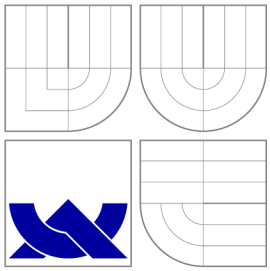
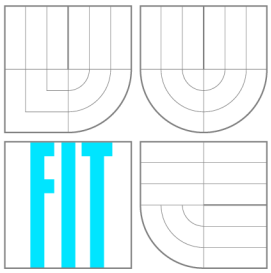


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

OPTIMALIZÁTOR ROZVRHU ZKOUŠEK NA FIT

OPTIMIZER FOR EXAM SCHEDULING AT THE FIT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MIROSLAV PAULÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Dr. Ing. DUŠAN KOLÁŘ

BRNO 2015

Abstrakt

Tématem této práce je automatizovaná tvorba rozvrhu zkoušek pro Fakultu informačních technologií Vysokého učení technického v Brně. Je zde popsán seznam požadavků (omezení), na které je nutné brát při návrhu rozvrhu ohled. Tyto omezení jsou dále klasifikována podle míry vlivu na kvalitu rozvrhu na nutné a volitelné. Problém plánování zkoušek je zde dekomponován na dílčí podproblémy a dále řešen pomocí Contraint logic programming. Výsledkem je množina řešení splňující všechna nutná omezení. Z nich je nakonec vybráno takové suboptimální řešení, které nejméně porušuje zbývající škálovatelná omezení.

Abstract

This paper describes automated examination scheduling for the Faculty of Information Technology of Brno University of Technology. It specifies a list of restrictions that must by satisfied. Furthermore, this limitations are classified due to their influence on a quality of the final version of the examination schedule. There are two types of restrictions; soft and hard. The task is to find such a solution that satisfies all hard constraints and breaks the minimum of soft constraints using techniques described in this paper.

Klíčová slova

Plánování, Plánování rozvrhu zkoušek, Constraint Logic Programming, SWI Prolog, Optimalizační úloha

Keywords

Scheduling, Examination scheduling, Constraint Logic Programming, SWI Prolog, Optimization problem

Citace

Miroslav Paulík: Optimalizátor rozvrhu zkoušek na FIT, diplomová práce, Brno, FIT VUT v Brně, 2015

Optimalizátor rozvrhu zkoušek na FIT

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Dr. Ing. Dušana Koláře

.....
Miroslav Paulík
26. května 2015

Poděkování

Rád bych poděkoval svému vedoucímu, Doc. Dr. Ing. Dušanu Kolářovi, za cenné rady a připomínky, které mi pomohly dovést tuto práci do konce. Dále bych chtěl poděkovat Ing. Petru Lampovi za ochotu a poskytnutí potřebných statistických dat. Nakonec bych chtěl vyjádřit díky svým spolužákům, kteří mi pomáhali s testováním.

Věnováno Bc. Igorovi Pavlů.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Plánování termínů zkoušek je úkol, který se každoročně řeší na všech vysokých školách. Podobně jako při tvorbě rozvrhu výuky jde o úlohu přiřazení předmětů do učeben v určité hodiny v předem vymezeném období. To, na rozdíl od výuky, trvá většinou v řádu týdnů a v rámci semestru se většinou neopakuje. Ačkoliv se může zdát, že plánování termínů zkoušek a tvorba rozvrhu výuky jsou už dále stejné problémy, tak tomu tak být nemusí v závislosti na konkrétních požadavcích [20, 17]. Mohou zde být drobné odlišnosti např. v tom, jak jsou vnímány různé vlastnosti výsledného rozvrhu. Zatímco u výuky jsou přednášky organizovány z důvodu efektivnosti do souvislých bloků, tak u zkoušek tomu tak není. Pro studenty je naopak výhodnější mít mezi jednotlivými termíny dostatek času na přípravu a regeneraci.

Všechny termíny zkoušek (včetně opravných) se plánují samostatně. Je zde navíc mnoho omezení, které musí jednotlivé termíny splňovat. Ty jsou dány školními předpisy či dispozicemi jednotlých fakult jakými jsou např. kapacity učeben, množství vyučujících či akce konané na akademické půdě dané instituce ve zkouškovém období. Je také nutné brát v potaz požadavky vyučujících a zároveň vytvořit takový plán zkoušek, aby je mohli studenti vůbec absolvovat (např. zajistit, aby se nekonaly 2 zkoušky zapsané stejným studentem v jeden a tentýž čas).

Tolik druhů požadavků a omezení spolu s proměnlivostí struktury předmětů vyučovaných v jednotlivých semestrech a letech dělá tuto úlohu velmi komplikovanou. Není totiž vždy zaručeno, že bude existovat plán zkoušek bez jediného konfliktu. V tomto ohledu jde tedy o hledání takového suboptimálního řešení, které bude obsahovat co nejméně takových konfliktů. Z těchto důvodů je zřejmé, že ruční plánování jednou či více osobami bude velmi časově náročné a zároveň náchylné k chybám. Cílem této práce je celý tento proces zautomatizovat a zajistit tak vznik takového rozvrhu zkoušek, který bude respektovat všechna omezení a zároveň bude mít i vlastnosti, které ocení nejen vyučující, ale i samotní studenti.

Kapitola 2

Způsoby řešení plánovacích úloh

Rozvrhování zkoušek je speciálním případem problému plánování rozvrhu, kdy je úkolem nalézt takové vzájemné přiřazení mezi předměty, učebnami, studenty a vyučujícím ve vymezeném čase, kdy pro určité zdroje musí být splněna určitá omezení [14]. Řešením podobných úloh se lidé zabývající už od padesátých let minulého století [25] a pro svou komplexnost a variabilitu je tato úloha i nadále zkoumána [17]. S postupem času se k řešení takové úlohy používaly různé nástroje a techniky počínaje těmi založených na grafech, přes lineální programování až po využití např. genetických algoritmů. V posledních letech je však jedním z hlavních směrů výzkumu zaměřen na přístup pomocí „Constraint logic programming“ [19, 15, 9].

V této kapitole jsou uvedeny postupy řešení tohoto problému pomocí vybraných technik. Nejprve jsou zmíněny postupy vycházející z OR grafů a transformací na lineární programování, dále je zde popsán nástin řešení využívající genetické algoritmy, „Tabu search“ a principy vycházející z „Constraint satisfaction problem“ (CSP).

2.1 OR grafy

2.1.1 Vrcholové barvení grafu

Uvažujme graf, jehož vrcholy představují konkrétní předmět (dvojice třída a vyučující) a kde hrany značí skutečnost, že předměty ve spojených uzlech sdílí studenta nebo vyučujícího. Problém naplánování rozvrhu do t časových úseků z takto popsaného grafu odpovídá úloze barvení uzlů grafu [7, 25], přesněji jde o obarvení vrcholů maximálně t barvami. Uzly obarvené stejným odstínem pak představují předměty, které je možné vyučovat (zkoušet) zároveň. Víme také, že obarvení takového grafu, kde $t \geq 3$, je $\mathcal{NP}$-úplný problém [7].

Výše uvedená grafová reprezentace plánovací úlohy však není úplně korektní. Nezohledňuje totiž značné množství dalších omezení. Taková omezení ale bývají velmi důležitá, neboť právě ony upřesňují charakter a kvalitatitvní vlastnosti výsledného rozvrhu. Například z definice hran uvedeného grafu již nepoznáme, zda hrana reprezentovala jednoho učitele či rovnou stovky studentů, stejně jako se ztrácí informace o tom, které předměty mají společné vyučující.

2.1.2 Toky v síti

Jiný způsob využívající grafové reprezentace je podobný bipartitnímu párování z vrcholů C reprezentující třídy do množiny vyučujících T. Zároveň platí, že graf obsahuje hrany

(C_i, T_j) právě tehdy, když učitel T_j přednáší ve třídě C_i.

Úloha bipartitního párování se obvykle řeší transformací grafu na síť a následném nalezení maximálního toku. Síť se vytvoří přidáním „startovacího“ a „koncového“ uzlu s resp. d, nových hran (s, x) pro všechna $x \in C$ a (y, d) pro všechna $d \in T$. Zároveň má každá hrana v síti svou minimální a maximální kapacitu viz tabulka 2.1.

Cesta	Minimální tok	Maximální tok
(s, C_i)	0	1, pokud všechny termíny třídy C_i jsou již naplánovány, jinak 0
(C_i, T_j)	0	1, pokud termín (přednáška) učitele T_j třídě C_i je naplánovaná, jinak 0
(T_j, d)	0	1, pokud jsou naplánovány všechny termíny (přednášky) vyučujícího T_j, jinak 0

Tabulka 2.1: Minimální a maximální tok v síti [14].

Algoritmů na řešení této úlohy je mnoho. Mezi velmi efektivní patří rodina algoritmů Ford-Fulkerson [4]. Nicméně, plánování rozvrhu pro období T vyžaduje nalezení T toků v síti, každý pro jedno období, což je opět $\mathcal{NP}$-úplný problém [14]. Navíc ani zde nejsme schopní zakomponovat všechny typy omezení (např. velikosti učeben či pořadí zkoušek), nehledě na fakt, že tento způsob plánování uvažuje všechny doby trvání za jednotné (délky 1), což je v praxi nereálné.

2.2 Lineární programování

Třetím metodou pro řešení plánovací úlohy je lineární programování s celočíselnými proměnnými [25]. Omezení jsou zde modelovány jako soustavy rovnic nad zdroji, což je zároveň hlavním problémem tohoto postupu, neboť počet takových zdrojů a omezení může být velmi velký. Nalezení optimálního řešení lineárního programu je navíc $\mathcal{NP}$-těžký problém, mimo situaci, kdy soustava rovnic omezení tvoří např. unimodulární matici (tj. matici, jejíž determinant je +/- 1), což je opět v praxi velmi nepravděpodobné.

Pro demonstraci, uvažujme plánovací úlohu, která zahrnuje m učitelů a n tříd. Každý učitel T_j vyučuje R_{ij} přednášek ve třídě C_i. Přednášky musí být naplánovány do ε časových úseků. Uvažujme proměnou x_{ijk}, jejíž hodnota je 1 v případě, že učitel T_j vyučuje třídu C_i v časovém úseku k, jinak je hodnota proměnné 0. Omezení popisující následující situaci lze zapsat takto:

$$\sum_{k=1}^{\varepsilon} x_{ijk} = r_{ij}, i = [n] \wedge j = [m] \tag{2.1}$$

$$\sum_{k=1}^{\varepsilon} x_{ijk} \leq 1, j = [m] \wedge k = [\varepsilon] \tag{2.2}$$

$$\sum_{k=1}^{\varepsilon} x_{ijk} \leq 1, i = [n] \wedge k = [\varepsilon] \tag{2.3}$$

Rovnice 2.1 modeluje případ, kdy všechny přednášky musí být naplánovány, rovnice 2.2 zastupuje fakt, že učitel nemůže mít více přednášek zároveň a konečně rovnice 2.3 značí, že student nemůže navštívit víc jak jednu přednášku konanou ve stejný čas.

Pro vyřešení této úlohy je možné použít různé úpravy jakými jsou například spojení studentů, místností či učitelů do skupin, což má za následek předefinování významu proměnných [21]. Konkrétního výsledku lze dosáhnout např. použitím Langrean relaxační metody [21, 14].

Tato metoda řešení má poměrně dobré výsledky, nicméně řešení je pevně svázáno s konkrétní specifikací problému. Přidání nového omezení nebo jen drobná změna stávajícího může způsobit, že značná část rovnic bude muset být přepracována, což se z hlediska opakovaného použití pro jiná období ukazuje jako nepraktické.

2.3 Genetické algoritmy

Genetický algoritmus (GA) je heuristický postup, který se snaží aplikací principů evoluční biologie nalézt řešení složitých problémů mezi které patří i plánování [6]. GA používá techniky známé z biologie – dědičnost, mutace, přirozený výběr a křížení, proto patří do kategorie tzv. evolučních algoritmů.

Každý problém řešený GA, je reprezentován (zakódován) do chromozomu, což je vektor symbolů resp. dvojic genů. Dva geny na i-té pozici pak mohou představovat např. čas a místnost konání i-té zkoušky [14].

GA vyžaduje k řešení nějaký počáteční stav chromozomů. Ten lze získat např. použitím algoritmu barvení grafu, obecně to ale může být i náhodný řetězec. Jedná se o iterativní algoritmus, kdy v každém cyklu vzniká nová generace vyhovujících řešení s lepšími vlastnostmi. Ty určuje tzv. „fitness“ funkce. V plánovací úloze musí tato funkce zohlednit nejen nutná omezení pro tvorbu rozvrhu, ale i kvalitativní vlastnosti jako jsou počet zkoušek jednoho studenta za den či vhodná obsazenost učeben. Zároveň je však nutné, aby implementace fitness funkce byla co nejrychlejší vzhledem k jejímu častému volání.

Genetické algoritmy mají velmi dobré výsledky při hledání efektivnějších řešení, nicméně obsahuje spoustu parametrů, které je nutné v konkrétní úloze určit poměrně náročným experimentováním. Zároveň GA nezaručují konvergenci řešení [6].

2.4 Tabu vyhledávání

Tabu vyhledávání [13] je algoritmus lokálního vyhledávání, který z počátečního řešení postupně krok za krokem přechází do jiného řešení, u kterého jsou očekávány optimálnější vlastnosti. V každém kroku lze přejít pouze do jedné z okolních kombinací. Okolí řešení s, $V(s)$ je pak množina jiných výsledků, které se od s liší pouze v jedné proměnné nebo hodnotě. V každém kroku se vybere nejlepší $x \in V(s)$ a to bez ohledu na to, zda je lepší nebo horší jak s. Aby nedocházelo k zacyklení (např. když x bylo horší jak s a v dalším kroku by s bylo nejlepším okolím x), existuje zde tzv. seznam „tabu“ T, který uchovává informace o N posledních krocích. Nejlepší $x \in V(s)$ musí zároveň splňovat $x \notin T$. Metoda končí po vykonání předem stanoveného počtu iterací a výsledkem je doposud nejlepší nalezené řešení.

Postup při využití této techniky pro řešení plánovací úlohy je popsán v [5]. Autor tohoto článku, J. P. Boufflet, zde popsal svoji plánovací úlohu jako barvení grafu, kde vrcholy představují přednášky a hrany různá omezení. Každé omezení a tedy i hrany mělo svou váhu. Cílem bylo obarvení grafu maximálně c barvami. Místo barvení však použil tabu vyhledávání, když v podstatě náhodně obarvil vrcholy grafu a poté redukoval počet konfliktů (hrana se stejně obarvenými uzly) tím, že prohledával nejbližší okolí.

Tabu vyhledávání je rychlá metoda, která však nemá zaručené nalezení nejlepšího řešení. Může se totiž stát, že se zasekne v lokálním extrému. Dokonce ani nemusí najít lepší, než počáteční řešení.

2.5 Constraint satisfaction problem

Constraint satisfaction problem (CSP) je skupina matematických problémů nad množinou proměnných. Každá proměnná má svou doménu (obor hodnot) a množinu omezení, které redukují množství hodnot, kterých mohou dosáhnout jednotlivé proměnné zároveň. Jedná se o úlohu přiřazení hodnot jednotlivým proměnným tak, aby byla splněna všechna kritéria.

Typickým rysem CSP je velká komplexnost vyžadující různé heuristiky a prohledávací metody k nalezení řešení v rozumném čase. Mezi ukázkové příklady řešené jako CSP patří problém N dam, barvení mapy či sudoku. CSP se řeší formou prohledávání stavového prostoru daného doménami proměných (nebo jejich kombinacemi). Techniky řešení lze rozdělit do následujících kategorií:

- backtracking (zpětné prohledávání) – je algoritmus založený na prohledávání stavového prostoru do hloubky. Je to vylepšená forma hledání „hrubou silou“, které však umožňuje dopředu vyloučit mnoho chybných kandidátů na řešení, aniž by bylo potřeba je doopravdy zkoušet [24].
- constraint propagation (propagace omezení) – je mechanismus, kdy se změna v jedné doméně vyvolá změny v doménách ostatních proměných. Tento postup se řetězovitě šíří a pokračuje dál.
- local search (lokální prohledávání) – je metoda, která neplní úplně všechny požadavky, ale snaží se jich splnit co nejvíc. Může najít řešení problému stejně jako může selhat, i když řešení existuje. Pracuje iterativně, kdy v každé iteraci se změnou několika proměnných snaží dosáhnout splnění více omezujících kritérií.

2.6 Constraint handling rules

Constraint handling rules (CHR) a od něj odvozený *Constraint handling in Prolog* (CHIP) jsou jazyky patřící do kategorie *Constraint Logic Programming* [23]. Jazyky CHR resp. CHIP pracují nad třemi typy domén: konečná doména (množina), boolovské hodnoty a racioální čísla. Pro účely této práce zde budou uvedeny pouze konečné domény (množiny).

2.6.1 Číselné a symbolické omezení

Každá proměnná musí mít svůj obor (skalárních) hodnot, který musí být jasně deklarovaný. CHIP obsahuje mj. základní definice omezení pro celá čísla: rovnost (#=), nerovnost ($\#\bar{}$) a porovnání (# <, # <=, # >, # >=). Každé z nich může být použito na libovolný *term* nad doménovou proměnnou či omezením [14].

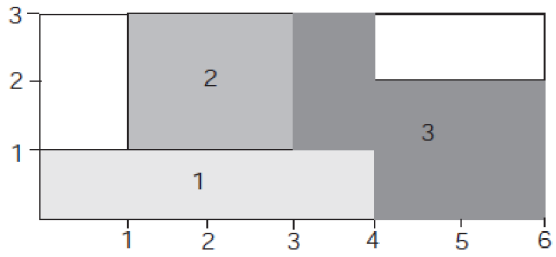
CHIP obsahuje i omezení nad symboly, například `element(N, List, Value)` pro zjištění, zda je `N`-tý prvek seznamu `List` má právě hodnotu `Value`. Toto omezení je v jazycích založených na Prologu zkráceně zapisováno jako `element/3`. V jazyce CHIP se dále uplatňuje princip *constraint propagation*, takže kdykoliv se změní `N` nebo `Value`, tak se automaticky

vytvoří nová omezení a nekonzistentní hodnoty jsou odstraněny. Příkladem, jak může taková událost vzniknout, je přidání omezení `atmost/3` popř. `atleast/3`, který značí, že maximálně resp. minimálně `N` prvků `List` musí mít hodnotu `Value`.

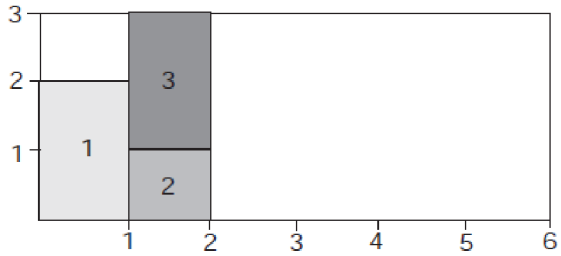
2.6.2 Kumulativní omezení

Některá omezení je však i nadále velmi obtížné popsat výše uvedeným způsobem, což bylo důvodem vzniku tzv. *kumulativního* omezení [3], jež má následující tvar:

`cumulative([S1, ..., Sn], [D1, ..., Dn], [R1, ..., Rn], L)`, kde `[S1, ..., Sn]`, `[D1, ..., Dn]`, `[R1, ..., Rn]` jsou neprázdné seznamy a `L` je kladné přirozené číslo. Toto omezení se používá pro řešení úlohy plánování jednoho zdroje. S_i pro $1 \leq i \leq n$ reprezentuje seznam startovních časů jednotlivých úloh, D_i doby trvání a R_i požadavky na množství zdrojů, které potřebují jednotlivé úlohy. L pak omezuje maximální množství zdrojů, které je v každém okamžiku k dispozici. Tento typ omezení pak zajišťuje, že v každém čase i nelze překročit maximální počet L přiřazených zdrojů. Toto je nejjednodušší typ použití tohoto omezení, ve skutečnosti má totiž až 8 parametrů [8].



Obrázek 2.1: cumulative([1, 2, 4], [4, 2, 3], [1, 2, 2], 3) [14].

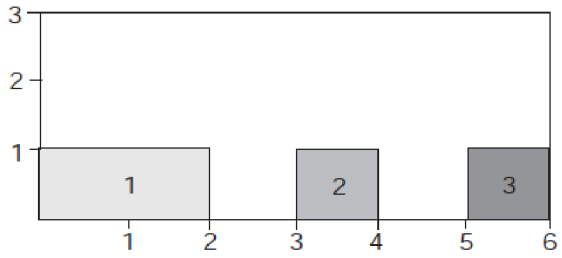


Obrázek 2.2: cumulative([1, 2, 2], [1, 1, 1], [2, 1, 2], 3) [14].

Základními typy použití tohoto omezení jsou následující [3]:

1. na obrázku 2.1 je schéma přiřazení zdrojů třem úlohám: první vyžaduje 1 zdroj po dobu 4 časových úseků, druhý úkol požaduje 2 zdroje po dobu 2 a třetí žádá opět o 2 zdroje a to na dobu 3 časových úseků. V žádném okamžiku nesmí být překročen maximální počet 3 zároveň přiřazených zdrojů.

2. na obrázku 2.2 mají všechny úlohy stejnou dobu trvání (1). Tato situace přesně koresponduje s problémem *rozřazení do košů* [18], kde se n prvků musí rozdělit do m košů o stejné velikosti.



Obrázek 2.3: cumulative([1, 4, 6], [2, 1, 1], [1, 1, 1], 1) [14].

3. na třetím schématu viz 2.3 je pak vyobrazena situace, když žádná z úloh nesmí mít přiřazen víc jak jeden zdroj, což přesně odpovídá situaci, kdy úlohy nemohou být prováděny zároveň (žádají o identický zdroj).

Kapitola 3

Rozvrhování zkoušek v prostředí FIT

Na Fakultě informačních technologií (FIT) Vysokého učení technického v Brně (dále jen VUT) se organizace zkouškového období řídí podle Studijního a zkušebního řádu VUT doplněného směrnicí děkana FIT [2, 1]. Ty obsahují základní nutné pravidla, jež musí být v rozvrhu dodržena. Zbývající podmínky a omezení plynou z typu úlohy (výlučné přiřazení zdrojů) a požadavků vyučujících. V této kapitole jsou postupně popsány všechny včetně jejich chakakteristiky.

3.1 Omezení daná předpisy

Na fakultě resp. na celé škole platí, že akademický rok se dělí na 2 semestry, zimní a letní. Každý semestr má 13 týdnů a je zakončen zkouškovým obdobím trvajícím zpravidla 5 týdnů. Předměty zakončené zkouškou mají 3 termíny zkoušek, které jsou plánovány centrálně. Předpisy umožňující vyučujícím si zvolit tzv. „variantní termíny“, kterých je minimálně 5, ale ty si plánují sami vyučující, tudíž je už dále nebudu uvádět. Pro centrálně plánované zkoušky platí, že nesmí dojít k souběžnému konání zkoušek vyučovaných ve stejném ročníku. Pro opravné termíny těchto předmětů je nutné zveřejnit výsledky předchozí zkoušky nejméně 2 dny dopředu. Zároveň je dáno, že rozvrh zkoušek musí být vytvořen nejméně 5 týdnů před koncem semestru. Z těchto předpisů plynou následující omezení:

- zkouškové období může být delší než 5 týdnů v případě zimního semestru, kdy jsou mezi koncem semestru a začátkem zkouškového období Vánoce. V případě, že svátky začínají např. v pátek, pak předchozí 4 dny lze využít ke konání zkoušek. Podobně i v případě, kdy svátek Nového roku vychází na první dny v týdnu, pak lze využít zbývající pracovní dny tohoto týdne. Důležité je, aby se zkouška konala až po skončení semestru.
- plán zkoušek je vyžadován 5 týdnů před koncem semestru, což je mnohem dřív, než jsou známy počty studentů, kteří budou skutečně připuštěni k absolvování zkoušky. Z tohoto důvodu je nutno počítat se všemi studenty, kteří se zapsali daný předmět. Opačný problém, kdy je alokováno zbytečně velké množství učeben, lze totiž řešit dodatečnou změnou rozesazení až těsně před samotným konáním zkoušky.
- interval mezi zkouškami je minimálně 2 dny kvůli lhůtě na zveřejnění výsledků. Prakticky však musí být mnohem větší, aby byl dostatek času na opravování. U předmětů

s větším počtem studentů lze dále předpokládat delší dobu opravování.

3.2 Požadavky ze strany vyučujících

Asi nejširší škála různých požadavků přichází od vyučujcích. Je to dáno zejména tím, že právě oni určují, jak má zkouška z jejich předmětu vypadat. V případě konkrétních termínů rozhodují o:

- rozesazení studentů, které může mít vliv na snížení kapacity učebny, což vede k navýšení počtu obsazených prostor,
- počtu „turnusů“ (počet po sobě jdoucích skupin v rámci jedné zkoušky)
- době trvání, kdy je nutné zohlednit i dobu přípravy a rezervu

Důležitá je však i dostupnost samotných vyučujících, kteří z různých důvodů nebudou moci v některé dny či hodiny dohlížet na průběh testu. Navíc mohou žádat o konkrétní den, učebny popř. specifikovat nejvíce vyhovující intervaly dnů či hodin. Podstatné jsou i doby potřebné k opravení, zveřejnění výsledků a případné reklamace. To vše musí výsledný rozvrh reflektovat.

3.3 Ostatní omezení

Školní předpisy a požadavky nejsou zdaleka jedinými zdroji omezení. Mezi další můžeme zahrnout různé pravidelné či ojedinělé akce konané na půdě fakulty. Například ve zkouškovém období v zimě se bude pořádá tradiční Den otevřených dveří a v roce 2015 i konference DevConf2015.cz, kvůli kterým nebude v době konání několik učeben k dispozici [11]. Je také nutné zohlednit i obhajoby semestrálních projektů, kdy bude hrozit kolize studentům i vyučujícím magisterského studijního programu.

3.4 Analýza dat

FIT má svůj vlastní informační systém (WIS), který mj. obsahuje údaje o všech studentech a jimi zapsaných předmětech na jednotlivé semestry. Administrátor může exportovat anonymizovaná data ve formátu XML.

Z obrázku 3.1 je patrné, že exportovaná data obsahují seznam studentů. Každý student má na fakultě právě jedno aktivní studium právě jednoho oboru (viz atribut zkr u elementu studium). V rámci tohoto studia má student zapsané předměty. O každém předmětu platí, že má vždy stejných 5 vlastností: zkratku (zkr), akademický rok (akr), semestr (sem), typ a kreditovou náročnost (kred).

Zkratka předmětu je řetězec alfanumerických znaků o délce až 4 znaky. Nejčastěji se jedná o trojici velkých písmen. Akademický rok je ve formátu yyyy a představuje rok, kdy byl předmět zapsán (např. student, který si zapsal předměty v září 2014 na akademický rok 2014/2015 bude mít u všech aktuálně zapsaných předmětů hodnotu 2014). Atribut semestr může nabývat pouze dvou hodnot: Z pro zimní semestr nebo L pro semestr letní. Předposlední vlastnost, typ, charakterizuje vztah studenta k předmětu (daný jeho oborem studia). Typ je má tedy buď hodnotu P pro povinný předmět, PV pro povinně volitelný nebo

```
<?xml version="1.0" encoding="utf-8"?>
<studenti>
	<student>
		<studium zkr="MIS">
			<predmet zkr="ZZN" akr="2014" sem="Z" typ="P" kred="5"></predmet>
			<predmet zkr="DIP" akr="2014" sem="L" typ="P" kred="13"></predmet>
			<predmet zkr="MSZ" akr="2014" sem="L" typ="P" kred="0"></predmet>
			...
		</studium>
	</studium>
	...
</studenti>
```

Obrázek 3.1: Vzorek dat ve formátu XML.

V pro volitený předmět. A nakonec kreditová náročnost, která určuje přibližnou (časovou) náročnost daného předmětu.

Tyto data obsahují úplně všechny zapsané předměty včetně třeba sportů. Pro plánování zkoušek na FIT je ale relevantní jen menší část z nich, které splňují následující:

- předmět musí být zakončen zkouškou
- zkouška musí proběhnout v areálu FIT
- předmět musí být veden jako centrálně plánovaný

V zimním semestru 2014 je takových předmětů 41 z celkového počtu 178. Pro další práci bude také potřeba znát, kolik studentů má společné předměty, přesněji kolik studentů sdílí jednotlivé dvojice předmětů. Pro získání takových údajů je nutné upravit vzorek dat následujícím způsobem:

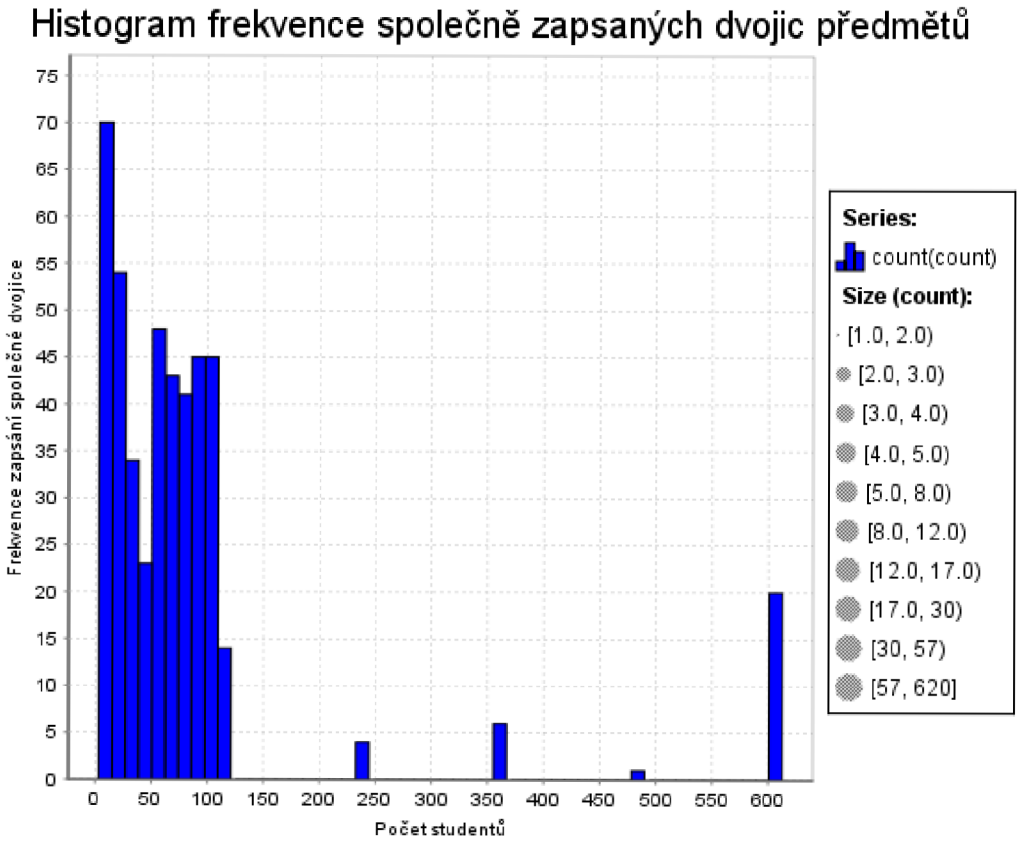
1. rozdělit předměty zapsané jednotlivými studenty do skupin podle semestru a roku
2. odstranit předměty, které nebudou plánovány (nesplňují výše uvedené podmínky)
3. pro každý semestr, rok a studenta je nutné vytvořit kombinaci všech dvojic jím zapsaných předmětů

Takto vzniklé dvojice reprezentují společně zapsané předměty. Počet stejných záznamů pak určuje frekvenci, kolik studentů tyto předměty sdílí.

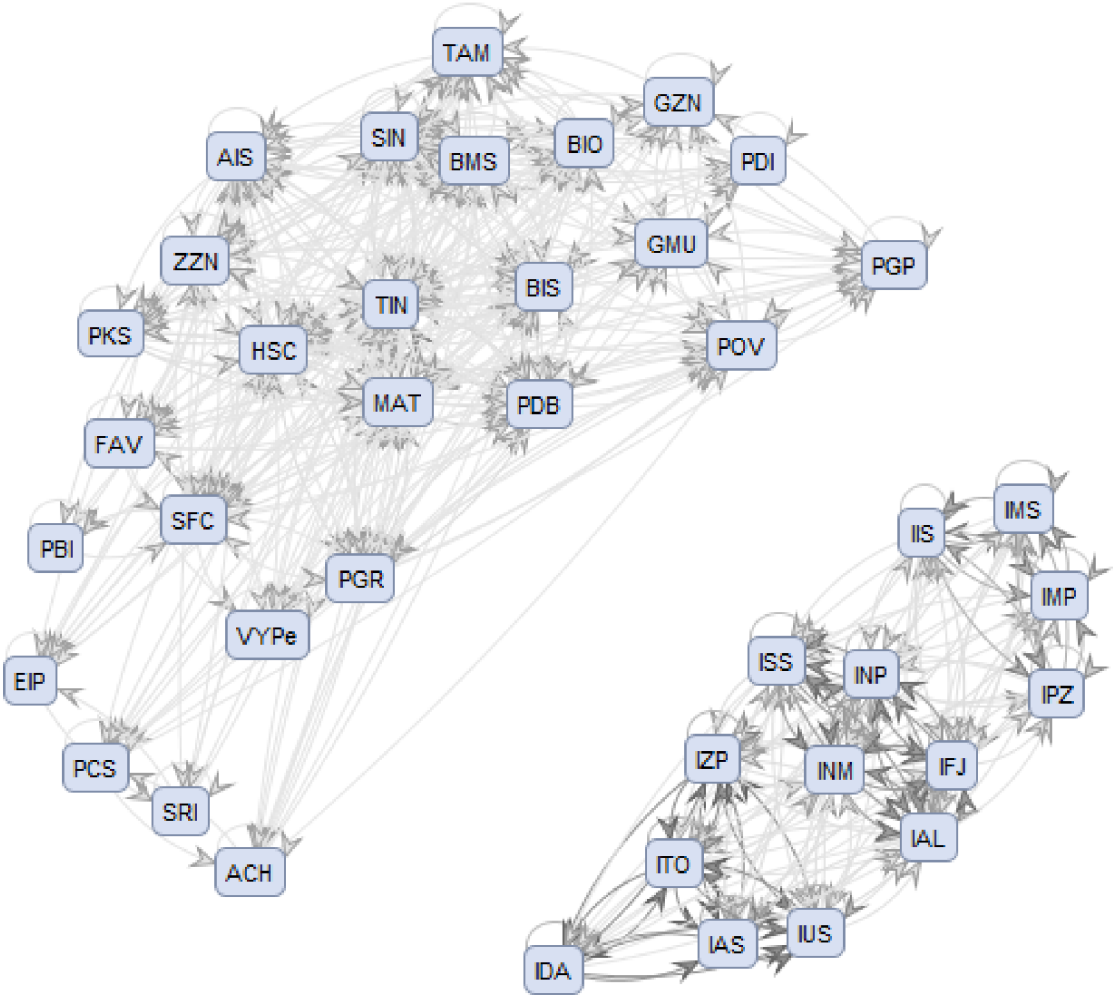
Počet studentů	1-10	11-50	51-100	101-200	201-400	≥ 400
Frekvence	263	134	20	4	7	20

Tabulka 3.1: Konkrétní frekvence výskytů společných předmětů.

Další užitečnou znalostí může být ověření, zda existují skupiny předmětů, které nesdílí žádného studenta. Jednotlivé skupiny pak totiž mohou být plánovány zcela nezávisle. Tento problém vyjádříme pomocí grafu, kde vrcholy jsou jednotlivé předměty a hrany reprezentují společně zapsaný předmět. Abychom mohli využít data z předchozí analýzy, tak musí být tento graf neorientovaný (jinak bychom museli duplikovat hrany v opačném směru). Jednotlivé skupiny předmětů pak získáme algoritmem nalezení silně souvislých komponent tohoto grafu. Výsledek je na obrázku 3.3.



Obrázek 3.2: Histogram frekvence společně zapsaných dvojic předmětů.



Obrázek 3.3: Graf dvojic společně zapsaných předmětů.

Kapitola 4

Návrh metody automatické tvorby rozvrhu zkoušek

4.1 Reprezentace dat

Zdroje v této jsou úloze jednotlivé dny, hodiny a místnosti. Hledané proměnné reprezentující předměty se pak nalézají v následujících doménách:

- *den* :: 1..N, kde N $\geq$ 35 (5 týdnů)
- *hodina* :: 8..18
- *mistnost* :: [D105, D0206, D0207, E112, E104, E105, G202, A112, A113]

Délka zkouškového období je však variantní a doména dní obsahuje i víkendy, kdy se sice nezkouší, ale je třeba tuto informaci zohlednit kvůli minimální době na zvěřejnění výsledků a reklamace. Hodiny jsou modelovány pro reálné časy zkoušek, aby nebylo nutné výsledné relativní časy dále upravovat. Délka zkoušky je interval $\langle a, b \rangle$, $a < b$ a $a, b \in$ *hodina*. 2 zkoušky konané ve stejný den v intervalech $< a, b >$ a $< c, d >$ se prolínají právě tehdy, když platí $c \in (a, b)$ nebo $d \in (a, b)$. Místnosti mají dále odlišný počet sedadel a různou kapacitu závislou na rozesazení studentů. Přehledný seznam místností je v tabulce 4.1.

místnost	kapacita	kapacita přes 1
D105	300	150
D0206	154	77
D0207	90	45
E112	156	78
E104	72	37
E105	72	37
G202	80	40
A112	64	32
A113	64	32
Celkem	1052	528

Tabulka 4.1: Kapacity učeben v závislosti na rozesaní [11].

4.2 CSP model kritérií

Model obsahuje následující pevné omezení:

- C_1 – každá zkouška může být naplánována pouze na jeden souvislý časový úsek
- C_2 – každá zkouška se koná v alespoň jedné místnosti
- C_3 – dvě zkoušky zapsané stejným studentem nesmí mít kolizní termín
- C_4 – dvě zkoušky se stejným vyučujícím nesmí mít kolizní termín
- C_5 – v jedné místnosti se zkouší vždy maximálně jeden předmět ve stejný čas
- C_6 – součet kapacit místností přiřazených ke zkoušce musí být větší než počet studentů účasnících se zkoušky
- C_7 – vyučující nemůže dohlížet na zkoušku, když je nedostupný
- C_8 – v místnosti nelze konat zkoušku, když není dostupná
- C_9 – všechny termíny pevně naplánovaných zkoušek musí být dodrženy
- C_{10} – všechny místnosti pevně přiřazené ke zkoušce musí být respektovány
- C_{11} – termíny některých zkoušek musí předcházet jiným
- C_{12} – předměty mohou mít určenou minimální dobu na opravení
- C_{13} – některé termíny mohou mít vlastní minimální dobu na opravení
- C_{14} – před prvním resp. druhým opravným termínem musí být další minimálně 2 dny (z toho alespoň 1 pracovní) dlouhý rozestup pro sdělení výsledků studentům a případné reklamace
- C_{15} – předměty mají dané rozesazení studentů (vedle sebe nebo přes jedno sedadlo)
- C_{16} – učebny mají různé kapacity
- C_{17} – zkoušky z některých předmětů probíhají ve více turnusech
- C_{18} – v některé dny se nezkouší
- C_{19} – zkoušky se musí konat v rozmezí 8 – 18 hodin

Omezení C_2 až C_4 jsou obecná kritéria pro plánovací úlohy, které zabraňují vícenásobnou alokaci stejných zdrojů (student, učitel a místnost). C_6 a C_{16} zajišťují adekvátní kapacitu místností pro konání zkoušky. V nejhorším možném případě (z pohledu alokace zdrojů) se zkoušky zúčastní všichni studenti, kteří si daný předmět zapsali. V praxi tomu tak není a požadavky na kapacitu učeben zpravidla klesají s každým termínem zkoušky ze stejného předmětu, ale není tomu tak vždy. C_{15} a C_{17} mají vliv na kapacitu učeben, C_{15} při rozesazení přes jedno sedadlo v podstatě zdvojnásobí požadavek na kapacitu, kdežto C_{17} při dvou turnusech snižuje požadavek na kapacitu učeben na polovinu. Mohou se tedy za jisté situace vzájemně vyrušit.

U plánovací úlohy, která obsahuje velké množství pevných omezení lze předpokládat, že nemusí obsahovat žádné řešení splňující všechna kritéria. Při ručním plánování je pak zodpovědná osoba nucena některá omezení porušit, aby zajistila alespoň nějaký výsledek. Zkušený organizátor zkoušek navíc dokáže odhadnout, jak se projeví porušení kterého omezení. Při automatické tvorbě rozvrhu je však nutné toto zahrnout a umožnit tak dynamickou relaxaci jednotlivých kritérií. K tomu je potřeba zavést určitou hierarchii mezi omezeními [12]. Hierarchie kritérií je typicky množina pojmenovaných omezení $c@level$, kde c je omezení na nějakých proměnných a $level$ představuje sílu tohoto omezení [16]. V této úloze bude mít každé omezení svou vlastní váhu. Kritéria s váhou 100 budou v této úloze reprezentovat pevná omezení. Ostatní omezení (s váhou v intervalu $< 0, 100)$) pak budou tvořit základní schéma relaxace omezení.

Porušením jednotlivých omezení takto vznikne penalizace, jehož hodnotu lze popsat následující funkcí:

$$Penalizace_1(P) = \sum_{i=1}^{1} 9(W_i nC_i) \tag{4.1}$$

kde nC_i reprezentuje četnost, kolikrát bylo porušeno omezení C_i, zatímco W_i je váha nebo také výše penalizace pro příslušné kritérium.

Doposud uvedená kritéria mají všechny stejnou (maximální) váhu, protože musí být vždy splněny. Nyní uvedeme seznam volitelných omezení, jejichž míra dodržení bude určovat kvalitu řešení. Model těchto volitelných omezení vypadá následovně:

- S_1 – počet zkoušek zapsaných stejným studentem a naplánované na stejný den
- S_2 – počet zkoušek zapsaných stejným studentem ve dvou po sobě následujících dnech
- S_3 – celkový počet místností pro konkrétní termín zkoušky
- S_4 – procentuální obsazení místností studenty

Omezení S_1 a S_2 určují kvalitu plánu zkoušek z pohledu studenta. Čím více jsou jednotlivé zkoušky od sebe vzdáleny, tím lépe pro studenty. S_3 a S_4 už podrobněji ohodnocují rozmístění studentů do učeben. Cílem je zajistit, aby předměty s velkým počtem studentů byly zkoušeny ve velkých místnostech. Zároveň je však vhodné, aby byly obsazovány efektivně. Pokud například přebývá několik studentů, tak je pro ně alokována nejmenší doposud dostupná učebna, do které se vejdou.

U těchto omezení se očekává, že budou porušeny. Celkovou míru nedodržení popisuje následující funkce:

$$Penalizace_2(P) = \sum_{i=1}^{4} (W_i f_i(S_i)) \tag{4.2}$$

kde $f_i(S_i)$ je funkce popisující míru porušení konkrétního omezení S_i. Tyto funkce ohodnocující nedodržení jednotlivých kritérií se vzájemně liší. Některé mohou být konstatní, jiné vyjádřené tabulkově. Konkrétní hodnoty jsou pak předmětem experimentů v další části práce, ve snaze dosáhnout co nejlepších výsledků. W_i je pak váha volného kritéria S_i.

Příkladem, jak může vypadat funkce pro ohodnocení volitelného omezení, vyjadřuje tabulka 4.2. Z tabulky je zřejmé, že čím víc zkoušek má student v jeden den, tím výrazně větší bude penalizace. Konkrétní hodnoty jsou však zatím pouze orientační.

Počet zkoušek ve stejný den	0	1	2	3	4	≥ 5
Označení	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}	S_{16}
Výče penalizace	0	0	2	5	10	20

Tabulka 4.2: Míra penalizace nedodržení volného omezení S_1.

Kombinací 4.1 a 4.2 máme všechny potřebné části nutné k vyjádření globální funkce ohodnocení rozvrhu zkoušek, kterou budeme minimalizovat:

$$Penalizace(P) = Penalizace_1(P) + Penalizace_2(P) \tag{4.3}$$

4.3 Algoritmus hledání

V této části je uveden algoritmus řešící úlohu plánování rozvrhu zkoušek na FIT na modelu ze sekce 4.2. Hledá tedy takový (sub)optimální plán, který nejvíce uspokojuje všechna omezení. Algoritmus se dělí do několika částí dle konkrétního problému, který je nutné vyřešit:

- nalezení všech kandidátních rozvrhů splňující všechna pevná omezení
- stanovení priorit (pořadí) alokace zdrojů (např. nejprve přiřadit zkoušky do dnů a až pak do místností)
- určit strategie obsazování místností
- výběr nejlepšího kandidáta minimalizací penalizační funkce

Algoritmy uvedené níže představují návrh metodou shora dolů, od abstraktního po konkrétní řešení.

Algoritmus 1 Kostra algoritmu plánování rozvrhu zkoušek

```
Input: Data
Input: C
Input: S
  Q = NajdiVsechnyKandidaty(Data, C)
  if |Q| > 0 then
    return                                  ▷ řešení nebylo nalezeno
  end if
  return VyberNejlepsiZ(Q)
```

Kostru hlavní části vyjadřuje algoritmus 4.3. Má 3 vstupy: *Data*, pevná omezení C a volná omezení S. *Data* zde zahrnují nejen konkrétní seznam učeben, předmětů či učitelů, ale i vzájemné jejich vazby a navíc i údaje z předzpracování, jakými jsou např. i statistiky (např. frekvence dvojic předmětů zapsané společným studentem).

Algoritmus počítá i s variantou, kdy nebylo nalezeno žádné řešení splňující všechna pevná omezení. Odstranění této části je podmíněno experimenty. V případě, že experimenty prokáží, že omezení jsou příliš restriktivní, bude nutné relaxovat jedno či více pevných kritérií.

Algoritmus 2 Nalezení plánu s minimální penalizací

```
function VyberNejlepsiZ(Q, S)
    NejlepsiPlan = null
    PenalizaceNejlepsihoPlanu = ∞
    for all P ∈ Q do
        H = Penalizace(P)
        if H < PenalizaceNejlepsihoPlanu then
            NejlepsiPlan = P
            PenalizaceNejlepsihoPlanu = H
        end if
    end for
    return NejlepsiPlan
end function
```

Funkce VyberNejlepsiZ popsaná v algoritmu 4.3 řeší výběr nejlepšího kandidáta s využitím ohodnocovací funkce 4.3 a představuje finální část celé úlohy.

Jádrem celé tvorby rozvrhu je pak funkce NajdiVsechnyKandidaty. Jejím výstupem je množina řešení, které splňují všechna pevná omezení. Konkrétně zde budou uvedeny dva přístupy realizace této funkce. Prvním z nich je využití automatických nástrojů pro kombinovaná omezení z oblasti *Constraint Logic Programing*, obsažených v nástrojích jako *Prolog* nebo *Haskell* pod souhrným názvem *Constraint Handling Rules* (CHR). Druhým přístup využije klasického úplného prohledávání stavového prostoru, avšak postupně pro jednotlivá omezení a s využitím znalostí konkrétní problematiky.

4.3.1 Nalezení kandidátů s využitím CHR

V této části bude popsána funkce NajdiVsechnyKandidaty formou zápisu pravidel v *CHR*. Některé omezení mohou být vyjádřeny přímo:

- C_2, C_6, C_9 – C_{14} jsou vyjádřeny pomocí omezení porovnáním
- C_{18} odpovídá nedostupnosti všech učeben po celý den
- C_7, C_8 a C_19 jsou omezením domén
- C_3 pro každého studenta nastavíme pro všechny jím zapsané předměty (a tím pádem i zkoušky) výlučný přístup vytvořením kumulativního omezení `cumulative(Zacatky, Trvani, Jedna, 1)`, kde `Zacatky` bude seznam začátků těchto zkoušek, `Trvani` seznamem dob trvání těchto zkoušek a `Jedna` bude obsahovat jen samé hodnoty `1`. V tomto případě budou zkoušky v roli úloh a budou sdílet 1 zdroj (studenta).
- C_4 a C_5 jsou řešeny obdobně, jen s tím rozdílem, že zdrojem je po řadě učitel a místnost
- C_15 je modelováno úpravou dat a to tak, že při rozesazení „přes 1 sedadlo“ bude zvýšen požadavek na kapacitu místností u dané zkoušky na dvojnásobek
- C_17 je podobně jako C_15 omezením požadavku minimální kapacity v učebnách, jen v tomto případě bude pro n turnusů celková minimální kapacita rovna výsledku výrazu p/n, kde p je původní požadovanou kapacitou

- C_1 je zajištěna modelováním C_3 – C_5 právě pomocí kumulativního omezení
- C_16 patří do dat

4.3.2 Nalezení kandidátů pomocí postupné eliminace

Zatímco předchozí přístup s využitím CHR umožňuje poměrně efektivní formu popisu problému a zajišťuje nalezení řešení (pokud existuje), musí se však vypořádat s výrazně více kombinacemi rozvrhů, než je skutečně nutné. Uvažujeme-li například, že potřebujeme naplánovat tři zkoušky do jednoho dne. Kombinací obsazení místností v čase bude mnoho. Některé z nich však budou lepší, než jiné v závislosti např. na omezení S_3 a S_4. Takových kombinací bude N. Pokud bychom uvažovali ještě jeden zkouškový den s obecně jinými předměty, tak opět vznikne obecně M možností obsazení místností, což dává dohromady $N \times M$ variant. V případě zimního semestru 2014, kde bylo celkem 123 zkoušek (41 předmětů, 3 termíny pro každý z nich) a pracovních dní, což bylo přibližně 5 zkoušek denně. Způsobů naplánování 5-ti zkoušek je obecně X, pro 25 dní to je přibližně X^{25} kombinací, což už je opravdu velké číslo.

Cílem této metody je eliminace tohoto velkého počtu na konstantní 1 dílčí řešení tak, že se postupně pro každý den provede minimalizace konkrétního obsazení místností. Tím vznikne pro každý den jediný plán a tedy i $1^{25} = 1$.

Algoritmus 3 Hledání jednotlivých plánů zkoušek

```
function NajdiVsechnyKandidaty
    K = ∅                                          ▷ Množina kandidátů
    Q                                              ▷ Kombinace dnů a zkoušek
    for all kombinace ∈ Q do
        kandidat = ∅
        for all (den, zkousky) ∈ kombinace do
            denniPlan = NapladujDen(den, predmety)
            kandidat = kandidat ∪ denniPlan
        end for
        K = K ∪ kandidat
    end for
    return K
end function
```

Funkce 4.3.2 ukazuje, jak by tato metoda pro odstranění zbytečných denních plánů mohla vypadat.

$$Penalizace_m(P) = \sum_{i=3}^{4}(W_i f_i(S_i)) \tag{4.4}$$

Vztah 4.4 vyjadřuje upravenou rovnici 4.2, ve které jsou vyjádřeny pouze taková volná omezení, která mají vliv na plánování rozmístění zkoušek do místností v rámci jednoho dne.

Poslední část této metody, funkce 4.3.2, řeší minimalizaci rozmístění zkoušek u místností tak, jak je uvedeno ve vztahu 4.4.

Algoritmus 4 Nalezení denního plánu s minimální penalizací

```
function NapladujDen(den, predmety)
    NejlepsiPlan = null
    PenalizaceNejlepsihoPlanu = ∞
    Q                                   ▷ Všechny kombinace zkoušek a místností
    for all P ∈ Q do
        H = Penalizace_m(P)
        if H < PenalizaceNejlepsihoPlanu then
            NejlepsiPlan = P
            PenalizaceNejlepsihoPlanu = H
        end if
    end for
    return NejlepsiPlan
end function
```

Kapitola 5

Implementace v prostředí SWI Prolog

V následující kapitole jsou uvedeny některé implementační problémy a zajímavosti, se kterými jsem se během realizace této práce setkal a které jsem musel vyřešit. Mezi vůbec první důležité rozhodnutí, které jsem učinil, byla volba programovacího jazyka. Po zralé úvaze jsem zvolil jazyk Prolog a to ve volně dostupné podobě ve verzi SWI Prolog v6.6. Tento logický programovací jazyk má totiž většinu potřebných algoritmů již zabudovaných ve svých knihovnách a výpočet podobných plánovacích úloh je v něm možné řešit velmi elegantně a efektivně.

V této kapitole tedy uvádím základní informace nutné k pochopení reprezentace dat a jejich vyhodnocení. Jsou zde popsány vlastnosti stěžejní knihovny pro řešení *constraint satisfaction* problému, její mechanismy pro vyčíslování proměnných a jejich vliv na výslednou podobu hledaného řešení. V závěrečné sekci této kapitoly 5.4 je pak popsán formát vstupu a výstupu do výsledné aplikace.

5.1 Constraint Logic Programming over Finite Domains v SWI Prolog

SWI Prolog je volně dostupná implementace jazyka Prolog. Je nutné zmínit, že se nejedná o *Constraint logic solver*, tedy není to jazyk určený výhradně k řešení podobných úloh, nicméně obsahuje několik knihoven jako rozšíření, které tuto funkcionalitu zajišťují. Při implementaci jsem použil knihovnu *Constraint Logic Programming over Finite Domains* (dále jen CLP/FD), neboť pracuje nad celými čísly (další knihovny jsou omezeny např. na binární aritmetiku nebo naopak rozšířeny třeba na racionální čísla). Knihovna CLP/FD je v podstatě realizací CHR v prostředí SWI Prolog. Od původního návrhu se liší jen v detailech a přidává pár dalších vlastních predikátů pro snadnější práci.

5.1.1 Predikáty a operace knihovny CLP/FD

Asi nejpodstatnější rozdíl je v realizaci predikátu `cumulative/2`, jež má v této knihovně pouze 2 argumenty, seznam úloh a seznam možností (reálně se však používá pouze omezení na maximální počet zdrojů). Seznam úloh má pak tvar podobný původnímu predikátu `cumulative/6`, tedy po řadě seznamy startujících časů, délek trvání, koncových časů, počet požadovaných zdrojů a jednoznačný identifikátor samotné úlohy.

Druhým nejdůležitějším predikátem je `tuples_in`. Tento predikát ověřuje, zda vstupní entice (první argument) je prvkem relace, jež je svým výčtem zadána jako druhý argument predikátu `tuples_in`. Zajištění pravdivosti uvnitř tohoto výrazu se realizuje velkým množstvím implikací, což je paměťově náročná operace, nicméně z hlediska vyčíslování proměnných jde o velmi rychlé řešení. Přiřazením hodnoty k proměnné dojde totiž k výraznému „ořezání“ domén závislých proměnných.

```
?- tuples_in([[X,Y]], [[1,2],[1,5],[4,0],[4,3]]), X = 4.
X = 4,
Y in 0\/3.
```

Příklad 5.1: Příklad predikátu `tuples_in`.

V příkladu 5.1 je ukázka použití predikátu `tuples_in` na dvojici proměnných X a Y. Vzhledem k tomu, že proměnná X je následně unifikována na hodnotu 4, tak dochází automaticky k „ořezání“ domény proměnné Y, která posléze může nabývat pouze hodnoty 0 nebo 3. Další ukázkou redukce domény proměnných X a Y sváznaných tímto predikátem je v příkladu 5.2, kde je proměnná Y shora omezená hodnotou 3.

```
?- tuples_in([[X,Y]], [[1,2],[1,5],[4,0],[4,3]]), Y #< 3.
X in 1\/4,
tuples_in([[X, Y]], [[1, 2], [4, 0]]),
Y in 0\/2.
```

Příklad 5.2: Demonstrace redukce domény a predikátu `tuples_in`.

Relace implikace (# ==>, resp. # <==) a ekvivalence (# <==>) jsou taktéž velmi důležité. Kromě toho, že doplňují standardní aritmetické a logické operace, mají i jednu speciální vlastnost: jako své argumenty akceptují pouze takové výrazy, jež jsou považovány za tvz. „reify“, tedy výrazy, které přímo omezují domény jiných proměnných. Je důležité zmínit, že mezi takové výrazy patří zejména operace porovnání (# >, # >=, # <, # <=, # = a #
=), ale např. i dříve zmíněný predikát `tuples_in`, zatímco predikát `cumulative` zde použít nelze. U `tuples_in` je to dáno právě tím, že vnitřně je reprezentován jako soustava implikací, což u `cumulative` neplatí.

5.1.2 Reprezentace proměnných

Doménové proměnné knihovny CLP/FD jsou výrazně odlišné od běžných proměnných v jazyce Prolog [22]. Nepracuje se s nimi jako s hodnotami, ale jako se soustavou omezení (typicky soustavou rovnic a nerovnic). Díky tomu s nimi lze deklarativně vyjádřit celočíselnou aritmetiku.

```
?- X #> 3, X #= 5 + 2.
X = 7.
```

Příklad 5.3: Příklad pravidel omezení.

Naproti tomu při použití nízkoúrovňových celočíselných aritmetických operátorů Prologu dostaneme:

```
?- X > 3, X is 5 + 2.
ERROR: >/2: Arguments are not sufficiently instantiated
```

Příklad 5.4: Příklad nízkoúrovňové celočíselné aritmetiky.

Při takové reprezentaci hodnot proměnných je však nutné být velmi opatrný. Existuje totiž mnoho užitečných predikátů např. `findall`, `bagof` nebo `assert`, které velmi pěkně simulují funkci cyklu, výběru z dat resp. uložení dat v databázi, nicméně za určitých okolností dochází k vytvoření nových instancí proměnných, místo použití již stávajících, a tudíž může dojít ke ztrátě informací o některých omezeních (dojde k přepsání těch stávajících). Pro iteraci jsem tedy využíval výhradně predikát `maplist`, jež jak název napovídá, mapuje n–tice stejné délky společnou operací.

```
maplist(sum_list, RoomsPowerset, M)
```

Příklad 5.5: Použití predikátu `maplist` pro výpočet celkových kapacit jednotivých kombinací učeben.

Na příkladě 5.5 lze vidět použití tohoto predikátu k získání hodnoty celkové kapacity učeben ke každé z kombinací učeben v `RoomsPowerset` a tyto výsledky jsou pomocí operace `sum_list` uloženy do seznamu `M`. Toto přesně odpovídá části úlohy řešené níže v 5.2.2.

5.2 Implementace jednotlivých omezení

Realizace většiny omezení popsaných v 4.2 je přímočará. V některých případech je však potřeba přidat něco navíc nebo omezující kritérium pojmout z jiného úhlu a vyjádřit ho např. jako kompozici více dílčích omezení. Příkladem budiž třeba omezení týkající se alokace učeben, viz dále, nebo požadavky limitující určité zkoušky na specifickou dobu a místo.

5.2.1 Pevně stanovené termíny a omezování dostupnosti

Pěvně stanovené termíny zkoušek a omezování dostupnosti (ať už učitelů nebo třeba učeben) spolu úzce souvisí. Dá se říct, že fixní termíny jsou speciálním případem omezení dostupnosti v tom smyslu, že všechny ostatní možnosti naplánování již nejsou možné. Například pro letní semestr 2015 [10] přibylo několik typů požadavků:

- „EVO: Kdykoliv od úterý do čtvrtka 9:00 - 17:00. (Doc. Ing. Josef Schwarz, CSc.)“
- „SNT: cokoliv po 9:00 je O.K. (ideálně >=10:00)“
- „PRL: . . . a aby zkouška z PRL nebyla hned na začátku prvního týdne zkouškového, ale nejlépe až po středě 13.5.2015“
- „IPP: . . . Řádný termín ne hned zkraje - kvůli zápočtům, resp. opravě projektů. V podstatě loni se to zvládlo, tak nějak podobně.“

V podstatě se jedná o totéž, nicméně kvůli těmto velmi specifickým požadavkům jsem byl donucen toto omezení dekomponovat na samostatné omezení dnů konání zkoušky, nejčasnějšího času začátku a nejpozdnějšího konce zkoušky a to jak v rámci vyučujících a popř. celého předmětu (všech jeho zkoušek), tak i u jednotlivých termínů viz požadavky ke konání řádných termínů u předmětům PRL a IPP.

5.2.2 Zajištění prostor pro konání zkoušek

Místo konání zkoušky je jedním ze 3 základních údajů o zkoušce, které potřebují studenti a zkoušející znát. V době tvorby rozvrhu je ale tato informace nejméně podstatná a to hned z několika důvodů:

- v době tvorby rozvrhu nejsou známy počty studentů, kteří mohou přistoupit ke zkoušce (např. kvůli nutnosti získat zápočet),
- není ani předem znám počet studentů, kteří budou chtít využít opravných termínů a tato informace bude známa nejdříve během samotného zkouškového období a to navíc jen za předpokladu, že se na zkoušku budou studenti sami přihlašovat (informaci o skutečné účasti by bylo možné odhadnout z předchozích let, nicméně na to nelze bezvýhradně spolenout, protože je nutné zajistit možnost absolvovat zkoušku pro potenciálně všechny studenty na všech termínech,
- zároveň nelze na 100% počítat i s jednotlivými učebnami, neboť může nastat neočekávaná situace a konkrétní učebna pak nemusí být k dispozici. V praxi se navíc rozmístění do učeben řeší až ve zkouškovém období podle skutečného počtu studentů přihlášených na daný termín.

Na druhou stranu striktním přiřazením konkrétních učeben v daný den a čas určitým předmětům zajistí jednoznačnost a bezkonfliktnost v rámci konání jednotlivých zkoušek.

Pro vyřešení úlohy zajištění dostatečných prostor pro konání zkoušky jsem navrh 2 přístupy:

- využít predikát `cumulative` na omezování požadované kapacity studentů a počtu učeben pro případ, kdy explicitně odděluji přiřazení zkoušek do jednotlivých dnů zkouškového období, od hledání existence rozvrhu v jednotlivých dnech , a nebo využít soustavu implikací `==>` pro dvojice zkoušek konané ve stejný den v případě, kdy se problém řeší jako jeden celek
- alternativou je přiřazování konkrétní podmnožiny učeben na základě požadované kapacity a dále jen zajistit vzájemnou disjunktnost podmnožin mezi všemi zkouškami konanými ve stejný den a čas. Jinými slovy využít predikát `tuples_in`.

Zatímco v prvním případě lze použít omezení `cumulative` resp. `==>` přímo, tak v tom druhém již nikoliv a je nutné určitou část výpočtu předzpracovat.

Nechť R je množina všech učeben, $\mathcal{P}(R)$ je potenční množina (množina všech podmnožin množiny R), $c(r)$ pro $r \in R$ značí kapacitu učebny r, pak celková kapacita jednotlivé kombinace učeben $p \in \mathcal{P}(R)$ vyjádřena vztahem

$$M(c) = \sum_{i \in p} c(i) \tag{5.1}$$

Množina všech kapacit všech podmnožin množiny R pak tvoří omezení na hodnoty reálného počtu míst, které mohou být během jednotlivé zkoušky obsazeny, tedy

$$M = \{M(c) | \forall c \in \mathcal{P}(R)\} \tag{5.2}$$

Pro termín zkoušky určený pro s studentů tedy bude platit, že bude nutné rezervovat učebny o celkové kapacitě $m \in M \wedge m \geq s$ míst. V praxi se tímto omezením sníží počet hodnot, jež je nutné prohledat. Například pro 9 učeben, ve kterých se běžně konají zkoušky na FITu vychází 512 kombinací učeben oproti jejich celkové kapacitě přesahující hodnotu 1000. Navíc mají některé učebny stejnou kapacitu, což výsledný rozsah hodnot výrazně snižuje a spolu s omezením na minimální velikost danou počtem studentů jednotlivých předmětů jde už o výraznou redukci domény hodnot. Tímto však ještě není vyřešeno omezení na maximální počet a kapacitu učeben. Toho však lze dosáhnout přímočaře na základě vyjádření kombinací učeben jako prvek potenční množiny všech učeben, tedy pokud jeden předmět obsadí učebny c_i, pak další zkouška z předmětu konaného ve stejný den a kolizní čas může obsadit pouze učebny $c_j \in \mathcal{P}(R) \wedge c_i \cap c_j = \varnothing$. Dvojice všech kombinací učeben, ve kterých se mohou konat zkoušky ve stejný čas bez vzájemného konfliktu místa konání vyjádříme jako

$$Kombinace\_uceben = \{(a, b) \mid a, b \in \mathcal{P}(R) \wedge a \cap b = \varnothing\} \tag{5.3}$$

A právě pro vztah 5.3 můžeme využít predikát `tuples_in` CLP/FD knihovny Prologu tak, že pro dvojici zkoušek zjistíme vhodné kombinace, kterým odpovídají množiny učeben resp. jejich kapacity. Tento predikát vyjádří všechny vyhovující n–tice jako implikace a nemůže se tedy při vyčíslení stát, že bychom se dostali mimo rozsah hodnot například tím, že překročíme hodnotu počtu všech učeben či jejich celkovou kapacitu. Při konání více zkoušek v kolizním termínu se totiž budou vyhodnocovat jednotlivé proměnné, které svým vyčíslením zároveň omezí rozsah hodnot ostatních, jež jsou spolu nějak propojeny (v tomto případě by např. jejich hodnota byla omezena jednotlivými implikacemi).

5.3 Strategie vyčíslení doménových proměnných

Vyčíslením proměnných, tzv. „labeling“, je myšleno systematické přiřazování hodnot proměným z jejich domén tak, aby každé proměnné byla přiřazena konkrétní hodnota z její domény a zároveň aby byly splněny všechny omezující podmínky. Aby k takovému ohodnocení mohlo dojít, musí mít každá proměnná konečnou doménu hodnot. Pro vyčíslení proměnných existuje několik strategií. Některé z nich určují pořadí proměnných, jiné pořadí hodnot z jejich domén či určují vlastní význam toho, co znamená, když proměnná má či naopak nemá konkrétní hodnotu. Predikát odpovídající této operaci se jmenuje příhodně `labeling/2` a za jeho argumenty jsou po řadě seznam proměnných a seznam strategií (pravidel).

Strategie určující pořadí proměnných jsou následující:

- **leftmost** – vyhodnotí proměnné v pořadí, v jakém jsou uvedeny v seznamu proměnných
- **ff** („first fail“) – vyčíslení proměnných podle velikosti jejich domén od nejmenší po největší (tento způsob umožňuje rychlejší odhalení neexistence řešení)
- **ffc** – podobně jako u *ff*, jen zde je určujícím měřítkem počet omezení (od proměnných s nejčastějším výskytem resp. od proměnných popsaných největším počtem rovnic)

- **min** – speciální pravidlo pro konkrétní proměnné, kde prohledávání začíná u proměnných s nejmenším dolním ohraničením domény
- **max** – obdoba pravidla *min* pro horní mez domény

Každá proměnná může nabývat kterékoliv hodnoty z její domény. Dalším typem strategií je preference výběru možných hodnot:

- **up** – výběr hodnot má vzestupnou tendenci
- **down** – výběr hodnot probíhá od největší po nejmenší

Poslední kategorií možností vyčíslení je určení významu samotných hodnot vzhledem k vyhodnocení ostatních proměnných i sebe sama:

- **step** – Pro každou proměnnou X se vybere buď přiřazení $X = V$ nebo omezení $X\# = V$, kde V je hodnota dle aktuálního řazení (*up* resp. *down*)
- **enum** – Postupné dosazení všech hodnot domény seřazených dle dané strategie
- **bisect** – Zúžení výběru hodnot půlením jejich domén. Tento způsob je vhodný zejména tehdy, nejsme–li si jisti rozložením vhodných přiřazení proměnné v rámci celé domény.

Možných strategií je tedy dost, ale platí, že lze použít pouze jednu z každé kategorie. Tato poslední vlastnost je ale problematická. Při implementaci totiž pracuji s proměnnými reprezentujícími různé aspekty rozvrhu (den konání, hodina začátku a konce zkoušky, požadovaná kapacita apod.), ale pro jejich vyhodnocení potřebuji aplikovat více vzájemně konfliktních strategií. Příkladem mohou být penalizační proměnné pro ohodnocení kvality rozvrhu. Pokud bych řešil počet konfliktů (počet dvojic zkoušek naplánovaných na stejný den a sdílících společného studenta), pak bych volil strategii *min*, avšak při současném řešení co největšího intervalu mezi termíny zkoušek pro co největší rozprostření zkoušek do hodin v průběhu jednoho dne, pak bych použil přesně opačnou strategii *max*. A když do toho přidáme i nutnost řešit dny konání ještě před konkrétními hodinami začátku či konce zkoušky, tak máme hned 3 vzájemně konfliktní strategie.

Z tohoto důvodu jsem byl nucen pro každý test volit právě jednu strategii výběru proměnných pro vyčíslení penalizace a samotný výběr pořadí pro vyřešení dnů konání zkoušek přesunout do předzpracování (výchozí stav je totiž dán dle pořadí proměnné v argumentu při volání predikátu `labeling`. Určujícím faktorem jsem zvolil počet požadavků na předmět (termín) a to nejvíce svázané proměnné po tu nejméně (obdoba *ffc*).

5.4 Formát vstupu a výstupu

Pro vytvoření rozvrhu zkoušek je potřeba velké množství různorodých dat, ať už se jedná o informace o zapsaných studentech, vyučujících nebo předmětech, které učí resp. budou osobně zkoušet, až po různé omezení a požadavky na konkrétní zkoušky či prostory.

Bylo by velmi uživatelsky nepřívětivé vkládat tyto údaje ručně a zároveň zbytečně komplikované takové data zpracovávat. Čistě z hlediska hledání rozvrhu zkoušek je způsob zpracování vstupu irelevantní, takže jsem zvolil formát odpovídající přímo faktům a predikátům Prologu samotného, který lze jako celek načíst do aplikace.

K tomuto řešení mě přiměl ještě jeden důvod. Právě vzhledem ke složitým vstupním datům by bylo velmi vhodné vytvořit k této aplikaci i grafické uživatelské rozhraní (GUI).

Jejím úkolem by bylo nejen umožnit vkládat a zobrazovat veškerá data, ale také vytvořit „most“ mezi webovým informačním systémem fakulty (WIS) a touto konzolovou aplikací. V případě existence takového GUI by pak syntaxe resp. API rozhraní této aplikace byla otázkou vnitřní domluvy a nebylo by nutné ji zpřístupnit ven. Stačilo by pouze, aby toto rozhraní bylo schopné vygenerovat validní soubor faktů. GUI však není součástí této práce, ale bude možné jej vytvořit dodatečně.

Formát výstupu, naproti tomu, už musí být velmi dobře čitelný, ale i nadále by měl být snadno strojově zpracovatelný.

```
Rozvrh IZG 1. termin: 2015-05-15 11:00-13:00
Rozvrh FLP 1. termin: 2015-05-15  8:00-11:00
Rozvrh KKO 1. termin: 2015-05-15 11:00-13:00
Rozvrh PRL 2. termin: 2015-05-18  8:00-11:00
Rozvrh IMA 1. termin: 2015-05-18 12:00-15:00
Rozvrh IPP 1. termin: 2015-05-18  8:00-11:00
Rozvrh IPR 2. termin: 2015-05-18 15:00-18:00
Rozvrh AGS 2. termin: 2015-05-18 13:00-15:00
Rozvrh ZRE 2. termin: 2015-05-18 11:00-13:00
Rozvrh IZU 2. termin: 2015-05-19  8:00-11:00
Rozvrh BIF 1. termin: 2015-05-19  8:00-10:00
Rozvrh FYO 1. termin: 2015-05-19 10:00-12:00
```

Příklad 5.6: Výstupní formát aplikace s detaily zkoušky na každém řádku.

Při zapojení striktní části je pak možné rovnou přiřadit i učebny, pak výstup vypadá následovně:

```
Rozvrh IJC 1. termin: 2015-05-11  8:00-10:00 v A112, A113, E104 a E105
Rozvrh IOS 1. termin: 2015-05-11  8:00-13:00 v G202, D0207, D0206,
E112 a D105
Rozvrh IFY 1. termin: 2015-05-12  9:00-11:00 v A112, A113, E105 a D105
Rozvrh IBS 1. termin: 2015-05-14  8:00-10:00 v A113
Rozvrh INC 1. termin: 2015-05-14  8:00-12:00 v A112, E104, E105 a D105
Rozvrh IDS 1. termin: 2015-05-15  8:00-11:00 v E105, E112 a D105
Rozvrh IJC 2. termin: 2015-05-18  8:00-10:00 v A112, A113, E104 a E105
```

Příklad 5.7: Výstupní formát aplikace s detaily zkoušky na každém řádku.

Kapitola 6

Experimenty a jejich vyhodnocení

Při navrhování generátoru zkoušek jsem očekával, že problém nebude jednoduché vyřešit a že výsledná aplikace bude mít tak velké nároky, že ji nebudu moct efektivně testovat na běžném osobním počítači. Odhadoval jsem, že výpočet pro kompletní rozvrh potrvá celý den a já budu moci testovat pouze část rozvrhu. Avšak i tvorba (sub)optimálního rozvrhu zkoušek pro samostatný bakalářský studijní program, čítající přibližně 15 předmětů, v rozumném čase by byl dostatečný úspěch. Tomu jsem také přizpůsobil prvotní vývoj a testování. Časem jsem však dokázal aplikaci natolik zrychlit, že jsem schopen vygenerovat rozvrh v řádu několika minut.

V této kapitole jsou popsány testy a měření, které jsem provedl během vývoje a které shrnují vlastnosti vytvořené aplikace. Jedná se o experimenty měřící zdroje potřebné k běhu programu, tedy čas a paměť. Dále jsem zkoušel různé možnosti počítání penalizace a na jejich základě dále zjišťoval doby nalezení nebo naopak nenalezení rozvrhu s danou hodnotou penalizace. Zkoušel jsem také vliv strategií predikátu `labeling` pro vyčíslení proměnných. Mimo jiné jsem se zaměřoval na největší slabiny celého algoritmu a např. hledal omezení, jež spotřebovávají velké množství zdrojů a snažil jsem se je posléze upravit tak, aby tolik zdrojů již nevyžadovaly. V závěru kapitoly se pak věnuji analýze kvalitativních vlastností vygenerovaných rozvrhů, tedy rozmístění jednotlivých termínů v rámci zkouškového období, rozložení intervalů časů zkoušek v rámci jednotlivých dní či počty kolizí (zkoušek z předmětů sdílících společného studenta).

Vzhledem k tomu, že různé počítače mají různý výkon, snažil jsem se všechny experimenty (vyjma těch s paměťovými nároky) realizovat na stejném stroji a tím se stal virtualizovaný operační systém Debian Wheezy (2 virtuální jádra, 6GB paměti), který běžel pod systémem MS Windows 7 (4 jádra Intel I5, 8GB RAM). Virtuální stroj jsem zvolil proto, abych co nejvíce zredukoval vliv okolního prostředí na běh aplikace (tj. bez připojení na internet, minimum aplikací běžících na pozadí, bez antiviru a podobných programů, jež mohou mít různé naplánované úlohy). Naopak při testech paměťové náročnosti jsem potřeboval zajistit co největší rezervu paměti pro výpočet s cílem co nejvíc oddálit možný *swap* paměti na pevný disk.

6.1 Doba (ne)nalezení vyhovujícího řešení

Jak už bylo psáno v předcházející kapitole, tak se při vyčíslování v SWI Prologu používají strategie výběru pořadí proměnných a pořadí hodnot z jejich domén, pro které se budou hledat řešení nejdříve. V tomto případě máme zajištěno, že jako první se vyhodnotí pena-

lizační proměnná a to od své nejmenší hodnoty tak, aby první nalezené ohodnocení všech proměnných bylo zároveň optimálním řešením, přesněji optimálním z pohledu zvoleného nastavení (strategie penalizace).

Z této stategie vyhledávání a ze značně velkého množství proměnných a velkých rozsahů jejich domén vyplývá, že zatímco nalezení jednoho vyhovujícího řešení, pokud existuje, bude daleko rychlejší, než ověření, že žádné řešení se zvolenými vlastnostmi neexistuje, neboť je nutné projít všechny kombinace hodnot všech proměnných. Tato vlastnost se nejvíce projevuje na rozsahu hodnot samotné penalizační proměnné. Pokud bychom např. hledali ohodnocení rozvrhu zkoušek v rozsahu penalizace 30–35 (bez ohledu na to, co se teď konkrétně myslí hodnotami této penalizace) a optimální řešení existovalo pro penalizaci rovnou hodnotě 31, pak by během výpočtu bylo nutné projít všechny kombinace všech proměnných vyhovujících ohodnocení penalizační proměnné tak, aby byla nejprve rovna 30 a až teprve poté se začne hledat řešení pro její hodnotu 31.

V tabulce 6.1 se nachází výsledky měření doby vyhledání existujícího řešení pro konkrétní hodnotu penalizace a různý počet předmětů. V tomto případě penalizace určuje počet kolizí 2 zkoušek z předmětů sdílících společného studenta a konaných ve stejný den (později se testuje i penalizace podle 3 zkoušek v jeden den). Z uvedených dat vyplývá, že řešení pro celý rozvrh (v tomto případě pro letní semestr 2015) je na virtuálním stroji nalezen za necelé 4 minuty a tato doba se pro snižující počet předmětů jen velmi lehce, téměř lineárně zmenšuje.

S ohledem na tyto měření bylo nutné upravit vyhledávací algoritmus tak, aby hledal rozvrh vždy pro jednu konkrétní hodnotu penalizace a při výrazně delší době hledání tento proces automaticky ukončil jako by řešení s danými parametry vůbec neexistovalo (velmi pravděpodobně nejspíš skutečně neexistuje). Tento přístup má však zásadní nedostatek. Na každém stroji může trvat nalezení řešení pro stejný případ různou dobu. Dokonce i při opakovaném spouštění na stejném stroji a stejnými vstupními hodnotami dochází k výkyvům a to díky rozdílném vytížení procesoru (jádra) dalšími procesy. Tento případ je demonstrován v tabulce 6.1 pro měření doby hledání rozvrhu pro 36 předmětů. Pro přesnější časy při ostrém provozu je tedy vhodné spustit aplikaci nejprve pro vždy vstup se známými výsledky a podle odlišnosti doby vyhledání pak patřičně přizpůsobit časový limit.

Číslo měření	1	2	3	4	5	6	7	8	9	10
Doba běhu [s]	153	156	154	154	155	176	178	173	161	154

Tabulka 6.1: Demonstrace vnějších vlivů na výsledky měření.

Optimálním způsobem hledání rozvrhu zkoušek se tedy stalo spouštění aplikace s konkrétní hodnotou penalizace. Toho lze využít a paralelně spouštět generátor rozvrhu s rozdílnou hodnotou této penalizace a tím zajistit včasnější nalezení řešení. Větší počet paralelně spuštěných procesů má však své limity – aplikace svým během výrazně, ne-li úplně, zatíží celé jádro (procesor), takže spuštěním většího počtu instancí, než je skutečný počet jader, může ve výsledku vést k výraznému zpomalení.

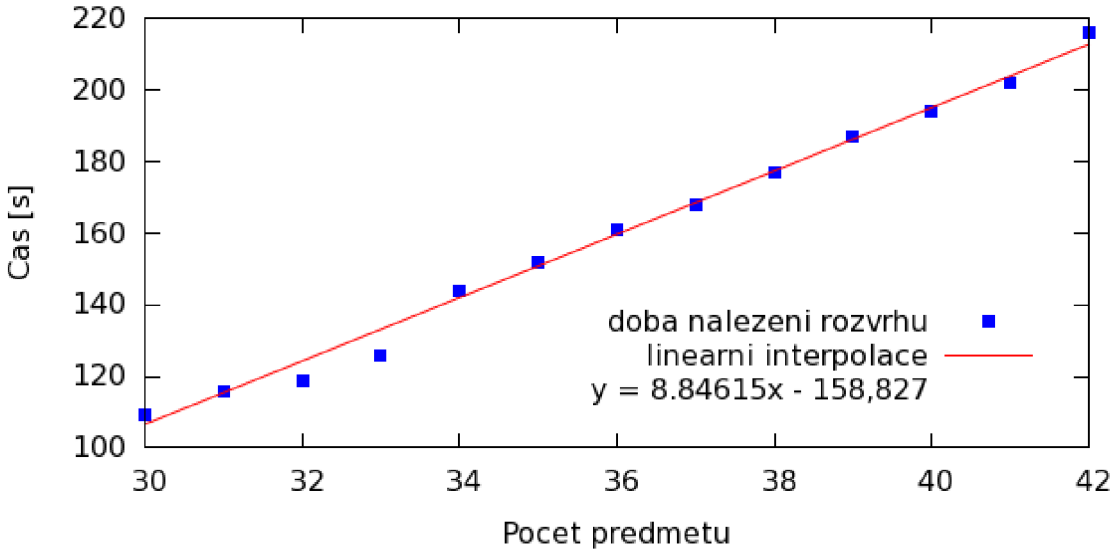
Jsou–li však známy údaje z rozvrhů vygenerovaných pro minulé roky, pak se dá množství konfliktů odhadnout a nalézt tak řešení výrazně rychleji, než při slepém hledání. V ideálním případě by aplikace měla tento údaj odhadnout i bez znalosti historických dat, ale tento úkol už vzhledem ke složitosti celého generování rozvrhu daleko přesahuje rámec této práce, neboť taková úloha by znamenala důkladnou sebeanalýzu (stroje, kde má být aplikace spuštěna) a netriviální schopnost učení z minulých dat (předměty a počty studentů se totiž

každý rok alespoň nepatrně mění).

6.2 Časová složitost výpočtu v závislosti na počtu předmětů

Máme–li tedy strategii pro spouštění a hledání řešení, můžeme měřit reálné doby nalezení rozvrhu pro konkrétní hodnoty penalizace. Toto měření probíhalo postupnou inkrementací hodnoty penalizace do doby nalezení rozvrhu pro n předmětů. Hledání rozvrhu pro $n + 1$ předmětů pak začalo právě na optimálním počtu konfliktů předcházejícího úspěšného měření. Tímto způsobem jsem byl schopen nalézt řešení pro úplně celý rozvrh na letní semestr 2015 čítající 42 předmětů (126 zkoušek). Během 24 dní zkouškového období (1 den bylo rektorské volno) bylo nalezeno celkem 95 konfliktů. Konkrétní výsledky měření jsou v shrnuty v tabulce 6.4.

Vzhledem k vnitřní implementaci jednotlivých omezení a vzájemné provázanosti těchto požadavků (opakovaný vztah každý s každým) jsem očekával, že se tato vlastnost projeví až exponenciálním nárůstem časů nutných k nalezení řešení. Velmi mne tedy překvapilo, že ani pro 42 předmětů se tento charakter růstu složitosti ještě moc neprojevuje. Graf 6.1 ilustruje nárůst doby hledání optimálního řešení vzhledem k počtu předmětů. Tento nárůst velmi připomíná lineární průběh, což potvrzuje i průběh lineární interpolace a hodnoty rozdílů naměřených hodnot od této přímky viz 6.2.

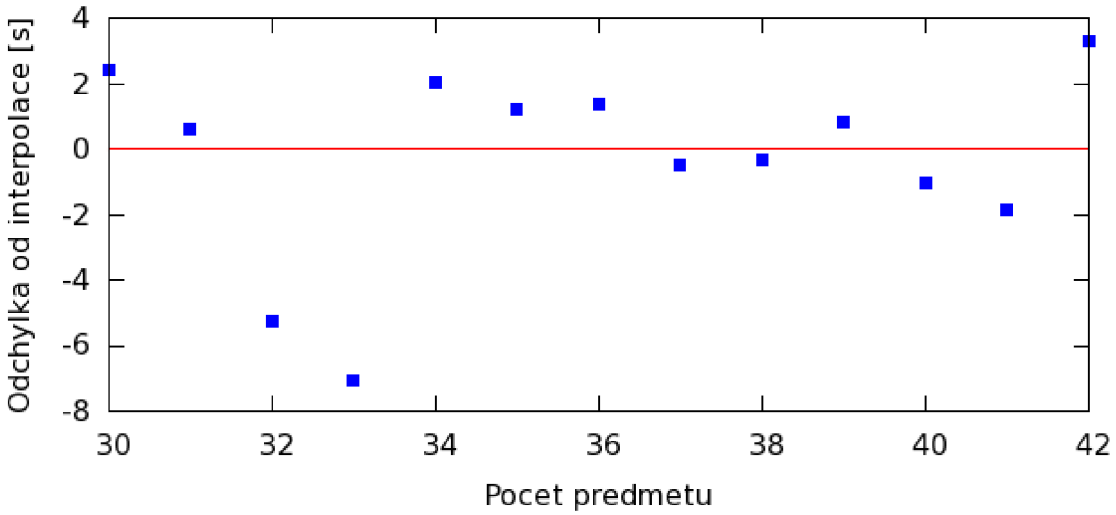


Obrázek 6.1: Graf závislosti doby hledání rozvrhu na počtu předmětů.

K těmto grafům je však nutné dodat, že hodnoty jsou zde pouze pro 13 posledních hodnot (30-42 předmětů). Důvodů je hned několik:

- máme 24 dní zkouškového období, každá zkouška má 3 termíny, takže pokud nemají tyto zkoušky fixně danou kolizi, pak je lze rozmístit zkoušky až z 8–mi předmětů tak, aby kolize mezi nimi neexistovala prostě tak, že se každá zkouška bude konat v jiný den

- při analýze předmětů v kapitole 3.4 se ukázalo, že předměty bakalářského a magisterského studijního programu nemohou mít společné studenty u předmětů zakončených



Obrázek 6.2: Odchylky od lineární interpolace časů hledání rozvrhů.

zkouškou

- čím méně prostoru (dní, hodin a učeben), tím víc roste pravděpodobnost kolize (zkoušku nelze konat v jinak ideálním termínu)

Pokud tyto důvody spojíme, tak zjistíme, že pro přibližně 16 předmětů (8 bakalářských a 8 magisterských) bude rozvrh mít rozvrh stále 0 kolizí, některé předměty pak bude možné vhodně doplnit, aby kolize nadále nerostla a pokud ano, tak třeba jen o jednotky. 30 a víc předmětů už je však už dostatečný počet na to, aby byl zajištěn nárůst počtu kolizí s každým dalším předmětem bez ohledu na pořadí přidávaných předmětů.

Zajímavostí zde zůstává i časová složitost jednotlivých částí výpočtu. Pro připomenutí se běh algoritmu skládá z následujících částí:

- zpracování vstupních dat
- příprava proměnných a navázání jednotlivých omezení
- vyčíslení proměnných
- úprava rozvrhu do čitelné podoby

Při testování jsem zjistil, že samotné vyčíslení je poměrně krátká operace (alespoň vzhledem k ostatním částem výpočtu). Pro všechny předměty letního semestru trvalo samotné vyčíslení pouhých 6 vteřin, zatímco celý běh aplikace trval skoro 4 minuty. Zpracování vstupu je lineárně závislé od jeho délky (pro všechny zde uvedené měření byla doba zpracování v podstatě konstatntní), rekonstrukce je lineárně závislá na počtu předmětů, tedy opět velmi rychlá operace. Nejpomalejší částí výpočtu se tedy stala fáze přípravy všech proměnných a omezení. Je to dáno hlavně tím, že je nutné vytvořit velmi velké množství vazeb ať už mezi termíny stejného předmětu či ošetření situací s možnými konflikty.

6.3 Konflikt strategií vyčíslování - změna pořadí předmětů

Při měření časů pro konkrétní hodnoty penalizace jsem narazil na zajímavou skutečnost. V okamžiku, kdy jsem chtěl přidat do rozvrhu i předmět Paralelní a distribuované algoritmy (dále jen PRL), tak se výpočet výrazně prodloužil (v řádu desítek vteřin) oproti předchozímu nalezenému optimálnímu řešení. Přidání dalších předmětů však již nemělo za následek podobný nárůst.

Za pozornost stojí, že tento předmět má pro letní semestr roku 2015 opravdu velmi velké omezení. Má totiž společného vyučujícího (doc. Zbořil mladší) spolu s předměty Základy umělé inteligence a Agentní a multiagentní systémy (AGS). Zároveň docent Zbořil požadoval, aby se jeho zkoušky nekonaly v intervalu přibližně 10–ti dní [10]. Intuitivně by mělo platit, že při velkém počtu omezení se výrazně sníží i doména možných hodnot (dní či hodin), kdy bude moct zkouška proběhnout. A taky to tak i funguje. Jenže je nutné k tomu připojit i strategii vyčíslení jednotlivých proměnných. Vzhledem k tomu, že se předmět PRL vyskytuje v seznamu předmětů nejméně na 30. pozici a to až za předměty IZU a AGS (předměty jsou přidávány v abecedním pořadí s tím, že nejprve jsou přidány bakalářské předměty a až teprve pak jsou plánovány předměty magisterské), tak interval dnů a hodin, kam by bylo možné zkoušky předmětu PRL naplánovat, je již obsazen jinými zkouškami. Dochází tedy k postupnému zamítání rozvrhu a následuje postupný návrat (*backtracking)* a přeplánování až do okamžiku, kdy se uvolní místo pro všechny termíny zkoušek předmětu PRL nebo se nevyzkouší všechny kombinace a rozvrh je zamítnut jako celek.

Vzhledem k tomu, jak fungují vyčíslovací strategie, je prakticky nemožné tento problém vyřešit přímo v prostředí SWI Prolog, jelikož řešení této situace je v přímém konfliktu s vyhodnocovací strategií a 2 různé strategie stejného typu SWI Prolog nepodporuje,. Je tedy vhodné, aby došlo k určitému předzpracování ještě před spuštěním samotného generátoru zkoušek. Na vhodném místě, v době kdy už budou známy všechny omezení (např. v grafickém uživatelském prostředí jako mezikrok před spuštěním samotného generátoru), je nutné seřadit samotné předměty podle počtu omezení na jednotlivé dny konání.

Pro měření časů jsem nakonec přesunul předmět PRL na první místo mezi magisterskými předměty a do tabulky měřením časů 6.4 jsem zanesl hodnotu času až po novém seřazení předmětů.

6.4 Přísné řešení kapacit a učeben a jeho vliv na paměťovou náročnost

V kapitole o implementaci 5.3 jsem nastínil problematiku řešení alokace učeben pro jednotlivé termíny zkoušek. Pro připomenutí se jednalo o rozpor mezi striktním dodržením pravidla o dostatečném počtu míst pro všechny studenty a reálným počtem studentů, který se zkoušky účastní. Tyto údaje jsou známy až těsně před konáním samotné zkoušky, zatímco rozvrhy musí být uzavřeny nejpozději 6 týdnů před začátkem zkouškového období. Popsal jsem tedy mechanismus, jak přesně zajistit učebny pro jednotlivé zkoušky. V následnujícím měření jsem tedy potřebovat zjistit, jak moc paměťově a časově náročný je takový přesný model výpočtu.

V tabulce 6.2 jsou zaneseny hodnoty jednotlivých měření, ale jen pro 6 až 15 předmětů, neboť se ukázalo, že tento způsob řešení je nepřijatelně paměťově náročný. Tento test jako jediný proběhl na stroji bez virtualizace. Liší se zde i časy. Pro srovnání, vygenerování rozvrhu pro 14 předmětů trvalo aplikaci s tímto rozšířením 165 vteřin, zatímco bez něj se

za stejnou dobu podaří sestavit rozvrh hned pro 37 předmětů, tedy přibližně 2,5krát více.

Počet předmětů	Doba hledání [s]	Paměťová náročnost [MB]
6	29	782
7	37	1075
8	49	1352
9	61	1878
10	83	2263
11	101	2634
12	118	3121
13	132	3775*
14	165	4879
15	289	6981**

** Toto je první případ, kdy už se pravidelně spouštěl garbage collector*
*** Výsledek je velmi ovlivněn swapem paměti.*

Tabulka 6.2: Paměťová náročnost přísného řešení alokace učeben.

Měření využité paměti jsem provedl v systému Windows 7 pomocí *Správce zdrojů* (dostupný ze *Správce úloh*), kde je možné monitorovat množství paměti přidělené jednotlivých procesům v čase. Samotné měření je však již z principu nepřesné. Absolutní množství paměti se z tohoto nástroje nedozvím, pouze mám šanci zahlédnout jednotlivé výkyvy (nárůsty) aktuálně využívané paměti.

Během testování jsem také zaregistroval 2 zajímavé události. První z nich je tzv. *garbage collector*, tedy nástroj programu SWI Prolog, který automaticky uvolňuje paměť. Při měření se spuštění tohoto mechanismu projevilo výrazným poklesem aktuálně využité paměti. Na druhou stranu se prodloužil i běh celého generátoru. Nebylo sice pravidlem, že se tento nástroj spustil, nicméně s rostoucím počtem předmětů se frekvence jeho spuštění zvětšovala.

Druhou nepříjemností se stal nedostatek paměti při měření 15–ti předmětů. Do výpočtu se začal výrazně projevovat systémových mechanismus správy paměti, tzv. *swap*, jež část paměti ukládá a načítá z disku, tedy podstatně pomalejšího média. Pokračovat v dalším měření na svém lokálním stroji již tedy nemělo další význam. Reálné použití tohoto rozšíření na běžném osobním počítači je tedy prakticky vyloučené. V případě FIT by však mělo být možné spustit aplikaci na daleko silnějším stroji, nicméně i tak je nárůst požadované paměti příliš strmý.

6.5 Minimalizace případů 3 zkoušek jednoho studenta v jeden den

Alternativním způsobem hodnocení kvality rozvrhu může být např. i počet případů, kdy mají studenti hned 3 zkoušky v rámci jediného dne. Této situaci se při průměrně 5–ti zkouškách denně nelze zcela vyvarovat. Někdy však ani tahle situace nemusí vadit. Pokud má například student takto naplánované zkoušky až v závěru zkouškového období, tak ho to třeba nemusí ani zajímat, neboť už mohl tou dobou všechny předměty absolvovat (nebo alespoň většinu z nich), takže se ho reálně tyto kolize netýkají. Avšak mít v jeden den 3 řádné termíny, to už se asi nebude líbit žádnému studentovi.

Z hlediska realizace je problém takřka totožný s minimalizací dvojic kolizních zkoušek, avšak situace se rapidně změní, pokud bychom místo minimalizace počtu takových situací

počítali s množstvím studentů, kterých se tyto kolize týkají. Je potřeba si totiž uvědomit, že zatímco počet kombinací trojic předmětů se dá dopředu odhadnout a je tedy možné určit nejhorší případ, tak pro počty studentů už nikoliv. Velikost prohledávané domény se tím pádem výrazně zvětší.

Částečným řešením této problematiky je minimalizace dvojic kolizních předmětů. Pokud má totiž student více zkoušek v jeden den, tak se to projeví i počtu konfliktních dvojic a to podle následujícího vztahu:

$$\binom{n}{2} \tag{6.1}$$

kde n je počet takových zkoušek. Pro 3 kolize se tedy penalizace zvýší o 3, při 4 zkouškách o 6, atd..

6.6 Srovnání současných a vygenerovaných rozvrhů zkoušek

V přílohách A a B se nacházejí vizualizace rozvrhů zkoušek pro letní semestr akademického roku 2014/2015. Už na první pohled je vidět rozdíl v časech konání jednotlivých zkoušek. Zatímco u aktuálního rozvrhu jsou zkoušky přibližně rovnoměrně rozloženy v průběhu celého dne, tak u generovaného tomu tak není. Všechny termíny zkoušek jsou plánovány na co nejdřívější termín, v tomto případě na 8 ráno. Až teprve, když je takto naplánovat nelze, tak se posouvají na pozdější čas.

Takové chování je však zcela v pořádku. Všechna nutná omezení jsou dodržena a je i dostatek volných míst, takže z tohoto hlediska v tom není žádný problém. Naopak, pro mnohé studenty může být takové uspořádání přínosné, neboť se nemusí stresovat dlouhým čekáním na začátek testu. Otázkou však zůstává, zda bude k dispozici dostatek vyučujících pro dozor a to zejména u souběžného zkoušení předmětů ze stejného oboru (ústavu).

U obou těchto rozvrhů je ještě jeden zásadní rozdíl, který opět plyne ze zvolené strategie vyhodnocování proměnných. U současných zkoušek je totiž stále viditelná snaha zachovat původní rozdělení zkouškového období na 3 části: řádné termíny (první 3 týdny), opravné termíny (4. týden) a poslední termíny (5. týden). Počet druhých termínů v prvních třech týdnech je minimální, podobně to platí i pro třetí termíny ve 4. týdnu. Naopak za pozornost stojí zvyšující se počet zkoušek konaných v jeden den v rámci jednotlivých týdnů. Zkouškové období začíná velmi pozvolna a na konci graduje. Naproti tomu generovaný rozvrh je v tomto úplně jiný. Zkoušky jsou v něm plánovány dle jejich pořadí a dny jsou přiřazovány na nejbližší volný termín. Není tedy výjimkou, že všechny 3 termíny zkoušek mohou proběhnout v prvních třech týdnech.

Je tu ještě jedna zajímavost. U současných rozvrhů je obvyklé, že studenti magisterského studijního programu mají zkoušky nahuštěny do prvního týdne a pak mají často pauzu, neboť uvolňují prostor pro konání zkoušek pro bakaláře, jejichž zkoušky často zablokují učebny v celém areálu fakulty. Tato vlastnost může být velmi jednoduše regulována právě změnou pořadí předmětů.

V tabulce 6.3 jsou shrnuty naměřené hodnoty, na jejich základě jsem v této práci měřil kvalitu rozvrhu. Je vidět, že v tomto ohledu se rozvrhy až tak neliší. Počet dvojic kolizních předmětů je u vygenerovaného rozvrhu menší, počet studentů majících 3 zkoušky v jeden den je naopak mírně větší. Pro srovnání časů nutných pro vytvoření bohužel nemám všechny potřebné údaje. Dobu manuální tvorby rozvrhů lze totiž zpětně pouze odhadovat a srovnání takových časů není úplně objektivní.

Rozvrh	Počet kolizí dvojic (trojic) zkoušek	Studentů se třemi zkouškami v 1 den
Manuálně vytvořený rozvrh léto 2015	94 (13)	84
Vygenerovaný rozvrh léto 2015	90 (18)	86
Manuálně vytvořený rozvrh zima 2014	116 (16)	91
Vygenerovaný rozvrh zima 2014	102 (16)	125

Tabulka 6.3: Srovnání rozvrhů pro akademický rok 2014/2015.

Počet předmětů	Doba hledání [s]	Počet kolizí dvojic (trojic) zkoušek	Studentů se třemi zkouškami v 1 den
30	109	30 (3)	15
31	116	33 (3)	15
32	119	35 (3)	15
33	126	39 (5)	84
34	144	44 (9)	66
35	152	51 (9)	66
36	161	56 (5)	24
37	168	59 (5)	24
38	177	61 (7)	26
39	187	67 (7)	26
40	194	73 (7)	26
41	202	80 (9)	34
42	216	90 (18)	86

Tabulka 6.4: Statistické údaje měření pro celý rozvrh na letní semestr 2015.

Dále jsem měřil i závislost počtu konfliktů na počtu předmětů. V tabulkách 6.4 a 6.4 jsou naměřené hodnoty pro zimní resp. letní semestr akademického roku 2014/2015. Z údajů vyplývá mj. i to, že výskytům 3 zkoušek v jeden den se společným studentem je opravdu obtížné se vyvarovat. Počet kolizí totiž výrazně roste s každým předmětem, ale intenzita růstu není ani pro 40 předmětů nijak extrémně vysoká, což je mnohem lépe vidět na obrázku 6.3. Graf je proložen kvadratickou interpolací spočtenou pomocí nástoje *Wolfram alpha*. Vhodnější by sice byla exponenciální interpolace, ale ta měla horší výsledky a jednotlivé naměřené hodnoty se značně odchylovaly.

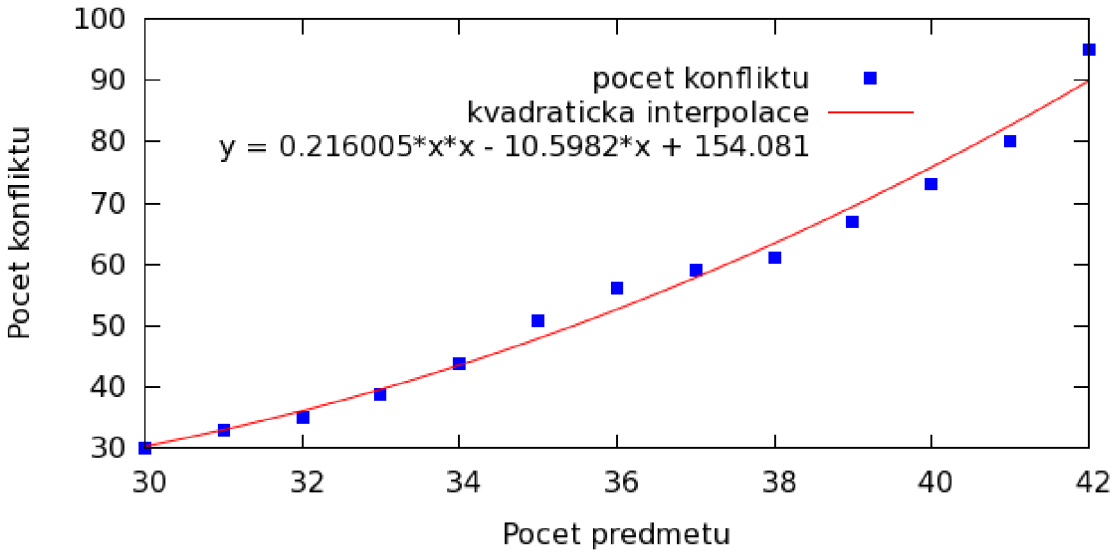
Pokud se ještě jednou vrátíme k více zkouškám absolvovaných v jeden den stejným studentem, tak určitě může nastat situace, kdy jedna zkouška přímo navazuje na druhou. Pro studenta by však bylo dobré, kdyby měl alespoň jednu hodinu volna na odpočinek. Při tvorbě rozvrhu by bylo možné tuto situaci zohlednit následující úpravou:

- doba konání každé zkoušky musí být zvětšena o 1 hodinu
- interval hodin, kdy se na fakultě konají zkoušky prodloužit také o jednu hodinu

Taková změna nemá na reálné konání zkoušek žádný vliv. Stanovené limity času pro psaní zkoušky se nemění, jedná se pouze simulované prodloužení zkoušek, které zabrání vzniku nových konfliktů. Zároveň však studentům zajistí zmiňovanou hodinu na přípravu na další zkoušku.

Počet předmětů	Doba hledání [s]	Počet kolizí dvojic (trojic) zkoušek	Studentů se třemi zkouškami v 1 den
30	117	47 (7)	23
31	121	49 (8)	29
32	133	54 (8)	29
33	139	58 (10)	41
34	148	63 (10)	41
35	153	66 (10)	41
36	168	72 (13)	68
37	169	79 (15)	43
38	182	87 (14)	77
39	191	91 (16)	125
40	202	102 (16)	125

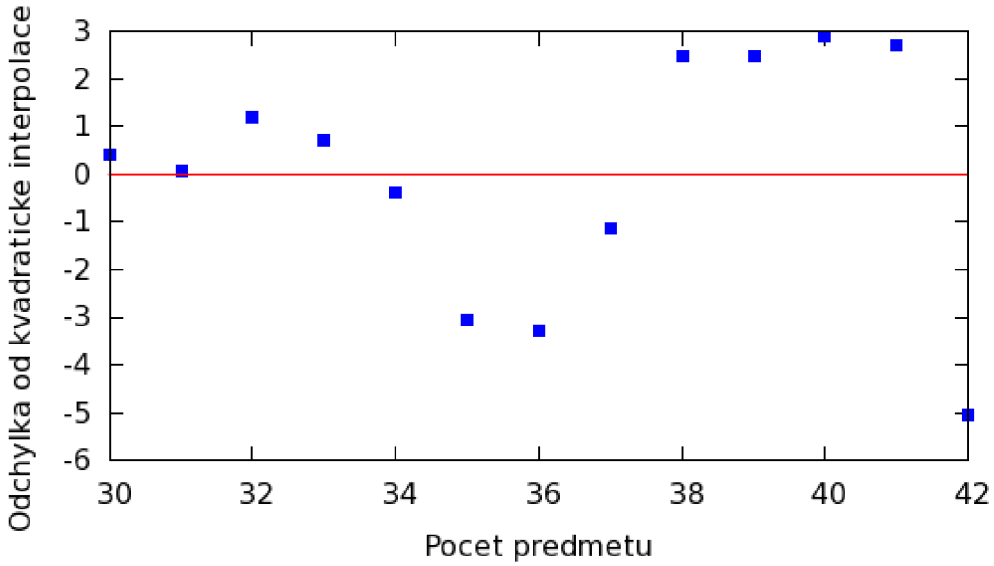
Tabulka 6.5: Statistické údaje měření pro celý rozvrh na zimní semestr 2014.



Obrázek 6.3: Graf závislosti počtu kolizí (dvojic předmětů ve stejný den a společnými studenty) na počtu předmětů.

6.7 Dodatečné úpravy rozvrhu

Rozvrhy generované touto aplikací nemusí mít jediné správné řešení na základě hodnoty penalizační proměnné. Takových řešení může být obecně mnoho a vyčíslovací predikát `labeling` vybere pouze jedno z nich, přesněji vybere první z nich, které najde. Vzhledem k tomu, že jako nejvhodnější se ukázala strategie vyčíslení od nejmenší hodnoty, tak se to projeví i v rozložení zkoušek v rámci jednotlivých dní. Všechny totiž budou v co nejčasnějším termínu. V příloze A je vizualizace jednoho takového řešení rozvrhu zkoušek pro letní semestr 2015. Je vidět, že mnoho zkoušek začíná už v 8 hodin, zatímco odpoledne je jich jen minimum. To mimo jiné znamená, že i při variantě rezervace míst pro všechny zapsané studenty na všechny termíny, zbývá spoustu míst, kde se mohou konat variantně organizované zkoušky.



Obrázek 6.4: Vizualizace odchylek od křivky interpolující nárůst počtu konfliktů.

Rozvrh tak, jak jej vytvoří tato aplikace, však nemusí být konečný. Při dodržení určitých pravidel je možné v něm provádět změny bez obav, že dojde k porušení některého z pevných omezení. Pokusím se zde formulovat základní pravidla pro přesuny předmětů v rámci jednoho dne:

- vybraná zkouška nemá časové omezení, které by zabránilo jejímu konání v nově zvoleném čase a zároveň:
- předmět může být přesunut právě tehdy, když se ve vybraném termínu nekoná žádná zkouška, nebo
- ve zvoleném termínu se konají pouze zkoušky z předmětů bakalářského studijního programu a přesouvaná zkouška patří mezi předměty magisterského studijního programu a naopak, nebo
- ve zvoleném termínu se konají maximálně takové zkoušky, jež jsou již nyní v časové kolizi se zvolenou zkouškou (v libovolném termínu)

Avšak ani porušení těchto pravidel (vyjma toho prvního) nemusí nutně znamenat problém. Bylo by ale nutné provést opravdu důslednou manuální kontrolu, které se však touto aplikací snažíme nahradit. Na obrázku 6.5 je pak demonstrován zcela bezpečný přesun zkoušky na jinou hodinu.

Obrázek 6.5: Ukázka dodatečného přesunutí předmětu ZRE na jinou hodinu.

Kapitola 7

Závěr

V této práci jsem popsal metody řešení plánovacích úloh používaných při plánování rozvrhů zkoušek. Dále jsem specifikoval problém navrhování zkoušek v prostředí Fakulty informačních technologií Vysokého učení technického v Brně a vytvořil model odpovídající této úloze. Popsal jsem a analyzoval vzorek dat s ohledem na získání užitečných informací pro další práci. Formuloval jsem postup pro nalezení řešení založený na algoritmech pro řešení *Constraint satisfaction* problému a implementoval jsem konzolovou aplikaci v jazykce SWI Prolog.

Výsledkem mé práce je generátor rozvrhu zkoušek, který v překvapivě krátkém čase několika minut dokáže nalézt vyhovující rozvrh zkoušek pro zvolený počet konfliktů. Nalezení optimálního rozvrhu bez předchozích znalostí je možné řešit paralelním spuštěním aplikace pro různé rozsahy penalizace. Množství takto spuštěných procesů už je závislé od výkonu konkrétního počítače (zejména od počtu jeho jader). Výsledné rozvrhy zkoušek jsem analyzoval a dospěl k závěru, že ačkoliv se rozložení jednotlivých zkoušek značně liší od současné podoby rozvrhů, tak z hlediska kvality jsou srovnatelné. V počtu konfliktů jsou nové rozvrhy lepší, ale např. opravné termíny se objevují často už v první polovině zkouškového období. Ačkoliv tedy vygenerované rozvrhy splňují daná kritéria, tak to nutně neznamená, že se budou používat ve stejné podobě. Kritérium kvality je totiž v tomto případě subjektivní pojem, na kterém se jen stěží shodnou sami studenti.

7.1 Otevřené problémy

Během práce na tomto projektu jsem narazil na několik problémů, které jsem nebyl schopen vyřešit dokonale nebo alespoň k mé vlastní spokojenosti. Jsou to dílčí vlastnosti celé problematiky, které možná ani nemusí mít sami o sobě úplné řešení, ale mohou být podnětem dalšího zkoumání a zdokonalování této práce.

7.1.1 Problém definice pojmu absolutní kvality rozvrhu zkoušek

Při realizaci jsem se zaměřil zejména na minimalizaci možných konfliktů zkoušek, ale tento pojem je sám o sobě nejednoznačný. Může představovat konání 2 zkoušek ve stejný den a sdílících společného studenta, může to být počet studentů mající dvojici zkoušek ve stejný den. O konfliktu můžeme uvažovat třeba až při 3 předmětech místo dvou. Nebo se na ten problém můžeme podívat z hlediska času studenta na přípravu na jednotlivé termíny a snažit se maximalizovat intervaly mezi zkouškami, aby měl v průměru co nejvíce času opakování a přípravu. Můžeme také zajistit rozvrh tak, aby v určité období byly pouze

řádné termíny (např. v první třech týdnech) a ve zbytku zkouškového pak „nějak“ rozdělit ostatní zkoušky, aby byla splněna alespoň základní nutná omezení bez ohledu na počet konfliktů. Každý z těchto přístupů má své klady a zápory a v mých silách nebylo určit, který z těchto přístupů je absolutně nejlepší. A co víc, těch konfliktů (ať už bereme počty předmětů či studentů) se nikdy nějak výrazně nesníží, protože se pořád jedná o organizaci více jak 120 zkoušek do přibližně 25–ti dní, tedy v průměru 5–ti zkoušek během jediného dne.

V této práci jsem nakonec zvolil jako hlavní měřítko kvality počet konfliktů dvojic předmětů v rámci jednoho dne se společným studentem, neboť mi subjektivně připadalo tohle řešení jako nejspravedlivější. Navíc, jak ukázaly experimenty, hledání hodnoty penalizační proměnné pro počet studentů je velmi pomalé, neboť rozsah hodnot nutný k prohledání je řádově větší než počet konfliktů u dvojic předmětů. Implementace by však mohla být rozšířena (např. volbou z případného grafického uživatelského rozhraní) o možnost zvolit si jednu z více strategií dle vlastního uvážení.

7.1.2 Počty studentů na opravných termínech zkoušek

Definice pojmu kvality rozvrhu není jediným otevřeným problémem, na který jsem při práci narazil. Dalším takovým byla otázka rozlišování počtů studentů na jednotlivých termínech zkoušek ze stejného předmětu. Rozhodl jsem se tyto počty zachovat stejné na všech termínech. Hlavním důvodem byl fakt, že pro konání zkoušky musí být k dispozici dostatečný počet míst a učeben, aby mohla zkouška bez problémů proběhnout a všichni studenti si měli kam sednout. Ačkoliv se počty studentů mohou na těchto termínech značně lišit a výchozím předpokladem je, že s více termíny se bude počet studentů snižovat, tak se i přesto se stále častěji objevují výjimky, kdy na první či druhou opravnou zkoušku dorazí nečekaně velké množství studentů. Pro toto rozhodnutí hovoří také fakt, že reálné počty studentů jdoucích ke zkoušce, nejsou v době tvorby rozvrhu známé a přidělování učeben ke zkouškách se řeší operativně až v průběhu zkouškového období podle počtu aktuálně přihlášených studentů na jednotlivé termíny.

Pro další vylepšení generátoru zkoušek by tedy bylo vhodné se zaměřit na nalezení všech takových případů, kdy počet studentů na zkoušce byl neočekávaně vysoký (např. předčil účast na řádném termínu) a zjistit důvody takového chování studentů. Předpokládám, že jednou z příčin může být případné dodělávání projektů ještě v průběhu zkouškového období, ale takové tvrzení zatím nemám čím podložit. Případné výsledky takového výzkumu by mohly přispět k predikci takových případů a celkově i k upřesnění údajů o požadovaných počtech studentů a tedy i učeben pro jednotlivé termíny zkoušek. Mohlo by to dokonce vést i k definici nových typů omezení s cílem takové situace zcela eliminovat. Zajímavý by mohl být i opačný extrém u předmětů s výrazně vyšší procentální úspěšností na řádném termínu. Takové předměty by pak mohly být vhodnými kandidáty na zvažované snížení počtu termínů zkoušek pouze na dvě, což by mělo opět vliv na další plánování a podobu rozvrhu zkoušek.

7.1.3 Grafické uživatelské rozhraní

V této práci jsem několikrát zmínil grafické uživatelské rozhraní pro tento generátor zkoušek a jeho možné další využití pro zdokonalení tvorby výsledných rozvrhů. Pokusil jsem se o jeho implementaci, ale z časových důvodů jsem toto rozšíření nestihl. Pokud se však tento generátor má prakticky využívat, vytvoření takového rozhraní bude v budoucnu nezbytné. Jeho součástí by měl být i nástroj pro grafickou vizualizaci rozvrhu, ve které by bylo

možné velmi snadno objevit případné opomenuté omezení či chyby. Zároveň by bylo určitě vhodné, kdyby uměl i manuální úpravy rozvrhu a případně implementoval mechanismus učení a pomáhal tak odhadnout počty studentů na opravných termínech.

Literatura

[1] Úplné znění studijního a zkušebního řádu Vysokého učení technického v Brně ze dne 21. července 2011 [online]. [cit. 6. 1. 2015].
URL http://www.fit.vutbr.cz/info/predpisy/szvut-2011-07-21.pdf

[2] Rozhodnutí děkana FIT č.43/2014: Směrnice děkana FIT doplňující Studijní a zkušební řád VUT v Brně [online]. [cit. 6. 1. 2015].
URL http://www.fit.vutbr.cz/info/rd/2014/rd43-141015.pdf

[3] Aggoun, A.; Beldiceanu, N.: Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, ročník 17, č. 7, 1993: s. 57–73, ISSN 0895-7177.

[4] Ahuja, R. K.; Magnanti, T. L.; Orlin, J. B.: Network flows: theory, algorithms, and applications. 1993.

[5] Boufflet, J. P.; Negre, S.: Three methods used to solve an examination timetable problem. In *Practice and Theory of Automated Timetabling*, Springer, 1996, ISBN 3-540-61794-9, s. 325–344.

[6] Burke, E.; Elliman, D.; Weare, R.: A genetic algorithm based university timetabling system. In *Proceedings of the 2nd East-West International Conference on Computer Technologies in Education*, ročník 1, 1994, s. 35–40.

[7] Čangalović, M.; Schreuder, J. A.: Exact colouring algorithm for weighted graphs applied to timetabling problems with lectures of different lengths. *European Journal of Operational Research*, ročník 51, č. 2, 1991: s. 248–258.

[8] Dechter, R.: *Constraint processing*. Morgan Kaufmann, 2003.

[9] Dimopoulou, M.; Miliotis, P.: Implementation of a university course and examination timetabling system. *European Journal of Operational Research*, ročník 130, č. 1, 2001: s. 202–213.

[10] Dytrych, J.: Plán zkoušek na letní semestr 2014/2015 [online]. [cit. 18. 5. 2015].
URL http://www.fit.vutbr.cz/~idytrych/zkousky/zkousky2014L.htm

[11] Dytrych, J.: Plán zkoušek na zimní semestr 2014/2015 [online]. [cit. 6. 1. 2015].
URL http://knot.fit.vutbr.cz/idytrych/zkouskyZS2014.htm

[12] Freuder, E. C.; Wallace, R. J.: Partial constraint satisfaction. *Artificial Intelligence*, ročník 58, č. 1, 1992: s. 21–70.

[13] Glover, F.: Tabu search: A tutorial. *Interfaces*, ročník 20, č. 4, 1990: s. 74–94, ISSN 0092-2102.

[14] Guéret, C.; Jussien, N.; Boizumault, P.; aj.: Building university timetables using constraint logic programming. In *Practice and Theory of Automated Timetabling*, Springer, 1996, ISBN 3-540-61794-9, s. 130–145.

[15] Lajos, G.: Complete university modular timetabling using constraint logic programming. In *Practice and Theory of Automated Timetabling*, Springer, 1996, ISBN 3-540-61794-9, s. 146–161.

[16] Menezes, F.; Barahona, P.; Codognet, P.: An incremental hierarchical constraint solver applied to a time-tabling problem. In *Proceedings of Avignon*, ročník 93, 1993.

[17] Müller, T.: Real-life Examination Timetabling. *Proceedings of the 6th Multidisciplinary In ternational Scheduling Conference*, 2013.

[18] Rayward-Smith, V.; Shing, M.: Bin packing. *Bulletin of the IMA*, ročník 19, č. 142-146, 1983: str. 49.

[19] Reis, E., Luís Paulo; Oliveira: A Constraint logic Programming Approach to Examination Scheduling. *Proc. Xth Irish Conference on Artificial Intelligence Cognitive Science*, 1999.

[20] Schaerf, A.: A survey of automated timetabling. *Artificial intelligence review*, ročník 13, č. 2, 1999: s. 87–127.

[21] Tripathy, A.: A lagrangean relaxation approach to course timetabling. *Journal of the Operational Research Society*, 1980: s. 599–603.

[22] Triska, M.: The Finite Domain Constraint Solver of SWI-Prolog. In *FLOPS*, *LNCS*, ročník 7294, 2012, ISBN 978-3-642-29821-9, s. 307–316.

[23] Van Hentenryck, P.: *Constraint satisfaction in logic programming*, ročník 5. MIT press Cambridge, 1989, ISBN 0-262-08181-4.

[24] Virius, M.: *Základy algoritmizace*. Česká technika-nakladatelství ČVUT, 2008, ISBN 80-01-01346-4.

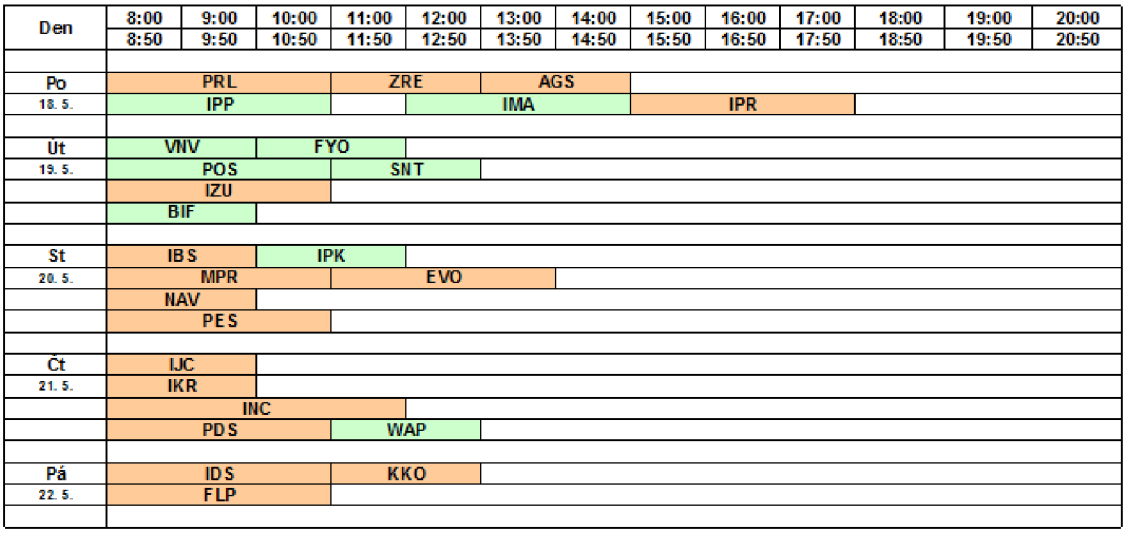
[25] de Werra, D.: An introduction to timetabling. *European Journal of Operational Research*, ročník 19, č. 2, 1985: s. 151–162.

Příloha A

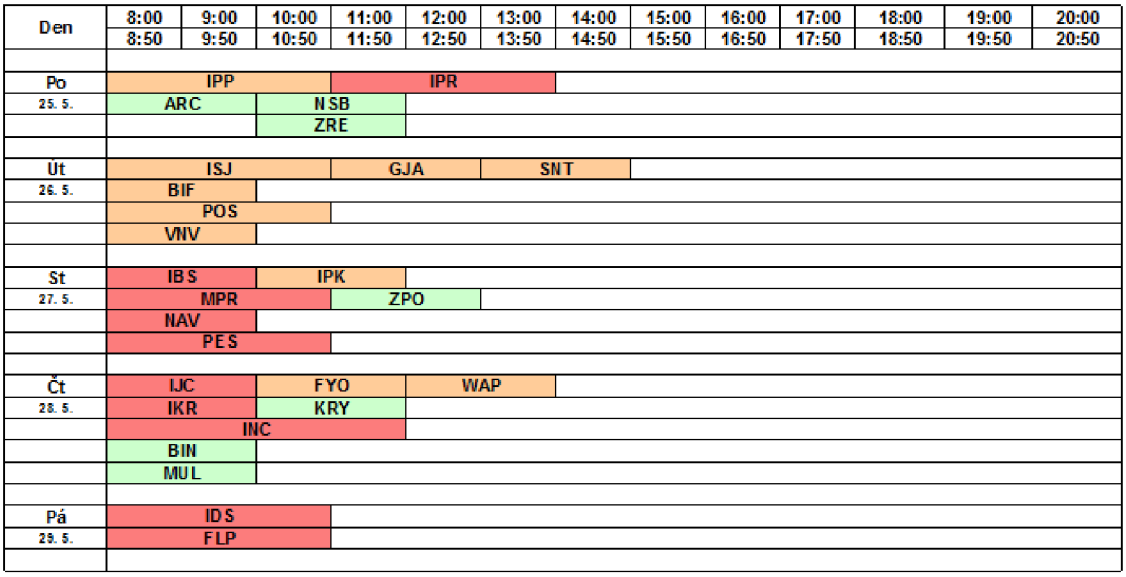
Vizualizace vygenerovaného rozvrhu pro letní semestr 2015

Den	8:00	9:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00
	8:50	9:50	10:50	11:50	12:50	13:50	14:50	15:50	16:50	17:50	18:50	19:50	20:50
Po	PRL			AGS		IPR			IKR				
11. 5.	ISJ			SPP		GJA		ZRE					
	IOS												
Út	IBS			IZU									
12. 5.		IFY		EVO									
	MPR												
	NAV												
	PES												
St													
13. 5.													
Čt	IJC		ITS		PIS		ITP						
14. 5.	INC				GIS		PDS						
Pá	IDS			IZG									
15. 5.	FLP			KKO									

Obrázek A.1: Vygenerovaný rozvrh zkoušek pro 1. týden zkouškového období na léto 2015.



Obrázek A.2: Vygenerovaný rozvrh zkoušek pro 2. týden zkouškového období na léto 2015.



Obrázek A.3: Vygenerovaný rozvrh zkoušek pro 3. týden zkouškového období na léto 2015.

Den	8:00 8:50	9:00 9:50	10:00 10:50	11:00 11:50	12:00 12:50	13:00 13:50	14:00 14:50	15:00 15:50	16:00 16:50	17:00 17:50	18:00 18:50	19:00 19:50	20:00 20:50
Po	PRL			ARC		NSB							
1. 6.	IPP				IMA								
Út	IOS					SNT		VNV					
2. 6.	ISJ			GJA									
	PDS												
St	IZU			IPK									
3. 6.	BIF			ZPO									
	POS												
Čt	ITP		BIN		KRY		WAP						
4. 6.	AGS												
		IFY											
	MUL												
	SPP												
Pá	EVO		IZG										
5. 6.	ITS		GIS										
			PIS										

Obrázek A.4: Vygenerovaný rozvrh zkoušek pro 4. týden zkouškového období na léto 2015.

Den	8:00 8:50	9:00 9:50	10:00 10:50	11:00 11:50	12:00 12:50	13:00 13:50	14:00 14:50	15:00 15:50	16:00 16:50	17:00 17:50	18:00 18:50	19:00 19:50	20:00 20:50
Po	ARC		NSB										
8. 6.													
Út	KKO		FYO										
9. 6.													
St	ZPO				IMA								
10. 6.													
Čt	IOS					ITP							
11. 6.	BIN		KRY		SPP								
	MUL												
Pá	GIS			IZG									
12. 6.	PIS												
	ITS												
		IFY											

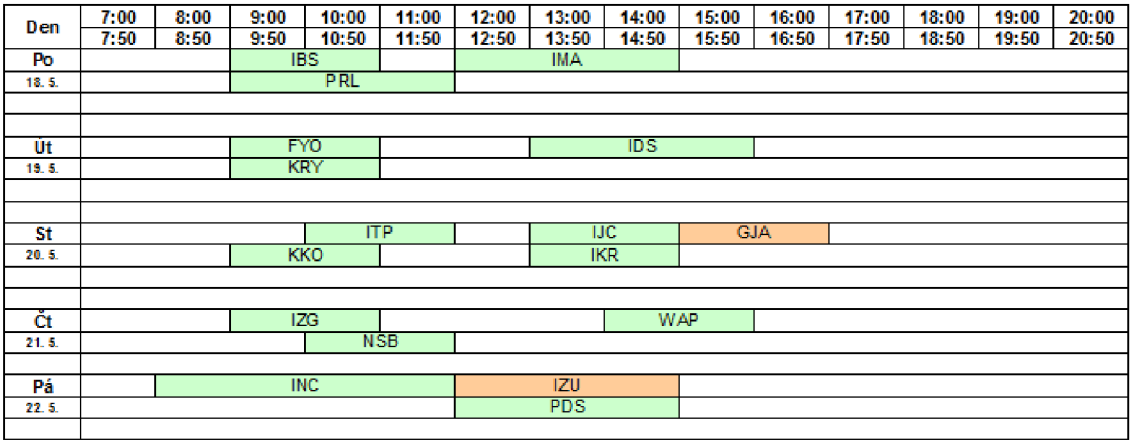
Obrázek A.5: Vygenerovaný rozvrh zkoušek pro 5. týden zkouškového období na léto 2015.

Příloha B

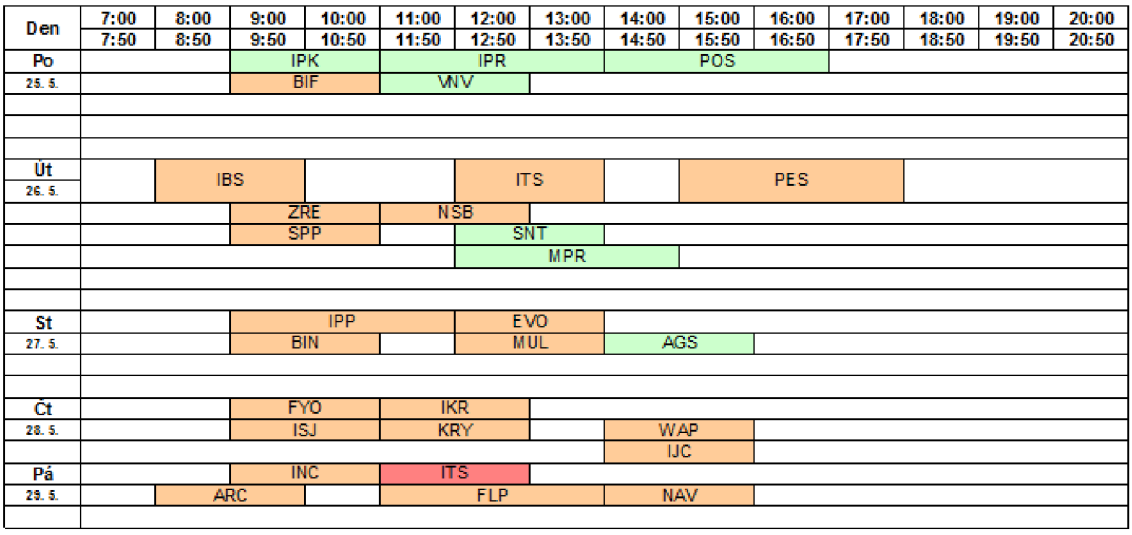
Vizualizace stávajícího rozvrhu pro letní semestr 2015

Den	7:00 7:50	8:00 8:50	9:00 9:50	10:00 10:50	11:00 11:50	12:00 12:50	13:00 13:50	14:00 14:50	15:00 15:50	16:00 16:50	17:00 17:50	18:00 18:50	19:00 19:50	20:00 20:50
Po 11. 5.			ISJ				IZU							
					BIF			ZPO		GIS				
Út 12. 5.			IFY					ITS						
			ZRE			SPP			PES					
			PIS			BIN								
St 13. 5.														
Čt 14. 5.				IPP										
		ARC					GJA		EVO					
									MUL					
Pá 15. 5.		FLP			IOS									
							NAV							

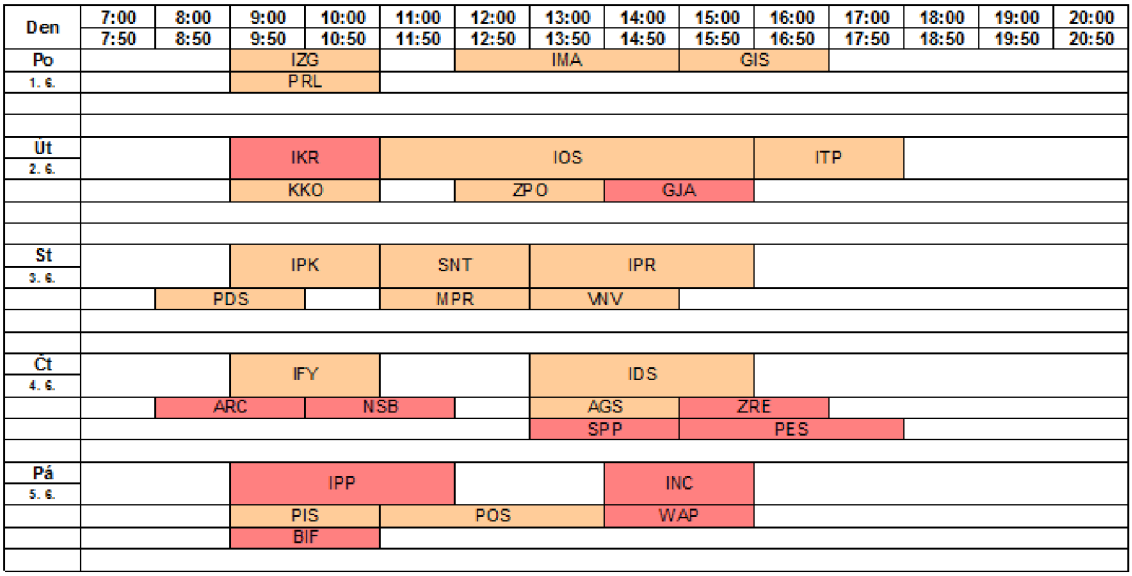
Obrázek B.1: Manuálně vytvořený rozvrh zkoušek pro 1. týden zkouškového období na léto 2015.



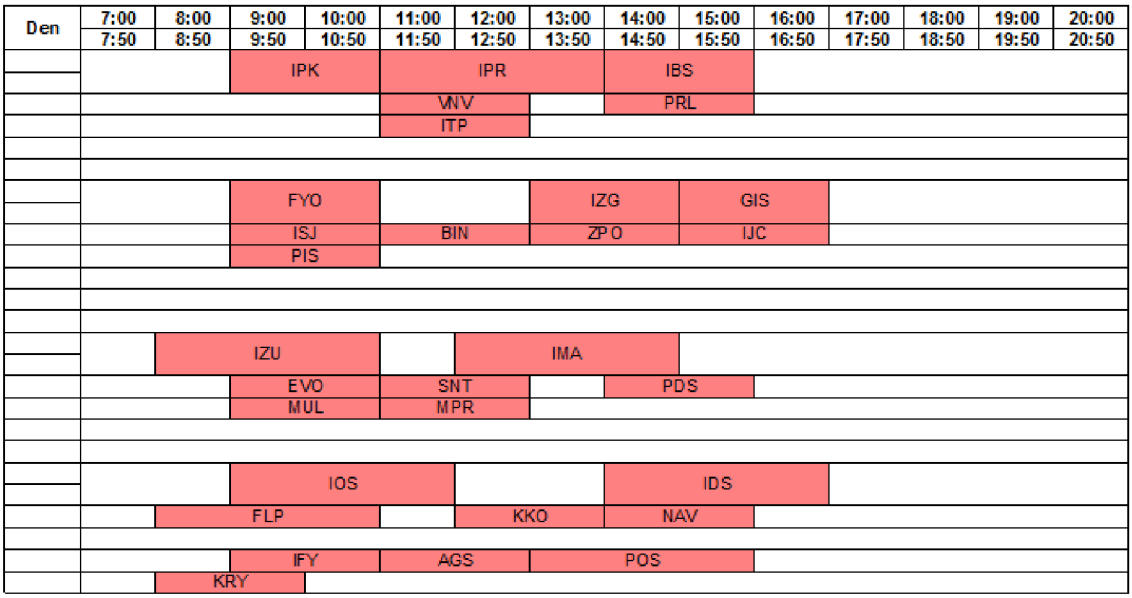
Obrázek B.2: Manuálně vytvořený rozvrh zkoušek pro 2. týden zkouškového období na léto 2015.



Obrázek B.3: Manuálně vytvořený rozvrh zkoušek pro 3. týden zkouškového období na léto 2015.



Obrázek B.4: Manuálně vytvořený rozvrh zkoušek pro 4. týden zkouškového období na léto 2015.



Obrázek B.5: Manuálně vytvořený rozvrh zkoušek pro 5. týden zkouškového období na léto 2015.

Příloha C

Obsah CD

Přiložený datový nosič má následující obsah:

- Adresář `src` – obsahuje zdrojové soubory v jazyce Prolog a skript umožňující paralelní spouštění
- Adresář `doc` – obsahuje zdrojové soubory textové části pro systém LaTeX
- Adresář `data` – obsahuje ukázky zpracovávaných vstupních dat a příklady výstupů
- `thesis.pdf` – tento dokument v elektronické podobě
- `README.txt` – dokument s požadavky a pokyny pro spuštění