnou interpretaci nalezených výsledků.

Reálná data navíc vyžadují pečlivou kontrolu na špatné nebo chybějící hodnoty, které navíc není snadné nahradit za platné. Zejména je však obtížné reálná data vůbec

získat, protože jejich vlastnící mají obavy z potenciálního úniku citlivých informací. Reálná data jsou tak obvykle získána pouze v silně anonymizované podobně, která výrazně ztěžuje porozumění a hlavně interpretaci výsledků. Podoba reálných dat také nemusí přesně odpovídat požadavkům výuky nebo testování. Naproti tomu ručně připravená data trpí často jasně viditelnými opakujícími se sekvencemi a také malým rozsahem, který není dostatečný pro všechny metody *data miningu*.

Zde prezentovaný přístup se snaží řešit výše zmíněné problémy tím, že automaticky generuje realisticky vypadající umělá data a přitom dovoluje přesně nastavit jejich strukturu i rozsah, hodnoty i jejich distribuce a v neposlední řadě přesně nastavit vztahy, které jsou v datech „skryty“. Vygenerovaná data pak mohou být analyzovaná nebo použita pro testování v libovolném nástroji pro *data mining* (nejen v LISp-Mineru, ale také například v systémech RapidMiner, Weka, Clementine nebo R). Jedna data s předem přesně známými vlastnostmi tak mohou být použita pro porovnání kvality i množství nabízených funkcí různých nástrojů a demonstraci jejich předností a nevýhod. Velkým pozitivem je snadná kontrola provedených studentských analýz a jejich výsledků. V neposlední řadě mohou být umělá data použita pro testování automatizace procesu *data miningu* v projektu *EverMiner* [Šimůnek&Rauch, 2011].

Smyslem tohoto textu není prezentovat vylepšenou verzi evolučního algoritmu nebo jeho speciální implementaci. Naopak, implementace evolučního algoritmu je záměrně co nejjednodušší s tím, že všechny jeho parametry je možné nastavovat uživatelsky. Hlavním snahou bylo zakomponování všech typů syntakticky bohatých vztahů, které se v datech mohou skrývat. Tyto vztahy a jejich míry zajímavosti jsou použity při konstrukci *fitness* funkce.

Článek je strukturován takto: v následující kapitole je vysvětlen použitý evoluční přístup, včetně navržených evolučních operací. Ve třetí kapitole je stručně představen systém LISp-Miner a metoda GUHA. Čtvrtá kapitola popisuje nový modul *ReverseMiner*, který výše uvedený přístup implementuje. Popis dosažených výsledků a zkušeností s použitím modulu je v následující, páté kapitole. Diskuze dříve publikovaných přístupů ke generování umělých dat je v šesté kapitole. Text zakončuje popis směrů dalšího vývoje a závěr.

2. Navržený přístup

Navržený přístup charakterizují dva hlavní znaky – evoluční algoritmy a syntakticky bohaté vztahy různých typů, které jsou jednotlivé moduly systému LISp-Miner (viz další sekce) schopné v datech hledat.

První částí zadání úlohy generování umělých dat je popis cílové databázové tabulky – jména sloupců, jejich datové typy, jaké hodnoty mohou obsahovat či přímo rozdělení hodnot a konečně, kolik má mít tabulka řádků. Jeden jedinec představuje jednu možnou variantu této databázové tabulky. Na začátku evolučního procesu je populace jedinců vytvořena náhodně. Při generování konkrétních hodnot v daném sloupci jsou dodržovány požadavky na minimální a maximální hodnotu, případně na povolené hodnoty dané výčtem. Zároveň histogram četností vygenerovaných hodnot odpovídá zadané volbě rozdělení – uniformní, Gaussovo nebo podle uživatelem zadaných frekvencí u výčtu hodnot.

Předpokládejme, že chceme vygenerovat jednoduchá data o klientech banky a jejich půjčkách. V této ukázce budeme mít pouze jednu tabulky o třech sloupcích – (1) *Loan*

Status (stav půjčky) s možnými hodnotami A (splaceno), B (odepsáno), C (spláceno) a D (opožděné splátky); (2) *District* (okres), ze kterého klient pochází; a (3) *Salary* (výše platu klienta). Tabulka má mít 1 000 řádků. Pro počáteční rozdělení hodnot stavu půjčky nastavíme poměry A: 30 %, B: 4 %, C: 58 % a D: 8 % – k nastavení co nejvěrohodnějších hodnot můžeme použít dostupné zprávy o splácení půjček, například z Českého statistického úřadu (ČSÚ) nebo Bankovní asociace. Všichni na počátku evoluce náhodně vygenerovaní jedinci budou mít rozdělení četnosti v prvním sloupci v daném poměru. K nastavení výčtu možných hodnot do sloupce *District* použijeme seznam okresů, stažený například opět ze stránek ČSÚ, doplněný o počty osob žijícím v každém okrese (ze sčítání obyvatelstva). Počty osob jsou následně automaticky přepočítány, aby v součtu daly 1 000 (cílový počet řádků naší tabulky). Nakonec ještě nastavíme omezení pro sloupec *Salary*, aby byl v intervalu od minimální mzdy až do 200 000 Kč s tím, že bude mít sice Gaussovo rozdělení, ale se střední hodnotou 19 000 Kč, tedy nižší než je průměrný plat. Posunutí střední hodnoty lépe vyjádří skutečnost, že průměrný plat je vyšší než medián (kvůli méně četným výskytům lidí s abnormálně vysokými platy, kteří průměr vychýlí). Počáteční hodnoty platů budou potom dodržovat dolní a horní limit a budou mít zadané rozdělení.

Druhou důležitou částí zadání pro generování umělých dat je množina *data miningových* úloh hledajících v uměle generovaných datech různé typy různě silných vztahů. Tyto požadavky na data se vyhodnotí pro každého jedince (jednu variantu databázové tabulky) v populaci a *fitness* jedince se vypočte jakou součet vážených odchylek mezi požadovaným počtem vztahů a počtem vztahů skutečně nalezených pro každou z *data miningových* úloh.

V naší ukázce bychom mohli požadovat, že počet nesplacených půjček v některém okrese postiženém vyšší nezaměstnaností (například v Šumperku) je dvakrát vyšší, než je průměr. Abychom tento fakt dále podpořili, budeme chtít, aby platy zde byly spíše nižší – např. více než 50 % klientů bude mít plat nižší než 10 000 Kč. Na druhou stranu musíme dbát i na to, aby v datech nebyly i nechtěné závislosti – proto budeme požadovat, že nadprůměrné množství nesplacených půjček se vyskytuje pouze v Šumperku a už v žádném dalším okrese. Realistický dojem dat můžeme umocnit opět s pomocí zpráv o trhu práce z ČSÚ a nižší plat požadovat i v některém dalším okrese – např. v Lounech a v České Lípě. Díky výše zadanému požadavku se však nebude v těchto okresech nižší plat odrážet na vyšším počtu nesplacených půjček.

Řízení evoluce pouze podle binárního kritéria – vztah v datech je $\times$ vztah v datech není – by bylo příliš hrubé. Proto byl implementován *fuzzy* přístup podle hodnoty míry zajímavosti zadaného vztahu (např. *konfidence*). Je-li tedy v datech požadován například vztah A $\Rightarrow$ B s *konfidencí* 90 %, tak jedinec, který tento vztah splňuje s hodnotou konfidence 60 %, je považován za lepšího než jedinec, který vztah splňuje pouze na 50 % (a lepšímu jedinci je v evoluci dána přednost).

Na počátku evoluce jsou jedinci vygenerování náhodně a pro každého je vypočtena *fitness* tak, že se na jeho variantě dat vyhodnotí všechny zadané *data miningové* úlohy. Více perspektivní jedinci (jejichž data lépe odpovídají uživatelským požadavkům) mají vyšší šanci předat své vlastnosti do další generace pomocí evolučních operací křížení, mutace a reprodukce – viz dále. Po provedení dostatečného množství evolučních kroků bude nejlepší jedinec odpovídat všem požadavkům a data budou připravena.

Podstatnou vlastností evolučního algoritmu je jeho *sub-optimální* povaha. Proto není zaručeno, že evoluce neuvízne v lokálním optimu a že výsledná data budou obsahovat úplně všechny požadované vztahy. Dopředu známý není ani celkový čas nutný pro získání požadovaných dat. Z těchto důvodů je evoluční proces implementován jako interaktivní, aby uživatel mohl sledovat postupné vylepšování podoby dat, včetně vztahů, které už do nich byly úspěšně zakomponovány. Dle potřeby může evoluci i předčasně ukončit a případně změnit zadání. Podrobněji viz kapitola *4 ReverseMiner* dále v tomto textu.

2.1 Evoluční operace

Evoluční algoritmus byl záměrně implementován v co nejjednodušší podobě. Jeho implementace vychází z práce [Weise, 2011] s tím, že všechny parametry je možné nastavovat uživatelsky s ohledem na velikost generovaných dat nebo typ a množství požadovaných vztahů.

V každém evolučním kroku se vytváří nová populace jedinců. Noví jedinci vznikají jedním ze tří druhů evolučních operací – *křížení*, *mutace* a *reprodukce*. Při *křížení* a *mutaci* vzniká nový jedinec (nová varianta dat). V případě *reprodukce* se jedinec zkopíruje do nové populace bez jakékoliv změny. Noví jedinci jsou přidávání do nové populace, dokud její velikost nedosáhne uživatelem zadané hodnoty. Druh operace je pro každého jedince zvolen náhodně tak, že procentní poměr *křížení*, *mutací* a *reprodukcí* opět odpovídá uživatelem zadaným parametrům. Kromě toho je možné zadat i velikost elity – počtu nejlepších jedinců, kteří jsou vždy zkopírováni do nové generace.

2.1.1 Výběr rodičů

Jedinci mající právo na rozmnožování se vybírají systémem turnaje, a to pouze na základě setřídění jedinců podle *fitness* (nikoliv podle velikosti rozdílu jejich *fitness*). Je-li třeba získat rodiče, provede se náhodný výběr tolika jedinců z populace, kolik je zadáno v parametru *Velikost turnaje*. Z tohoto počtu náhodně vybraných jedinců se rodičem stane ten s nejlepší (nejnižší) hodnotou *fitness*. Čím je hodnota parametru *Velikost turnaje* vyšší, tím se **snižuje** pravděpodobnost, že šanci na rozmnožení dostane horší jedinec (ve velké skupině je větší pravděpodobnost, že tam bude i jedinec s lepší *fitness* a ten vyhraje turnaj). Vyšší hodnoty *Velikost turnaje* tak vedou ke snižování diverzity populace.

2.1.2 Křížení

Pro křížení jsou zapotřebí dva rodiče (každý vybrán systémem turnaje – přitom je kontrolováno, že nebyl v obou případech vybrán stejný jedinec). Následně dojde k vlastnímu křížení, kterým vzniknou **dva** noví jedinci. Křížení je možné ve dvou směrech:

a) Křížení ve smyslu sloupců – první nově vznikající jedinec převezme celý sloupec *C* od prvního rodiče a druhý vznikající jedinec tento sloupec převezme od rodiče druhého. Případně opačně – je voleno náhodně pro každý sloupec. Tento typ křížení je možné použít vždy.

b) Křížení ve smyslu řádků/záznamů – první nově vzniklý jedinec bude mít jednu podmnožinu hodnot ve sloupci *C* od prvního rodiče a druhou od druhého rodiče; druhý jedinec naopak. Řada hodnot ve sloupci se rozdělí na dvě

poloviny počínaje *n*-tým záznamem. Parametr *n* je zvolen náhodně v rozmezí *1* až *počet_řádků_datové_matice* a záznamy se považují za „zatočené samy do sebe“ – po posledním následuje zase první. Tento typ křížení je možný pouze v případě, že není zakázána možnost změny distribuce hodnot v daném sloupci.

Oba noví jedinci vznikají postupně sloupec po sloupci. Pro každý sloupec se nejprve zjistí, jestli je vůbec možné vybírat z obou typů křížení. Je-li na výběr, tak je poměr mezi oběma typy dán uživatelem zadanou hodnotou.

2.1.3 Mutace

Mutací rozumíme změnu hodnot v náhodně vybraném sloupci. Jako rodič figuruje pouze jeden jedinec (vybraný turnajem) a vznikne také jeden jedinec, ale s (drobně) pozměněnými hodnotami oproti svému rodiči. Opět jsou dva možné typy:

a) Prohození hodnot – souvislá posloupnost hodnot počínaje záznamem na pozici n je prohozena se stejně dlouhou posloupností hodnot na pozici m ve stejném sloupci. Délka posloupnosti je zvolena jako náhodné číslo od 1 do hodnoty dané zadaným parametrem. Přitom se hodnoty ve sloupci opět považují za „zatočené samy do sebe“ – po poslední hodnotě následuje zase první. Je zaručeno, že se obě posloupnosti nepřekrývají. Při vytváření jedince je možné provést prohození dvou úseků hodnot vícekrát, opět podle hodnoty parametru. Tento typ mutace je možné použít vždy.

b) Změna hodnot – souvislá posloupnost hodnot počínaje záznamem na pozici n a délce m je nastavena na náhodně zvolenou hodnotu (přitom se dodržuje povolený rozsah/výčet a rozdělení četností zadané pro tento sloupec). Sloupec je opět zatočený sám do sebe. Délka posloupnosti je zvolena jako náhodné číslo od 1 do maximálního hodnoty zadané uživatelem. Při vytváření jedince je možné provést nastavení hodnoty vícekrát. Počet opakování se určí náhodně v rozmezí 1 až hodnota zadaného parametru. Délka posloupnosti zůstává stále stejná. Tento typ mutace je možný pouze v případě, že je povolena změna rozdělení hodnot u daného sloupce.

Je-li na výběr, tak je poměr mezi oběma typy mutací opět dán uživatelem zadanou hodnotou.

2.1.4 Reprodukce

Jedinec vybraný turnajem je bez jakékoliv změny zkopírován do nové populace. Procento takto kopírovaných jedinců je dáno jako doplněk do 100 % po součtu procentní pravděpodobnosti křížení a procentní pravděpodobnosti mutace. Do nové populace je vždy reprodukován aktuálně nejlepší jedinec, případně více nejlepších jedinců (podle zadání parametru velikosti elity).

3. Systém LISp-Miner

Systém LISp-Miner [LISp-Miner] je akademický systém pro dobývání znalostí z databází (DZD) a je vyvíjen na Fakultě informatiky a statistiky VŠE Praha od přelomu let 1995/96 – více viz např. [Šimůnek, 2010]. Od počátku vývoje je systém stavěn na metodě GUHA [Hájek&Havránek, 1978] a na celé řadě dalších teoretických pracích, které se jí týkají. Aktuální shrnutí metody GUHA je v [Hájek a kol., 2010].

Hlavním rysem systému je bohatá syntaxe hledaných vztahů, která mohla být následně využita i pro velmi jemnou definici požadavků na generovaná data. Podrobný popis stavu systému k roku 2010 naleznou zájemci v [Šimůnek, 2010]. Celý systém LISp-Miner je volně ke stažení na adrese http://lispminer.vse.cz.

Systém se v současné době skládá z osmi GUHA-procedur (*4ft-Miner*, *CF-Miner*, *KL-Miner*, *SD4ft-Miner*, *SDCF-Miner*, *SDKL-Miner*, *Ac4ft-Miner* a *ETree-Miner*) a dalších modulů – zejména pro předzpracování dat (*LM DataSource*) a komunikaci se systémy třetích stran pomocí dokumentů ve formátu PMML, HTML či XML (*LM SwbImporter* a *LM SwbExporter*).

Každá z implementovaných GUHA-procedur hledá v analyzovaných datech odlišný typ vztahů (*patterns*), případně porovnává míru platnosti vztahu mezi dvěma podmnožinami analyzovaných dat (procedury rodiny *SD-*). V případě *4ft-Mineru* se tak hledají *asociační pravidla* s rozšířenou syntaxí; v případě *KL-Mineru* pak *K×L polní tabulky četností* popisující vztah dvou kategoriálních atributů; v *Ac4ft-Mineru* dvojice *asociačních pravidel*, které vyjadřují *změnu stavu*, resp. *akci* od dosavadního stavu ke stavu žádoucímu; a konečně v případě *ETree-Mineru* (podmíněné) *rozhodovací stromy* nebo dokonce tzv. *exploračí stromy*, tedy struktury zastupující celý les *rozhodovacích stromů* (více viz [Berka, 2011] a [Šimůnek, 2012]).

Důležitou vlastností systému je velká rychlost verifikace všech možných vztahů v datech díky celé řadě implementovaných optimalizací. Kromě toho je k dispozici i automatické rozdělení *data miningových* úloh na části, které jsou pak řešeny paralelně na vícejádrovém procesoru nebo dokonce na počítačovém gridu, resp. v *cloudu*.

Systém LISp-Miner je v prvé řadě používán pro výuku a pro výzkum v oblasti DZD. Je však využíván i jako nástroj pro analýzu reálných dat, a to jak ve výzkumných projektech (zejména v lékařských studiích), tak i v komerčních aplikacích (v oblasti bankovnictví, marketingu i CRM). V neposlední řadě je systém používán při práci na bakalářských, diplomových i disertačních pracích, a to opět jak pro analýzu dat, tak i pro návrh nových postupů či technik v oblasti DZD.

V naší ukázce jsme definovali první požadavek, že ve vygenerovaných datech má být dvakrát vyšší četnost nesplacených půjček v okrese Šumperk, než je průměr. Tento požadavek bude vyjádřen zadáním úlohy pro *4ft-Miner* hledající *4ft-asociační pravila* tvaru

$$District(\ Šumperk) \Rightarrow^{+}_{1} Status(\ Bad),$$

kde $\Rightarrow^{+}$ je *4ft-kvantifikátor* „nadprůměrného souvisení“ (*above-average dependency)* s parametrem *p=1* požadujícím alespoň o 100 % vyšší frekvenci špatných půjček v okresu Šumperk v porovnání s frekvencí špatných půjček v celých datech. Pod *Status(Bad)* zahrnujeme jak půjčky již definitivně nesplacené (*B*), tak i půjčky s prodlením ve splácení (*D*). Abychom zdůraznili důležitost existence tohoto vztahu v datech, přiřadíme mu váhový koeficient 2, takže bude mít dvojnásobný podíl na výsledné *fitness* daného jedince.

Druhý požadavek na data se týkal vychýleného rozdělení platů ve vybraných okresech. K tomu využijeme jiný typ vztahu, tzv. *podmíněné frekvence* (nebo také *podmíněný histogram*). Zadaná úloha pro *CF-Miner* bude hledat zvýšenou četnost nízkých platů v podmnožinách celé datové matice dané okresy Šumperk, Louny a Česká Lípa. Míra zajímavosti vztahu je definována tak, že počet klientů s nízkými platy musí být alespoň 50 % počtu všech klientů z daného okresu. Počet všech klientů

v daném okrese je spočítán automaticky, takže práh může být zadán relativně. Výsledkem takto zadané úlohy pro *CF-Miner* může být žádné, jedno, dvě nebo tři pravidla – podle toho, v kolika okresech je požadovaný podíl klientů s nízkým platem skutečně nalezen. Cílem je získat data, ve kterých vztah platí ve všech třech okresech.

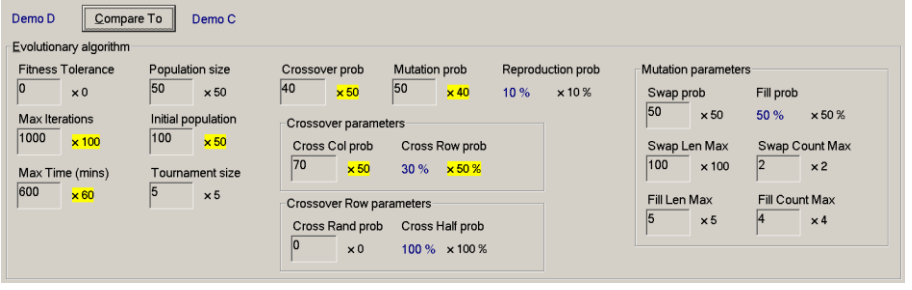
Poslední požadavek na data zabraňuje vzniku nadprůměrného počtu nesplacených půjček ve všech ostatních okresech, kromě okresu Šumperk. Je opět zadán jako *4ft-asociační pravidlo*, tentokrát v podobě

$$District(\ Any\_Except\_Šumperk) \Rightarrow^{+}_{0.5}\ Status(\ Bad)$$

Procedura *4ft-Miner* automaticky projde všechny okresy (kromě Šumperku) a zjišťuje, jestli v některém z nich není nadprůměrný výskyt špatných půjček (v tomto případě stačí, že jich je o 50 % více, než je průměr v celých datech). Důležitá je negativní váha tohoto požadavku, která zajistí, že se každý nalezený vztah podílí negativně na celkové *fitness* daného jedince.

I když jsme v naší jednoduché ukázce další typy vztahů nepoužili, tak například *K×L polní tabulky četností* (viz [Lín et al., 2005]) jsou vhodné pro hledání (podmíněných) vztahů mezi hodnotami dvou vícekategoriálních atributů. Stejně tak mohou být složitější podmínky pro definici podmnožin, na kterých má závislost platit – např. muži mezi 25 až 35 lety, bez univerzitního vzdělání a žijící v Praze nebo Plzni.

Uvedený způsob zadávání požadavků na vztahy v datech je poněkud specifický. Při klasické analýze dat se obvykle zadávají mnohem obecnější úlohy, jako například „*Jsou nějaké statisticky významné závislosti mezi atributy popisujícími sociální charakteristiky klienta na straně jedné a kvalitou půjčky na straně druhé?*“. Pro tento typ úloh byly i konstruovány možnosti jednotlivých modulů systému LISp-Miner. Přesto nic nebrání tomu, aby byly pro účely generování umělých dat zadávány i velmi konkrétní úlohy a více obecná zadání jsou používána pro negativní požadavky (viz výše).



Obr. 1 Evoluční parametry pro úlohu *Demo D* a porovnání s hodnotami pro *Demo C*

4. ReverseMiner

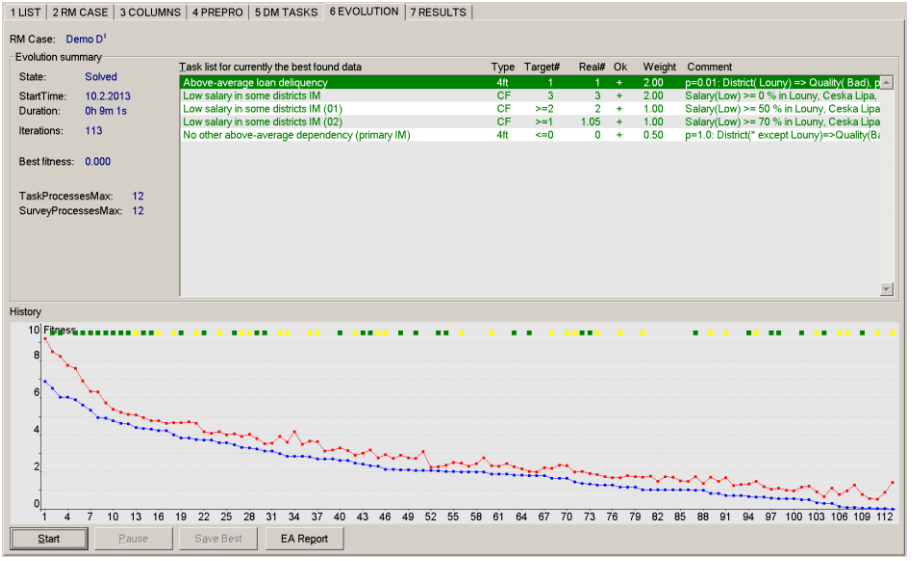
ReverseMiner je nový modul implementovaný do systému LISp-Miner. Využívá výše uvedený přístup a umožňuje uživateli vygenerovat umělá data podle zadaných požadavků. Modul je volně ke stažení na webových stránkách systému LISp-Miner, včetně ukázkových příkladů (viz také další sekce).

Modul umožňuje nadefinovat cílovou tabulku, včetně sloupců, jejich datových typů i počtu řádků – viz výše. Pro všechny parametry evolučního procesu nabízí předdefinované hodnoty, které však může uživatel měnit. Zároveň je možné nechat porovnat aktuální nastavení hodnot parametrů mezi dvěma úlohami – viz Obr. 1. Žlutě jsou zvýrazněny hodnoty, které se liší.

V neposlední řadě ReverseMiner umožňuje nastavení požadavků na data ve formě zadání *data miningové* úlohy, požadovaného počtu vztahů (které má takto zadaná úloha nalézt) a váhy, kterou se požadavek podílí na výpočtu celkové *fitness* jedince.

Novou úlohu generování dat je možné vytvořit i „klonováním“ již existující, takže se převezmou všechna uživatelská nastavení. Tímto způsobem je možné dělat drobné úpravy v zadání (např. změnit velikost populace jedinců nebo požadovaný počet vztahů v datech). Zároveň je v seznamu úloh zachována celá historie změn.

Po zadání úlohy generování dat je možné spustit evoluční proces – viz Obr. 2. Evoluční proces je interaktivní, takže uživatel může průběžně sledovat, které z požadavků jsou již úspěšně zakomponovány do generovaných dat (resp. do aktuálně nejlepšího jedince) – viz znaménko „+“ ve sloupci Ok. Ve spodní části obrazovky se zobrazuje graf historie evoluce znázorňující aktuálně nejlepšího a nejhoršího jedince. Vidíme tedy jak rychlost konvergence k požadovanému řešení, tak i aktuální diverzitu populace. Údaje o aktuálním stavu evoluce, počtu kroků i celkovém času jsou v levém horním rohu obrazovky.



Obr. 2 Generování dat – hlavní obrazovka se seznamem požadavků a historií evoluce

Modul *ReverseMiner* nemá prostředky pro hledání všech možných zadaných typů vztahů v datech. Místo toho opakovaně volá již dříve implementované moduly systému LISp-Miner, které provedou vlastní analýzu dat a modulu *ReverseMiner* vrátí veškeré potřebné údaje pro výpočet *fitness* každého jedince. Konkrétně jde o moduly *LMTaskPooler* a *LMGridPooler*. První je určen pro výpočet úloh na lokálním počítači

(včetně paralelního výpočtu na jeho vícejádrovém procesoru, je-li k dispozici), druhý pak pro výpočet úloh na počítačovém *gridu* (nebo v *cloudu*). Oba moduly byly původně vyvinuty pro dávkové zpracování mnoha úloh na pozadí a byly volány přímo uživateli z modulů systému LISp-Miner. Pro využití v *ReverseMineru* bylo přidáno aplikační rozhraní pro volání *poolerů* z jiného procesu a pro vzájemnou *mezi-procesovou* komunikaci.

Pro každého jedince je spuštěna nová instance modulu *LMTaskPooler* (resp. *LMGridPooler*) s tím, že se pro danou verzi dat mají spočítat všechny *data miningové* úlohy představující požadavky. Po úspěšném výpočtu všech úloh pro daného jedince vyexportuje *pooler* informace o počtu nalezených vztahů do PMML souboru a zasílá notifikační *mezi-procesovou* zprávu *ReverseMineru*. Import a export PMML souborů byl původně implementován pro výměnu zadání úloh a výsledků s jinými systémy, zejména v projektu SEWEBAR, viz [Kliegr et al., 2010]. Export je řízen šablonami, které mohou být uzpůsobeny specifickým potřebám, jaké má například *ReverseMiner*. Ten soubor načte a spočítá *fitness* tak, že porovnává skutečně nalezené počty vztahů s počty požadovanými.

Důležitou vlastností implementace evolučního algoritmu v *ReverseMineru* je opakovatelnost generování. Přestože jsou evoluční algoritmy teoreticky založeny na generátoru náhodných čísel, tak v praxi se u současných počítačů musíme spokojit pouze s generátory tzv. *pseudo-náhodných* čísel. Jejich hlavním znakem je nutnost inicializace pomocí počáteční hodnoty, která by měla být sama o sobě „náhodná“. Obvykle se pro tyto účely používá aktuální čas. Potom je již generátor schopen generovat řadu čísel, která splňuje základní požadavky na náhodné rozdělení. V případě, že generátor opakovaně inicializujeme stejnou počáteční hodnotou, dostaneme identickou řadu „náhodných“ čísel. Tato „nevýhoda“ je v *ReverseMineru* přeměněna na plus. Generátor *pseudo-náhodných* čísel pro evoluci se záměrně inicializuje stále stejnou hodnotou, takže i evoluce probíhá stále stejně (za předpokladu, že nedošlo ke změně parametrů nebo požadavků na data). Počáteční hodnotu je však možné měnit jako uživatelský parametr a vynutit tak jiný průběh evoluce (a pravděpodobně tedy i jinou podobu vygenerovaných dat). Tohoto chování je využíváno při výuce, v ukázkových příkladech, tak i při testování funkčnosti *ReverseMineru*.

Nalezne-li evoluce jedince, jehož varianta dat splňuje všechny požadavky na ně kladené, je aktuální podoba dat zobrazena v podobě tabulky. Tu je možné exportovat buď ve formě textového souboru (CSV), nebo ve formě databáze (MDB).

5. První výsledky

Během experimentů s nově naimplementovaným modulem a při přípravě ukázkových dat byly získány první zkušenosti. Kromě toho byl *ReverseMiner* úspěšně použit ve studentských pracích.

Doba nutná pro vygenerování dat je dostatečně krátká pro dobře nadefinované úlohy a středně rozsáhlá data – viz Tab. 1. Úlohy *Demo C* a *D* jsou stejné, v úloze *D* je však využit *fuzzy* koncept výpočtu *fitness* přímo z hodnoty míry zajímavosti (viz výše). Tím se jednak zkrátila doba nutná pro vygenerování požadovaných dat, ale také zjednodušilo zadávání požadavků na dat pomocí *data miningových* úloh.

Tab. 1 Ukázkové úlohy, nastavení parametrů a doba trvání (Intel Core i7 2630QM Sandy Bridge, 8GB RAM)

Úloha	Počet Řádků	DB Sloupců	Omezení na Data	Velikost Populace	Počet Iterací	Doba Trvání
Demo A	1000	2	3	10	4	4 sekundy
Demo B	1000	3	11	50	12	1m 5s
Demo C	1000	3	11	50	24	2m 11s
Demo D	1000	3	5	50	15	56 sekund
Demo D'	5000	3	5	50	113	9m 1s
Demo D''	10000	3	5	50	254	27m 20s

V tabulce jsou dále tři úlohy nazvané *Demo D*, *D'* a *D''*. Ty se liší v cílovém počtu řádků generované databázové tabulky. Vidíme, že doba generování je lineárně závislá na počtu iterací. Počet iterací je pak o něco víc než lineárně závislý na cílovém počtu řádků. Další experimenty ukazují, že i tabulky o velikosti sto tisíc řádků jsou vygenerovány v řádu hodin.

Tab. 2 Rozdělení doby generování Demo D' podle hlavních fází (v sekundách)

Fáze	Celkem	Přepočet na Jedince	Přepočet na Řádek	Přepočet na Sloupec
Inicializace evolučního algoritmu	0.639	0.013	0.000	0.213
Evoluční operace	1.063	0.021	0.000	0.354
Příprava souborů na disku	37.473	0.749	0.007	12.491
Spouštění *LMTaskPooleru*	120.402	2.408	0.024	40.134
Čekání na výsledky z *LMTaskPooleru*	373.990	7.480	0.075	124.663
Import výsledků	3.615	0.072	0.001	1.205
Odstraňování souborů na disku	3.818	0.076	0.001	1.273
Celkem	541.000	10.820	0.108	180.333

V tabulce 2 vidíme podíl hlavních fází generování dat na celkovém času generování. Nejvíce času zabírá výpočet *data miningových* úloh v modulu *LMTaskPooler*. Určitý čas zaberou diskové operace. Ty je možné zkrátit použitím rychlejšího pevného disku nebo SSD. Čas potřebný na vlastní evoluční operace je zanedbatelný.

Uvědomíme-li si množství *data miningových* úloh, které bylo nutné vyřešit (v populaci je 50 jedinců, každý z nich obsahuje pět úloh – požadavků na data, a pro každého nově vytvořeného jedince se musí všechny spočítat), je výpočet poměrně rychlý. To je umožněno i díky celé řadě optimalizací, které byly v systému LISp-Miner implementovány a kvůli kterým je doba řešení *data miningových* úloh lineárně závislá na počtu řádků analyzované datové matice (více viz [Rauch&Šimůnek, 2005]). Zároveň byla využita možnost paralelního řešení více úloh najednou na vícejádrovém procesoru.

Výpočet je dále možné zrychlovat využitím distribuovaného výpočtu na počítačovém gridu *Techila*. Ten dovoluje jak zapojení nevyužitých počítačů na síti (v kancelářích, v učebnách – podobně jako projekt *SETI@home*), tak i zakoupení výpočetního výkonu v *cloudu* (na platformě *Windows Azure*). Více o distribuovaném výpočtu viz [Šimůnek&Tammisto, 2010].

K další analýze průběhu evoluce můžeme použít detailní zprávu generovanou *ReverseMinerem* – viz Tab. 3. Jedinec může vzniknout pouze jedním z pěti způsobů –

náhodně (při inicializaci populace na začátku evoluce), jedním ze dvou typů křížení nebo jedním ze dvou typů mutace (viz evoluční operace výše). V řádcích tabulky 3 je pak pět významných stavů, ve kterých se jedinec může nacházet. Číselné hodnoty udávají, kolik jedinců v daném stavu vzniklo daným způsobem:

Tab. 3 Jedinci z ukázky Demo D' podle způsobu, kterým byli vytvořeni

	NÁHODNĚ	KŘÍŽENÍ		MUTACE	
JEDINEC		SLOUPCE	ŘÁDKY	PROHOZENÍ	VYPLNĚNÍ
Nově vytvořený	100	930	2154	1695	343
Stal se nejlepším	1	6	29	28	2
Stal se rodičem	52	958	2398	1371	343
Zkopírován beze změny	5	73	232	160	27
Odstraněn	100	922	2135	1676	339

Na začátku bylo náhodně vytvořeno 100 jedinců, z nichž 50 nejlepších utvořilo populaci první generace. V dalším průběhu evoluce vznikla většina nových jedinců křížením (3084), z toho 930 křížením přes sloupce a 2154 křížením přes řádky. Pouze 2038 jedinců vzniklo mutacemi (1695 prohozením hodnot a 343 vyplněním hodnot). Poměry jsou dané uživatelským nastavením 40/50/10 pro křížení/mutaci/reprodukci, protože při křížení vznikají vždy dva noví jedinci.

Naproti tomu u jedinců, kteří se v nějakém kroku stali aktuálně nejlepšími, jsou jedinci vzniklí křížením a mutací zastoupeni přibližně stejně (35 ku 30), přestože jedinců vzniklých křížením je v populace více (viz výše). Jedinci vzniklí křížením se však častěji stávají rodiči (i po zohlednění faktu, že jich je v populaci více).

Zjištění podporují fakt, že křížení jsou účelnější v první části evolučního procesu, kdy je možné využít široké diverzity populace. Mutace se uplatňují v pozdější fázi evoluce pro hledání mírně odlišných (ještě lepších) verzí dříve vyšlechtěných dobrých jedinců.

Spolu se zkušenostmi o průběhu evoluce byly identifikovány i tipy pro zadávání parametrů generování a požadavků na data. V prvé řadě je třeba volit vhodná počáteční rozdělení náhodně generovaných hodnot sloupců datové tabulky. Rozdělení by měla co nejvíce odpovídat následným požadavkům na data. I když by evoluční proces ve většině případů dokázal pomocí evolučních operací počáteční nevhodné rozdělení kompletně změnit na požadovaná, je zde riziko uváznutí v lokálním optimu a každopádně jde o zbytečnou ztrátu času.

Dalším důležitým tipem je definice „negativních" požadavků na data, jak bylo ukázáno i v příkladě výše. Negativní požadavky zabraňují vzniku celé řady dalších nechtěných vztahů v datech, ve kterých by se zamýšlené vztahy ztratily. Negativní požadavky mohou být definovány buď pomocí vhodně nastavené syntaxe zadání *data miningových* úloh, nebo pomocí pozitivně formulovaného požadavku, kterému přiřadíme zápornou váhou, kterou se má podílet na výpočtu celkové *fitness*.

Graf vývoje evoluce pomáhá při experimentech s nastavením evolučních parametrů i vah dílčích požadavků. I když je možné ponechat evoluci dostatečně dlouhý čas, aby našla požadovanou variantu dat, tak se v souladu s teorií ukazuje, že největších kladných evolučních změn je dosaženo během několika prvních evolučních kroků. Rychlá konvergence na počátku evoluce slouží jako vodítko pro správně zadané parametry a požadavky na data. Nepozorujeme-li konvergenci v prvních deseti až dvaceti krocích, je úloha patrně špatně nastavena, resp. některé požadavky na data mohou být vzájemně protichůdné.

6. Jiné přístupy

Nejjednodušším postupem přípravy umělých dat je generování generátorem náhodných čísel, bez jakýchkoliv dalších požadavků na ně, viz [Ross, 2002], [Hamuro et al., 2002]. Jde o velmi jednoduchou a rychlou metodu, ale podoba výsledných dat není obvykle pro *data miningovou* analýzu vhodná.

Opačný postup je pečlivě nastudovat vlastnosti reálných dat a zkonstruovat jejich statistický model, viz [Abowd&Lane, 2004]. Jde o pracnou činnost, která může být částečně (nikoliv úplně) automatizovaná, viz [Žabkar et al., 2011]. Umělá data jsou spočtena na základě zkonstruovaného modelu, případně doplněna náhodným šumem pro zvýšení dojmu realističnosti. Problém je složitost statistického modelu, pokud by měl zahrnout všechny vlastnosti a výjimky reálných dat. V praxi se pak setkáváme spíše se zjednodušenými modely, které obvykle ignorují zajímavé výjimky, které jsou však cílem *data miningové* analýzy. Druhým problém je případně dodaná náhodná složka, která musí být opatrně nastavena tak, aby zakryla umělou povahu dat, přitom v nich však nesmí přebít chtěné vztahy. Za tímto účelem bylo vyvinuto mnoho metod zajišťujících, že vybrané podstatné vlastnosti jsou v datech zachovány, např. vztahy mezi vysvětlujícími a závislými proměnnými, viz [Potharst&Wezel, 2005].

Přístup nejpodobnější zde prezentovanému je simulované žíhání, viz [Hunniford &Hickey, 1999]. Tato technika je podobná evolučním algoritmům, zejména kvůli své *sub-optimální* povaze a nutnosti výpočtu *fitness* pro aktuální stav.

Další částečně podobný přístup je diskutován v práci [Eno&Thompson, 2008]. Ten se pokouší jít cestou porozumění výsledkům dříve provedené *data miningové* analýzy a nalezené vztahy (v daném případě rozhodovací stromy) převést přímo na formule jazyka SDDL (*Synthetic Data Definition Language*) pro generování dat. Generátory založené na jazyku SDDL jsou už dostupné (viz např. [Hoag&Thompson, 2007]), chybí jim však syntaktická bohatost vztahů používaných v systému LISp-Miner.

Z dostupných aplikací uvádíme přístup používaný v nástroji *Rapidminer*, jak je popsán v [Akthar&Hahne, 2012]. K dispozici je operátor *Generate data* nabízející dvě základní možnosti vytvoření umělých dat – data z jedné z předem připravených doménových oblastí, s předem danými atributy (např. *Direct Mailing Data*, *General Sales Data*); nebo obecná data podle uživatelem zadaných parametrů. Ty se omezují pouze na zadání počtu atributů a jejich minimální a maximální hodnoty (v případě numerických) a počtu hodnot (v případě nominálních). Funkce generování umělých dat v *RapidMineru* je zamýšlena zejména pro začínající uživatele, kteří si chtějí nástroj vyzkoušet a nemají data vlastní. Proto byla funkce navržena co nejjednodušší na použití. Z pokročilejších možností obsahuje nadefinování odvozeného atributu, jehož hodnoty jsou vypočteny matematických výrazem z hodnot jiných atributů. Dodatečnou možnost ovlivnění podoby vygenerovaných dat pak nabízí operátor *Přidání šumu* (*Add Noise*). Jde však o přidání další náhodné složky, nikoliv o možnost zakomponování požadovaných závislostí do dat.

7. Směry dalšího rozvoje

Kromě původně zamýšleného použití *ReverseMineru* pro přípravu výukových a testovacích dat se zdá jako vhodné jeho začlenění přímo do výuky *data miningu*. Místo, aby studenti dostali za úkol data analyzovat, mohou alternativně data vytvořit. Didaktický záměr je v tom, že při vytváření umělých dat (nejen pomocí *ReverseMineru*)

se velmi dobře procvičí i činnosti nutné pro jejich analýzu: podrobně poznat doménovou oblast; navrhnout způsob kódování znaků v databázové tabulce i předzpracování dat; rozmyslet různé typy vztahů, které se v datech mohou vyskytovat; a v neposlední řadě se zamýšlet, co a jak by fiktivní vlastník chtěl o datech zjistit a jak mu nalezené výsledky srozumitelně podat. Velkou výhodou je fakt, že student si doménovou oblast vybírá sám, nejlépe podle oblasti svého zájmu. Naproti tomu striktně přidělená data z oblasti, která je studentovi cizí, nemohou vést k dobrým výsledkům analýzy a navíc mohou zapříčinit i potlačení zájmu o *data mining* jako takový.

Po dalším rozšíření okruhu uživatelů *ReverseMineru* očekáváme nejen rozšíření zkušeností a tipů pro práci, ale i nové požadavky na jeho funkčnost. Místem, kde v budoucnu dojde k dalšímu vylepšení, bude zcela jistě přidání více typů inicializace počáteční populace. Kromě již implementovaného náhodného vytvoření jedinců jsou rozmyšleny i možnosti inicializace podle vzorových dat, včetně možné změny cílového počtu řádků, či drobných změn hodnot. Dalším směrem může být vylepšení stávající jednoduché implementace evolučního algoritmu, například ve formě „omlazení“ populace s cílem zvýšit diverzitu v případě, kdy dojde k uváznutí v lokálním optimu. Důležitou podmínkou jakýchkoliv změn evolučního algoritmu je však zachování jeho opakovatelnosti – jak kvůli potřebám výuky, tak i testování.

8. Závěr

Příprava umělých dat pro výuku je důležitá, přitom pracná činnost, která však díky svým specifikům – na přípravu dat je dostatek času (například o prázdninách), nejde o životně kritická data (takže se spokojíme třeba i se sub-optimálním řešením) – umožňuje nasazení evolučních algoritmů. Nově naimplementovaný modul *ReverseMiner* využívá široké možnosti různých typů vztahů implementovaných v systému LISp-Miner, včetně jejich bohaté syntaxe. Modul byl úspěšně vyzkoušen pro přípravu umělých dat výše popsaným přístupem. Prezentovaný postup generování umělých dat může být navíc implementován i pomocí jiného nástroje, pokud tento podporuje dostatečné množství různých typů vztahů a zároveň má prostředky k automatickému opakovanému spouštění mnoha *data miningových* úloh na pozadí.

Prohlášení: *Tato práce byla podpořena grantem IGA 20/2013 interní grantové agentury VŠE.*

Použitá literatura

Abowd, J. M., Lane, J., 2004: *New Approaches to Confidentiality Protection: Synthetic Data, Remote Access and Research Data Centers*. In: Privacy in Statistical Databases. Springer-Verlag, pp. 282-289

Akthar, F., Hahne, C., 2012: *RapidMiner 5 Operator Reference*. Rapid-I. cit: 28. 6. 2013. Dostupný z WWW: http://rapid-i.com/content/view/306/233/lang,en/

Berka, P., 2011: *ETree Miner: a new GUHA procedure for building exploration trees*. In: Foundations of Intelligent Systems. New York: Springer, s. 96–101. ISBN 978-3-642-21915-3. ISSN 0302-9743.

Eno, J., Thompson, C.W., 2008: Generating Synthetic Data to Match Data Mining Patterns. *IEEE Internet Computing*, Vol. 6. pp. 78-82

Hájek, P., Havránek, T., 1978: *Mechanising Hypothesis Formation – Mathematical Foundations for a General Theory.* Berlin – Heidelberg – New York, Springer-Verlag, 396 pp.

Hájek, P., Holeňa, M.; Rauch, J., 2010: The GUHA method and its meaning for data mining. *Journal of Computer and System Sciences,* 76, pp. 34-48

Hamuro, Y., Katoh, N., Yada, K., 2002: MUSASHI: Flexible and Efficient Data Preprocessing Tool for KDD based ON XML. In: *Proceedings of the First International Workshop ON Data Cleaning and Preprocessing* and 2002/12, pp.38-49

Hoag, J., Thompson, C., 2007: A Parallel General-purpose Synthetic Data Generator. *ACM SIGMOD Record,* Vol. 36, No. 1, pp. 19–24

Hunniford, T.J.C., Hickey, R.J., 1999: A simulatedannealingtechnique for generatingsyntheticdata to assess concept learning algorithms. *Intelligent Data Analysis.* Vol. 3, issue 3, pp. 177–189

Kliegr, T., Svátek, V., Ralbovský, M., Šimůnek, M., 2010: SEWEBAR-CMS: semantic analytical report authoring for data mining results. *Intelligent Information Systems* [online], pp. 1-25. ISSN 0925-9902. URL: http://dx.doi.org/10.1007/s10844-010-0137-0

Lín, V., Dolejší, P., Rauch, J., Šimůnek, M., 2013: The KL-Miner Procedure for Datamining. *Neural Network World,* 2004, Vol. 5, pp. 411–420. ISSN 1210-0552. LISp-Miner [online]. 2013. cit. 20. 4. 2013. Dostupný z WWW: http://lispminer.vse.cz

Rauch, J., Šimůnek, M., 2005: *An Alternative Approach to Mining Association Rules.* In: LIN, Tsau Young et.al.(eds.). Foundations of Data Mining and Knowledge Discovery. Berlin, Springer, s. 211–231. ISBN 3-540-26257-1. ISSN 1860-949X

Potharst, R., Wezel, M.C., 2005: *Generating synthetic data with monotonicity constraints*. No EI 2005-06, Econometric Institute Report, Erasmus University Rotterdam, Econometric Institute

Ross, S.M., 2006: *Simulation*, Fourth Edition (Statistical Modeling and Decision Science). Academic Press. ISBN: 978-0125980630

SEWEBAR – SEmantic WEB and Analytical Reports [online]. 2011. cit.15. 2. 2012. Dostupný z WWW: http://sewebar.vse.cz/

Šimůnek, M., 2010: *Systém LISp-Miner – akademický systém pro dobývání znalostí z databází, Historie vývoje a popis ovládání.* skripta VŠE, Praha, Oeconomica, 106 stran, ISBN: 978-80-245-1699-8

Šimůnek, M., Rauch, J., 2011: *EverMiner – Towards Fully Automated KDD Process.* In: FUNATSU, K., HASEGAWA, K. New Fundamental Technologies in Data Mining. Rijeka : InTech, s. 221–240. 584 s. ISBN 978-953-307-547-1

Šimůnek, M., 2012: Nová GUHA-procedura ETree-Miner v systému LISp-Miner. *Systémová integrace,* roč. 19, č. 2, s. 62–72. ISSN 1210-9479 (print). ISSN 1804-2716 (online). URL: http://www.cssi.cz/cssi/system/files/all/SI_2012_02_06_Simunek.pdf

Weise, T., 2011: *Global Optimization Algorithms – Theory and Application.* Third edition. Available from URL http://www.it-weise.de/projects/bookNew.pdf

Žabkar, J., Možina, M., Bratko, I., Demšar, J., 2011: Learning qualitative models from numerical data. *Synthetic Intelligence.* Volume 175, Issues 9–10, pp. 1604–1619

JEL: D83, C15