ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra kybernetiky

Predikce v prostředí inteligentního domova

Diplomová práce

květen 2010

Student: Miroslav Řehoř

Vedoucí práce: Ing. Lenka Nováková, Ph.D.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 14.5.2010

Podpis

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Miroslav Ř e h o ř

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný

Obor: Kybernetika a měření , blok KM2 – Umělá inteligence

Název tématu: Predikce v prostředí inteligentního domova

Pokyny pro vypracování:

1. Seznamte se se systémem inteligentního domova a jeho schopnostmi z hlediska uživatele. Sestavte demonstrační soubor akcí reprezentující činnost uživatele s omezenou schopností pohybu.
2. Seznamte se s algoritmy, které mohou být využity jako predikční systémy pro inteligentní domovy.
3. Zvolte vhodný predikční algoritmus a vytvořte univerzální knihovnu, která bude použitelná jako predikční modul pro systém inteligentního domova.
4. Činnost vytvořené knihovny ověřte na předpřipraveném scénáři z bodu 1. Navrhněte možnosti využití logu systému pro optimalizaci jeho činnosti.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí diplomové práce: Ing. Lenka Nováková, Ph.D.

Platnost zadání: do konce letního semestru 2010/2011

L.S.

prof. Ing. Vladimír Mařík, CSc.
vedoucí katedry

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 12. 2. 2010

Poděkování

Chtěl bych poděkovat Ing. Lence Novákové, Ph.D. za vedení této diplomové práce, za cenné rady, za trpělivost a také za ochotu, se kterou se mi vždy věnovala.

Dále bych chtěl poděkovat Ing. Petru Novákovi za poskytnutí materiálů a konzultace, které umožnily, aby byla tato práce úspěšně dokončena.

Poděkování patří také mé matce za podporu, kterou mi poskytla během celého studia.

Abstrakt

Tato diplomová práce se zabývá problematikou využití metod umělé inteligence pro predikci akcí uživatele v prostředí inteligentního domova a vytvořením predikčního modulu. Stručně je zmíněn pojem inteligentní domov. Dále je popsán generátor akcí, který simuluje akce uživatele a který je použit pro testování predikčního modulu. Další část rozebírá dostupné algoritmy vhodné k predikci v inteligentním domově. Následuje popis implementace zvoleného algoritmu - asociačních pravidel. V poslední části jsou uvedeny experimenty provedené s pomocí dat vytvořených zmíněným generátorem akcí. Všechny metody byly implementovány v prostředí Microsoft Visual Studio 2008, v jazyce C#.

Abstract

This Master's thesis deals with the use of artificial intelligence methods for user's action prediction in an environment of a smart home. First, the term smart home is defined followed by a brief introduction of the subject area. Then, a description of a program for user's action generation is provided. In the next section there is a discussion of algorithms suitable for prediction in a smart home environment. Following the discussion of algorithms, specifications of implemented method – association rules – are described. The last section looks at performed experiments using the aforesaid action generator. All methods have been implemented in Microsoft Visual Studio 2008, in C#.

1. Obsah

2. Úvod

Provádění běžných akcí, jako je například zapínání televize nebo rozsvícení světla, je pro hendikepované spoluobčany s omezenou schopností pohybu velmi náročné. V současné době neexistují pro tyto osoby dostupné systémy, které by jim usnadňovaly běžné akce vykonávat bez pomoci ošetřovatele. Je nutné pro ně vytvořit speciální zařízení, které jsou schopni sami i se svým postižením ovládat. Takové zařízení jim nabízí akce, ze kterých následně uživatel vybírá tu, kterou chce, aby systém následně vykonal.

V rámci systému je žádoucí uživateli nabízet akce pro danou situaci vhodné, například když se zešeří nabídnout rozsvícení světel. Potřebujeme se chování uživatele naučit a následně předpovídat jeho budoucí akce. Tyto akce mu pak bude možné přednostně nabízet.

Cílem této diplomové práce je prozkoumat algoritmy, které by mohly být použity pro predikci akcí v inteligentním domově, a naimplementovat predikční modul. Tento predikční modul bude následně zakomponován do systému vyvíjeného na katedře kybernetiky.

3. Prostředí inteligentního domova

3.1 Co je to inteligentní domov?

Inteligentní domov, v angličtině *Smart Home*, je domácnost vybavená spoustou senzorů, elektronických zařízení a řídicím systémem. Dohromady všechny tyto prvky vytvářejí celek, který je na první pohled velmi efektní a na laika působí dojmem, že je schopen sám uvažovat a že je inteligentní. Nechci se zde pouštět do definice a rozboru termínu „inteligence“, ale myslím, že lidé, kteří se alespoň trochu orientují v problematice umělé inteligence, se mnou budou souhlasit, když intuitivně prohlásím, že dnešní inteligentní domovy vlastně inteligentní nejsou.

Obr. 1 Ilustrace inteligentního domova 1

Rád bych se ale přece jen krátce zastavil s malou vsuvkou. Nedávno jsem četl knihu Jeffa Hawkinse: On Intelligence (Hawkins, 2004) a velmi mě zaujala. Autor prezentuje svoji teorii o fungování mozku, tzv. Memory prediction framework a částečně se také dotýká pojmu inteligence. Ve zkratce říká, že systém je inteligentní, když si vytváří model skutečnosti, který pak používá k predikci budoucnosti, tu následně porovnává s realitou a svůj model koriguje. Tak funguje i náš mozek. Řekl bych, že inteligence není jen true-false hodnota, ale měla by se stanovovat její míra. Například pes má rozhodně poměrně sofistikovaný mozek, takže máme tendenci říci, že je inteligentní, ale oproti člověku je stále o několik řádů jinde.

Ale zpět k našemu tématu inteligentního domova. Přestože dnešní systémy v pravém slova smyslu inteligentní nejsou, jsou schopny poměrně zajímavých výsledků. Co vše mohou umět?

3.2 Situace na trhu

Obr. 2 Ilustrace inteligentního domova 2

Stačí zadat do Googlu frázi „smart home solutions“ a objeví se celá řada společností nabízejících kompletní řešení vašeho inteligentního domova. Při svém průzkumu jsem ale nenarazil na jedinou firmu, která by nabízela něco skutečně pokrokového. Jedná se vždy o standardní funkce jako:

- rozsvícení světel po vstupu do místnosti,
- nastavování úrovně vytápění v závislosti na čase a poloze uživatelů,
- vypnutí všech spotřebičů při odchodu z domu/bytu,
- ovládání všech zařízení z mobilního telefonu,
- nouzový stav v případě požáru apod. (např. rozsvícení světel k východu),
- optimalizace svícení, topení.

V zásadě se tedy jedná o pevně naprogramované a tudíž přísně deterministické akce typu „if A then B“. Z pohledu dodavatelských firem je tento přístup naprosto pochopitelný, neboť jsou schopni ručit za kvalitu, bezpečnost a funkčnost celého systému – na rozdíl od novátorského a zajímavého, ale částečně nepředvídatelného systému. Všechny tyto systémy jsou tak spíše podle mého názoru určeny k ohromení hosta nebo k uspokojení fanoušků moderních technologií, kteří mají více než plnou kapsu. K praktickému masovému využití mají poměrně daleko. A řekl bych, že se svými stávajícími funkcemi ani nejsou pro obyčejného člověka moc potřebné.

Situace je však trochu jiná, pokud se začneme bavit o osobách s tělesným postižením, a to zejména s poruchou pohybového ústrojí. Těmto lidem by podobný systém velmi usnadnil život. Pro ně jsou však tyto již existující systémy příliš drahé. Tito lidé nepotřebují kompletně vybavit celou svou domácnost novými spotřebiči jedné luxusní značky, propojených LAN sítí, za stovky tisíc korun. Televizi, video a další zařízení již doma mají a jen by potřebovali spojit je do jednoho systému centrálně ovladatelného například z jejich invalidního vozíku.

A právě takový systém vyvíjí na katedře kybernetiky Ing. Petr Novák, jehož součástí bude i predikční modul, který je předmětem zájmu této diplomové práce.

3.3 Situace ve výzkumu

Soukromá sféra však není jediná, která se tématem inteligentního domova zabývá – je tu ještě sféra akademická. Asi nejzajímavějším, na co jsem při svém hledání narazil, je projekt MavHome z University of Texas at Arlington (S. K. Das, 2002).

Jak uvádějí autoři projektu: „Cílem projektu MavHome je vytvořit dům, který se bude chovat jako racionální agent. Tento agent se snaží maximalizovat komfort uživatelů a minimalizovat provozní náklady. K dosažení těchto cílů je zapotřebí, aby agent byl schopen predikovat vzorce pohybu uživatele a vzorce používání jednotlivých zařízení.“ Pro predikci jsou využívány 3 algoritmy:

- LeZi-update – algoritmus pro určování polohy uživatele,
- SHIP – metoda pro předpověď další interakce uživatele s prostředím,
- ED – algoritmus hodnotící význam jednotlivých epizod, tedy sérií akcí.

Tyto 3 metody jsou nasazeny současně, a společně tak umožňují efektivní fungování celého systému. To autoři dokazují použitím na konkrétních datech.

Další projekt pochází z Duke University v USA a nese název „Smart Home Program“ (Smarthome, 2010). Vysoce moderní dům, který je vidět na Obr. 3, je vybaven celou řadou moderních přístrojů a technologií. Jeho primárním účelem však není inteligentní domov v pravém slova smyslu. Je to spíše příležitost k vyzkoušení týmové spolupráce a k praktickému poznání studovaných technologií. Z projektů, které se zde realizují, bych jmenoval například „Smart Floor“, „RFID Tracking“ či „Alternative Energy“.

Obr. 3 Duke University Smart Home

Další článek, který mě zaujal, pochází z univerzity ETSI Informática Avda, Sevilla. Jeho titulem je „Modeling Smart Homes for Prediction Algorithms“ (Fernández-Montes, 2007). Autoři stručně a velmi přehledně popisují, jak by měl vypadat ideální predikční algoritmus, jaké vstupy je vhodné zvolit. Také koncept „okna akcí“ jsem ve svém programu využil.

3.4 Co by měl umět inteligentní domov?

Zamysleme se nyní tedy krátce, co vlastně bychom od inteligentního domova chtěli. Myslím, že odpověď je nasnadě. V ideálním případě bychom chtěli, aby náš domov uměl předpovědět libovolnou naši další akci se stoprocentní přesností a zároveň by ji rovnou vykonal. Již nikdy bychom nemuseli sami otvírat dveře, zatahovat žaluzie, pouštět vodu z kohoutku, ani rozsvěcovat světlo. Je zřejmé, že toto je ovšem velice složitý a snad až nereálný požadavek. I pro člověka, který má spoustu zkušeností a výbornou doménovou znalost, by to byl velmi náročný úkol a ani jeho předpovědi by nikdy nebyly neomylné. Nemůžeme tedy takovou schopnost požadovat ani po relativně primitivním počítačovém programu.

Co však vyžadovat můžeme, je alespoň hrubý odhad následujících událostí, či akcí. Náš inteligentní domov by mohl být schopen v každém časovém okamžiku na základě již vykonaných akcí sestavit seznam dalších kroků. Tento seznam by mohl být seřazen například podle pravděpodobnosti následujícího kroku. Většinu času by tyto pravděpodobnosti byly patrně nízké a nevýznamné. Pokud by však překročily určitou stanovenou hranici, systém by nás upozornil a nabídl nám několik alternativ, mezi kterými bychom si vybrali. V případě, že pravděpodobnost bude znatelně vysoká, mohl by systém tuto akci provést rovnou sám.

Obr. 4 Ilustrace inteligentního domova 3

4. Simulace akcí uživatele

Pro testování a ověření funkčnosti predikčního algoritmu, který je hlavním výstupem této diplomové práce, je zapotřebí mít data, na kterých algoritmus budeme moci vyzkoušet. Ideální by samozřejmě bylo mít data reálná, z nějakého fungujícího inteligentního domova. Ty však bohužel k dispozici nemám. Mým úkolem je tedy taková data vytvořit. Nyní bych rád stručně popsal, jak jsem při vytváření simulovaného logu postupoval. Chci tím osvětlit logiku svého přístupu a poukázat na některá negativa, která by mohla být při případném vytváření nové verze odstraněna.

Začal jsem nejprve tím, že jsem ručně vytvořil záznam jednoho dne obsahující základní atributy jako poloha uživatele, teplota, úroveň osvětlení, stav dveří, oken, televize, rádia a podobně. Záhy jsem však zjistil, že záznam jednoho dne nedostačuje a že by bylo potřeba mnohem delší log. Zároveň jsem si také uvědomil, že je velmi neefektivní psát každý záznam stavu domácnosti ručně a že by bylo lepší kódovat akce v nějakém pseudo-jazyce a ten pak převádět na finální seznam stavů. Dodatečně mě napadlo, že by bylo vhodné do mého pseudo-jazyka zavést nějakou neurčitost týkající se času provádění akce a umožnit variabilitu pořadí vybraných akcí.

V této sekci se budu nejprve věnovat tomu, jaké veličiny jsou relevantní pro prostředí inteligentního domova. Dále budu popisovat algoritmus, jakým vytvářím seznam stavů neboli log. Začnu představením algoritmu jako celku a následně se budu věnovat struktuře programu a detailům implementace.

4.1 Simulované veličiny

Chceme-li kvalitně predikovat akce uživatele, musíme se rozhodnout, jak budeme vytvářet data, ve kterých budeme hledat příčiny těchto akcí. Obecně asi můžeme prohlásit, že čím víc veličin sledujeme, tím lepší naše predikce bude. Na druhou stranu za zbytečné informace platíme zvýšením náročnosti hledání potřebných souvislostí, a tedy snížením výpočetního výkonu predikčního modulu jako takového. Intuitivně je jasné, že poloha uživatele bude klíčovým atributem pro předpověď jeho akcí a například barva stěny v kuchyni je naprosto irelevantní. To však náš

predikční algoritmus, který nedisponuje rozumným modelem prostředí, neví. Přistupuje tedy ke všem proměnným se stejnou vážností.

Při přemýšlení o tom, které veličiny jsou vhodné a relevantní, jsem se inspiroval v již zmiňované článku autorů z univerzity v Seville. (Fernández-Montes, 2007). Vytvořil jsem pak následující tabulku Tab. 1, která se snaží rozdělit atributy do několika skupin. Pravý sloupec obsahuje příklad z dané kategorie:

Zařízení	Status	zapnuto/vypnuto, teplota trouby
	Poloha	vysavač je v obývacím pokoji
Obyvatel	Osobní data	jméno, věk, pohlaví
	Poloha	v kuchyni
	Fyzický stav	má rýmu, má zlomenou ruku
	Mentální stav	rozzlobený, spokojený
Prostředí	Datum, čas	pátek 13. 5., 13:20
	Parametry prostředí	teplota venku/uvnitř, prší, světlo/tma
	Poloha nábytku	postel je v ložnici

Tab. 1 Relevantní atributy

Finální rozhodnutí, které veličiny budou použity, však není na mém systému. Ten pouze zpracovává data, která jsou mu předána. Jaké senzory budou v domácnosti nasazeny a jaké vstupy tak bude můj systém dostávat, záleží na celkové koncepci inteligentního domova a není součástí této práce. Uvedená tabulka slouží pouze pro představu. Veličiny, které jsem použil ve svém generátoru akcí, jsou vybrány z této tabulky intuitivně.

4.2 Algoritmus vytváření simulovaných akcí

Algoritmus tvorby simulovaných akcí, v dalším textu jej budu nazývat log, je přehledně zobrazen na schématu na další straně – Obr. 5. Zde nyní naznačím celou ideu a detailům se budu věnovat v dalších sekcích.

Celý proces vychází ze **souboru předlohy**, pracovně nazývaném *template*. Každý řádek v tomto souboru popisuje jednu akci uživatele. Poskytuje však zároveň další dvě možnosti:

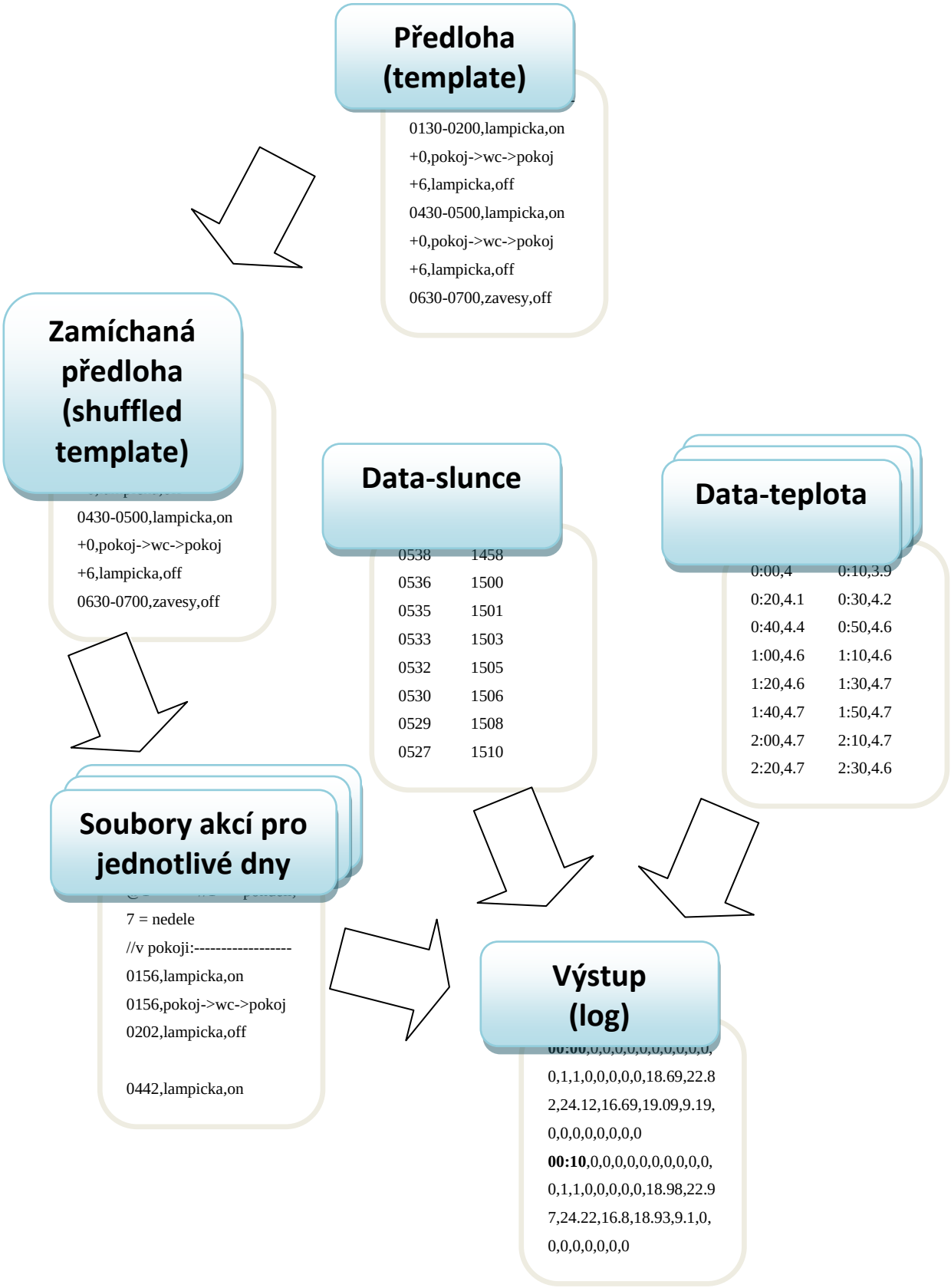
1. nespecifikovat čas provedení akce přesně, nýbrž intervalem, ze kterého je konkrétní čas vybrán náhodně,
2. vytvořit množinu akcí a každému jejímu prvku přiřadit pravděpodobnost, na jejímž základě pak bude pořadí těchto akcí stanoveno.

Syntaktické podrobnosti přiblížím později. V dalších krocích se pak postupně vytvoří tyto soubory:

- **zamíchaný soubor předlohy** (*shuffled template*) – akce již mají pevné pořadí (bod 2), ale jejich časová neurčitost stále zůstává zachována (bod 1),
- **soubory akcí pro jednotlivé dny** – akce mají pevné pořadí, časově jsou již přesně určeny; každý soubor odpovídá jednomu dni,
- **log jednoho dne** – zde do hry vstupují soubor s daty o východu a západu **slunce** a soubory obsahující informace o **teplotě**. Soubor se slunečními daty obsahuje 28 záznamů. Rovněž teplotních souborů je 28. Celkem tedy můžeme s těmito daty generovat 28 unikátních dnů, tedy 4 týdny. Kombinací jednoho souboru akcí, jednoho souboru s teplotami a jednoho řádku slunce tak vytvoříme vždy log jednoho dne.

Další možností, kterou jsem přidal na závěr, je možnost generování naprosto **náhodného logu**. Výsledek je však opravdu zcela nahodilý, kontrolované jsou pouze rozsahy veličin, ale už se neověřuje, jestli se například uživatel nachází ve dvou místnostech zároveň. Tento log jsem využil při testování predikčního algoritmu.

Při popisu jednotlivých kroků budu postupovat tak, jak postupuje i můj program.



Obr. 5 Schéma algoritmu generování logu

4.2.1 Předloha (template)

V prvním kroku se z předlohy – jak ji vidíme na Obr. 6 – vytváří zamíchaná předloha.

```
//v pokoji:---------------------------------------
0130-0200,lampicka,on
+0,pokoj->wc->pokoj
+6,lampicka,off         //protoze musime pocitat se zpozdenim wc

0430-0500,lampicka,on
+0,pokoj->wc->pokoj
+6,lampicka,off         //obdobne

0630-0700,zavesy,off
+1,pokoj->wc->pokoj

0727-0731,radio,1
0740-0745,radio,off

0810-0817,pokoj->wc_koupelna->pokoj

//snidane:---------------------------------------
+10,pokoj->kuchyn //protoze musime pocitat se zpozdenim wc_koupelna

[+0,0.7],zavesy,off
[+0,0.2],radio,1
[+1,0.1],topeni,8
###   // = konec sekvence
```

Obr. 6 Ukázka předlohy

Z množiny akcí se sestaví pořadí těchto akcí na základě specifikovaných pravděpodobností. Zde například z množiny:

```
[+0,0.7],zavesy,off
[+0,0.2],radio,1
[+1,0.1],topeni,8
###   // = konec sekvence
```

Syntaxe je následující. První hranatá závorka určuje počátek sekvence, '###' pak její konec. Každý řádek představuje jednu akci. První číslo uvnitř závorek je časové určení, druhá hodnota je pravděpodobnost. V tomto případě tedy bude první akcí:

- akce **„+0,zavesy,off“** s pravděpodobností 70%,
- akce **„+0,radio,1“** s pravděpodobností 20%,
- a akce **„+1,topeni,8“** s pravděpodobností 10%.

Druhá akce se pak ze zbylých vybere opět vzhledem k pravděpodobnosti. A tak dále. V tabulkách Tab. 2 a Tab. 3 jsou podrobně rozepsány všechny použitelné příkazy:

Makro akce		
	Makro-akce	**Elementy této makro-akce**
Větší akce	pokoj->wc->pokoj	pokoj->chodba; svetlo_chodba,on; chodba->wc; wc->chodba; svetlo_wc,off; svetlo_chodba,off; chodba->pokoj
	pokoj->wc_koupelna->pokoj	kuchyn->chodba; svetlo_chodba,on; svetlo_wc,on; chodba->wc; wc->chodba; svetlo_wc,off; svetlo_chodba,off; chodba->pokoj
	pokoj->wc_koupani->pokoj	pokoj->chodba; svetlo_chodba,on; svetlo_wc,on; chodba->wc; wc->chodba; svetlo_wc,off; svetlo_koupelna,on; chodba->koupelna; koupelna->chodba; svetlo_koupelna,off; svetlo_chodba,off; chodba->pokoj
	kuchyn->wc->pokoj	pokoj->chodba; svetlo_chodba,on; svetlo_wc,on; chodba->wc; wc->chodba; svetlo_wc,off; svetlo_koupelna,on; chodba->koupelna; koupelna->chodba; svetlo_koupelna,off; svetlo_chodba,off; chodba->pokoj
Goto akce	kuchyn->pokoj	moveto_chodba
	kuchyn->chodba	moveto_chodba
	pokoj->kuchyn	dvere_pokoj,on; moveto_chodba; dvere_pokoj,off; moveto_kuchyn
	pokoj->chodba	dvere_pokoj,on; moveto_chodba; dvere_pokoj,off
	chodba->wc	dvere_wc,on; moveto_wc; dvere_wc,off
	chodba->koupelna	dvere_koupelna,on; moveto_koupelna; dvere_koupelna,off
	chodba->pokoj	dvere_pokoj,on; moveto_pokoj; dvere_pokoj,off
	wc->chodba	dvere_wc,on; moveto_chodba; dvere_wc,off
	koupelna->chodba	dvere_koupelna,on; moveto_chodba; dvere_koupelna,off

Tab. 2 Makro akce

Elementární akce		
	Akce	**Možné hodnoty**
Různé	lampicka	on, off
	zavesy	on, off
	radio	off, {cislo kanálu}
	topeni	0, 1, 2, 3, 4, 5, 6, 7, 8
	tv	off, {cislo kanálu}
Světlo	svetlo_chodba	on, off
	svetlo_wc	on, off
	svetlo_koupelna	on, off
	svetlo_kuchyn	on, off
	svetlo_pokoj	on, off
Dveře	dvere_koupelna	on, off
	dvere_wc	on, off
	dvere_pokoj	on, off
Pohyb	moveto_kuchyn	bez parametru
	moveto_pokoj	bez parametru
	moveto_chodba	bez parametru
	moveto_koupelna	bez parametru
	moveto_wc	bez parametru

Tab. 3 Elementární akce

4.2.2 Zamíchaná předloha (shuffled template)

Na Obr. 7 vidíme zamíchanou předlohu. Pořadí akcí je již určeno a nyní musíme určit časy konání. Vidíme 2 typy časového popisu:

1. **rozsah** – například v prvním řádku „0130-0200,lampicka,on“
   Tento zápis znamená, že akce nastane někdy mezi časy 1h30min a 2h00min.
2. **posun** – například ve druhém řádku „+0,pokoj->wc->pokoj“
   Hodnota za symbolem plus určuje, o kolik bude časová značka této akce větší než akce předchozí. To znamená, že pokud se předchozí akce vyhodnotí jako např. „0139,lampicka,on“, tato akce bude určena jako „0139,pokoj->wc->pokoj“
3. **posun s rozsahem** – např. řádek „+20-30,radio,off“
   Toto je kombinace předchozích dvou pravidel. K předchozímu řádku se přičte číslo z intervalu 20 až 30.

```
//v pokoji:---------------------------------------
0130-0200,lampicka,on
+0,pokoj->wc->pokoj     //protoze musime pocitat se zpozdenim wc
+6,lampicka,off

0430-0500,lampicka,on
+0,pokoj->wc->pokoj
+6,lampicka,off         //obdobne

0630-0700,zavesy,off
+1,pokoj->wc->pokoj

0727-0731,radio,1
0740-0745,radio,off

0810-0817,pokoj->wc_koupelna->pokoj

//snidane:---------------------------------------
+10,pokoj->kuchyn //protoze musime pocitat se zpozdenim wc_koupelna

+0,zavesy,off
+0,radio,1
+1,topeni,8
```

Obr. 7 Ukázka zamíchané předlohy

4.2.3 Teplotní data

Data potřebná pro simulaci teplot jsem získal ze stránek http://www.fifeweather.co.uk/. Konkrétně se jedná o meteorologická data získaná měřením ve skotském městečku Cowdenbeath během února 2005. Datový soubor obsahoval celou řadu měřených veličin. Pro svou potřebu jsem však vybral pouze teplotu. Měření bylo prováděno v desetiminutových intervalech, což je naprosto dostačující. Pro účely simulace tato data zcela vyhovují.

Data jsem rozdělil do 28 textových souborů pojmenovaných 0.txt až 27.txt. Jsou uložena ve složce „Temperatures“, kde je také očekává můj program. Čas je ve formátu {h:mm}, teplota pak ve formátu desetinného čísla odděleném desetinnou tečkou. Mezi těmito dvěma hodnotami je čárka. Vše je vidět v tabulce Tab. 4.

0:00,9.2	0:00,4	0:00,5.6		0:00,1.6
0:10,9.1	0:10,3.9	0:10,5.5		0:10,1.6
0:20,8.9	0:20,4.1	0:20,5.6		0:20,1.6
0:30,8.7	0:30,4.2	0:30,5.4		0:30,1.6
0:40,8.6	0:40,4.4	0:40,5.4		0:40,1.5
0:50,8.6	0:50,4.6	0:50,5.5		0:50,1.4
1:00,8.4	1:00,4.6	1:00,5.4		1:00,1.3
1:10,8.3	1:10,4.6	1:10,5.4		1:10,1.3
1:20,8.2	1:20,4.6	1:20,5.6		1:20,1.2
1:30,7.9	1:30,4.7	1:30,5.7		1:30,1.2
1:40,7.7	1:40,4.7	1:40,5.4		1:40,1.2
1:50,7.4	1:50,4.7	1:50,5.4		1:50,1.2
2:00,7.2	2:00,4.7	2:00,5.5		2:00,1.1
2:10,7	2:10,4.7	2:10,5.7		2:10,1.1
2:20,6.8	2:20,4.7	2:20,5.8		2:20,1.2
2:30,6.6	2:30,4.6	2:30,5.7		2:30,1.2
2:40,6.3	2:40,4.6	2:40,5.4		2:40,1.2
2:50,6.1	2:50,4.5	2:50,5.1		2:50,1.1
3:00,6.2	3:00,4.5	3:00,4.7		3:00,1.2
3:10,6.3	3:10,4.4	3:10,4.4		3:10,1.3
3:20,6.5	3:20,4.4	3:20,4.2		3:20,1.4
3:30,6.7	3:30,4.4	3:30,4.2		3:30,1.4
3:40,6.9	3:40,4.4	3:40,4.8		3:40,1.4
3:50,7.1	3:50,4.4	3:50,5.2		3:50,1.4
4:00,7.3	4:00,4.4	4:00,5.4		4:00,1.4

Tab. 4 Ukázka teplotních dat

4.2.4 Sluneční data

0538	1458
0536	1500
0535	1501
0533	1503
0532	1505
0530	1506
0529	1508
0527	1510
0525	1512
0524	1513
0522	1515
0520	1517
0518	1519
0517	1520
0515	1522
0513	1524
0511	1525
0509	1527
0508	1529
0506	1531
0504	1532
0502	1534
0500	1536
0458	1537
0456	1539
0454	1541
0452	1542
0450	1544

Tab. 5 Data o východu a západu slunce

Únorová data popisující východ a západ slunce jsem získal ze stránek „Astronomical Applications Department of the U.S. Naval Observatory“ - http://aa.usno.navy.mil/. Opět se jedná o 28 záznamů – řádků – ve formátu:

východ_slunce {tabulátor} západ_slunce

Pro potřeby simulace jsou tato data dostačující.

4.2.5 Soubory akcí

Na Obr. 8 vidíme dva úryvky ze souborů akcí vzniklých specifikováním časů ze zamíchané předlohy. Těchto souborů vznikne tolik, kolik chceme vytvořit dnů. Nutno však poznamenat, že vzhledem k omezenému množství teplotních a slunečních dat se hodnoty teplot a světla budou opakovat.

Číslo na prvním řádku udává den v týdnu, 1 je pondělí, 2 úterý atd. Prázdné řádky jsou přeskakovány, symbol „//“ značí komentář. Přejděme ale k syntaxi jednotlivých akcí. Algoritmus pracuje se dvěma typy akcí, které jsou již popsány v tabulkách Tab. 2 a Tab. 3:

1. elementární akce – např. „0823,topeni,0“
2. makro akce – např. „0812,pokoj->wc_koupelna->pokoj“.

```
@1    //1 = pondeli, 7 = nedele
//v pokoji:--------------------
0132,lampicka,on
0132,pokoj->wc->pokoj
0138,lampicka,off

0446,lampicka,on
0446,pokoj->wc->pokoj
0452,lampicka,off

0630,zavesy,off
0631,pokoj->wc->pokoj

0728,radio,1
0743,radio,off

0812,pokoj->wc_koupelna->pokoj

//snidane:----------------------
0822,pokoj->kuchyn

0822,zavesy,off
0822,radio,1
0823,topeni,8

0823,topeni,0
0827,radio,off

0827,kuchyn->pokoj
```

```
@2    //1 = pondeli, 7 = nedele
//v pokoji:--------------------
0133,lampicka,on
0133,pokoj->wc->pokoj
0139,lampicka,off

0436,lampicka,on
0436,pokoj->wc->pokoj
0442,lampicka,off

0637,zavesy,off
0638,pokoj->wc->pokoj

0729,radio,1
0743,radio,off

0816,pokoj->wc_koupelna->pokoj

//snidane:----------------------
0826,pokoj->kuchyn

0826,zavesy,off
0827,topeni,8
0827,radio,1

0827,topeni,0
0832,radio,off

0832,kuchyn->pokoj
```

…

Obr. 8 Ukázka výsledných souborů akcí

4.2.6 Výstupní data – log

Nyní již vidíme výsledek celého procesu. Na Obr. 9 je ukázka ze začátku výstupního souboru. První řádek obsahuje názvy sloupců. Hodnoty jsou odděleny čárkou.

Generátorem vytvořený jeden den má zhruba 500 řádek. To znamená, že rok bude mít řádově 180 000 řádek, což odpovídá zhruba 10 MB dat. I kdyby vstupů (atributů) byl například pětinásobek, stále se pohybujeme na hodnotě kolem 50 MB/rok. Tento odhad provádím z toho důvodu, že při predikci dalších akcí budeme muset všechna tato data stále uchovávat. I kdybychom však naši databázi udržovali po dobu velmi přehnaných 100 let, bude mít velikost 5 TB – a to v nekomprimované podobě.

```
cas,tv_POKOJ,radio_POKOJ,radio_KUCHYN,svetlo_KUCHYN,svetlo_WC,svetlo_KOUPELNA,svetlo_CHODBA,
svetlo_POKOJ,svetlo_POKOJLAMPICKA,okno_POKOJ,okno_KUCHYN,zavesy_POKOJ,zavesy_KUCHYN,
topeni_POKOJ, topeni_KUCHYN,dvere_WC,dvere_KOUPELNA,dvere_POKOJ,teplota_KUCHYN,teplota_WC,
teplota_KOUPELNA, teplota_CHODBA,teplota_POKOJ,teplota_VENKU,urovenOsvetleni_KUCHYN,
urovenOsvetleni_WC,urovenOsvetleni_KOUPELNA,urovenOsvetleni_CHODBA,urovenOsvetleni_POKOJ,
urovenOsvetleni_VENKU,den,poloha
00:00,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,18.79,22.97,24,16.99,18.96,9.19,0,0,0,0,0,0,1,0
00:10,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,19,23,23.85,17.05,18.8,9.1,0,0,0,0,0,0,1,0
00:20,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,18.72,22.75,24.2,16.8,19.02,8.9,0,0,0,0,0,0,1,0
00:30,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,18.8,23.05,24.12,17.05,18.57,8.69,0,0,0,0,0,0,1,0
00:40,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,18.96,22.92,23.97,16.83,18.68,8.6,0,0,0,0,0,0,1,0
00:50,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,18.91,22.75,24,16.61,18.63,8.6,0,0,0,0,0,0,1,0
01:00,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,18.76,22.87,24.12,16.81,18.71,8.4,0,0,0,0,0,0,1,0
01:10,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,18.67,23.15,24.17,16.57,18.62,8.3,0,0,0,0,0,0,1,0
01:20,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,18.98,23.05,23.9,16.5,18.53,8.19,0,0,0,0,0,0,1,0
01:30,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,18.49,23.15,23.77,16.66,18.49,7.9,0,0,0,0,0,0,1,0
01:32,0,0,0,0,0,0,0,0,1,0,0,1,1,0,0,0,0,0,18.49,23.15,23.77,16.66,18.49,7.9,0,0,0,0,0.2,0,1,0
01:32,0,0,0,0,0,0,0,0,1,0,0,1,1,0,0,0,0,1,18.49,23.15,23.77,16.66,18.49,7.9,0,0,0,0,0.2,0,1,0
01:32,0,0,0,0,0,0,0,0,1,0,0,1,1,0,0,0,0,1,18.49,23.15,23.77,16.66,18.49,7.9,0,0,0,0,0.2,0,1,1
01:32,0,0,0,0,0,0,0,0,1,0,0,1,1,0,0,0,0,0,18.49,23.15,23.77,16.66,18.49,7.9,0,0,0,0,0.2,0,1,1
01:32,0,0,0,0,0,0,0,1,0,1,0,0,1,1,0,0,0,0,0,18.49,23.15,23.77,16.66,18.49,7.9,0,0,0,1,0.2,0,1,1
01:32,0,0,0,0,0,1,0,1,0,1,0,0,1,1,0,0,0,0,0,18.49,23.15,23.77,16.66,18.49,7.9,0,1,0,1,0.2,0,1,1
01:32,0,0,0,0,0,1,0,1,0,1,0,0,1,1,0,0,1,0,0,18.49,23.15,23.77,16.66,18.49,7.9,0,1,0,1,0.2,0,1,1
01:32,0,0,0,0,0,1,0,1,0,1,0,0,1,1,0,0,1,0,0,18.49,23.15,23.77,16.66,18.49,7.9,0,1,0,1,0.2,0,1,3
```

Obr. 9 Ukázka výstupu - logu

Na následujícím Obr. 10 vidíme přehledné zobrazení logu v Excelu. Jedničky jsou u binárních hodnot vybarveny červeně.

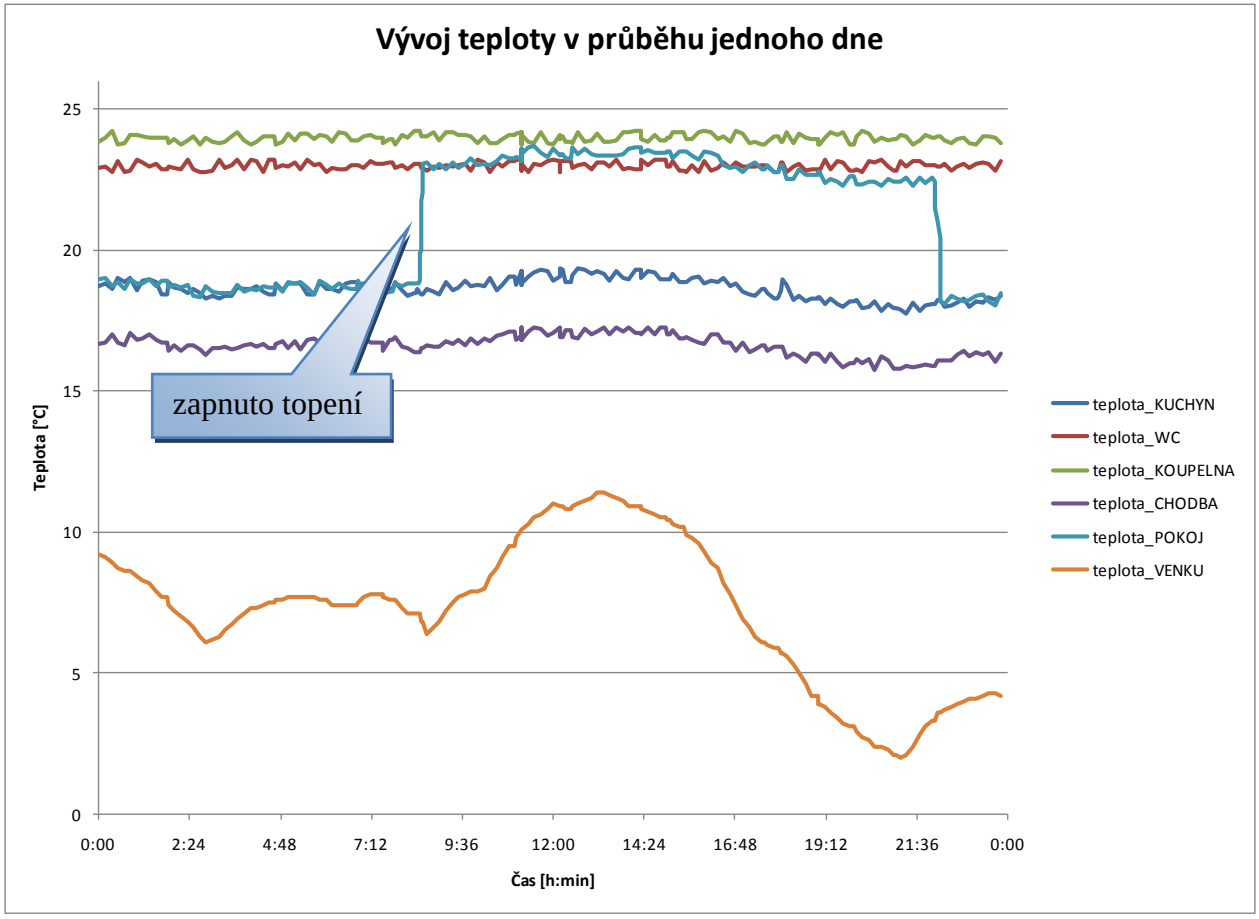
cas	poloha_KUCHYN	poloha_WC	poloha_KOUPELNA	poloha_CHODBA	poloha_POKOJ	tv_POKOJ	radio_POKOJ	radio_KUCHYN	svetlo_KUCHYN	svetlo_WC	svetlo_KOUPELNA	svetlo_CHODBA	svetlo_POKOJ	svetlo_POKOJLAMPICKA	okno_POKOJ	okno_KUCHYN	zavesy_POKOJ	zavesy_KUCHYN	topeni_POKOJ	topeni_KUCHYN	dvere_WC	dvere_KOUPELNA	dvere_POKOJ	teplota_KUCHYN	teplota_WC	teplota_KOUPELNA	teplota_CHODBA	teplota_POKOJ	teplota_VENKU	urovenOsvetleni_KUCHYN	urovenOsvetleni_WC	urovenOsvetleni_KOUPELNA	urovenOsvetleni_CHODBA	urovenOsvetleni_POKOJ	urovenOsvetleni_VENKU	den
0:00	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	19	23	24	17	19	9.2	0	0	0	0	0	0	1
0:10	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	19	23	24	17	19	9.1	0	0	0	0	0	0	1
0:20	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	19	23	24	17	19	8.9	0	0	0	0	0	0	1
0:30	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	19	23	24	17	19	8.7	0	0	0	0	0	0	1
0:40	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	19	23	24	17	19	8.6	0	0	0	0	0	0	1
0:50	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	19	23	24	17	19	8.6	0	0	0	0	0	0	1
1:00	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	19	23	24	17	19	8.4	0	0	0	0	0	0	1
1:10	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	19	23	24	17	19	8.3	0	0	0	0	0	0	1
1:20	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	19	23	24	17	19	8.2	0	0	0	0	0	0	1
1:30	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	19	23	24	17	19	7.9	0	0	0	0	0	0	1
1:40	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	18	23	24	17	19	7.7	0	0	0	0	0	0	1
1:42	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	18	23	24	17	19	7.7	0	0	0	0	0.2	0	1
1:42	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	18	23	24	17	19	7.7	0	0	0	0	0.2	0	1
1:42	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	18	23	24	17	19	7.7	0	0	0	0	0.2	0	1
1:42	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	18	23	24	17	19	7.7	0	0	0	0	0.2	0	1
1:42	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0	0	18	23	24	17	19	7.7	0	0	0	1	0.2	0	1
1:42	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	0	0	0	0	18	23	24	17	19	7.7	0	1	0	1	0.2	0	1
1:42	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	0	1	0	0	18	23	24	17	19	7.7	0	1	0	1	0.2	0	1
1:42	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	0	1	0	0	18	23	24	17	19	7.7	0	1	0	1	0.2	0	1
1:42	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	0	0	0	0	18	23	24	17	19	7.7	0	1	0	1	0.2	0	1
1:42	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	0	1	0	0	18	23	24	17	19	7.7	0	1	0	1	0.2	0	1
1:42	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	0	1	0	0	18	23	24	17	19	7.7	0	1	0	1	0.2	0	1
1:42	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	0	0	0	0	18	23	24	17	19	7.7	0	1	0	1	0.2	0	1
1:45	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0	0	18	23	24	17	19	7.7	0	0	0	1	0.2	0	1
1:45	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	18	23	24	17	19	7.7	0	0	0	0	0.2	0	1
1:45	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	18	23	24	17	19	7.7	0	0	0	0	0.2	0	1
1:45	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	18	23	24	17	19	7.7	0	0	0	0	0.2	0	1
1:45	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	18	23	24	17	19	7.7	0	0	0	0	0.2	0	1
1:48	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	18	23	24	17	19	7.7	0	0	0	0	0	0	1
1:50	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	19	23	24	16	19	7.4	0	0	0	0	0	0	1
2:00	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	19	23	24	17	19	7.2	0	0	0	0	0	0	1

Obr. 10 Ukázka zobrazení logu v Excelu

V zájmu přehlednosti je čas ponechán ve formátu {hodiny: minuty}, poloha není sloučena do jediného sloupce, teplota je ve stupních Celsia a úroveň osvětlení je v rozsahu <0,1>. Tyto veličiny jsou pak pro účely predikce kvantizovány. To by však bylo v tento okamžik málo názorné.

4.2.7 Grafické znázornění některých veličin z logu

Na Obr. 11 vidíme vývoj teploty v rámci jednoho dne. Venkovní teplota je brána přímo ze zdrojových souborů.



Obr. 11 Graf vývoje teploty během jednoho dne

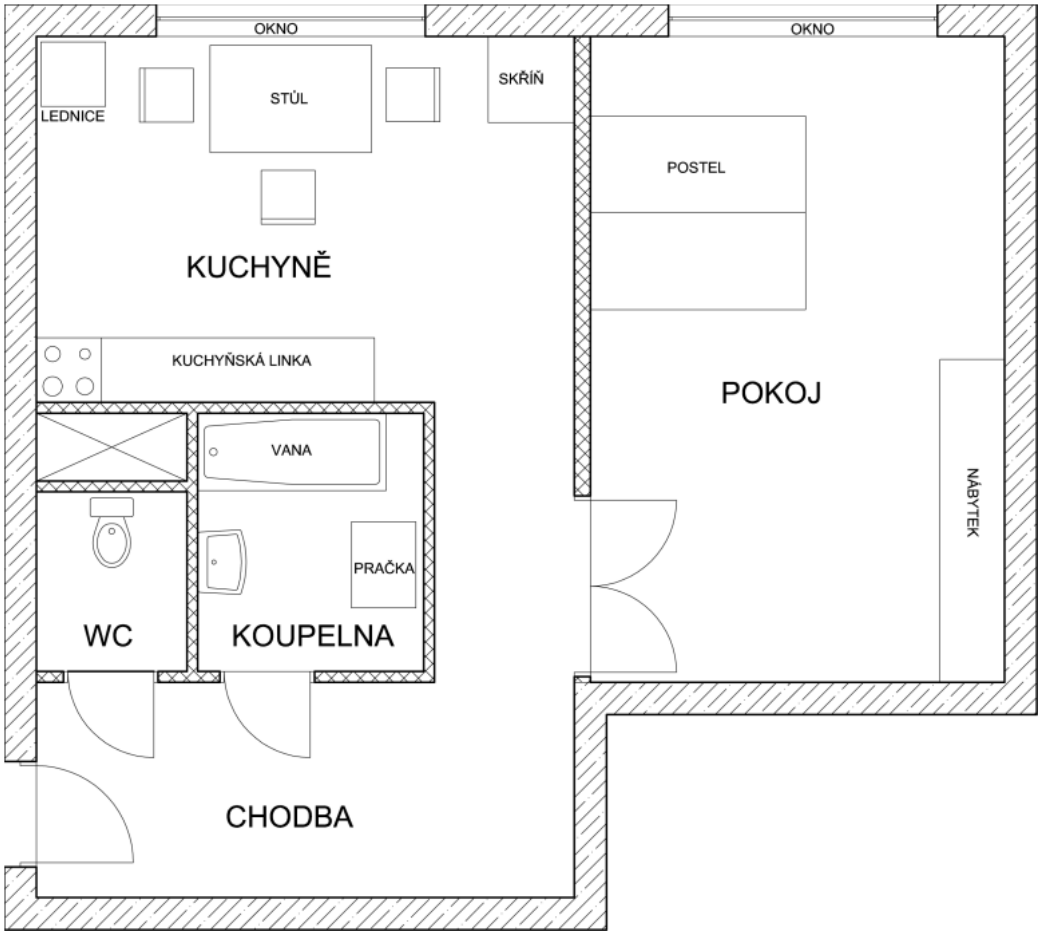
Ostatní teploty jsou počítány následujícím způsobem (viz Obr. 12):

```
public bool AddTemperatures(Time time0, Time time1)
{
    ... //ostatni teploty:
    double venku = logLine.teplota[ActivityElem.VENKU];
    const double DIFF = 4d / 30d;       //-20 venku odpovida 15 vevnitr;
                                        //+10 -> 19 vevnitr
    const double DIVISOR = 40d; //+- 0.25 deg
    logLine.teplota[ActivityElem.KUCHYN] = 15d + (venku + 20d) * DIFF +
             Rand.Next(-10, 10) / DIVISOR;
    logLine.teplota[ActivityElem.POKOJ] = 15d + (venku + 20d) * DIFF +
             Rand.Next(-10, 10) / DIVISOR;
    logLine.teplota[ActivityElem.CHODBA] = 13d + (venku + 20d) * DIFF +
             Rand.Next(-10, 10) / DIVISOR;
    logLine.teplota[ActivityElem.KOUPELNA] = 24d +
             Rand.Next(-10, 10) / DIVISOR;
    logLine.teplota[ActivityElem.WC] = 23d + Rand.Next(-10, 10) / DIVISOR; ...
}
```

Obr. 12 Výpočet teplot - část kódu

U všech teplot se konstanta upraví o venkovní teplotu a přičte se náhodný šum. S výjimkou koupelny – tam teplota na venkovní závislá není.

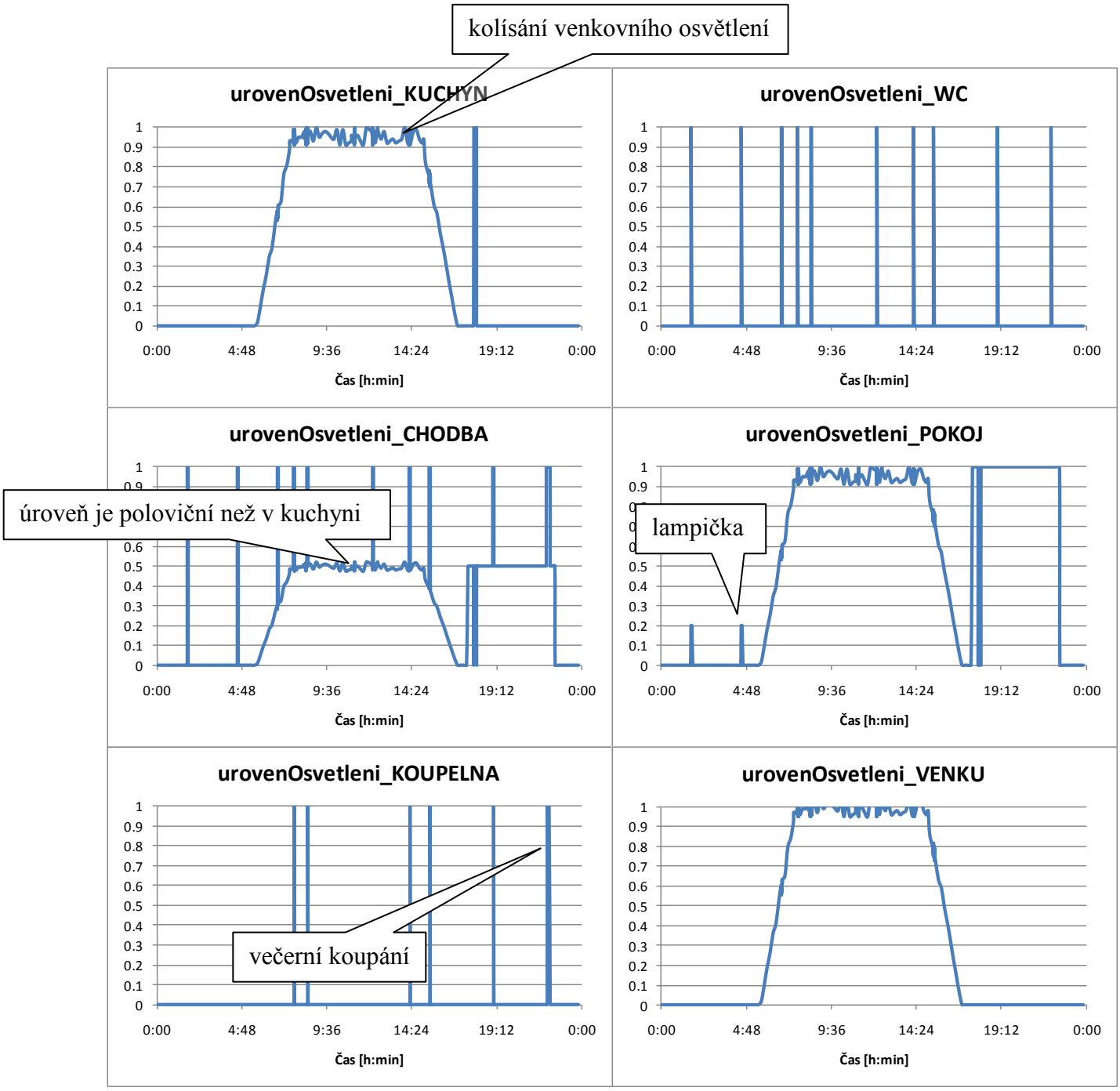
Obdobná situace nastává při výpočtu úrovní osvětlení. Zde jsou již závislosti trochu složitější a snaží se modelovat reálnou situaci, která odpovídá bytu na Obr. 13[1]. V tomto bytě bydlí moje babička, která mi také byla inspirací při tvorbě tohoto programu.



Obr. 13 Schéma modelovaného bytu

[1] Za tento nákres děkuji svému kamarádovi Ing. Karlovi Veselému.

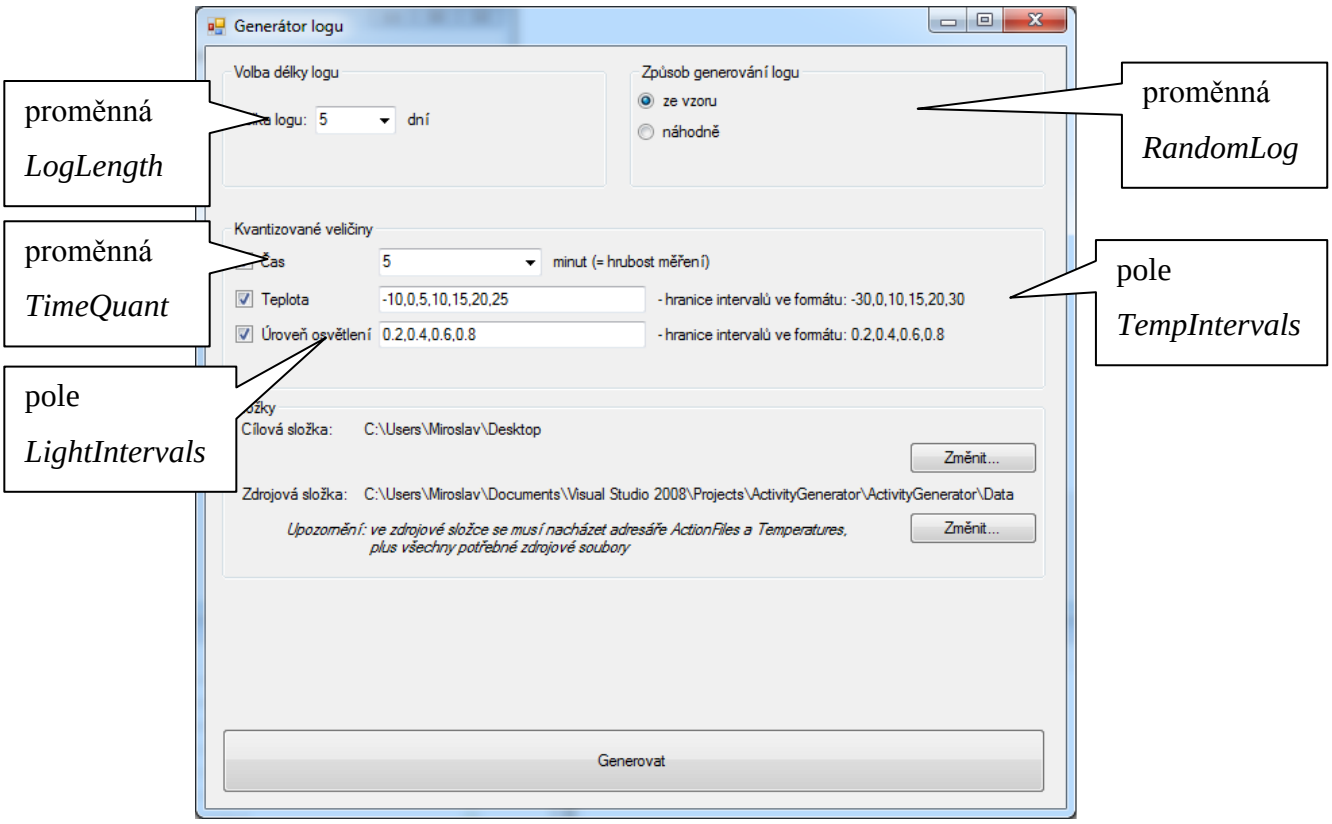
Následující grafy na Obr. 14 jsou opět vytvořené z jednoho denního logu:



Obr. 14 Grafy vývoje úrovně osvětlení během jednoho dne

4.3 Aplikace Generátor logu

Na následujícím obrázku (Obr. 15) je uživatelské rozhraní generátoru akcí. Význam jednotlivých ovládacích prvků je zřejmý a jeho ovládání je intuitivní. V popiscích uvádím názvy korespondujících proměnných, které se vyskytují v ukázce kódu.



Obr. 15 Okno programu "Generátor logu"

4.3.1 Náhodný log – kód

Na Obr. 16 je část kódu, který se stará o tvorbu náhodného logu. Jak je vidět z tohoto výpisu, po nastavení parametrů náhodného generátoru (`TimeQuant`, `TempIntervals`, `LightIntervals`) následuje cyklus generující jeden den v každé iteraci. Na závěr je kompletní log uložen do souboru „outRandLog*{LogLength}*.txt“.

```
...
if (RandomLog == true)
{
      //vytvorime objekt generator; obsahuje vsechny metody potrebne pro
      //tvorbu logu:
      Generator randG = new Generator();
      //nastavime kvantizovani jednotlivych velicin:
      randG.SetQuantization(TimeQuant, TempIntervals, LightIntervals);

      //cyklus delky dle poctu generovanych dni:
      for (int i = 0; i < LogLength; i++)
      {
            //vytvorime nahodne log jednoho dne:
            randG.CreateRandomDayLog();
            //pridame ho do celkoveho logu:
            randG.AddDayLogToCompleteLog();

            //vizualizace pokroku pri vytvareni logu:
            lStatus.Text = (i + 1).ToString() + "/" + LogLength.ToString();
            lStatus.Update();
            progressBar1.PerformStep();
      }

      //na zaver log vyexportujeme; soubor jmena typu 'outRandLog100.txt':
      randG.ExportQuantizedLogForDatabase(outputFolder + "outRandLog" +
            LogLength + ".txt");
}
else
...
```

Obr. 16 Tvorba náhodného logu – kód

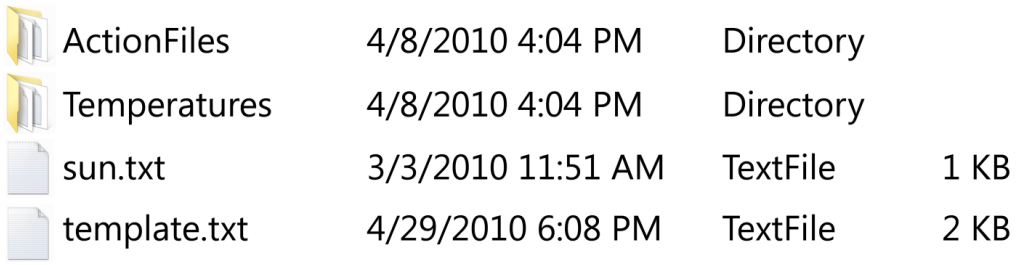
4.3.2 Log ze vzoru – kód

Kód pro vytváření logu ze vzoru, který je zobrazený na Obr. 18, přesně kopíruje schéma z Obr. 5. Nejprve se opět nastaví kvantizace – parametry (`TimeQuant`, `TempIntervals`, `LightIntervals`). Následuje cyklus, který vždy do celkového logu přidává denní log. Iterace obsahují tyto operace:

1. zamíchání vzoru,
2. vytvoření souboru akcí pro daný den,
3. načtení těchto akcí spolu s teplotami a údaji o slunci,
4. vytvoření logu jednoho dne,
5. úprava teplot a intenzit osvětlení,
6. přidání hotového denního logu do celkového záznamu,
7. vizualizace průběhu.

Na závěr je kompletní log uložen do souboru „outLog*{LogLength}*.txt“.

Následující obrázek (Obr. 17) ukazuje, jak musí vypadat struktura zdrojové složky:



Obr. 17 Struktura souborů pro tvorbu logu ze vzoru

```
...
else
{
    //vytvorime objekt generator; obsahuje vsechny potrebne metody:
    Generator g = new Generator();
    //nastavime kvantizovani jednotlivych velicin:
    g.SetQuantization(TimeQuant, TempIntervals, LightIntervals);
    //cyklus delky dle poctu generovanych dni:
    for (int i = 0; i < LogLength; i++)
    {
      //v templatu nejprve prohazime specifikovane akce dle jejich
      //pravdepodobnosti,(vysledkem je soubor 'shuffledtemplate.txt'):
      g.ShuffleTemplate(sourceFolder + "template.txt", sourceFolder +
            "shuffledtemplate.txt");
      //na zaklade tohoto zamichaneho templatu vytvorime i-ty soubor akci,
      //(vysledkem je soubor typu '10.txt'):
      g.CreateActionFileFromTemplate(sourceFolder + "shuffledtemplate.txt",
            sourceFolder + "\\ActionFiles\\" + i + ".txt", i);

      //z tohoto souboru '10.txt' napr. nacteme akce do generatoru,
      //uvnitr Generatoru se vytvori objekt Actions:
      g.LoadActions(sourceFolder + "ActionFiles\\" + i + ".txt");
      //nacteme rovnez teploty, je nutne mit presne 28 souboru typu txt a
      //daneho formatu, uvnitr Generatoru se vytvori objekt Temperatures:
      g.LoadTemperatures(sourceFolder + "Temperatures\\" + i%28 + ".txt");
      //na zaver nacteme casy vychodu a zapadu slunce, tvorba objektu Sun:
      g.LoadSun(sourceFolder + "sun.txt");

      //nyni jiz muzeme vytvorit log jednoho dne na zaklade objektu
      //Actions, teploty a urovne osvetleni nejsou nastaveny jeste:
      g.CreateDayLog();

      //na zaklade objektu Temperatures upravime teploty v pokoji:
      g.UpdateLogTemperatures(ActivityElem.POKOJ);
      //... a jeste v kuchyni:
      g.UpdateLogTemperatures(ActivityElem.KUCHYN);
      //nastavime i hodnoty urovni osvetleni:
      g.UpdateLightIntensities(g.Sun[i%28].sunrise, g.Sun[i%28].sunset);

      //pridame ho do celkoveho logu:
      g.AddDayLogToCompleteLog();

      //vizualizace pokroku pri vytvareni logu:
      lStatus.Text = (i + 1).ToString() + "/" + LogLength.ToString();
      lStatus.Update();
      progressBar1.PerformStep();
    }
    //na zaver log vyexportujeme; soubor jmena typu 'outLog100.txt':
    //(tato funkce jeste navic slucuje sloupce tykajici se polohy uzivatele
    //do jedineho, mela by byt snadno upravitelna pro slouceni i jinych
    //sloupcu)
    g.ExportQuantizedLogForDatabase(outputFolder + "outLog" + LogLength +
      ".txt", new List<int>(new int[] { 1, 2, 3, 4, 5 }), "poloha");
}
```

Obr. 18 Tvorba logu ze vzoru - kód

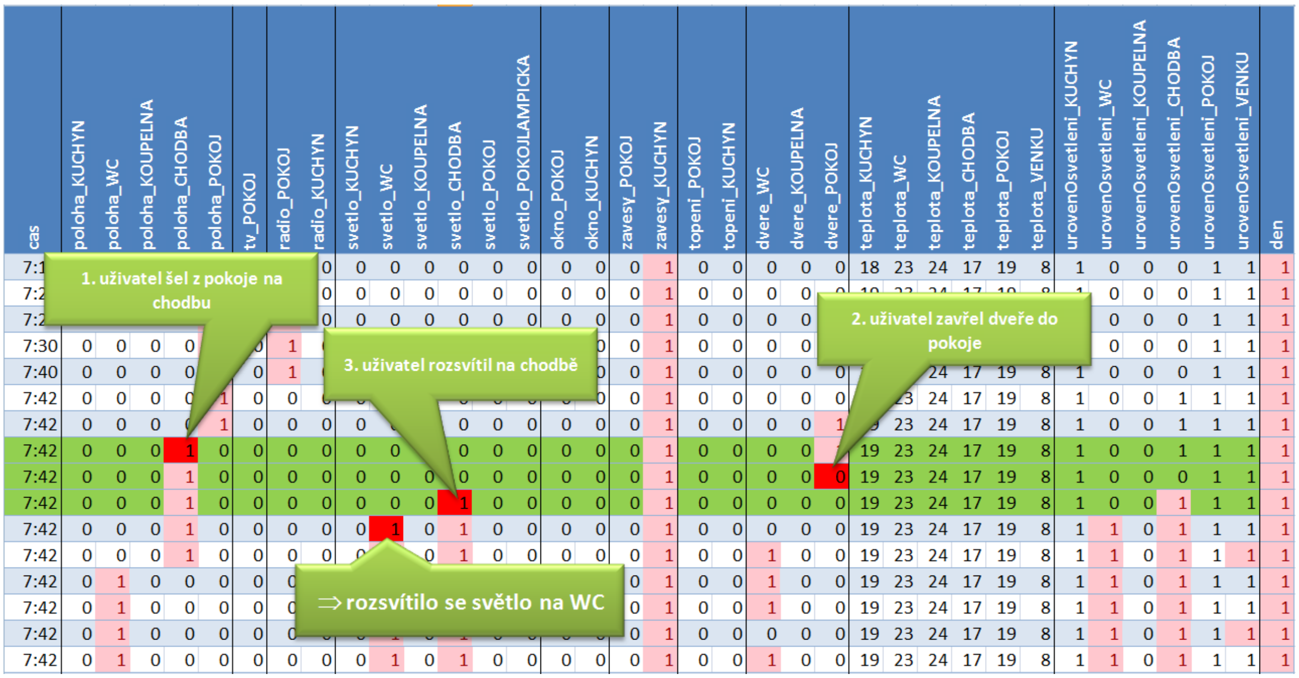
5. Predikční algoritmy

5.1 Predikce

Tématem této diplomové práce je predikce. Co to ale vlastně znamená? Co si pod tímto pojmem představit? Slovo predikce pochází z latiny (*prae*-, před, a *dicere*, říkat) a znamená předpověď nebo prognóza. Predikce nám říká, co by se mělo nebo nemělo stát v budoucnosti. Vytvořit zařízení, které předvídá budoucnost, je však poměrně netriviální úloha. I za předpokladu, že by svět kolem nás byl deterministický a my bychom uměli velmi přesně namodelovat chování všech částic, byl by tento úkol patrně stále neřešitelný – vzhledem k výpočetní náročnosti a současnému stavu výpočetní techniky.

V našem případě je úloha predikce následující. Máme souhrn atributů popisujících stav našeho inteligentního domova – např. poloha uživatele, stav dveří (otevřené, zavřené), stav oken, teplota v místnostech a tak dále. Pro jednoduchost nyní přepokládejme, že tento stav domácnosti známe v každém okamžiku. V tomto seznamu akcí se nachází i informace o tom, co uživatel udělal. Tuto informaci musíme vyextrahovat a zjistit souvislost s ostatními stavy, které tomuto předcházely.

Přestavme si následující situaci – viz Obr. 19. Vidíme například, že v čase 7:42 se rozsvítilo světlo na WC. Je evidentní, že tato událost nastala díky tomu, že uživatel zmáčkl vypínač. My bychom nyní chtěli zjistit závislost mezi akcí „světlo_WC“ a stavy, které této akci předcházely. Začněme tedy třeba tím, že se podíváme na stav, který bezprostředně předcházel. Na tomto řádku vidíme, že došlo k rozsvícení světla na chodbě a zároveň se také zvýšila intenzita osvětlení na chodbě. To je logické – uživatel si rozsvítil na chodbě. Nyní se můžeme podívat ještě hlouběji do historie – vidíme, že v ještě dřívějším kroku uživatel zavřel dveře do pokoje. Takto bychom mohli pokračovat stále dál. Můžeme se například domnívat, že uživatel zrovna jde na WC – přišel na chodbu, zavřel dveře do pokoje, rozsvítil si na chodbě, rozsvítil si na WC a v dalším kroku patrně otevře dveře na WC.



Obr. 19 Část logu – ukázka souvislostí akcí

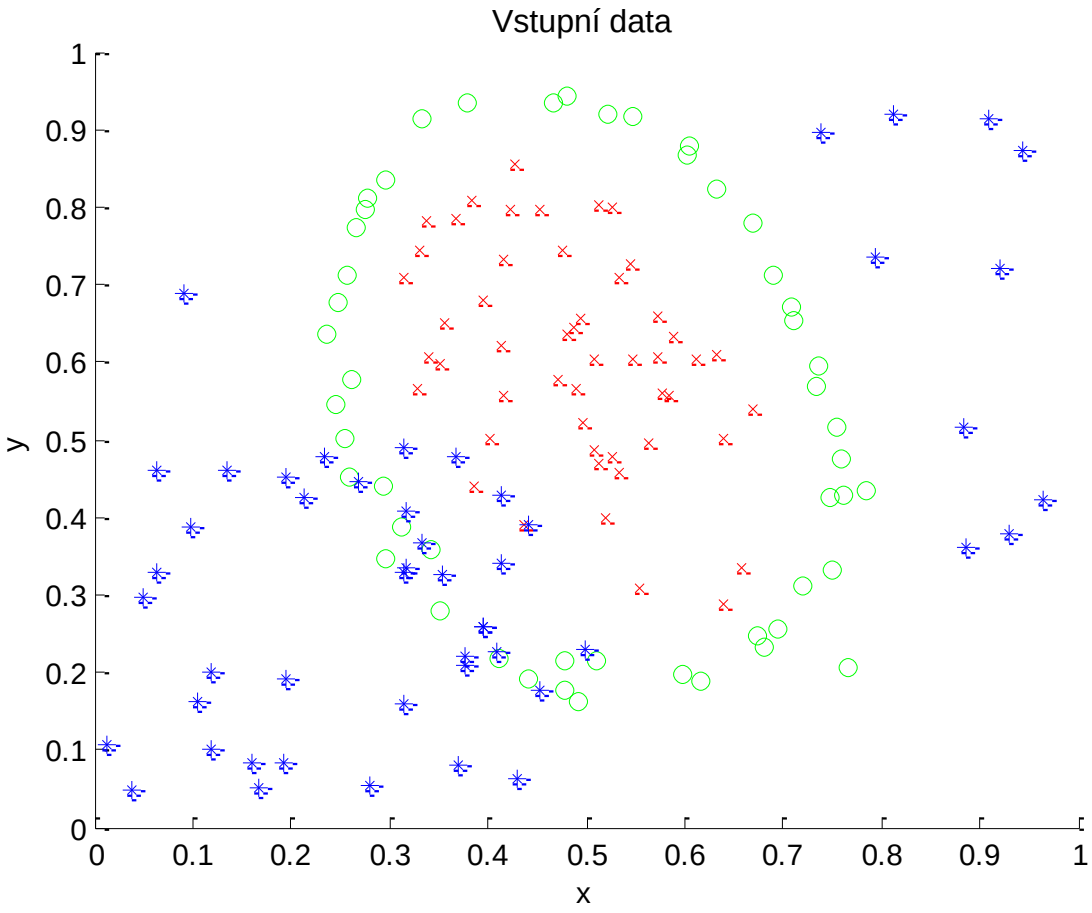
Naši úlohu predikce tak můžeme přeformulovat na úlohu klasifikace – známe stavy, které nastaly až do současnosti, a třída, do které je zařadíme, určuje akci, která bude následovat. Na našem příkladu by to znamenalo, že sekvence

$$[poloha\_CHODBA=1 \rightarrow dveře\_POKOJ=0 \rightarrow světlo\_CHODBA=1]$$

patří do třídy „světlo_WC=1“. V budoucnu bychom pak stále tuto sekvenci hledali a v případě jejího nalezení bychom predikovali, že dalším krokem bude rozsvícení světla na WC.

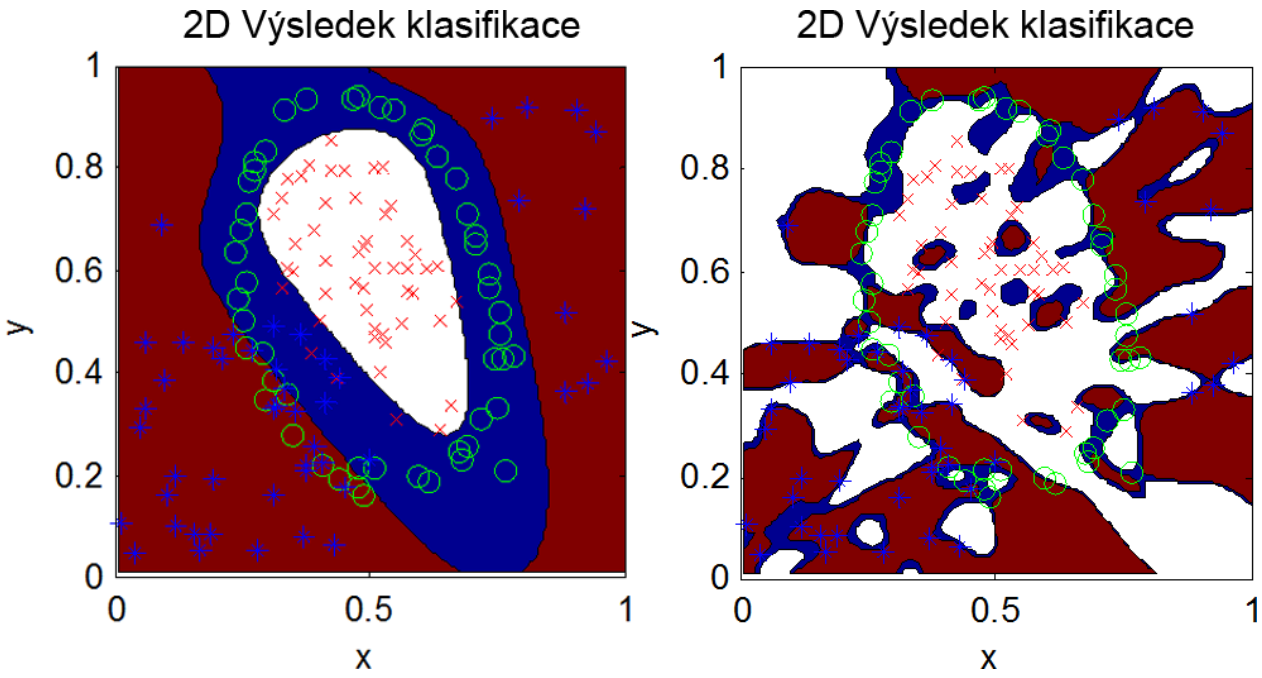
Takto jednoduché to však není. Musíme se potýkat se dvěma zásadními problémy. Za prvé – co když se někdy přihodí, že přestože naše sekvence nastala, uživatel udělá úplně něco jiného? Původně měl třeba namířeno na WC, ale pak mu zazvonil telefon a on se vrátil do pokoje. Druhou otázkou pak je, jak hluboko do historie se dívat, tedy jak dlouhé sekvence hledat. Na první pohled je zřejmé, že jeden stav historie je málo a že 100 stavů je moc – rozsvícení světla na WC dnes asi nemá souvislost se zapnutím rádia včera odpoledne.

První problém je standardní komplikací klasifikace – neseparovatelné oblasti. Pro ilustraci se podívejme na Obr. 20. Osy x a y jsou atributy (spojitý obor hodnot), barva značek je třída. Naším úkolem je rozdělit rovinu xy na oblasti, které ideálně obsahují pouze jeden typ značky. Tyto oblasti pak slouží k tomu, abychom mohli klasifikovat nově získaný bod. Měly by tedy být co nejobecnější.



Obr. 20 2D-data ke klasifikaci; převzato z (Řehoř, 2007)

Na Obr. 21 je vidět srovnání dvou různých klasifikací stejných dat. Na levém výsledku vidíme obecnější, ale méně přesnou klasifikaci, napravo je tomu naopak.



Obr. 21 Výsledky klasifikace 2D-dat; převzato z (Řehoř, 2007)

K analogickému problému bude docházet i v našem případě přiřazování minulých stavů k predikované akci. Myslím, že je intuitivně jasné, že k tomu, abychom určili budoucí akci přesně, bychom potřebovali další informace, které nemáme. Například náladu uživatele, co říká, jak se tváří. Náš systém by musel mít velmi komplexní model jak prostředí, tak uživatele. Něco takového, jako má náš mozek. A je zřejmé, že i my máme problém přesně predikovat, co náš spolubydlící udělá. Natož pak počítačový program, který má k dispozici pouze tabulku s několika desítkami atributů popisujících stav domácnosti.

Podívejme se nyní na různé metody, které můžeme pro řešení naší úlohy použít. Z nejznámějších bych jmenoval tyto:

- nejbližší soused, K-nejbližších sousedů
- rozhodovací stromy, asociační pravidla
- neuronové sítě
- shlukování (např. K-means, EM algoritmus)

Rozhodl jsem se vybrat 3 kandidáty na řešení svého problému – neuronové sítě, rozhodovací pravidla a asociační pravidla. Všechny tyto přístupy nyní stručně popíšu.

5.2 Neuronové sítě

Neuronovými sítěmi se zabývá moje bakalářská práce (Řehoř, 2007). V ní jsem používal neuronové sítě typu Back-propagation pro klasifikaci spánkových EEG záznamů. Díky tomu lze pak diagnostikovat některé choroby strojovým zpracováním hypnogramů. Zde uvádím pouze úvod a důvody, proč není vhodné neuronové sítě použít pro predikci akcí v inteligentním domově.

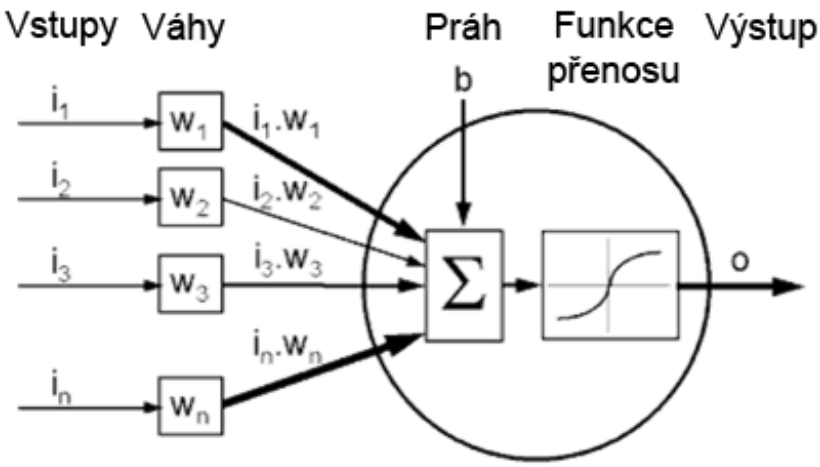
5.2.1 Úvod do neuronových sítí

Funkce neuronové sítě je inspirována činností lidského neuronu, základního stavebního kamene mozku. Každý neuron má velké množství vstupů a vždy jen jeden výstup. Na vstupy přicházejí informace od ostatních neuronů a tyto signály jsou různě zesilovány (zeslabovány) podle počtu transmiterů uvolněných při přenosu informace v rámci synapse. Na základě těchto vstupních parametrů se neuron „rozhodne“, jestli pošle signál dál, nebo ne. Toto je v podstatě celý jednoduchý princip, jak lidský mozek funguje.

Přestože podstata činnosti neuronu je známa již od počátku 20. století, inspiraci v ní nalezli až ve 40. letech pánové McCulloch a Pitts. V roce 1943 spolu publikovali první článek (A Logical Calculus of the Ideas Immanent in Nervous Activity), ve kterém popsali formální umělý neuron. Lze jej charakterizovat vztahem:

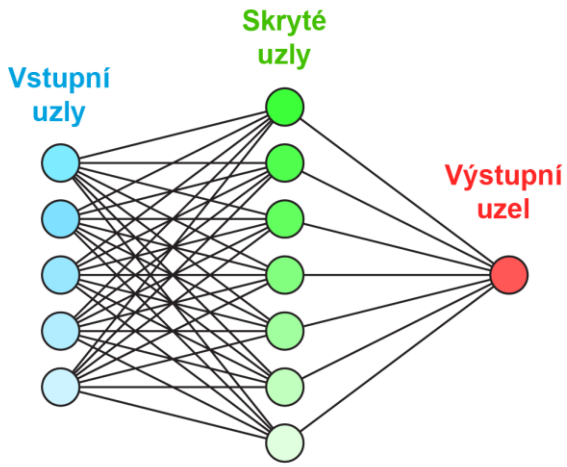
$$o = T\left(\sum_{i=1}^{N} i_i w_i + b\right), \text{ kde}$$

- y je výstup neboli aktivita neuronu *(output)*,
- i_i je i-tý vstup neuronu; vstupů je celkem N *(input)*,
- w_i představuje hodnotu i-té synoptické váhy *(weight)*,
- T je (nelineární) přenosovou funkcí *(transfer function)* a
- b představuje prahovou hodnotu *(bias)*.



Obr. 22 Schéma umělého neuronu; zdroj (Warwick, 2007)

Toto je pouze první model, který byl později vylepšen. Pro představu o činnosti neuronové sítě a její analogie s lidskou však postačuje. Rozvoj této nové oblasti zkoumání byl rychlý. Následoval Rosenblattův perceptron, sítě ADALINE, MADALINE a GMDH. Na základě kritiky Minského s Papertem došlo v letech 1969 až 1978 ke stagnaci. Domnívali se totiž, že neuronové sítě nejsou schopné simulovat funkci XOR. Tento omyl byl později napraven zjištěním, že je to možné při použití více vrstev. Od té doby byly navrženy další typy sítí – Hopfieldova síť, SOM (samoorganizující se mapy), síť typu backpropagation, RBF.



Obr. 23 Schéma jednoduché neuronové sítě

Nejznámějším zástupcem neuronových sítí jsou sítě typu **feedforward**. Jsou charakterizovány tím, že v nich nejsou žádné cykly. Tedy jak vyplývá z názvu – signál se šíří pouze jedním směrem. Jejich důležitým rysem je také to, že jejich odpověď na vstup je velmi rychlá.

Učícím algoritmem, který se často používá, je **algoritmus zpětného šíření** (back-propagation). Na vstup sítě se vždy přiloží vzor a síť vyprodukuje výstup. Ten je srovnán s referenčním a případné odchylky jsou propagovány zpět k první vrstvě a na jejich základě jsou přenastavovány váhy jednotlivých neuronů.

5.2.2 Nevýhody použití v inteligentním domě

a) Black box

- asi největší minus nasazení neuronových sítí je netransparentnost jejich fungování; jinými slovy to znamená, že přestože síť dává rozumné výsledky, vůbec si nemůžeme být jisti, jestli to není pouze náhoda a proč se chová tak, jak se chová

b) Minimalizace střední kvadratické chyby na trénovacích datech

- chyba, kterou se snažíme minimalizovat v procesu učení, se týká pouze testovacích dat, ne reálných

c) Uváznutí v lokálním minimu

- metoda učení back-propagation je jednou z tzv. gradientních metod; to znamená, že minimum chybové funkce hledá tak, že postupuje ve směru největšího poklesu (gradientu); úskalí tohoto postupu je v tom, že má tendenci uvíznout v lokálním minimu

d) Ochrnutí sítě (network paralysis)

- během učení může dojít k tomu, že váhy budou nabývat velkých hodnot; potom i vstup sigmoidy může být velké číslo a její výstup se pak bude blížit 0 nebo 1; při přenastavování pak v důsledku vztahů, které zde neuvádím (Šnorek, 2002), bude docházet k tomu, že se váhy nebudou měnit a síť „ochrne“

e) Učení nemusí skončit

- jak vyplývá z výše uvedeného – při použití gradientní metody nemusí algoritmus vždy konvergovat a minimum můžeme překročit (při volbě velkého kroku)

5.3 Rozhodovací pravidla – algoritmus pokrytí

Rozhodovací pravidla se stejně jako neuronové sítě používají ke klasifikaci. Každé pravidlo má tvar:

IF Ant THEN Class,

kde Ant je předpoklad – výrok vytvořený z kombinace atributů – a Class je třída, do které výrok zařazujeme.

Jedním z algoritmů pro tvorbu rozhodovacích pravidel je algoritmus pokrytí. Tato metoda, spíše známá jako algoritmus pokrývání množin (neboli set covering algorithm) pochází od Ryzsarda Michalskiho (Michalski, 1969), který jej vytvořil v roce 1969 pod označením algoritmus AQ.

Podstatou tohoto algoritmu je nalézt pravidla, která pokrývají nějaké příklady hledaného konceptu. Tyto příklady je pak potřeba oddělit od jiných příkladů téhož konceptu a od příkladů jiné třídy. V n-rozměrném prostoru atributů se tak hledají mnohorozměrné hranoly, které obsahují pouze prvky stejné třídy.

5.3.1 Popis algoritmu pokrytí

Základní varianta tohoto algoritmu je uvedena na Obr. 24. Takto ji prezentuje (Berka, 2003).

Algoritmus pokrývání množin

1. najdi pravidlo, které pokrývá nějaké pozitivní příklady a žádný negativní,
2. odstraň pokryté příklady z trénovací množiny D,
3. pokud v D zbývají nějaké nepokryté pozitivní příklady, vrať se k bodu 1, jinak skonči.

Obr. 24 Algoritmus pokrývání množin

Uvedený algoritmus funguje pouze pro kategoriální data, čili data, ve kterých každý atribut může nabývat více než dvou hodnot. Tomuto omezení se vyhneme tak, že takový atribut rozdělíme na příklady a protipříklady této třídy.

Dalším omezením algoritmu z Obr. 24 je, že nefunguje pro data, která jsou zatížena šumem. Abychom s takovými daty mohli pracovat, je nutné modifikovat krok 1. Nebudeme vyžadovat, aby pravidlo pokrývalo příklady pouze jedné třídy, nýbrž i více tříd.

5.3.2 Příklad použití algoritmu pokrytí

Při tvorbě pravidel máme dvě možnosti – metodu generalizace (pravidla postupně zobecňujeme) a metodu specializace (do zprvu jednoduchých pravidel přidáváme další kategorie).

Ukažme si nyní, jak funguje postup první – „zdola nahoru“. Opět budu citovat z (Berka, 2003):

„Krok 1 z algoritmu na Obr. 24 bude mít podobu:

1. vezmi jeden pozitivní příklad jako jádro (seed),
2. najdi jeho generalizaci, která pokrývá nějaké pozitivní příklad a žádný negativní.“

Aplikujme nyní tento postup na datech z Tab. 6, jejichž zdrojem jsou přednášky z předmětu Umělá Inteligence 1.

id	Hlava	Úsměv	Ozdoba krku	Tělo	Předmět	Přátelský
k1	kruh	ne	kravata	čtverec	šavle	ne
k2	čtverec	ano	motýlek	čtverec	nic	ano
k3	kruh	ne	motýlek	kruh	šavle	ano
k4	trojúhelník	ne	kravata	čtverec	balón	ne
k5	kruh	ano	nic	trojúhelník	květina	ne
k6	trojúhelník	ne	nic	trojúhelník	balon	ano
k7	trojúhelník	ano	kravata	kruh	nic	ne
k8	kruh	ano	kravata	kruh	nic	ano

Tab. 6 Ukázková data pro tvorbu rozhodovacích pravidel

- Začneme řádkem 2:

  Hlava(čtverec) ∧ Úsměv(ano) ∧ Ozdoba_krku(motýlek) ∧ Tělo (čtverec) ∧ Předmět(nic)

  ...pokrývá **(k2)** a žádný negativní příklad

- Použijeme řádek 3 a zobecníme toto pravidlo:

  Ozdoba_krku(motýlek) …pokrývá **(k2, k3)** a opět žádný negativní příklad

- Oba příklady odstraníme z trénovací množiny. Obdobně postupujeme dále. Získáme tato pravidla a skončíme:

  Hlava(trojúhelník) $\wedge$ *Úsměv(ne)* $\wedge$ *Ozdoba_krku(nic)* $\wedge$ *Tělo (trojúhelník)* $\wedge$ *Předmět(balon)* **(k6)**

  Hlava(kruh) $\wedge$ *Úsměv(ano)* $\wedge$ *Ozdoba_krku(kravata)* $\wedge$ *Tělo (kruh)* $\wedge$ *Předmět(nic)* **(k8)**

5.4 Asociační pravidla

Termín „asociační pravidla“ je používán zhruba od poloviny 90. let 20. století, kdy byl zpopularizován Rakeshem Agrawalem a jeho kolektivem. Samotným konceptem asociačních pravidel se však zabývala skupina kolem českého vědce Petra Hájka zhruba o 30 let dříve, kolem roku 1966. Ti svou metodu nazývali GUHA, což znamená General Unary Hypotheses Automaton.

Agrawal a kol. se ve své práci z roku 1993 (Agrawal, Imieliski, & Swami, 1993) zabývali tzv. analýzou nákupního košíku – známou ekonomickou úlohou. V ní jde o to najít souvislosti mezi produkty prodávanými např. v supermarketu. Na jejich základě pak management podniku rozhoduje jaké zboží dát do slevy, jak vytvořit slevové kupony, jak uspořádat produkty v regálech atp.

Zajímá nás tedy kupříkladu, že zákazníci, kteří si koupili chleba a máslo, si koupili také sýr. Máme tabulku všech nákupů a hledáme souvislosti (asociace) mezi jednotlivými položkami – vytváříme asociační pravidla.

5.4.1 Základní pojmy

Obecně můžeme asociační pravidlo zapsat v následujícím formátu:

$$Ant \Rightarrow Suc,$$

kde Ant znamená Antecedent (= předchůdce), tedy levá strana pravidla, a Suc znamená Sukcedent (= závěr), tedy pravá strana. Tato symbolika je převzata z (Berka, 2003), Agrawal ve své práci hovoří o Consequentu namísto Suc (Agrawal, Imieliski, & Swami, 1993).

Základní nástroj pro určování charakteristik pravidel je pak následující kontingenční tabulka:

	Suc	**¬Suc**	**Σ**
Ant	a	b	r = a + b
¬Ant	c	d	s = c + d
Σ	k = a + c	l = b + d	n

Hodnota **a** tedy vyjadřuje počet objektů (řádků databáze), kde platí jak Ant, tak Suc:

$a = n\ (Ant \wedge Suc)$

Hodnota **b** je počet objektů, kde platí jak Ant, ale neplatí Suc:

$b = n\ (Ant \wedge \neg Suc)$

Hodnota **c** je počet objektů, kde neplatí Ant, ale platí Suc:

$c = n\ (\neg Ant \wedge Suc)$

Hodnota **d** je počet objektů, kde neplatí ani Ant, ani Suc:

$d = n\ (\neg Ant \wedge \neg Suc)$

Dle Agrawala jsou základními charakteristikami asociačních pravidel 2 veličiny – podpora (support) a spolehlivost (confidence) (Agrawal, Imieliski, & Swami, 1993).

Podpora je počet objektů splňující předpoklad i závěr. Může být buď absolutní:

$$\text{Support} = a$$

nebo relativní:

$$\text{Support} = P(Ant \wedge Suc) = \frac{a}{a + b + c + d}$$

Spolehlivost je podmíněná pravděpodobnost závěru, pokud platí předpoklad:

$$\text{Confidence} = P(Ant|Suc) = \frac{a}{a + b}$$

Z dalších charakteristik, které uvádí Berka, bych zmínil alespoň následující:

- absolutní, popř. relativní, počet objektů, které splňují předpoklad:

$$a + b, \text{resp. } P(\text{Ant}) = \frac{a + b}{a + b + c + d}$$

- absolutní, popř. relativní, počet objektů, které splňují závěr:

$$a + c, \text{resp. } P(\text{Suc}) = \frac{a + c}{a + b + c + d}$$

- pokrytí (coverage), tj. podmíněná pravděpodobnost předpokladu pokud platí závěr:

$$P(\text{Suc}|\text{Ant}) = \frac{a}{a + c}$$

- kvalita, tj. vážený součet spolehlivosti a pokrytí:

kde w_1 a w_2 se obvykle volí tak, aby $w_1 + w_2 = 1$, tedy například $w_1 = 0{,}5$ a $w_2 = 0{,}5$ nebo $w_1 = 0{,}8$ a $w_2 = 0{,}2$.

5.4.2 Tvorba pravidel

Nejnaivnější postup pro vytváření asociačních pravidel by byl následující:

1. vytvoř všechny možné kombinace (o délce menší nebo rovno počtu objektů),
2. vytvoř všechny varianty rozdělení každé kombinaci kombinace na Ant a Suc (tedy možná pravidla),
3. vypočti podporu všech těchto pravidel a zahoď ta, která mají podporu moc nízkou.

Již na první pohled je zřejmé, že tento přístup by jistě fungoval, ale jeho výpočetní náročnost je značná. Uvažujme pouhých 10 objektů, tedy například 10 různých druhů zboží.

1. vytvoříme všechny kombinace:

$$\text{počet kombinací} = \binom{10}{1} + \binom{10}{2} + \binom{10}{3} + \ldots + \binom{10}{9} + \binom{10}{10} =$$
$$= 10 + 45 + 120 + \cdots + 10 + 1 = 1023$$

2. každou kombinaci rozdělíme na **Ant** a **Suc**:
   a. **pro** $\binom{\mathbf{10}}{\mathbf{1}}$; máme 10 prvků {1,2,3,4,5,6,7,8,9,10}, ty budeme rozdělovat:

b.

Ant	Suc
1	2,3,4,5,6,7,8,9,10
2	1,3,4,5,6,7,8,9,10
3	1,2,4,5,6,7,8,9,10
…	
10	1,2,3,4,5,6,7,8,9
1,2	3,4,5,6,7,8,9,10
1,3	1,2,4,5,6,7,8,9,10
…	
9,10	1,2,3,4,5,6,7,8
…	
1,2,3,4,5,6,7,8,10	9
1,2,3,4,5,6,7,8,9	10

Tab. 7 Kombinace Ant a Suc

Je vidět, že se jedná opět o výpočet kombinací bez opakování:

$$\binom{10}{1} + \binom{10}{2} + \binom{10}{3} + \ \ldots + \binom{10}{9} + \binom{10}{10} =$$
$$= 10 + 45 + 120 + \cdots + 10 + 1 = 1023$$

c. **pro** $\binom{10}{2}$; máme 45 prvků {[1,2], [1,3], [1,4]… [1,10], [2,3], [2,4]…[2,10]… …[9,10]}, ty budeme opět rozdělovat. Také jde o výpočet všech podkombinací bez opakování:

$$\binom{45}{1} + \binom{45}{2} + \binom{45}{3} + \ \ldots + \binom{45}{44} + \binom{45}{45} =$$
$$= 45 + 990 + 14190 + \cdots + 45 + 1 = 3.51844 \cdot 10^{13}$$

d. **a tak dále...**

Souhrn počtu kombinací z bodu 2 by byl následující:

Počet kombinací	Počet variant rozdělení na Ant a Suc
$\binom{10}{1} = 10$	1023
$\binom{10}{2} = 45$	$3.51844 \cdot 10^{13}$
$\binom{10}{3} = 120$	$1.32923 \cdot 10^{36}$
$\binom{10}{4} = 210$	$1.6455 \cdot 10^{63}$
$\binom{10}{5} = 252$	$7.23701 \cdot 10^{75}$
$\binom{10}{6} = 210$	$1.6455 \cdot 10^{63}$
$\binom{10}{7} = 120$	$1.32923 \cdot 10^{36}$
$\binom{10}{8} = 45$	$3.51844 \cdot 10^{13}$
$\binom{10}{9} = 10$	1023
$\binom{10}{10} = 1$	1
Součet:	$7.23701 \cdot 10^{75}$

Tab. 8 Počty kombinací pro 10 prvků

Je vidět, že počet možných pravidel roste velmi rychle a tento primitivní přístup není zvládnutelný. Pro n = 11 je počet pravidel v řádu 10^{137}, pro n = 12 již dokonce 10^{276}, pro vyšší n již číslo překračuje pracovní rozsah tabulky v Excelu.

Jedním z řešení této kombinatorické exploze je algoritmus apriori, navržený R. Agrawalem a jeho kolektivem v roce 1996 (Agrawal, Mannila, Srikant, Toivonen, & Verkamo, 1996).

5.4.3 Algoritmus Apriori

Algoritmus apriori vychází z toho, že pokud nějaká kombinace nesplňuje požadovanou úroveň podpory, nemusíme ji již uvažovat při hledání nadkombinací. Jinými slovy – pokud víme, že ze 100 nákupů v naší databázi byla majonéza koupena pouze jednou (její podpora je tedy 0,01) a naše minimální požadovaná podpora je 0,5 (minsup = 0,5), vyřadíme majonézu ze hry. Nebude pak již zkoumat podporu pravidel jako

(majonéza ∧ rohlík), (majonéza ∧ chleba), (majonéza ∧ párek) apod.

To znamená, že majonéza nebude obsažena v žádné kombinaci délky 2. Z toho dále vyplývá, že nebude ani v žádných dalších kombinacích.

Obdobným způsobem, pokud například kombinace *(hořčice ∧ máslo)* bude mít pouze podporu 0,1, zatímco my vyžadujeme minsup = 0,5, prohlásíme tuto kombinaci za nevyhovující. Libovolná další pravidla, která by v sobě obsahovala jak hořčici, tak máslo, budou rovněž přeskočena.

Díky tomuto přístupu zabráníme kombinatorické explozi a vytvoříme asociační pravidla v rozumném čase. Rozdělování na Ant a Suc si však již neulehčíme, a musíme proto vytvořit všechny kombinace.

Následuje pseudokód shrnující celý apriori algoritmus tak, jak jej uvádí Berka (Berka, 2003):

Algoritmus Apriori

1. do L_1 přiřaď všechny hodnoty atributů, které dosahují alespoň požadované četnosti
2. polož k = 2
3. dokud $L_{k-1} \neq \emptyset$
   3.1. pomocí funkce *apriori-gen* vygeneruj na základě L_{k-1} množinu kandidátů C_k
   3.2. do L_k zařaď ty kombinace z C_k, které dosáhly alespoň požadovanou četnost
   3.3. zvětši počítadlo k

Funkce apriori-gen (L_{k-1})

1. pro všechny dvojce kombinací p, q z L_{k-1}
   1.1. pokud p a q se shodují v prvních k-2 položkách přidej do C_k sjednocení $p \cup q$
2. pro každou kombinaci c z C_k
   2.1. Pokud některá z jejich podkombinací délky k-1 není obsažena v L_{k-1} odstraň c z C_k

Tab. 9 Algoritmus Apriori

5.4.4 Přizpůsobení algoritmu Apriori pro naši úlohu

Používat algoritmus Apriori v jeho čisté podobě by nebylo ideální. Naše atributy totiž nejsou úplně ekvivalentní. Hledat souvislosti mezi všemi by tedy nedávalo smysl. Respektive bylo by to dosti časově náročné a zbytečné. Například najít pravidlo typu:

$$okno\_POKOJ=zavřené \wedge topení\_POKOJ=zapnuté \Rightarrow teplota\_VENKU<0$$

by nám asi moc informací nepřineslo. Asi vždy bude platit, že když máme zapnuté topení, tak venku mrzne. Hlavně ale je nám toto pravidlo k ničemu z důvodu, že venkovní teplotu stejně neovlivníme. Atributy proto musíme rozdělit na ty, co můžeme změnit, resp. uživatel může změnit, a na ty, co ne. A pouze ty „změnitelné“, říkejme jim třeba akční, budeme používat pro konstrukci pravé strany pravidel, tedy Suc.

Další věcí, se kterou se musíme vyrovnat, je **hloubka historie**, kterou budeme používat pro vytváření asociačních pravidel. Jak jsem již psal na začátku této kapitoly o predikci – jednou hranicí intervalu, ze kterého můžeme vybírat, je pouze jedna, poslední akce; druhou je pak historie

celá. Obojí má však své nevýhody – přeučenost versus nedoučenost. Optimum bude ležet někde mezi těmito extrémy a bude záviset na řadě dopředu neodhadnutelných parametrů. Bude tedy nutné s tímto parametrem experimentovat a vybrat jeho vhodnou velikost.

Kvantizace veličin je další komplikací, která bude ovlivňovat kvalitu naší predikce. Některé vstupy můžeme nechat rovnou v jejich hrubé formě a ihned je používat pro tvorbu pravidel. Takovým vstupem bude například poloha uživatele. Existuje však skupina atributů, které nemůžeme ponechat v jejich syrové formě a musíme je upravit, kvantizovat. Příkladem tohoto vstupu může být třeba teplota. Jelikož teplotu budeme získávat s poměrně velkou přesností, řekněme v řádech desetin stupně Celsia, musíme její hodnotu nějak rozumně zaokrouhlit, resp. zařadit do nějakého intervalu. V opačném případě bychom dostávali přeučená pravidla, která by měla velice malou vypovídací hodnotu. Dalším příkladem atributů charakterově podobných teplotě může být čas nebo úroveň osvětlení. Jak konkrétně ale veličiny kvantizovat je opět těžké obecně říci. Bude záležet na celé řadě okolností a bude potřeba vyzkoušet různé varianty nastavení.

Důležité je také rozhodnutí, u kterých atributů používat jejich absolutní hodnotu a u kterých **diferenci** – neboli diskrétní podobu derivace. K této úpravě atributů vede zejména výpočetní náročnost. Pokud bychom nechali všechny atributy, aby vyjadřovaly přímo svůj stav, měli bychom v našich datech zbytečně moc redundantních informací. Tuto situaci ukazuje Obr. 25. Například stav závěsů se změní pouze dvakrát za den – ráno a večer – při roztažení a zatažení. Pokud ale v naší databázi necháme přímo stav závěsy_KUCHYŇ, algoritmus bude vytvářet obrovské množství pravidel obsahující stav závěsy_KUCHYŇ=1. Relevantní informace ale je pouze to, že ráno se závěsy roztáhnou a večer zatáhnou. Pamatujme si tedy pouze tuto věc. Při jejich roztažení zapišme do tabulky hodnotu +1, při zatažení -1, a když se nic neděje, tak hodnotu 0. To v podstatě znamená totéž jako „zderivovat“ první tabulku; získáme tím novou tabulku zobrazenou na Obr. 26. Změny jsou vyznačeny červeně. Udržování se ve stavu původně označeném jako aktivní je vybarveno žlutě.

Při nasazení našeho predikčního systému pak opět bude uživatel (ve smyslu navrhovatel systému inteligentního domova) muset vyzkoušet přínos různých nastavení – které atributy diferencovat a které ne.

cas	poloha_KUCHYN	poloha_WC	poloha_KOUPELNA	poloha_CHODBA	poloha_POKOJ	tv_POKOJ	radio_POKOJ	radio_KUCHYN	svetlo_KUCHYN	svetlo_WC	svetlo_KOUPELNA	svetlo_CHODBA	svetlo_POKOJ	svetlo_POKOJLAMPICKA	okno_POKOJ	okno_KUCHYN	zavesy_POKOJ	zavesy_KUCHYN	topeni_POKOJ	topeni_KUCHYN	dvere_WC	dvere_KOUPELNA	dvere_POKOJ	teplota_KUCHYN	teplota_WC	teplota_KOUPELNA	teplota_CHODBA	teplota_POKOJ	teplota_VENKU	urovenOsvetleni_KUCHYN	urovenOsvetleni_WC	urovenOsvetleni_KOUPELNA
7:10	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	18	23	24	17	19	7.8	0.8	0	0
7:20	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	19	23	24	17	19	7.8	0.8	0	0
7:29	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	19	23	24	17	19	7.8	0.9	0	0
7:30	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	19	23	24	16	18	7.7	0.9	0	0
7:40	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	19	23	24	17	19	7.6	0.9	0	0
7:42	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	19	23	24	17	19	7.6	1	0	0
7:42	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	19	23	24	17	19	7.6	1	0	0
7:42	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	19	23	24	17	19	7.6	1	0	0
7:42	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	19	23	24	17	19	7.6	0.9	0	0
7:42	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	19	23	24	17	19	7.6	0.9	0	0
7:42	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	19	23	24	17	19	7.6	1	1	0
7:42	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	19	23	24	17	19	7.6	1	1	0
7:42	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	19	23	24	17	19	7.6	1	1	0
7:42	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	19	23	24	17	19	7.6	1	1	0
7:42	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	19	23	24	17	19	7.6	1	1	0
7:42	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	19	23	24	17	19	7.6	1	1	0

Obr. 25 Log obsahující absolutní hodnoty atributů

rádio ponecháme v absolutní podobě

cas	poloha_KUCHYN	poloha_WC	poloha_KOUPELNA	poloha_CHODBA	poloha_POKOJ	tv_POKOJ	radio_POKOJ	radio_KUCHYN	svetlo_KUCHYN	svetlo_WC	svetlo_KOUPELNA	svetlo_CHODBA	svetlo_POKOJ	svetlo_POKOJLAMPICKA	okno_POKOJ	okno_KUCHYN	zavesy_POKOJ	zavesy_KUCHYN	topeni_POKOJ	topeni_KUCHYN	dvere_WC	dvere_KOUPELNA	dvere_POKOJ	teplota_KUCHYN	teplota_WC	teplota_KOUPELNA	teplota_CHODBA	teplota_POKOJ	teplota_VENKU	urovenOsvetleni_KUCHYN	urovenOsvetleni_WC	urovenOsvetleni_KOUPELNA
7:10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	23	24	17	19	7.8	0.8	0	0
7:20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	23	24	17	19	7.8	0.8	0	0
7:29	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	23	24	17	19	7.8	0.9	0	0
7:30	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	23	24	16	18	7.7	0.9	0	0
7:40	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	23	24	17	19	7.6	0.9	0	0
7:42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	23	24	17	19	7.6	1	0	0
7:42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	19	23	24	17	19	7.6	1	0	0
7:42	0	0	0	1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	23	24	17	19	7.6	1	0	0
7:42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	19	23	24	17	19	7.6	0.9	0	0
7:42	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	19	23	24	17	19	7.6	0.9	0	0
7:42	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	19	23	24	17	19	7.6	1	1	0
7:42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	19	23	24	17	19	7.6	1	1	0
7:42	0	1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	23	24	17	19	7.6	1	1	0
7:42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19	23	24	17	19	7.6	1	1	0
7:42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	19	23	24	17	19	7.6	1	1	0
7:42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	19	23	24	17	19	7.6	1	1	0

Obr. 26 Log obsahující diferencované hodnoty atributů

5.4.5 Příklad vytváření pravidla

Ukažme nyní, jak probíhá vytváření jednoho asociačního pravidla v našem případě. Předpokládejme, že veličiny jsou již kvantizované a vhodné atributy jsou také diferencované.

1. Zvol minimální hodnotu podpory **minsup**, kterou budou mít všechna výsledná pravidla.
2. Zvol pravou stranu pravidla. Např. „světlo_WC=1“ (na WC se rozsvítilo světlo).
3. Zvol hloubku historie – **h**. Toto číslo udává počet řádků v databázi, ve kterých budeme zkoumat souvislosti atributů.
4. Prohledej celou databázi a vyber řádky, které obsahují hledanou pravou stranu pravidla, resp. akci. V našem příkladu to znamená najít řádky, ve kterých je hodnota „světlo_WC“ rovna jedné.
5. Pro každý z vybraných řádků **r** proveď následující (podrobněji rozebráno v kapitole 6 – implementace a na Obr. 30):
    - Vyber z databáze **h**-řádků, které řádku **r** předchází.
    - Vytvoř z těchto řádků jeden nový, „dlouhý“ řádek.
    - Tento nový řádek vlož do nové tabulky. (V této tabulce budeme počítat podporu pravidel.)
6. V nově vytvořené tabulce aplikuj algoritmus apriori.
    - Algoritmus apriori nejprve zkoumá podporu jednotlivých prvků, pak dvojic, pak trojic atd. Stále si ale pamatuje, které atributy jsou tzv. „na živu“; tedy které používáme ke konstrukci n-tic.
    - Při vytváření n-tic délky 1 si zapamatujeme atomy, které mají podporu rovnu jedné. Ty pak přidáme do pravidel až na závěr. Kdybychom toto neudělali, velmi by se zvětšil počet zkoumaných kombinací a program by se často mohl zasekávat.

6. Implementace

Pro implementaci algoritmu jsem zvolil prostředí C# Microsoft Visual Studio 2008 Professional. Důvodem k tomuto rozhodnutí bylo zejména to, že nadřazený systém inteligentního domova, do kterého bude můj modul zakomponován, je rovněž programován v jazyce C# na platformě .NET. Jako databázový server jsem použil Microsoft SqlServer 2005, který je součástí VS 2008. Licenci k tomuto softwarovému balíku jsem získal v rámci MSDN Academic Alliance.

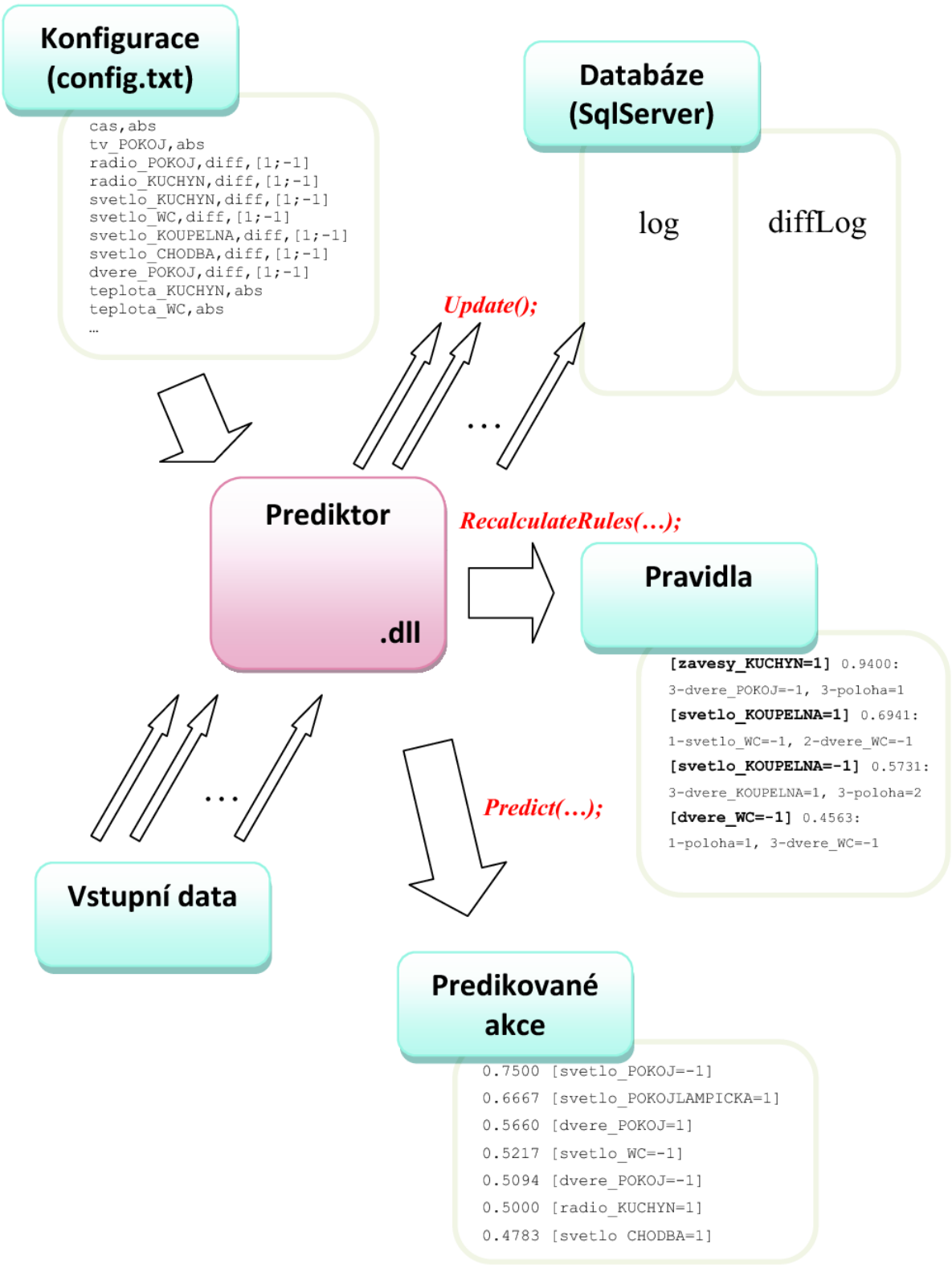
V této kapitole nejprve popíšu funkci programu jako celku, budu pokračovat strukturou jednotlivých tříd a konfiguračního souboru a na závěr ukážu některé klíčové metody, jejichž funkci popíšu.

6.1 Schéma programu

Schéma funkce celého programu je zobrazeno na Obr. 27. Program je koncipován jako dynamická knihovna, která bude posléze využita v systému inteligentního domova. Jeho nastavení se upraví prostřednictvím textového souboru *„config.txt“*. V něm jsou obsaženy názvy atributů a jejich typy. Podrobněji se tomuto souboru věnuji v kapitole 6.3. Vstupní data jsou ukládána do databáze prostřednictvím SQL dotazů, o to se stará funkce **Update()**. Pro jistotu uchovávám v databázi 2 tabulky – **log** (data v té formě, jak byla zaznamenána systémem) a **diffLog** (diferencovaná data; na základě konfiguračního souboru).

Nejvýznamnější jsou pak funkce:

- `void RecalculateRules(int historyDepth, double minSupport)` – na základě žádané délky historie a minimální podpory vytvoří sadu asociačních pravidel pro všechny kombinace akčních veličin,
- `List<Rule> Predict(int historyDepth)` – s použitím již vytvořených pravidel vrátí seznam aplikovatelných pravidel v okamžiku svého zavolání, tedy s použitím posledních řádků v databázi; `historyDepth` určuje jejich počet.



Obr. 27 Schéma funkce celého programu

6.2 Hlavní třídy

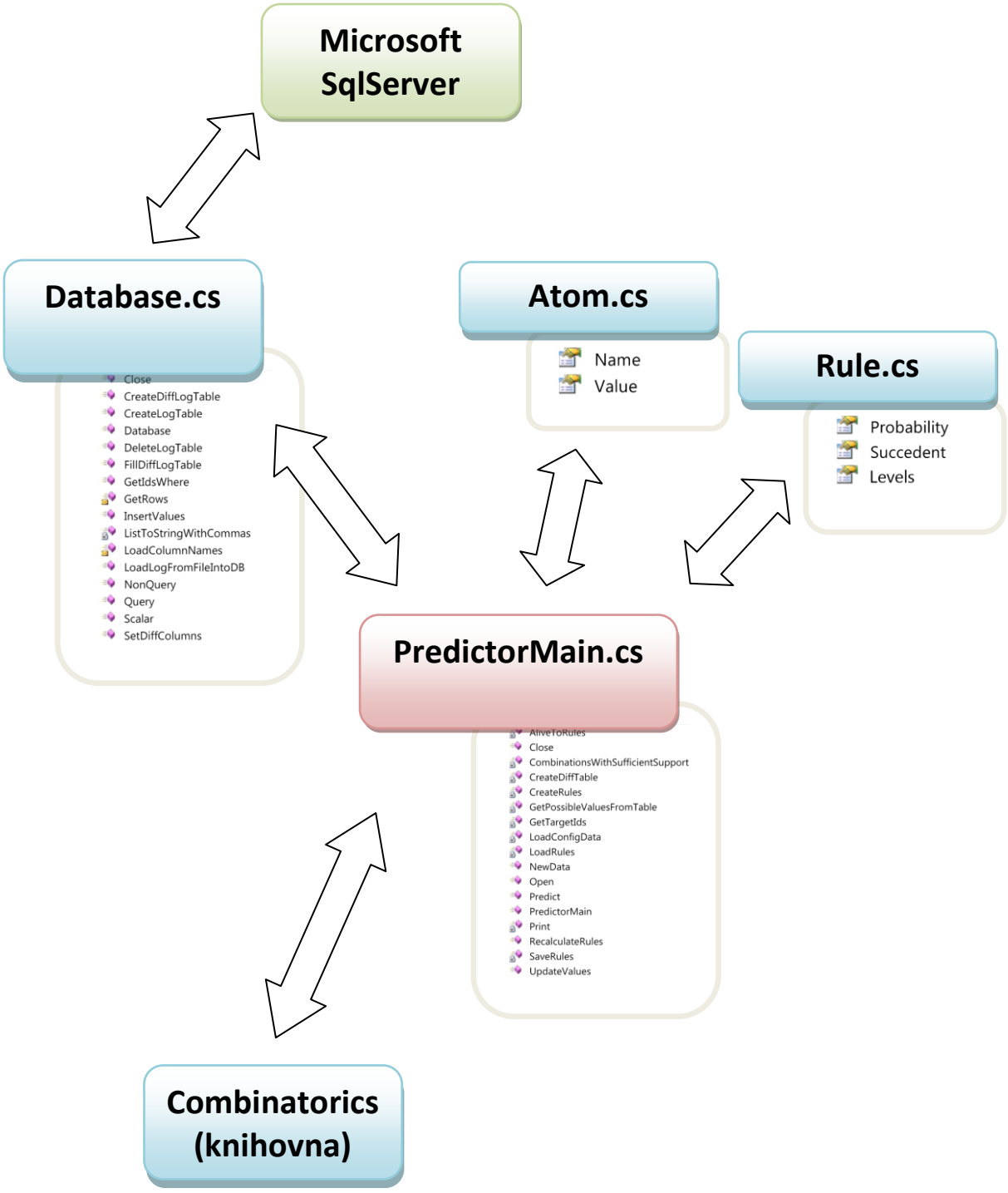
Schéma nejdůležitějších tříd je vidět na obrázku Obr. 28. Jak již z názvu vyplývá – jádrem programu je třída **PredictorMain.cs**. Ta obsahuje několik veřejných metod, které jsou v podstatě jediným ovládacím prvkem celého programu. Některé už jsem zmínil v kapitole 6. Zde je jejich úplný seznam:

- `public PredictorMain()` – konstruktor; inicializace objektu *Database*
- `public void Open()` – vytvoření tabulek log a diffLog, načtení konfiguračních dat
- `public void Close()` – uzavření spojení s databází, uložení pravidel
- `public void NewData(string nameDevice, object obj)` – přijetí nových dat
- `public void UpdateValues()` – uložení nových dat do databáze
- `public List<Rule> Predict(int historyDepth)` – viz kapitola 6
- `public void RecalculateRules(int historyDepth, double minSupport)` – viz kapitola 6

Je zde pak množství funkcí typu private. Jejich rozborem se však podrobně zabývat nebudu; na to zde není prostor.

Další větší třídou je třída **Database.cs**. Ta se stará o komunikaci s SqlServerem. Umožňuje zadávat SQL dotazy, vytvářet a mazat tabulky log a diffLog, vkládat do nich hodnoty. Obsahuje i složitější funkce, např. funkci, která rovnou vrací tabulku na základě zadaných id.

Za zmínku ještě stojí balík tříd nazvaný **Combinatorics**. Ten umožňuje práci s kombinatorikou. Já jej využívám pro vytváření kombinací bez opakování. Toho je zapotřebí při konstrukci asociačních pravidel. Tento balík jsem získal ze stránek www.codeproject.com (Akison, 2008). Jeho použití je podmíněno licencí „Distributed under license terms of CPOL http://www.codeproject.com/info/cpol10.aspx“.



Obr. 28 Schéma tříd

6.3 Konfigurační soubor

Ukázku konfiguračního souboru vidíme na Obr. 29. Každý řádek tohoto souboru specifikuje jeden vstupní atribut. Formát řádku je pak následující:

{nazev_atributu},{parametr},{hodnoty_pro_hledani_pravidel – nepovinné},

kde:

- **`nazev_atributu`** je libovolný řetězec znaků bez mezer a pomlček (z důvodu pozdějšího parsování)
- **`parametr`** může nabývat dvou hodnot:
  - `abs` ... hodnota se ponechá ve své absolutní hodnotě
  - `diff` ... hodnota bude diferencována
- **`hodnoty_pro_hledani_pravidel`** je výčet celočíselných hodnot oddělených středníkem v hranatých závorkách; např. `[-1;1;2]` bude znamenat, že program bude hledat pravidla, jejichž Sukcedent bude:
  - „{nazev_atributu}=-1"
  - „{nazev_atributu}=1"
  - „{nazev_atributu}=2"

Dvě lomítka „//" na začátku řádku znamenají komentář.

```
//---------------------------------
//Konfiguracni soubor
//---------------------------------
cas,abs
tv_POKOJ,abs
radio_POKOJ,diff,[1;-1]
radio_KUCHYN,diff,[1;-1]
svetlo_KUCHYN,diff,[1;-1]
svetlo_WC,diff,[1;-1]
svetlo_KOUPELNA,diff,[1;-1]
svetlo_CHODBA,diff,[1;-1]
svetlo_POKOJ,diff,[1;-1]
svetlo_POKOJLAMPICKA,diff,[1;-1]
okno_POKOJ,diff,[1;-1]
okno_KUCHYN,diff,[1;-1]
zavesy_POKOJ,diff,[1;-1]
zavesy_KUCHYN,diff,[1;-1]
topeni_POKOJ,diff,[1;-1]
...
```

```
...
topeni_KUCHYN,diff,[1;-1]
dvere_WC,diff,[1;-1]
dvere_KOUPELNA,diff,[1;-1]
dvere_POKOJ,diff,[1;-1]
teplota_KUCHYN,abs
teplota_WC,abs
teplota_KOUPELNA,abs
teplota_CHODBA,abs
teplota_POKOJ,abs
teplota_VENKU,abs
urovenOsvetleni_KUCHYN,abs
urovenOsvetleni_WC,abs
urovenOsvetleni_KOUPELNA,abs
urovenOsvetleni_CHODBA,abs
urovenOsvetleni_POKOJ,abs
urovenOsvetleni_VENKU,abs
den,abs
poloha,abs
```

Obr. 29 Ukázka konfiguračního souboru

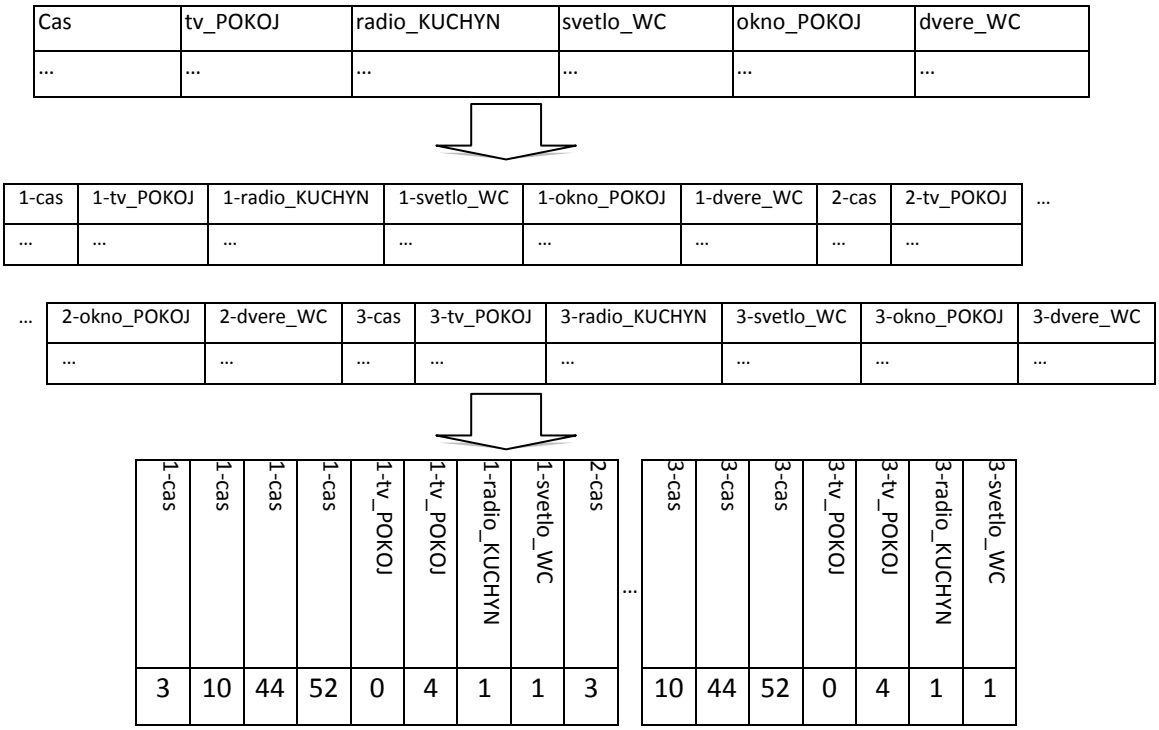
6.4 Tvorba pravidel – kód

Na obrázcích Obr. 31 a Obr. 32 je okomentovaný kód funkce, která vytváří asociační pravidla pro jeden Sukcedent. Parametry jsou:

- `int` **`historyDepth`** – počet řádků databáze, které předchází řádku obsahující `succedent` a na jejichž základě se vytváří pravidla
- `double` **`minSupport`** – minimální podpora pravidla
- `Atom` **`succedent`** – pravá strana hledaných pravidel

Funkce pak postupuje následujícím způsobem:

1. Z databáze vybereme řádky, obsahující `succedent - targetIds`.
2. Vytvoříme názvy sloupců obsahující historii – `namesWithHistory`.
3. Z databáze vyselektujeme vždy x řádků, které předcházejí řádku specifikovaném v `targetIds`. A to tak, že hodnota x se rovná hodnotě `historyDepth`. Zjistíme také, jaké všechny hodnoty jsou v této tabulce použity – `usedValues`.
4. Zkonstruujeme jakousi „expandedTable“, která bude obsahovat všechny kombinace `namesWithHistory` a `usedValues`. To proto, abychom mohli počítat podporu. Schéma celého tohoto procesu je patrné z Obr. 30.



Obr. 30 Schéma vytváření tabulky obsahující všechny kombinace názvů a hodnot

```
private List<Rule> CreateRules(int historyDepth, double minSupport,
    Atom succedent)
{
    //vytvorime seznam indexu radku, ktere potrebujeme prohledavat pro
    //vypocet supportu:
    List<int> targetIds = GetTargetIds(historyDepth, succedent);
    if (targetIds == null)
        return null;

    //sestavime nazvy sloupcu, aby byly unikatni vzhledem k historii:
    List<string> namesWithHistory = new List<string>();
    for (int depth = 1; depth <= historyDepth; depth++)
        //'Names' je globalni promenna s nazvy sloupcu:
        for (int i = 0; i < Names.Count; i++)
            namesWithHistory.Add(depth.ToString() + "-" + Names[i]);
        //vysledkem jsou nazvy typu '1-cas', '1-poloha' ... '{hisDepth}-cas'

    //data z databaze ulozime do tabulky v pameti, v ni pak budeme
    //pocitat support:
    DataTable table = CreateDiffTable(historyDepth, targetIds,
                  namesWithHistory);

    //zjistime, jake hodnoty atomu jsou pouzity (funguje jen pro int):
    List<HashSet<int>> usedValues = GetPossibleValuesFromTable(table);

    //vytvorime vsechny kombinace hodnot atomu:
    List<string> expandedColumnNames = new List<string>();
    List<int> expandedColumnCorrespondingValues = new List<int>();
    for (int i = 0; i < namesWithHistory.Count; i++)
    {
        foreach (int value in usedValues[i])
        {
            expandedColumnNames.Add(namesWithHistory[i]);
            expandedColumnCorrespondingValues.Add(value);
        }
    }

    //a ted jiz vytvorime pravidla:
    int total = table.Rows.Count;
    int numberOfAtoms = expandedColumnNames.Count;

    List<int> atomsAlive = new List<int>();
    for (int i = 0; i < numberOfAtoms; i++)
        atomsAlive.Add(i);

    List<HashSet<int>> deadCombinations = new List<HashSet<int>>();
    List<HashSet<int>> aliveCombinations = new List<HashSet<int>>();
    List<double> supports = new List<double>();
    List<int> newAtomsAlive = new List<int>();
    List<int> atomsWithSupport1 = new List<int>();

    ...
```

1

2

3

4

Obr. 31 Vytváření pravidel pro zadanou pravou stranu (Suc) – kód 1/2

Dále postupujeme takto: délku kombinace nastavíme na 1 a spustíme cyklus. V něm vždy zavoláme funkci `CombinationsWithSufficientSupport` (5), která se postará o výpočet podpory a vrátí atomy, které ji mají dostatečně vysokou. Pokud již žádné nejsou (6), cyklus se ukončí a pravidla jsou vytvořena a vrácena (7).

```
    ...

    //delku kombinaci nastavime na 1:
    int n = 1;
    while (true)
    {
        //vytvorime kombinace delky n, spocitame jejich support,
        //vratime atomy a kombinace s dostatecnou podporou:
        newAtomsAlive = CombinationsWithSufficientSupport(
            atomsAlive.ToArray(), ref aliveCombinations,
            ref deadCombinations, ref supports,
            expandedColumnNames.ToArray(),
            expandedColumnCorrespondingValues.ToArray(),
            minSupport, total, table, n);

        //pri prvnim behu zkontrolujeme, jestli nektere atomy
        //nemaji podporu rovnu 1:
        if (n == 1)
        {
            for (int i = 0; i < supports.Count; i++)
            {
                if (supports[i] == 1)
                    atomsWithSupport1.Add(newAtomsAlive[i]);
            }
            //odstranime je jeste z atomsAlive...
            foreach (int i in atomsWithSupport1)
                newAtomsAlive.Remove(i);
        }
        //pokud jsou newAtomsAlive prazdne, znamena to, ze jsme skoncili:
        if (newAtomsAlive == null)
            break;

        //v opacnem pripade pokracujeme kombinacemi delky n+1:
        atomsAlive = newAtomsAlive;
        n++;
    }

    Console.WriteLine("No more atoms alive. Terminating...");

    //na zaver prevedeme aliveCombinations na pravidla (Rule):
    List<Rule> rules = AliveToRules(aliveCombinations, supports,
        expandedColumnNames.ToArray(),
        expandedColumnCorrespondingValues.ToArray(),
        historyDepth, succedent);

    //tato pravidla pak vratime:
    return rules;
}
```

5 6 7

Obr. 32 Vytváření pravidel pro zadanou pravou stranu (Suc) – kód 2/2

6.5 Tvorba kombinací délky n – kód

Obsah funkce `CombinationsWithSufficientSupport` jsem se rozhodl uvést v plné délce. Vidíme jej na obrázcích Obr. 33 a Obr. 34. Její parametry jsou tyto:

- **`int[] atomsAlive`** – atomy, ze kterých skládáme další kombinace,
- **`ref List<HashSet<int>> combinationsAlive`** – kombinace s dostatečnou podporou, tedy vyšší než `minSupport`,
- **`ref List<HashSet<int>> combinationsDead`** – kombinace, které mají podporu nižší než `minSupport`,
- **`ref List<double> supports`** – seznam hodnot podpory příslušející jednotlivým kombinacím naživu,
- **`string[] atomNames`** – jména atomů; čili `expandedColumnNames`,
- **`int[] atomCorrespondingValues`** – hodnoty těchto sloupců; čili `expandedColumnCorrespondingValues`,
- **`double minSupport`** – minimální požadovaná podpora,
- **`int total`** – celkový počet řádků,
- **`DataTable table`** – datová tabulka v paměti; v ní počítáme podporu,
- **`int n`** – požadovaná délka kombinací.

Algoritmus pracuje takto:

1. Z atomů na živu vytvoříme kombinace délky n (s pomocí tříd Combinatorics). (Obr. 33)
2. U každé z těchto kombinací zkontrolujeme, zda není nadmnožinou nějaké již mrtvé kombinace (čili nesplňující požadavek na podporu). Pokud je, přidáme ji mezi mrtvé a pokračujeme další kombinací. (Obr. 33)
3. Postupně projdeme všechny řádky tabulky `table` a pokud nějaký řádek obsahuje naši aktuální kombinaci, zvýšíme hodnotu proměnné `frequency` o 1. Podporu pak vypočítáme jako počet výskytů lomeno celkový počet řádků – `frequency / total`. (Obr. 34)
4. Pokud je podpora dostatečná, kombinace i její atomy jsou přidány do živých. V opačném případě se kombinace stává mrtvou. (Obr. 34)
5. Objekty `combinationsAlive`, `combinationsDead` a `supports` jsou vráceny referencí. Nové živé atomy `newAtomsAlive` jsou návratovou hodnotou. (Obr. 34)

```
private List<int> CombinationsWithSufficientSupport(int[] atomsAlive,
    ref List<HashSet<int>> combinationsAlive,
    ref List<HashSet<int>> combinationsDead,
    ref List<double> supports,
    string[] atomNames, int[] atomCorrespondingValues,
    double minSupport,
    int total,
    DataTable table,
    int n)
{
    if (atomsAlive.Length == 0 || n > atomsAlive.Length)
        return null;

    List<HashSet<int>> newCombinationsAlive = new List<HashSet<int>>();
    List<HashSet<int>> newCombinationsDead = new List<HashSet<int>>();
    HashSet<int> newAtomsAlive = new HashSet<int>();

    //z atomu na zivu vytvorime vsechny kombinace delky n:
    Combinations<int> combinations = new Combinations<int>(atomsAlive, n);

    int frequency = 0;
    bool containsDeadCombination = false;
    HashSet<int> combinationAsSet = null;
    foreach (List<int> combination in combinations)
    {
        //z kombinace vytvorime mnozinu:
        combinationAsSet = new HashSet<int>(combination);
        //overime, ze nase kombinace neobsahuje nejakou jiz mrtvou
        //kombinaci:
        foreach (HashSet<int> dead in combinationsDead)
        {
            if (combinationAsSet.IsSupersetOf(dead))
            {
                containsDeadCombination = true;
                break;
            }
        }
        if (containsDeadCombination == true)
        {
            //...pokud obsahuje, pridame ji do Mrtvych:
            newCombinationsDead.Add(combinationAsSet);
            continue;   //pokracujeme dalsi iteraci
        }

        ...
```

1

2

Obr. 33 Generování kombinací délky n - kód 1/2

```
private List<int> CombinationsWithSufficientSupport(...)
{
    ...
    foreach (List<int> combination in combinations)
    {
        ...

        //pokud podkombinace vyhovuji, pocitame support:
        frequency = 0;
        foreach (DataRow row in table.Rows)
        {
            bool stillHoping = true;
            //projdeme postupne vsechny prvky nasi kombinace:
            for (int i = 0; i < n; i++)
            {
                int index = combination[i];
                string column = atomNames[index];
                int value = atomCorrespondingValues[index];
                string baseName = column.Split('-')[1];
                //kontrolujeme, zda se bunka lisi od value, nebo
                //v pripade, ze je sloupec diferencovan, preskakujeme 0:
                if ((int)row[column] != value ||
                    (value == 0 && namesToBeDiffed.Contains(baseName)))
                {
                    stillHoping = false;
                    break;
                }
            }
            if (stillHoping == true)
                frequency++;
        }
        double sup = (double)frequency / total;
        //je support dost velky?
        if (sup >= minSupport)
        {
            //vsechny atomy z nasi kombinace zustavaji na zivu:
            foreach (int i in combination)
                newAtomsAlive.Add(i);
            //zrovna tak i cela kombinace:
            newCombinationsAlive.Add(combinationAsSet);
            //zapamatujeme si i jeji support:
            supports.Add(sup);
        }
        else
            newCombinationsDead.Add(combinationAsSet); //je mrtva
    }

    //mrtve a zive kombinace vracime referenci (support je take vracen
    //timto zpusobem):
    combinationsDead.AddRange(newCombinationsDead);
    combinationsAlive.AddRange(newCombinationsAlive);
    //navrat atomu, co jsou na zivu (konvertujeme z HashSetu):
    return newAtomsAlive.ToList<int>();
}
```

3

4

5

Obr. 34 Generování kombinací délky n - kód 2/2

7. Použití systému na simulovaných datech

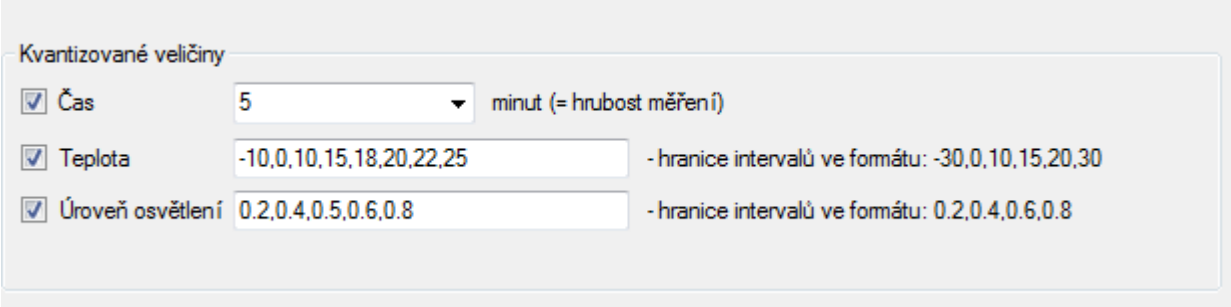
Všechny testy popsané v této kapitole jsem prováděl na svém notebooku (dnes již téměř dva roky starém), který má tyto parametry:

- Model: Dell XPS 1530
- Procesor: Intel Core2 Duo T8300, 2.4 GHz, FSB 800 MHz
- Paměť: 2 GB, 667 MHz
- Pevný disk: 250 GB, 5400 rpm
- Operační systém: Microsoft Windows 7 Professional, 64 bit

Pomocí generátoru popsaného v kapitole 4 jsem vytvořil 3 záznamy:

1. záznam ze vzoru o délce 100 dní (44 000 řádků),
2. náhodný záznam o délce 300 dní (o stejné délce jako záznam 100 dní),
3. záznam ze vzoru o délce 1000 dní (440 000 řádků),

s nastaveními zobrazenými na Obr. 35. Zaměřím se na rychlost tvorby pravidel v závislosti na hloubce historie a požadované minimální podpoře. Sledovat budu také kvalitu pravidel – jejich počet a obsah.



Obr. 35 Nastavení generátoru

Následující obrázek Obr. 36 ukazuje konfigurační soubor, který jsem použil pro specifikaci atributů:

```
//Konfiguracni soubor
//
//Vsechny hodnoty atributu musi byt typu int a musi byt vetsi nebo rovny 0.
//Maximum neni omezeno. Algoritmus prohlizi vsechny pouzite hodnoty kazdeho
//atributu a na zaklade toho vytvari asociacni pravidla.
//
//Radky ve formatu: {nazev_atributu},{parametr 'diff' nebo 'abs'},
// {hodnoty hledanych pravidel v hranatych zavorkach oddeleny ';'}
//Napriklad: cas,abs
// topeni_POKOJ,diff
// dvere_POKOJ,diff,[1;-1] ... tzn. Budeme hledat
// pravidla, kde Suc = 'dvere_POKOJ=1' nebo 'dvere_POKOJ=-1'
//
//POZOR! Nazev atributu nesmi obsahovat '-' pomlcku!
//------------------------------------------------------------------------
cas,abs
tv_POKOJ,abs
radio_POKOJ,diff,[1;-1]
radio_KUCHYN,diff,[1;-1]
svetlo_KUCHYN,diff,[1;-1]
svetlo_WC,diff,[1;-1]
svetlo_KOUPELNA,diff,[1;-1]
svetlo_CHODBA,diff,[1;-1]
svetlo_POKOJ,diff,[1;-1]
svetlo_POKOJLAMPICKA,diff,[1;-1]
okno_POKOJ,diff,[1;-1]
okno_KUCHYN,diff,[1;-1]
zavesy_POKOJ,diff,[1;-1]
zavesy_KUCHYN,diff,[1;-1]
topeni_POKOJ,diff,[1;-1]
topeni_KUCHYN,diff,[1;-1]
dvere_WC,diff,[1;-1]
dvere_KOUPELNA,diff,[1;-1]
dvere_POKOJ,diff,[1;-1]
teplota_KUCHYN,abs
teplota_WC,abs
teplota_KOUPELNA,abs
teplota_CHODBA,abs
teplota_POKOJ,abs
teplota_VENKU,abs
urovenOsvetleni_KUCHYN,abs
urovenOsvetleni_WC,abs
urovenOsvetleni_KOUPELNA,abs
urovenOsvetleni_CHODBA,abs
urovenOsvetleni_POKOJ,abs
urovenOsvetleni_VENKU,abs
den,abs
poloha,abs
```

Obr. 36 Použité nastavení konfiguračního souboru

7.1 Záznam za vzoru – 100 dní

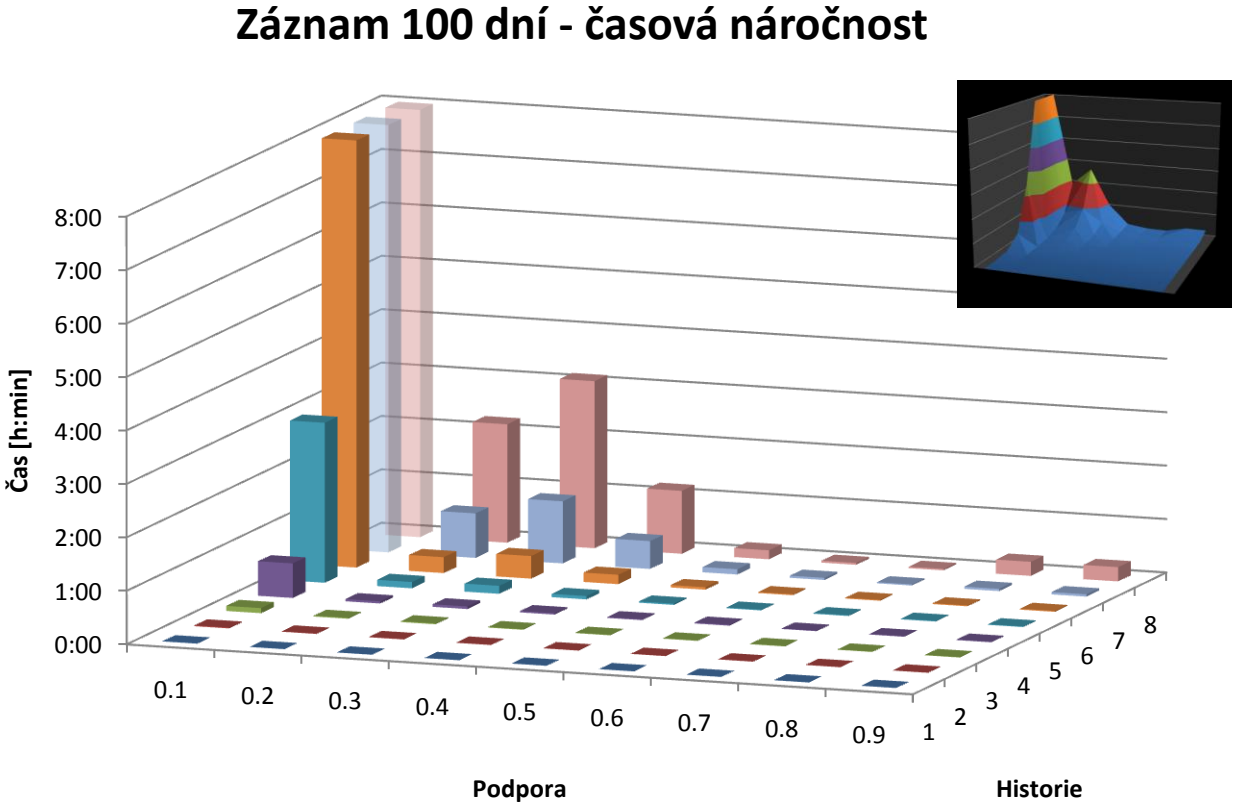
7.1.1 Časová náročnost

Graf na Obr. 37 zobrazuje časovou náročnost tvorby pravidel v závislosti na minimální požadované podpoře a historii, do které pravidla sahají. Testy jsem provedl pro rozsahy:

- $historie \in \{1,2,\dots,8\}$,
- $podpora \in \{0.1, 0.2, \dots, 0.9\}$.

Nejdéle trvaly výpočty pro malé hodnoty podpory a vysoké hodnoty historie. Vzhledem k tomu, že kombinace $\{historie = 7, podpora = 0.1\}$ a $\{historie = 8, podpora = 0.1\}$ byly časově neúměrně náročné, výpočty jsem neprováděl.

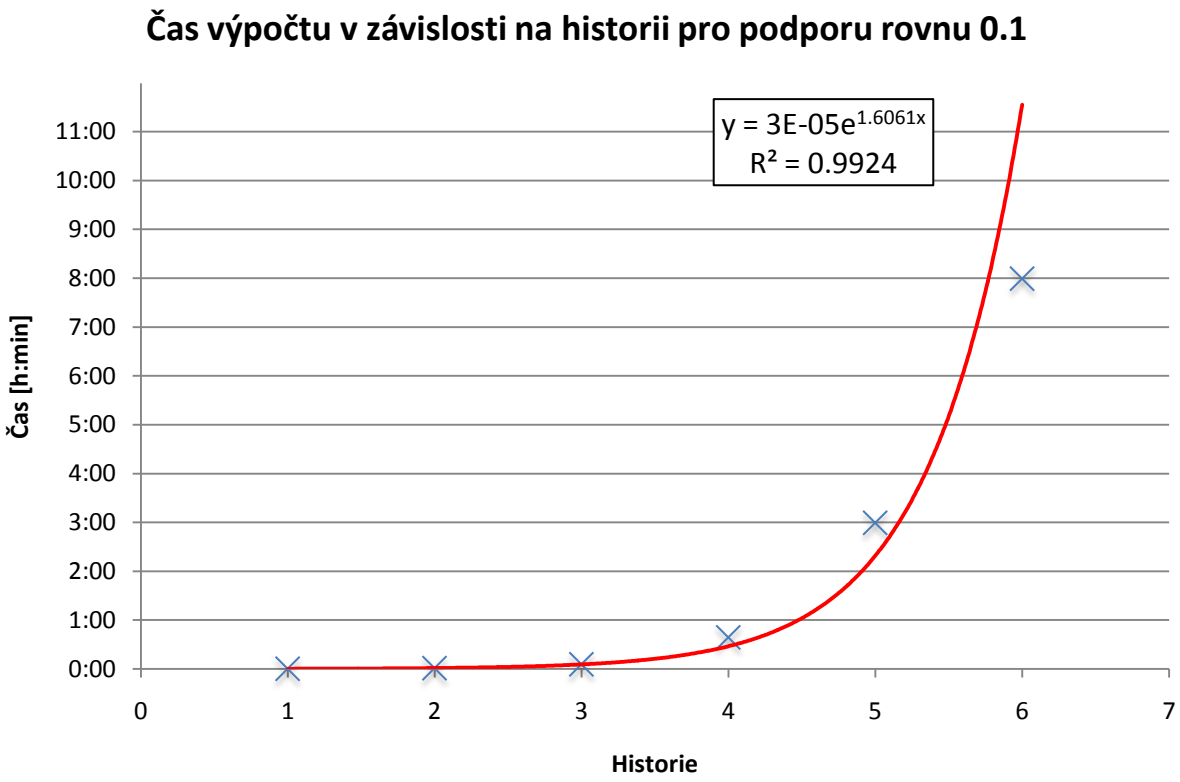
Celkový čas potřebný k provedení těchto testů byl 23 hodin 30 minut.



Obr. 37 Záznam 100 dní – časová náročnost

Provedl jsem extrapolaci zmíněných dvou hodnot na základě ostatních hodnot s podporou 0.1. Vývojový trend je zobrazen na Obr. 38 spolu s rovnicí regrese a její přesností. Jedná se o exponenciálu. Dopočítané hodnoty mají velikost:

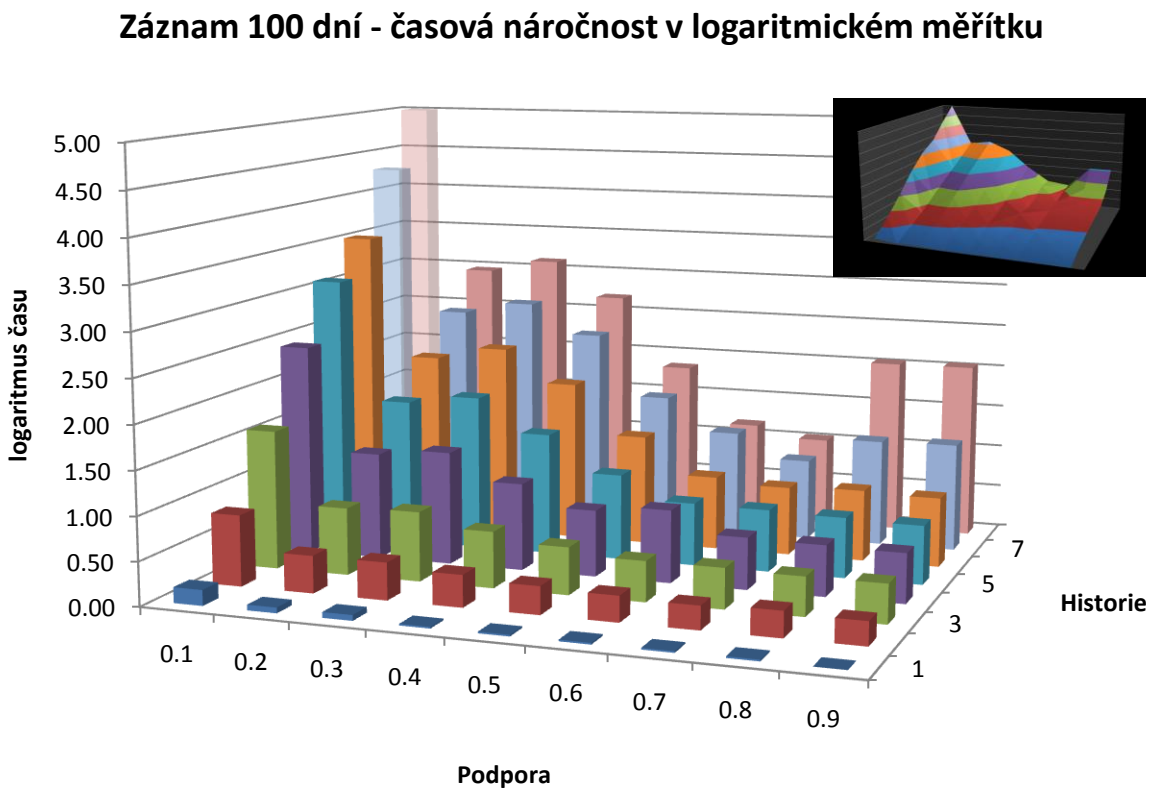
- $čas(historie = 7, podpora = 0.1) = \ 2\ dny$
- $čas(historie = 8, podpora = 0.1) = \ 10\ dní$



Obr. 38 Graf závislosti času výpočtu na hloubce historie, pro podporu rovnu 0.1

Tyto odhady používám i v dalších grafech. Odpovídající sloupce jsou vždy odlišeny od ostatních – jsou průhledné.

Graf z Obr. 37 jsem vykreslil ještě v logaritmickém měřítku – viz Obr. 39. Vidíme, že u většiny hodnot podpory má tento graf lineární charakter, a tudíž ukazuje exponenciální trend původního grafu.



Obr. 39 Záznam 100 dní – časová náročnost v logaritmickém měřítku

7.1.2 Výsledná pravidla

Nastavení kvantizace teploty a úrovně osvětlení má velký vliv na výslednou kvalitu pravidel. Při nevhodném nastavení program generuje velké množství zbytečných pravidel, která pak obsahují málo vypovídající kombinace atributů – v tomto případě teploty a osvětlení. Příklad takových pravidel vidíme na Obr. 40.

Záměrně jsem se nesnažil optimální nastavení nalézt – z důvodu otestování robustnosti a výkonnosti svého algoritmu. Vytvořená pravidla ze sekce 7.1.1 jsem profiltroval tak, že každé pravidlo obsahující atribut teploty nebo osvětlení, jsem odstranil. Výsledkem je pak sada nových pravidel. Ukázka je vidět na Obr. 41.

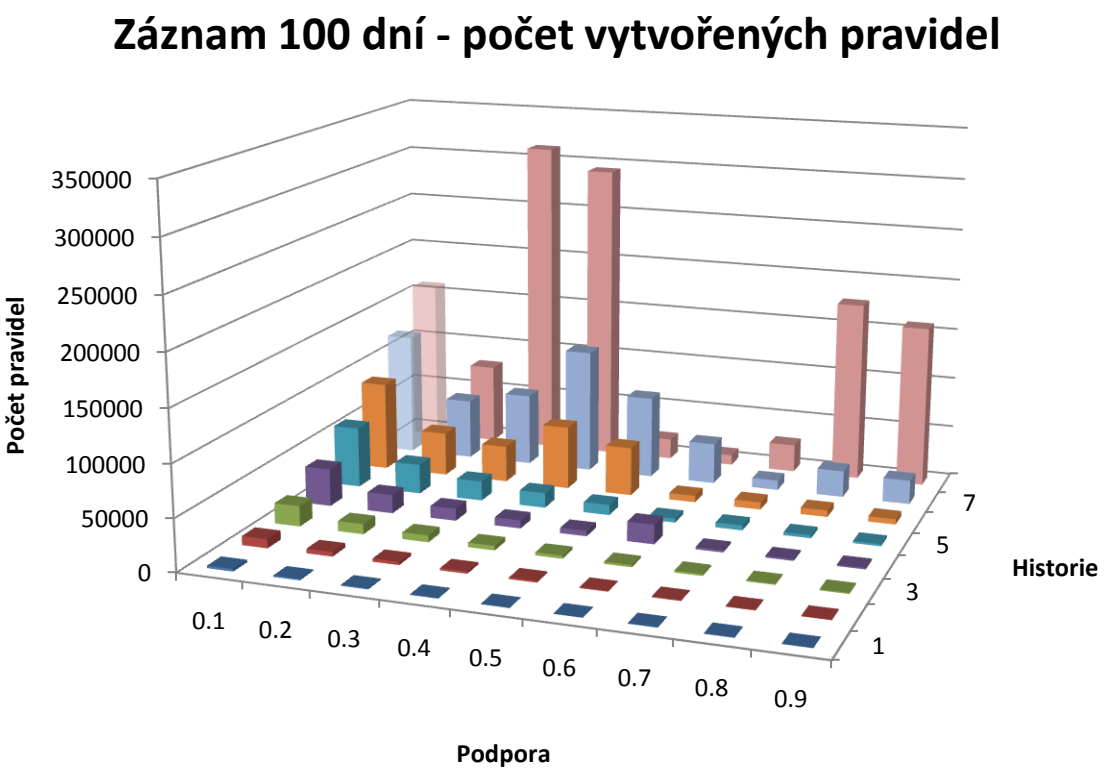
```
[svetlo_KUCHYN=1] 0.5800: 1-teplota_VENKU=2, 2-teplota_VENKU=2, 2-poloha=1,
3-teplota_VENKU=2, 4-dvere_POKOJ=1, 5-svetlo_POKOJ=-1, 6-teplota_VENKU=2,
6-urovenOsvetleni_CHODBA=2, 6-urovenOsvetleni_POKOJ=5
[svetlo_KUCHYN=1] 0.5800: 1-teplota_VENKU=2, 2-teplota_VENKU=2, 2-poloha=1,
3-teplota_VENKU=2, 4-dvere_POKOJ=1, 5-svetlo_POKOJ=-1, 6-teplota_VENKU=2,
6-urovenOsvetleni_CHODBA=2, 6-poloha=0
[svetlo_KUCHYN=1] 0.5800: 2-dvere_POKOJ=-1, 2-teplota_VENKU=2, 2-poloha=1,
4-teplota_VENKU=2, 6-teplota_VENKU=2, 6-urovenOsvetleni_CHODBA=2
...
[dvere_POKOJ=-1] 0.5122: 1-urovenOsvetleni_CHODBA=2, 2-urovenOsvetleni_KUCHYN=5
[dvere_POKOJ=1] 0.5122: 1-urovenOsvetleni_VENKU=5, 2-urovenOsvetleni_POKOJ=5
...
[dvere_KOUPELNA=1] 0.4367: 1-teplota_POKOJ=6, 1-urovenOsvetleni_VENKU=5
[dvere_KOUPELNA=1] 0.4367: 1-dvere_KOUPELNA=-1, 1-urovenOsvetleni_POKOJ=5
[svetlo_CHODBA=1] 0.4365: 1-urovenOsvetleni_VENKU=5, 2-poloha=1
...
```

Obr. 40 Ukázka výsledných pravidel

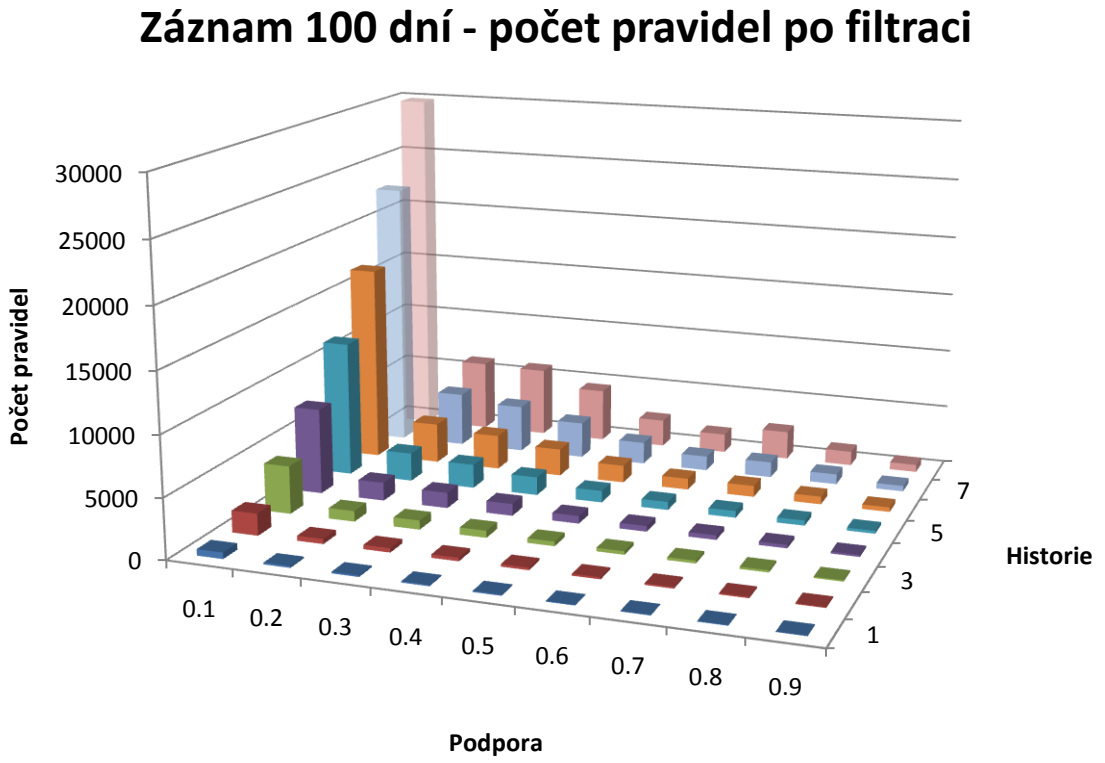
```
[svetlo_WC=1] 0.8317: 5-poloha=0, 6-poloha=0
[svetlo_CHODBA=1] 0.8317: 2-poloha=1, 3-poloha=0
[svetlo_CHODBA=1] 0.8317: 1-dvere_POKOJ=-1, 5-poloha=0
...
[svetlo_KOUPELNA=1] 0.6841: 2-dvere_WC=-1, 4-poloha=3
[dvere_KOUPELNA=-1] 0.6841: 2-dvere_KOUPELNA=1, 4-tv_POKOJ=0
[dvere_KOUPELNA=-1] 0.6841: 2-tv_POKOJ=0, 2-dvere_KOUPELNA=1
[zavesy_POKOJ=1] 0.6700: 2-tv_POKOJ=4
...
[svetlo_CHODBA=-1] 0.5060: 1-svetlo_KOUPELNA=-1
[radio_KUCHYN=-1] 0.5000: 2-cas=50
...
[dvere_KOUPELNA=1] 0.5000: 2-poloha=1, 4-poloha=1
[dvere_WC=1] 0.5000: 2-poloha=1, 3-poloha=1
[dvere_KOUPELNA=1] 0.5000: 2-poloha=1, 3-poloha=1
...
```

Obr. 41 Ukázka výsledných pravidel po filtraci

Počet pravidel před a po filtraci – opět v závislosti na požadované podpoře a hloubce historie – je vidět na Obr. 42 a Obr. 43:

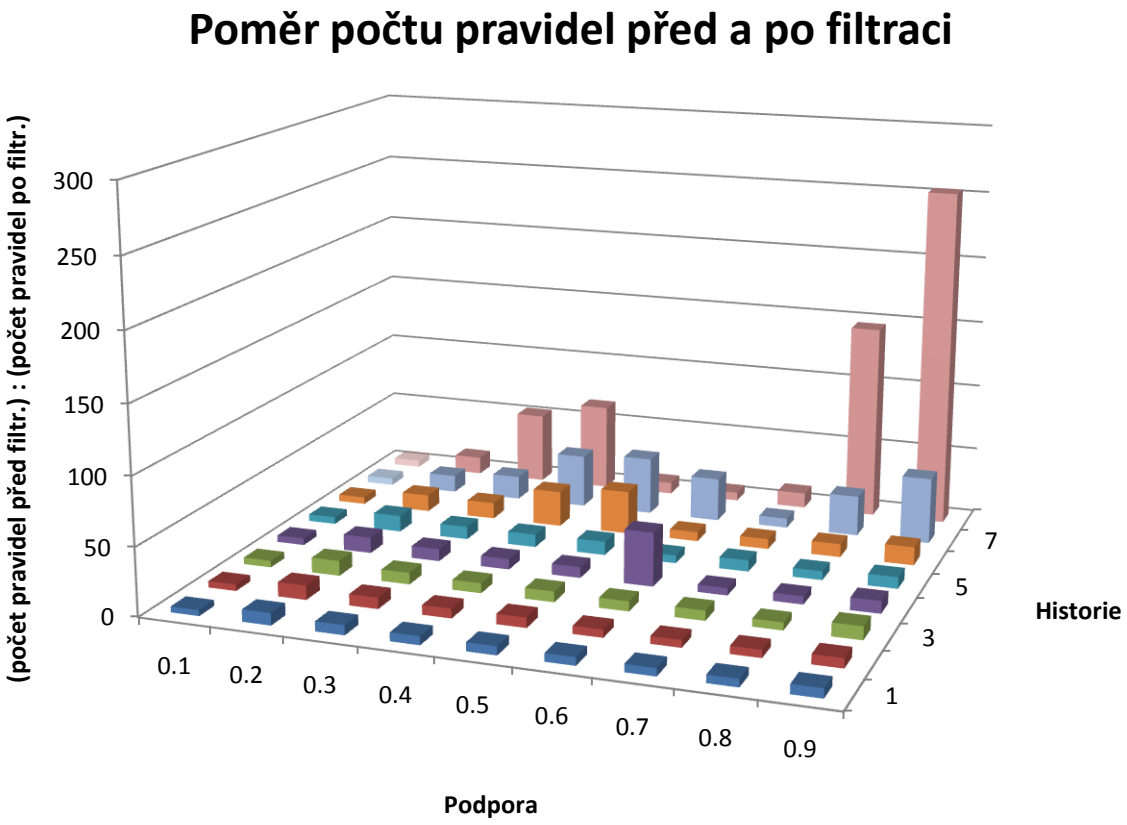


Obr. 42 Záznam 100 dní – počet vytvořených pravidel



Obr. 43 Záznam 100 dní – počet pravidel po filtraci

Pro srovnání obou předchozích grafů jsem hodnoty z jednotlivých měření navzájem podělil a získal tento graf (Obr. 44):



Obr. 44 Záznam 100 dní - poměr počtu pravidel před a po filtraci

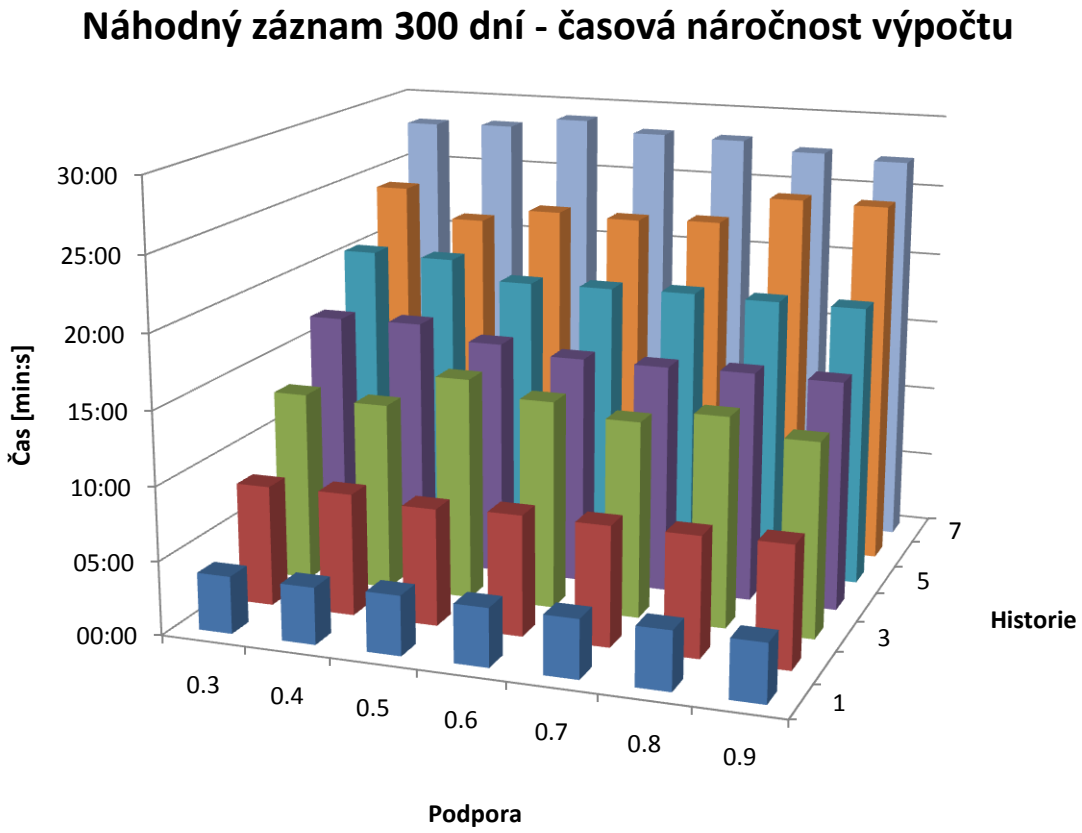
Vidíme, že s rostoucí historií a rostoucí podporou rostl i počet zbytečně generovaných kombinací teplot a úrovní osvětlení.

7.2 Náhodný záznam – 300 dní

Obdobně jako v sekci 7.1 jsem i s náhodným záznamem prováděl časové testy. Rozsahy parametrů jsem zvolil takto:

- $historie \in \{1,2,\dots,7\}$,
- $podpora \in \{0.3, 0.4, \dots, 0.9\}$.

Výsledek je zakreslen na Obr. 45. S hloubkou historie se časová náročnost zvyšuje. Změna podpory nemá téměř žádný vliv. To je způsobeno s největší pravděpodobností tím, že náhodný záznam je zcela nahodilý, a jednotlivé kombinace atributů tak mají stejnou četnost.



Obr. 45 Náhodný záznam 300 dní – časová náročnost výpočtu

Vzhledem k tomu, že pravidla vytvořená z náhodného záznamu nemají žádnou vypovídací hodnotu, další testy s těmito daty provádět nebudu.

7.3 Záznam ze vzoru – 1000 dní

Jako poslední testovací data jsem zvolil záznam z předlohy o délce 1000 dní. Počet řádků v databázi pak dosahoval počtu 440 tisíc. Při běhu programu se využití paměti pohybovalo v relacích:

- 496 MB pro SqlServer a
- 70 MB pro samotný program.

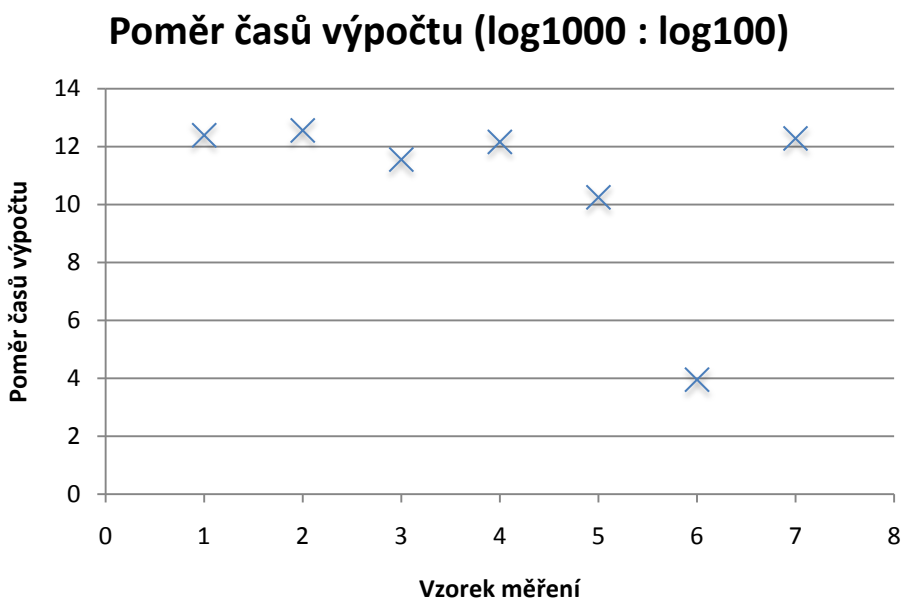
7.3.1 Časová náročnost

Tabulka Tab. 10 obsahuje srovnání časové náročnosti záznamu pro 1000 dní a záznamu pro 100 dní ze sekce 7.1. Parametry pro srovnání jsou voleny intuitivně – aby časové zatížení bylo v rozumných mezích.

Historie	Podpora	Délka výpočtu [h:min:s]		Poměr délek výpočtu
		1000 dní	100 dní	
1	**0.3**	0:02:04	0:00:10	**12.5**
1	**0.5**	0:01:53	0:00:09	**12.6**
1	**0.8**	0:01:44	0:00:09	**11.6**
3	**0.8**	0:04:52	0:00:24	**12.2**
5	**0.2**	1:12:06	0:07:02	**10.3**
5	**0.3**	0:35:26	0:08:58	**3.9**
5	**0.8**	0:08:48	0:00:43	**12.3**

Tab. 10 Záznam 1000 dní – časová náročnost; srovnání se záznamem 100 dní

Poměr délek výpočtu (poslední sloupec tabulky Tab. 10) vidíme graficky znázorněn na Obr. 46. S výjimkou hodnoty 3.9 se výsledky pohybují kolem čísla 12. To zhruba odpovídá tomu, že i datový vzorek byl 10 krát větší.



Obr. 46 Záznam 1000 dní – grafické srovnání časů výpočtů se záznamem 100 dní

Za předpokladu, že tento poměr by zůstal zachován pro většinu výpočtů, celková doba by místo jednoho dne (doba tvoření pravidel pro stodenní data) byla 10 dní.

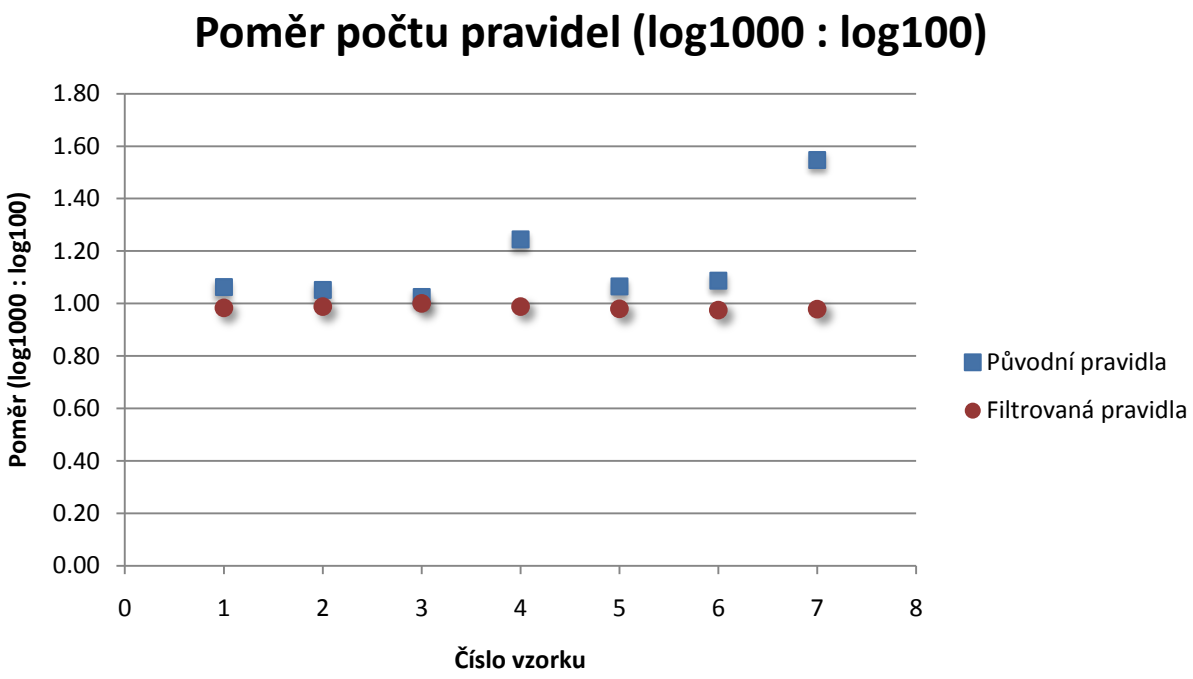
7.3.2 Výsledná pravidla

Stejně jako v sekci 7.1.2 i zde provedu filtraci pravidel – dle stejného kritéria (odstranění teploty a osvětlení). Výsledky budu porovnávat se záznamem pro 100 dní. Jsou shrnuty v tabulce Tab. 11.

Historie	Podpora	Počet pravidel		Poměr	Počet filtrov. pravidel		Poměr
		1000 dní	100 dní		1000 dní	100 dní	
1	**0.3**	948	892	**1.06**	121	123	**0.98**
1	**0.5**	548	521	**1.05**	80	81	**0.99**
1	**0.8**	290	283	**1.02**	49	49	**1.00**
3	**0.8**	1259	1012	**1.24**	175	177	**0.99**
5	**0.2**	30924	29030	**1.07**	2352	2401	**0.98**
5	**0.3**	20872	19200	**1.09**	1967	2017	**0.98**
5	**0.8**	4347	2811	**1.55**	418	427	**0.98**

Tab. 11 Záznam 1000 dní – srovnání počtu pravidel před a po filtraci s daty záznamu 100 dní

Pro větší názornost je vynesu do grafu (Obr. 47):



Obr. 47 Záznam 1000 dní - grafické srovnání počtu pravidel před a po filtraci (oproti záznamu 100 dní)

Vidíme, že poměry se pohybují okolo jedničky. A to jak u původních, tak u filtrovaných pravidel. Důvodem tohoto jevu je patrně pravidelnost generování dat. Přestože se snažím pořadí některých akcí měnit a posouvat je ještě v čase, na výsledek to nemá velký vliv.

7.4 Shrnutí

Sestavil jsem 3 sady záznamů různých délek a parametrů – 100 dní, 300 dní náhodných a 1000 dní. Tato data jsem použil k vytváření asociačních pravidel.

V experimentech jsem se soustředil na dobu výpočtu a na vlastnosti výsledných pravidel. Sledoval jsem závislost potřebného času na požadované minimální podpoře a hloubce historie, do níž výsledné pravidlo sahá. U záznamu délky 100 dní jsem provedl téměř 24 hodinový test, ve kterém se potvrdilo, že nižší požadovaná podpora a vyšší hloubka historie zvyšují dobu výpočtu. Ke stejnému závěru jsem došel u záznamu délky 1000 dnů. Tam byla časová náročnost zhruba 10 krát větší. U náhodného záznamu neměla velikost požadované podpory na výsledek vliv, roli

hrála pouze hloubka historie. To odpovídá předpokladům, neboť záznam byl generován rovnoměrně náhodně.

Kvalita výsledných pravidel velmi závisí na volbě kvantizace jednotlivých veličin. V mém experimentu bylo vytvořeno velké množství zbytečných pravidel. Ta obsahovala různé varianty kombinací špatně kvantizované teploty a úrovně osvětlení. Záměrně jsem tento nedostatek ponechal bez povšimnutí, abych vyzkoušel robustnost svého algoritmu. Ta se ukázala jako postačující, neboť program fungoval pro velkou škálu nastavených parametrů.

Možným vylepšením by bylo vytvořit vlákno provádějící konstrukci pravidel, kterému bychom měřili čas. V případě, že tento čas překročí stanovené maximum, algoritmus by pokračoval další pravou stranou, nebo by se ukončil.

8. Závěr

Úkolem této práce bylo seznámit se s metodami vhodnými pro predikci akcí uživatele v prostředí inteligentního domova a vytvořit predikční modul.

Predikční modul je koncipován jako dynamická knihovna využívající asociační pravidla. Tento modul bude posléze využit v systému inteligentního domova. Pro implementaci algoritmu jsem zvolil prostředí C# Microsoft Visual Studio 2008 Professional. Důvodem k tomuto rozhodnutí bylo zejména to, že nadřazený systém, do kterého bude můj modul zakomponován, tento jazyk používá také. Predikční modul se bude moci průběžně učit z logu získaného z nadřazeného systému a tím optimalizovat svou funkci. Výstupem budou predikované akce.

Pro ověření funkčnosti predikčního modulu bylo zapotřebí mít data, na kterých je možné modul testovat. Ideální by samozřejmě bylo mít data reálná, ta však bohužel k dispozici nebyla. Mým úkolem bylo data vygenerovat. Vytvořil jsem generátor akcí, který umožňuje pomocí pseudo-jazyka vytvářet rozsáhlé množiny testovacích dat doplněné například o data o teplotě. Pro větší rozmanitost umožňuje generátor nastavit variabilitu pořadí a časového určení akcí. Výstupem je textový soubor (log) pro jednotlivé dny.

Pro účely testování jsem vytvořil pomocí generátoru akcí 3 sady záznamů různých délek a parametrů – 100 dní, náhodných 300 dní a 1000 dní. Tato data jsem použil ke konstrukci asociačních pravidel pomocí predikčního modulu. V experimentech jsem se soustředil na dobu výpočtu a na vlastnosti výsledných pravidel. Sledoval jsem závislost potřebného času na požadované minimální podpoře a hloubce historie, do níž výsledné pravidlo sahá. U záznamu délky 100 dní jsem provedl téměř 24 hodinový test, ve kterém se potvrdilo, že nižší požadovaná podpora a vyšší hloubka historie zvyšují dobu výpočtu. Ke stejnému závěru jsem došel u záznamu délky 1000 dnů. Tam byla časová náročnost zhruba 10 krát větší.

Možným vylepšením by bylo vytvořit vlákno provádějící konstrukci pravidel, kterému bychom měřili čas. V případě, že tento čas překročí stanovené maximum, algoritmus by pokračoval další pravou stranou, nebo by se ukončil. Predikční modul bude nutné ještě důkladně otestovat na reálných datech.

9. Použitá literatura

[1] Agrawal, R., Imieliski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data* (stránky 207-216). New York: ACM.

[2] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast Discovery of Association Rules. V *Advances in Knowledge Discovery and Data Mining* (stránky 307–328). AAAI/MIT Press.

[3] Akison, A. (2008). Combinatorics. *www.codeproject.com* .

[4] Berka, P. (2003). *Dobývání znalostí z databází.* Academia, Praha.

[5] Fernández-Montes, A. (2007). Modeling Smart Homes for Prediction Algorithms. *Knowledge-Based Intelligent Information and Engineering Systems* (stránky 26-33). Springer Berlin / Heidelberg.

[6] Hawkins, J. (2004). *On Intellligence.* Times Books.

[7] Řehoř, M. (2007). Metody klasifikace EEG signálu. *Bakalářská práce - ČVUT, FEL*, 33-34.

[8] S. K. Das, D. J.-Y. (2002). The role of prediction algorithms in the MavHome smart home architecture. *Wireless Communications, IEEE In Wireless Communications, IEEE, Vol. 9, No. 6.* , 77-84.

[9] *Smarthome*. (2010). http://smarthome.duke.edu

[10] Šnorek, M. (2002). *Neuronové sítě a neuropočítače.* skripta FEL ČVUT.

[11] Warwick. (2007). http://www.warwick.ac.uk/atc/tig/whatwedo/interests/neuron.gif

[12] Michalski R. S. (1969). *On the Quasi-minimal solution of the general covering problem.* Proc. 5th Int. Symposium on Information Processing FCIP'69, Bled, Yugoslavia, 125-128

10. Obsah přiloženého CD

`DP_rehor_miroslav_2010_05_14.pdf`

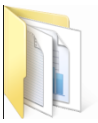
- samotná diplomová práce

`readme.txt`

- soubor obsahující stručný popis obsahu CD

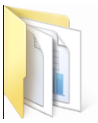
`SmartHomePrediction`

- projekt Visual Studia 2008, kompletní kódy



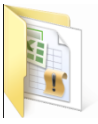
`Vysledky_testu`

- výsledky testování; grafy těchto dat jsou uvedeny v kapitole 6



`Zdrojova_data_pro_testovani`

- data vygenerovaná programem Generátor akcí, použita pro testování v kapitole 6



`Ostatni`

- ostatní pomocné soubory