Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Luděk Cigler

Analýza a implementace algoritmů pro sestavování středoškolských rozvrhů

Katedra aplikované matematiky

Vedoucí bakalářské práce: RNDr. Martin Pergel
Studijní program: Obecná informatika

2006

Poděkování patří mému vedoucímu RNDr. Martinu Pergelovi za jeho pomoc při přípravě této práce. Rovněž bych chtěl poděkovat Adamu Böckovi z Gymnázia Postupická v Praze 4 za poskytnutí testovacích dat.

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

V Praze dne 22. května 2006 Luděk Cigler

Obsah

Název práce: Analýza a implementace algoritmů pro sestavování středoškolských rozvrhů
Autor: Luděk Cigler
Katedra (ústav): Katedra aplikované matematiky
Vedoucí bakalářské práce: RNDr. Martin Pergel
e-mail vedoucího: perm@kam.mff.cuni.cz

Abstrakt: Rozvrhování představuje zajímavý a obtížný problém kombinatorické optimalizace. V této práci studujeme přístupy používané pro automatické sestavování rozvrhů pro střední školy. Popisujeme implementaci rozvrhovacího systému, který kombinuje rozvrhování pomocí programování s omezujícími podmínkami a techniky lokálního prohledávání. Systém umožňuje uživateli provádět ruční úpravy rozvrhu během výpočtu.

Klíčová slova: rozvrhování, lokální prohledávání, programování s omezujícími podmínkami

Title: Analysis and implementation of algorithms for secondary school timetables scheduling
Author: Luděk Cigler
Department: Department of Applied Mathematics
Supervisor: RNDr. Martin Pergel
Supervisor's e-mail address: perm@kam.mff.cuni.cz

Abstract: Scheduling represents an interesting and difficult problem of combinatorial optimization. In the present work we study approaches used for automated secondary school timetabling. We describe an implementation of a timetabling system, which combines constraint programming with local search techniques. The system allows user to make manual changes in the timetable during the computation.

Keywords: timetabling, local search, constraint programming

Kapitola 1

Úvod

Na počátku každého školního roku očekávají studenti, ale i učitelé s napětím, jaký že pro ně osud připravil rozvrh jejich dnů a hodin ve školních škamnách. Někdo se na nový rozvrh těší, jiný ho přijímá se smíšenými pocity, pro dalšího je to o důvod více, proč si v době školní docházky zařizovat návštěvy lékařů, babiček, či sportovních utkání.

Skutečně, při tvorbě rozvrhu asi nelze vyhovět všem – o to více nezáviděníhodná je pozice školního rozvrháře, který často mnoho hodin vytváří rozvrh ručně, povětšinou z již existujícího rozvrhu loňského. Počítačový program řešící tento problém je tedy více než žádoucí, ať už z pohledu úspory práce, či z pohledu „přenosu zodpovědnosti“ na stroj[1].

Ve školním rozvrhování (v anglické literatuře označovaném obvykle *timetabling*), speciálním případu rozvrhování obecného (anglicky *scheduling*), je cílem přiřadit rozvrhované položky (studenty, učitele a vyučované předměty) do předem vymezených časových oken a místností. Přitom je potřeba zaručit, aby nedocházelo ke kolizím: jeden učitel nesmí učit více hodin současně, studenti mají jen jednu hodinu najednou, v místnosti probíhá v daný čas jen jedna výuka.

Klasické rozvrhování je problém statický, tj. po zadání vstupních dat se již podmínky úlohy nemění. Vstupní podmínky ovšem nemohou přesně vyjádřit všechny nuance rozvrhu, a proto je vhodné, aby byl průběh výpočtu ovlivnitelný člověkem. Cílem této práce je popsat rozvrhovací systém, který umožňuje sledovat průběh automatického rozvrhování a ručně upravovat aktuální výsledek výpočtu.

[1] Ukazuje se totiž, že lidé jsou mnohem svolnější k činnostem, které jsou jim nařízeny strojem (teoreticky nestranným), než člověkem, na kterého si mohou patřičně došlápnout.

Tato práce je organizována následujícím způsobem. V následující kapitole se nejprve zmíníme o různých pohledech na rozvrhování a zamyslíme se nad tím, proč je tento problém tak zajímavý a těžký. Podrobněji se podíváme na přístupy využívající programování s omezujícími podmínkami a lokální prohledávání. Nakonec se zamyslíme nad různými postupy pro interaktivní rozvrhování.

Třetí kapitola popisuje model rozvrhovacího problému použitý v rozvrhovacím systému. Tento model umožňuje definovat komplexní požadavky na výsledný rozvrh. Vznikl na základě analýzy požadavků na rozvrh pro konkrétní gymnázium, je však použitelný i pro jiné školy.

Čtvrtá kapitola se zabývá konkrétními algoritmy použitými pro rozvrhování. Tyto algoritmy jsou založeny na použití obecné knihovny pro řešení problémů s omezujícími podmínkami a na lokálním prohledávání.

Nakonec v páté kapitole prezentujeme výsledky experimentů. Data pro rozvrh vychází z dat získaných z několika gymnázií lišících se velikostí i organizací studijního plánu.

Kapitola 2

Základy

2.1 Rozvrhování

Rozvrhování je proces, při kterém jsou úlohy rozdělovány do množiny zdrojů. Zdroje mohou být různého typu – čas, místnosti, výrobní linky, apod. Výsledný rozvrh musí přitom splňovat celou řadu podmínek; tyto podmínky se obvykle dělí na *tvrdé* (*hard constraint*) a *měkké* (*soft constraint*) (jak popisuje Burke, Kingston a Jackson v [BKJ97]).

Tvrdé podmínky nesmí přípustný rozvrh porušovat. Co konkrétně je tvrdá podmínka záleží na zadání úlohy – ve školním rozvrhování se používají například tyto podmínky ([BP02]):

- Žádný učitel nesmí učit dvě hodiny najednou,
- žádní studenti nesmí navštěvovat dvě hodiny najednou,
- v místnosti nesmí probíhat najednou více než jedna hodina.

Měkké podmínky mohou být zadány podobně jako tvrdé podmínky, tj. výčtem (ne)žádoucích situací, obvyklé je však jejich vyjádření formou ohodnocovací funkce ([BP02], [BM02]). Taková ohodnocovací funkce vyjadřuje kvalitu řešení vzhledem ke zvoleným kritériím – může to být například počet mezer v rozvrzích studentů nebo míra naplnění časových preferencí učitelů.

Obtížnost rozvrhování silně závisí na konkrétních podmínkách úlohy. Existují úlohy, které lze řešit snadno polynomiálním algoritmem, pokud přidáme některá omezení navíc (například pokud hodiny mají vždy pevnou délku, učitelé jsou dostupní vždy a máme dostatek místností) nebo se naopak některých omezení zřekneme.

2.1.1 Složitost rozvrhování

Školní rozvrhování je obecně NP-těžký problém – přehled převodů některých NP-úplných problémů (většinou *K-COLOR* nebo *BIN-PACKING*, jejichž definice najdete v [Ka72]) na různě definované rozvrhovací problémy podává Cooper a Kingston v [CK95].

Náš problém rozvrhování na středních školách splňuje taktéž některé charakteristiky NP-těžkých problémů. Učitelé nemusí být dostupní vždy, položky rozvrhu mohou trvat různý počet vyučovacích hodin a studenti si mohou vybírat semináře dle své chuti. Tak mohou existovat skupiny studentů s neprázdným průnikem.

Definujme rozvrhovací problém takto: Vstupem problému je seznam dvojic (učitel, skupina studentů) – položek rozvrhu. Různé skupiny studentů mohou mít neprázdný průnik. Kromě toho máme dán pevný počet časových oken v rozvrhu (označme ho k). Cílem je rozhodnout, zda existuje takové rozdělení dvojic (učitel, skupina studentů) do časových oken, při kterém nedochází ke kolizím, tj. do jednoho časového okna není rozvrženo více položek pro jednoho učitele, ani více položek pro skupiny studentů s neprázdným průnikem. Označme takto definovaný problém jako *TT*. Ukážeme, že problém TT je NP-úplný.

Lemma 1. *K-COLOR* $\propto$ *TT*

Důkaz. Máme-li instanci problému K-COLOR, tedy graf $G = (V, E)$ a číslo k, vytvoříme instanci problému TT s k časovými okny. Vrcholům z V nyní přiřadíme učitele a studenty následujícím způsobem: každému vrcholu přiřadíme jiného učitele a každé hraně přiřadíme jednoho studenta (pro každou hranu jiného). Skupinu studentů u vrcholu dostaneme jako množinu studentů přiřazených hranám sousedícím s vrcholem. Tak bude každému vrcholu ve V odpovídat jedna položka rozvrhu. Pokud jsou dva vrcholy (označme je u a v) v grafu G spojeny hranou, jsou jím odpovídající položky nekompatibilní (mají totiž společného alespoň toho studenta, který byl přiřazen hraně uv). Pokud naopak mezi dvěma vrcholy u' a v' hrana nevede, jsou odpovídající položky kompatibilní – učitele mají obě různé; pokud by jejich skupiny studentů měly nějakého společného studenta, nemohl by se do nich dostat jinak než hranou spojující tyto vrcholy u', v'.

Pokud nyní existuje rozvrh bez kolizí, získáme z něj obarvení grafu k barvami tak, že vrcholy odpovídající položkám rozvrženým do i-tého časového okna obarvíme barvou i. Naopak, každé obarvení grafu G k barvami lze převést na rozvrh bez kolizí. □

Věta 2. *Problém TT je NP-úplný.*

Důkaz. Z lemmatu 1 již víme, že pro každý problém $P \in NP$ platí, že $P \propto TT$ (problém TT je tedy NP-těžký). Ukážeme, že $TT \in NP$.

Máme-li již rozdělení položek rozvrhu do časových oken, je jednoduché ověřit, že všechny položky rozvržené do jednoho časového okna mají různé učitele a že skupiny studentů rozvržené do jednoho časového okna mají prázdný průnik. Pro všechny položky rozvrhu to zvládneme nejhůře v čase $O(n^2)$, kde n je počet položek rozvrhu. Ověření je tedy polynomiální vzhledem k délce vstupu (pro každou položku rozvrhu potřebujeme v binárním zápisu vstupu alespoň konstantní počet znaků). □

2.2 Programování s omezujícími podmínkami

Programování s omezujícími podmínkami (obvykle označované jako *CP*, *constraint programming*) je technika řešení problémů kombinatorické optimalizace. Problém je popsán pomocí proměnných s konečnou či nekonečnou doménou a podmínek, jež mají tyto proměnné splňovat ([Ba]). Podmínky mohou být různého druhu, od číselných vztahů (např. $x < y$) až po komplikované funkce - stačí, když funkce vrací pravdivostní hodnotu. Výhodou programování s omezujícími podmínkami je přirozený deklarativní popis problému: prostě jen popíšeme požadavky, které má jeho řešení splňovat a zbytek necháme na počítači. Eugene C. Freuder ([Fr97]) v této souvislosti poznamenává: *"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."*.

V programování s omezujícími podmínkami rozlišujeme dva základní typy problémů: *Constraint satisfaction* nebo-li *CSP* a *constraint solving*. První z nich se zabývá pouze problémy s konečnými doménami a dále budeme mluvit pouze o něm. Druhý naopak dovoluje i nekonečné domény proměnných; není třeba zdůrazňovat, že řešení druhého problému je mnohem obtížnější. I první z problémů je však již NP-těžký, není pro něj tedy znám polynomiální algoritmus.

Pro řešení problémů typu constraint satisfaction se používá celá řada technik. Některé z nich jsou úplné (garantují nalezení řešení, pokud existuje), jako například různé varianty backtrackingu – backjumping, backmarking, apod. Neúplné metody, založené většinou na lokálním prohledávání, se snaží o postupné zlepšování aktuálně nalezeného (neúplného) řešení problému pro-

hledáváním sousedních řešení (třeba řešení vzniklých změnou hodnoty jedné proměnné).

Velkou roli v řešení problémů CSP hraje *propagace omezení*. Jde o proces, ve kterém jsou z domén proměnných vyřazovány hodnoty, které nemohou být součástí žádného řešení problému. Ať už proto, že existuje nějaké omezení přímo tuto hodnotu zakazující ($x \neq 5$), nebo daná hodnota není kompatibilní s žádnou hodnotou proměnné, která je s ní svázána jiným omezením. Mějme například omezení $x < y$ pro proměnné $x \in \{1, 2, 3\}$, $y \in \{1, 2, 3\}$ – proměnná x nemůže nabývat hodnoty 3 v žádném přípustném řešení. K propagaci omezení se používají různé (tzv. *konzistenční*) techniky, převážně techniky *hranové konzistence* (viz [Ba]). Často se také využívají jiné typy podmínek, především tzv. globální – například podmínka *all-different* požadující vzájemnou nerovnost pro množinu proměnných. Jejich výhodou je silnější propagace nekonzistencí v porovnání s běžným popisem problému prostřednictvím sady binárních nerovností.

Při aplikaci technik CP se často používají již hotové knihovny tyto problémy řešící, a to jak v jazycích neprocedurálních (Prolog), tak procedurálních (C++, Java, Python).

2.2.1 Rozvrhování jako CSP

Pro rozvrhování jsou techniky CP velmi lákavé – celá řada rozvrhovacích systémů je využívá (namátkou [BM02]). Popis požadavků na rozvrh lze totiž na CSP přirozeně transformovat. Jedná se především o tvrdá omezení, měkká omezení jsou již technikami CP popsatelná obtížněji. Pro vyjádření měkkých podmínek se používají různé varianty problému částečného splňování omezujících podmínek (*partial CSP*), které nevyžadují splnění všech omezujících podmínek.

Hlavní výhodou použití CP je však snadná modifikace popisu problému a také přímočarost tohoto popisu. V rozvrhování se totiž často objevují nové požadavky, se kterými se na začátku nepočítalo. Mohou to být speciální požadavky na umístění konkrétních předmětů do určitých dnů a další. Rozvrhovací systém v takovém případě musí snadno tyto nové požadavky absorbovat.

2.3 Lokální prohledávání

Lokální prohledávání je metaheuristická technika pro řešení optimalizačních problémů. Používá se tam, kde lze hledání řešení převést na maximalizaci (nebo minimalizaci) nějaké ohodnocovací funkce. Probíhá iterativně: nejprve je nalezeno počáteční řešení problému (buď náhodně, nebo některou jinou řešící technikou). V každém kroku se pak hledá zlepšení aktuálního řešení mezi tzv. sousedními řešeními. Ta jsou definována pomocí jednoduchých změn řešení – u rozvrhů je to například prohození dvou položek nebo změna času u jedné položky.

Lokální prohledávání je neúplnou technikou, negarantuje tedy nalezení řešení, ani nemůže podat důkaz, že nalezené řešení je optimální. Přesto při použití dobré heuristiky může pomoci v řešení problémů, pro které není znám efektivní úplný algoritmus. Takovými problémy jsou například problém vrcholového pokrytí, SAT nebo problém obchodního cestujícího. Přehled algoritmů pro problém SATu založených na lokálním prohledávání podává Selman, Kautz a Cohen v [SKC93].

V rozvrhování je lokální prohledávání používáno typicky ve dvou fázích. První fáze hledá přípustný rozvrh splňující všechna tvrdá omezení, druhá fáze se poté zaměřuje na zlepšování rozvrhu vzhledem k měkkým omezením, resp. jejich ohodnocovací funkci (více například v [RBK02]).

Problémem lokálního prohledávání je uvíznutí v lokálním maximu – „obyčejné“ lokální prohledávání, které pouze maximalizuje ohodnocovací funkci, se proto příliš nepoužívá. Daleko vhodnější jsou techniky, které za jistých okolností umožňují provést změny vedoucí k (dočasnému) poklesu hodnoty ohodnocovací funkce, jako například simulované žíhání, tabu search nebo různé druhy evolučních algoritmů (srovnání technik lokálního prohledávání aplikovaných na rozvrhování najdete v [RSB+02]).

Zvláštní otázkou při implementaci algoritmů lokálního prohledávání je také nastavení parametrů algoritmu – jejich hodnota může například určovat, jak je algoritmus náchylný k přijetí zhoršujícího řešení nebo jak velké okolí prohledává. V závislosti na hodnotách parametrů pak může algoritmus produkovat diametrálně odlišné výsledky. V poslední době se proto experimentuje s automatickým laděním parametrů v průběhu výpočtu.

2.4 Interaktivní rozvrhování

Interaktivní rozvrhování se snaží kombinovat automatické rozvrhování s rozvrhování ručním. Hlavním důvodem je to, že některé požadavky na výsledný rozvrh je obtížné vyjádřit pomocí vstupních dat a je lepší, když mohou být tyto nuance upraveny ad-hoc ručně. Kromě toho také interaktivita umožňuje (drobné) změny zadání i v průběhu výpočtu nebo po jeho dokončení.

Při interaktivním rozvrhování lze postupovat buď tak, že v každém kroku výpočtu zachováváme přípustný rozvrh, který je ovšem neúplný, tj. má některé nerozvržené položky (jak to prezentuje Barták a Müller v [BM02]). Jiným postupem je použití technik lokálního prohledávání, kdy ve výpočtu pracujeme vždy s úplným rozvrhem, který ovšem nemusí být přípustný (tj. nesplňuje některá tvrdá omezení). Zásahy uživatele se pak realizují stejným způsobem jako jeden krok lokálního prohledávání, tedy přesunutím položky do jiného časového okénka. Tento přístup je prezentován v této práci.

Kapitola 3

Model rozvrhovacího problému

Model rozvrhovacího problému použitý v této práci vznikl na základě požadavků z jednoho gymnázia, nicméně je použitelný i v širším okruhu středních škol. Cílem je co nejjednodušší popis požadavků na rozvrh s minimální redundancí, který je přitom dostatečně flexibilní a umožňuje popsat širokou škálu požadavků.

3.1 Popis školy a požadavků na rozvrh

3.1.1 Učitelé a jejich časové preference

Učitelé mohou být ve škole dostupní jen určité hodiny v týdnu, například proto, že učí na zkrácený úvazek. Kromě toho mohou mít specifická přání, aby jejich hodiny začínaly třeba dopoledne, ať už z rodinných či jiných důvodů.

Učitelé si mohou pro každé časové okno v rozvrhu nastavit úroveň preference celým číslem od 0 do 10; vyšší hodnota značí vyšší preferenci, přitom 0 znamená, že učitel v daný čas nemůže vůbec učit. Pokud učitel pro některé časové okno neuvede vůbec žádnou hodnotu, bere se jako by v daný čas nemohl vůbec učit a jeho preference pro toto časové okno je tedy 0. Při výpočtu rozvrhu je hodnota preference 0 považována za tvrdé omezení, pokud tedy má učitel rozvrženu výuku do času, kdy není dostupný, je to ohodnoceno jako kolize.

3.1.2 Požadavky předmětů na místnosti

Předměty mohou vyžadovat pro svou výuku speciální místnosti. Tělocvik je dobré učit v tělocvičnách, hodiny výpočetní techniky je vhodné umístit do počítačových učeben, a k výuce fyziky bývá často potřeba například zpětný projektor. Je tedy nutné, aby bylo možné vyjádřit preference předmětů vzhledem k místnostem.

Protože ale vyjádření preferencí k jednotlivým místnostem by bylo příliš komplikované, používá se odlišný postup. Místnosti jsou zařazeny do kategorií a předměty vyjadřují své požadavky vzhledem k těmto kategoriím. Hodnotou preference je opět celé číslo od 0 do 10; 0 znamená, že daný předmět se v místnostech dané kategorie nemůže vůbec vyučovat. Jedna místnost může být (a obvykle i bývá) zařazena do více kategorií. Při výpočtu konkrétní preference předmětu pro místnost se berou v úvahu všechny kategorie, do kterých místnost patří a pro které má předmět nastavenu nějakou preferenci. Preference pro místnost se pak spočítá jako maximum preferencí přemětu pro tyto kategorie. Pokud pro nějakou místnost nemá předmět nastavenu vůbec žádnou preferenci, je výsledná preference předmětu vůči této místnosti nastavena na výchozí hodnotu 5.

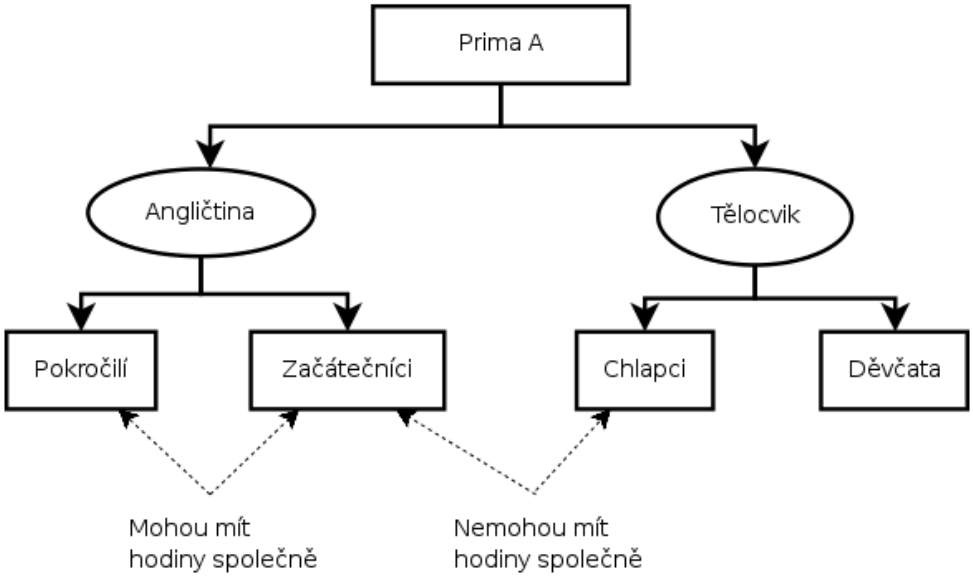
3.1.3 Hierarchie studentů

Pro reprezentaci studentů se nabízí dva základní přístupy – každého studenta reprezentovat zvlášť, nebo studenty sdružit do skupin. První přístup bývá obvyklý při univerzitním rozvrhování, kde každý student má svůj individuální rozvrh podle předmětů, které si zapsal. Na střední škole však tento přístup není příliš obvyklý. Jeho nevýhodou je značný nárůst velikosti problému (skupin studentů je typicky mnohem méně než studentů samotných). Kromě toho většina hodin studenta střední školy probíhá v rámci jeho třídy. Proto byl zvolen popis pomocí skupin studentů, kde každá má svou předem určenou velikost.

Skupiny studentů nestačí popsat pouhým seznamem tříd – na různé předměty se třídy dělí na poloviny (a na každý předmět jiné), naopak některých výběrových seminářů se zúčastňují studenti z různých tříd jednoho ročníku, či dokonce více ročníků zároveň. Jejich popis proto musí tuto hierarchickou strukturu odrážet.

Každá skupina studentů může mít definována svá *rozdělení*. Ta odpovídají disjunktnímu rozkladu skupiny na podskupiny. Jedno rozdělení může obsahovat jednu či více podskupin studentů. Skupiny v jednom rozdělení

Obrázek 3.1: Příklad části hierarchie studentů (obdélníky označují skupiny studentů, elipsy jejich rozdělení)



mohou mít současně rozvržené hodiny, aniž by docházelo ke kolizím. Naopak pokud by byly pro nějaké dvě podskupiny z různých rozdělení jedné skupiny studentů rozvrženy hodiny současně, ke kolizi by došlo.

Příkladem takovéto hierarchie je různé dělení třídy na poloviny při tělocviku a při angličtině. Na tělocvik je třídy rozdělena na skupinu chlapců a skupinu děvčat, pro angličtinu je ovšem třída rozdělena podle stupně pokročilosti (viz obrázek 3.1).

Na vrcholu hierarchie je skupina studentů obsahující všechny studenty školy. Má zpravidla jediné rozdělení, které obsahuje například skupiny studentů odpovídající jednotlivým ročníkům.

3.1.4 Položky rozvrhu

Položkou rozvrhu rozumíme čtveřici (skupina studentů, učitel, předmět, počet hodin týdně). Je tím, co budeme skutečně rozvrhovat. Předpokládáme totiž, že pro konkrétní třídu a vyučovaný předmět bude znám učitel ještě před začátkem rozvrhování – to bývá na středních školách obvykle splněno, jeden učitel totiž učí danou třídu někdy i během celého studia. Položky roz-

vrhu mohou trvat jednu či více hodin, v závislosti na vyučovaném předmětu. Pokud položka trvá déle než jednu hodinu, musí být po celou dobu trvání rozvržena do jediné místnosti.

3.2 Ohodnocení rozvrhu

Ohodnocovací funkce rozvrhu sestává ze dvou částí – počtu nesplněných tvrdých omezení (kolizí) a měkkého ohodnocení. Při porovnávání má přednost první část, rozvrh neobsahující žádné kolize (byť s velkou hodnotou ohodnocovací funkce) je tedy lepší než rozvrh s malou hodnotou měkké ohodnocovací funkce, ale s kolizemi.

3.2.1 Tvrdá omezení na rozvrh

Kromě omezení vynucených fyzikální realitou našeho světa (není možné, aby jeden člověk byl najednou na dvou různých místech) bývají na střední škole ještě další omezení, která musí rozvrh bezpodmínečně splňovat. Je to například požadavek, aby studenti měli jeden předmět maximálně jednou denně. Tvrdá omezení na rozvrh aplikovaná v našem rozvrhovacím systému jsou tato:

- Žádný učitel nesmí učit dvě hodiny najednou
- Žádný student nesmí mít rozvrženy dvě hodiny najednou
- Do jedné místnosti může být naráz rozvržena jen jedna hodina
- Jedna skupina studentů nesmí mít jeden předmět vícekrát za den.
- Učitelé učí pouze v časech, kdy jsou dostupní
- Skupiny studentů jsou rozvrhovány do místností tak, aby se tam vešly
- Pokud nějaký předmět nemůže být vyučován v nějaké místnosti, není do této místnosti rozvrhován

Počet porušených tvrdých omezení pak tvoří první část ohodnocovací funkce rozvrhu.

3.2.2 Měkká ohodnocovací funkce

Měkká ohodnocovací funkce rozvrhu je vypočítávána na základě těchto kritérií:

- Jak dobře odpovídá umístění položek do místností preferencím u jednotlivých předmětů
- Jak odpovídá rozvržení položek časovým preferencím učitelů
- Jak vyhovuje rozvrh studentům

Část měkké ohodnocovací funkce odpovídající místnostem se spočítá na základě definice preferencí předmětů vzhledem k místnostem z části 3.1.2. Za každou položku, které je přiřazena do nějaké místnosti, se do k ohodnocovací funkci přičte preference jejího předmětu vzhledem k této místnosti. Z toho vyplývá, že čím vyšší součet, tím lépe.

Učitelská část ohodnocovací funkce se spočítá takto: Pro každou rozvrženou položku rozvrhu se započítá hodnota učitelské preference pro čas, do kterého je položka rozvržena. Pokud položka trvá déle než jednu hodinu, je započítána učitelova preference za každou hodinu, do které položka zasahuje. Hodnotu učitelského ohodnocení se opět snažíme maximalizovat.

Ohodnocení rozvrhu z pohledu studentů se počítá pouze pro ty skupiny studentů, které jsou „listy“ studentské hierarchie, nemají tedy již žádné podskupiny (viz část 3.1.3). Sečteme ohodnocení jejich rozvrhů pro každý den v týdnu – ohodnocení denního rozvrhu se spočítá takto:

$$s + \sum_{i=1}^{n} d_i$$

kde s je čas první hodiny v rozvrhu, d_i délky mezer v rozvrhu a n počet mezer v rozvrhu; studentskou část ohodnocovací funkce se algoritmus snaží minimalizovat.

Kapitola 4

Rozvrhovací algoritmus

Rozvrhovací algoritmus implementovaný v této práci je kombinací několika přístupů: Použití obecné knihovny pro řešení problémů s omezujícími podmínkami (nazývané též *constraint solver*) a lokálního prohledávání založeného na minimalizaci kolizí v rozvrhu. Z technik lokálního prohledávání bylo použito simulované žíhání a tabu search (jejich podrobnější popis najdete níže v části 4.4). Rozvrhovací systém nicméně umožňuje snadnou implementaci dalších algoritmů lokálního prohledávání nebo jiného typu.

4.1 Obecné schéma algoritmu

Obecné schéma algoritmu je popsáno v algoritmu 1. Nejprve vytvoříme popis problému pro constraint solver (podle části 4.2). Poté nalezneme pomocí constraint solveru počáteční řešení problému (to je v algoritmu naznačeno příkazem `constraintSolver.getInitialTimetable`).

Nalezené řešení pak dále vylepšujeme pomocí lokálního prohledávání. Prohledávání ukončíme, když je proveden maximální počet kroků (změn řešení), které smí prohledávání udělat.

Algoritmus 1 Schéma rozvrhovacího algotimu

```
constraintSolver.createProblem()
timetable = constraintSolver.getInitialTimetable()
while nenastala ukončovací podmínka do
  udělej krok lokálního prohledávání
end while
```

4.2 Hledání počátečního řešení pomocí CP

V první fázi je rozvrhovací problém převeden na problém CSP a pomocí obecného software na řešení CSP (knihovny *python-constraint* vytvořené Gustavo Niemeyerem – [Ni]) je získáno počáteční řešení. Popis problému vypadá takto:

Proměnné Pro každou položku (viz část 3.1.4) se vyrobí tolik proměnných, kolikrát týdně má být daná položka vyučována. Pokud navíc trvá položka déle než jednu hodinu, je každá proměnná ještě vytvořena v tolika „kopiích“, kolik je doba trvání položky.

Domény Pro proměnné odpovídající položce rozvrhu jsou doménami časová okna, ve kterých je učitel vyučující danou položku dostupný. Hodnoty v doméně jsou celá čísla, časová okna jsou reprezentována číslem ($den * hodina$).

Omezení Pro popis problému se používají pouze binární omezující podmínky. V první fázi není vyžadováno splnění všech tvrdých podmínek na rozvrh, požadovány jsou pouze tyto:

- Žádné kolize pro učitele – hodnoty všech proměnných odpovídajících jednomu učiteli musí být různé
- Žádné kolize pro studenty – hodnoty všech proměnných odpovídajících skupinám studentů, které mohou mít neprázdný průnik, musí být různé.
- Studenti mohou mít jeden předmět pouze jednou denně – hodnoty všech proměnných odpovídající konkrétní skupině studentů a předmětu se musí lišit ve dni; toto neplatí, pokud předmět trvá déle než jednu hodinu, pak se požaduje různost pouze pro proměnné odpovídající počátečním úsekům hodin
- Proměnné pro jednu déletrvající položku musí být rozvrženy po sobě, tj. jejich hodnoty se musí shodovat v dni a hodiny musí být vždy po sobě

Knihovna, kterou k řešení problému CSP používáme, pracuje na principu backtrackingu s dopřednou propagací omezení, někdy též označovaném jako MAC (*maintaining arc consistency*). Postupně zkouší různá dosazení

hodnot do proměnných a při každém dosazení vyřazuje z domén proměnných hodnoty, které nemohou být součástí řešení problému – snaží se tak udržovat hranovou konzistenci (viz část 2.2). Domény proměnných před začátkem výpočtu předtřídíme podle následujícího uspořádání – hodnota x z domény nějaké proměnné je menší než hodnota y z téže domény, jestliže odpovídá vyšší hodině, nebo menšímu dni. Toto uspořádání vychází z pozorování vlastností použitého constraint solveru. Ten totiž při takovémto uspořádání distribuoval položky rozvrhu rovnoměrně do jednotlivých dnů a navíc blíže k první hodině rozvrhu. Pokud byly hodnoty v doménách proměnných setříděny podle velikosti, všechny položky byly rozvrhovány přednostně na poslední den rozvrhu do pozdních odpoledních hodin; kromě toho také trvalo mnohem déle, než bylo nalezeno přípustné řešení.

Poté co constraint solver vydá své první řešení, jsou sloučeny déletrvající položky do jedné – dále již na ně budeme pohlížet jako na jednu položku, zasahující ovšem do více časových oken.

4.3 Přiřazení položek do místností

Přiřazování položek do místností probíhá postupně v jednotlivých časových oknech. Pro jeho výpočet se používá algoritmu maximálního váženého párování maximální mohutnosti v bipartitním grafu popsaného v [MN99]. Vrcholy bipartitního grafu tvoří položky, které byly do časového okna rozvrženy, a místnosti. Mezi vrcholem položky a vrcholem místnosti vede hrana pokud může být položka do místnosti rozvržena – to znamená, že preference předmětu položky vzhledem k místnosti není 0 (viz definice preference předmětu k místnosti 3.1.2). Hrana je pak ohodnocena hodnotou preference předmětu vzhledem k místnosti.

Problém nastává, pokud je do časového okna rozvržena položka, která začíná již v některém dřívějším časovém okně. Pak není možné, aby byla rozvržena do jiné místnosti, než do jaké byla rozvržena na svém počátku. Proto je v takovém případě vrchol odpovídající této položce z grafu vyřazen; stejně tak je vyřazen i vrchol odpovídající místnosti, do které byla položka dříve rozvržena. Ospravedlněním tohoto postupu je pozorování, že položek, které trvají déle než jednu hodinu, je na střední škole obvykle málo a tak výsledek tohoto algoritmu je jen o málo horší, než kdyby se místnosti přiřazovaly optimálně (a přitom je rychlejší, nemusí se totiž vracet k již vypočítaným přiřazením místností).

Pokud není položce přiřazena párovacím algoritmem žádná místnost, je to považováno za porušení tvrdého omezení.

4.4 Lokální prohledávání

Po nalezení úvodního řešení výpočet pokračuje lokálním prohledáváním. V první fázi je cílem zmenšení počtu kolizí v rozvrhu a nalezení rozvrhu, ve kterém budou splněna veškerá tvrdá omezení. Toho je dosaženo pomocí přesouvání položek, které porušují nejvíce tvrdých omezení. Přesuny jsou dvojího druhu:

- Přesun položky, která porušuje nejvíce omezení, do jiného časového okna
- Prohození položky, která porušuje nejvíce omezení, s jinou položkou, která nějaké omezení také porušuje

Konkrétní metaheuristiky pak buď vybírají vhodný přesun z části nebo z celého „sousedství“, tj. ze všech přesunů, které lze z aktuálního řešení provést, nebo přesun vybírají náhodně. Při výběru pouhé části všech možných přesunů je v závislosti na konkrétním algoritmu určena pravděpodobnost, s jakou je daný přesun vyzkoušen. Například naše implementace tabu search vybírá vždy 10 % ze všech přesunů, které jsou z aktuálního řešení možné, minimálně však deset přesunů (tyto hodnoty jsou ovšem nastavitelné prostřednictvím konfiguračního souboru).

Když je nalezen přípustný rozvrh, pokračuje algoritmus ve zlepšování hodnoty měkké ohodnocovací funkce. Pro přesuny jsou pak brány v úvahu všechny položky rozvrhu.

4.4.1 Simulované žíhání

Simulované žíhání je pravděpodobnostní metoda hledání globálně optimálního řešení problémů kombinatorické optimalizace. Byla objevena S. Kirkpatrickem, C. D. Gelattem a M. P. Vecchi v roce 1983 a V. Černým v roce 1985. Využívá myšlenky žíhání z metalurgie, kde se řízeným ochlazováním a opětovným ohříváním materiálu dosahuje vyšší kvality. Jednou z prvních úspěšných aplikací simulovaného žíhání bylo řešení problému obchodního cestujícího ([Ce85]).

Varianta simulovaného žíhání implementovaného v našem algoritmu vychází z práce [RSB+02]. Základní idea je popsána v algoritmu 2.

Algoritmus 2 Simulované žíhání

```
e_T = energy(timetable)
while nevypšel časový limit do
  m = timetable.getRandomMove()
  e_M = energy(move) {Energie nalezené změny}
  if dlouho se nic nezměnilo then
    n = timetable.getNeighbourhood()
    σ = průměrný rozptyl ohodnocení v n
    t = c_reheat * σ
  else
    t = λ * t
  end if
  if tah je lepší or rand() < e^{-(e_M - e_T)/t} then
    timetable = timetable.applyMove(move)
    e_T = e_M
  end if
end while
```

Algoritmus nejprve spočítá ohodnocení počátečního řešení získaného řešením problému CSP (viz. 3.2). Toto ohodnocení je pro účely simulovaného žíhání označeno jako „energie“ rozvrhu. Poté začíná první iterace algoritmu – v každé iteraci je vybrána jedna možná změna rozvrhu z množiny sousedních rozvrhů. Pokud tato změna vede k lepšímu řešení než je to aktuální, je přijata a provede se. Pokud dává řešení horší, je přijata pouze s pravděpodobností $e^{-(e_M - e_T)/t}$, kde e_M je energie změny, e_T energie rozvrhu a t aktuální teplota. Pokud je však již aktuální řešení bez kolizí a navrhovaná změna nějaké kolize způsobuje, je zamítnuta vždy. Výpočet končí, pokud byl proveden daný počet tahů, případně po vypršení časového limitu.

Energie rozvrhu (nebo změny) se spočítá takto: Pokud je počet kolizí změny a aktuálního řešení různý, je energií právě počet kolizí (tvrdá ohodnocovací funkce). Pokud je počet kolizí stejný, je energií hodnota měkké ohodnocovací funkce.

Průběh výpočtu a náchylnost k přijetí zhoršující změny se řídí tzv. *teplotou*. Ta se v průběhu výpočtu mění ve fázích. Na začátku každé fáze dojde k takzvanému *ohřátí* – spočítá se nová hodnota teploty, a to na základě vy-

počteného výběrového rozptylu hodnot ohodnocovací funkce pro sousedních řešení; rozptyl je navíc vynásoben koeficientem c_{reheat}. Poté je v každé fázi teplota snižována s pevným koeficientem (v našem algoritmu je to $\lambda = 0.85$), čímž se snižuje pravděpodobnost, že bude přijata zhoršující změna. Pokud po pevném počtu kroků nedojde k žádné změně aktuálního řešení, je teplota znovu zvýšena a začíná další fáze.

Výpočet lze řídit několika parametry. Je to již zmíněná λ vyjadřující pokles teploty v každé fázi. Dále pak koeficient c_{reheat}, kterým je násoben průměrný rozptyl při výpočtu nové teploty a nakonec n_{empty} určující počet iterací bez provedení změny, než dojde k ohřátí teploty.

4.4.2 Tabu search

Tabu search je další metaheuristikou pro lokální prohledávání, popsanou Fredem Gloverem v roce 1986. Tato technika se snaží zabránit vzniku (krátkých) cyklů při procházení sousedních řešení. K tomu využívá pomocnou datovou strukturu – *tabu list* – s pevnou délkou, do které ukládá poslední navštívená řešení. Díky tomu může snáze uniknout z lokálního maxima.

Nástin verze algoritmu implementovaného v našem systému najdete v algoritmu 3. Při hledání změny se neprohledává celé okolí aktuálního řešení, ale jen jeho část, vyjádřená parametrem λ metody `getNeighbourhood`. V seznamu změn, které jsou tabu, nejsou uchovávány celé rozvrhy, nýbrž jen položky, které byly přesunuty a časové okno určující odkud byly přesunuty. Přesun takových položek zpátky do tohoto časového okna je pak zakázán, leda by zlepšoval nejlepší dosud nalezené řešení.

Algoritmus 3 Tabu search

while nevypršel časový limit **do**
  n = timetable.getNeighbourhood(λ)
  m = nejlepší tah v n, který není tabu nebo zlepšuje nejlepší dosud nalezené řešení
  timetable.applyMove(m)
  tabulist.insert(m)
end while

4.5 Ruční úpravy rozvrhů

Ruční úpravy rozvrhů mohou být dvou druhů – změna času položky a změna místnosti. Ručně rozvržené položky algoritmus považuje za pevně rozvržené – dále je již nepřesouvá. Samotné změny jsou pak prováděny stejně jako u lokálního prohledávání – měněná položka je prostě přesunuta na své nové místo.

Rozdíl oproti předchozím postupům použitým u lokálního prohledávání nastává v přiřazování místností. Pokud přiřazujeme místnosti v časovém okně, kde některá z položek má již pevně určenu místnost, do které má být rozvržena, tuto místnost vyřadíme z dalšího rozvrhování v tomto časovém okně.

Kapitola 5

Výsledky experimentů

Rozvrhovací algoritmus (spolu s dalšími součástmi rozvrhovacího systému) byl implementován v Pythonu. Všechny testy probíhaly na PC s procesorem Intel Centrino 1.73GHz, 1024MB RAM, operačním systémem Linux s jádrem 2.6.15 a Pythonem 2.4.3.

5.1 Vstupní data

Data pro testování byla získána ze tří gymnázií, lišících se počtem studentů i organizací studia. Všechna data odpovídají rozvrhům ze školního roku 2005/2006. Konkrétně se jedná o tato gymnázia:

- Gymnázium Pacov – šestileté gymnázium, v každém ročníku je jedna třída
- Gymnázium Postupická, Praha 4 – šestileté a čtyřleté gymnázium s rozšířenou výukou tělocviku, v každém ročníku 2–3 třídy
- Gymnázium Voděradská, Praha 10 – osmi-, šesti- a čtyřleté gymnázium, v každém ročníku je 2–5 tříd.

Tabulka 5.1 obsahuje orientační informace o jednotlivých sadách dat. Sloupec *Třídy* udává počet kmenových tříd ve škole – počet skupin studentů nutných pro popis celé školy je však vyšší. Sloupec *Učitelé* udává počet učitelů zaměstnaných ve škole, včetně těch, kteří učí na zkrácený pracovní úvazek. Sloupec *Místnosti* popisuje počet všech učeben ve škole a sloupec *Položky rozvrhu* obsahuje počet položek, které je třeba rozvrhnout.

Tabulka 5.1: Vstupní data experimentů

Data	Kmenové třídy	Učitelé	Místnosti	Položky rozvrhu
Pacov	6	16	12	119
Postupická	19	50	34	362
Voděradská	23	63	42	400

Tabulka 5.2: Počet omezení pro constraint solver

Data	Studenti	Učitelé	Předměty v jeden den
Pacov	5443	2632	130
Postupická	29022	8971	374
Voděradská	24906	8414	470

5.2 Testy

5.2.1 Constraint solver

Pro popis problému pro constraint solver jsme použili omezení dle části 4.2. Tabulka 5.2 dává základní představu o velikosti jednotlivých problémů. Sloupec *Studenti* obsahuje počet binárních omezení zakazujících kolize mezi hodinami studentů, sloupec *Učitelé* obsahuje počet binárních omezení zakazujících kolize hodin učitelů a sloupec *Předměty v jeden den* obsahuje počet binárních omezení zakazujících to, aby studenti měli jeden předmět vícekrát za den.

5.2.2 Průběh výpočtů

Při testování průběhu výpočtů jsme vycházeli z ohodnocovací funkce (část 3.2). Pro všechny sady dat jsme provedli oba implementované algoritmy – simulované žíhání a tabu search, a přitom jsme měřili různé charakteristiky rozvrhu. Přehled výsledků těchto testů najdete v tabulkách 5.3, 5.4 a 5.5. Kromě počtu kolizí (řádek *Počet kolizí*) to byla kvalita rozvrhu z pohledu studentů, učitelů a přiřazení položek do místností. Kvalitu rozvrhu jsme měřili na základě více kritérií, které jsou součástí měkké ohodnocovací funkce. Pro studenty jsme zjišťovali, kterou hodinu jim začíná rozvrh (průměrně) a jaký je rozptyl začátků výuky tříd (řádky *Začátek výuky* a *Rozptyl začátku výuky*

Tabulka 5.3: Ohodnocení rozvrhu – Pacov

Data	CS	Simulované žíhání		Tabu search	
		60s	240s	60s	240s
Počet kolizí	1	0 ± 0,00	0 ± 0,00	0 ± 0,00	0 ± 0,00
Začátek výuky	0,88	0,87 ± 0,03	0,89 ± 0,05	0,89 ± 0,00	0,85 ± 0,15
Rozptyl začátku výuky	1,19	1,18 ± 0,04	1,22 ± 0,14	1,20 ± 0,00	1,63 ± 0,55
Díry v rozvrhu	2,09	2,11 ± 0,03	2,00 ± 0,14	2,10 ± 0,01	1,80 ± 0,13
Roptyl děr	2,51	2,54 ± 0,06	2,49 ± 0,21	2,53 ± 0,05	2,08 ± 0,20
Ohodnocení učitelů	3,88	3,97 ± 0,10	4,19 ± 0,10	3,88 ± 0,02	4,16 ± 0,17
Ohodnocení místností	6,54	6,66 ± 0,07	6,74 ± 0,02	6,59 ± 0,01	6,71 ± 0,03

– čím menší hodnota, tím lépe). Dále jsme zjišťovali také průměrný počet děr v rozvrhu a rozptyl počtu děr pro všechny skupiny studentů (řádky *Díry v rozvrhu* a *Rozptyl děr* – rovněž čím menší, tím lepší). Pro učitele jsme měřili průměrné ohodnocení jim rozvržených položek (řádek *Ohodnocení učitelů* – čím vyšší hodnota, tím lépe) a pro místnosti průměrnou hodnotu preference předmětů položek rozvržených do místnosti (řádek *Ohodnocení místností* – opět čím vyšší hodnota, tím lépe).

Měření jsme prováděli vždy pro nejlepší dosažené řešení po určité době – po vyřešení problému CSP (sloupec *CS*), po jedné minutě lokálního prohledávání a po čtyřech minutách lokálního prohledávání. Pro každý typ výpočtu a každou sadu dat jsme provedli osm opakování měření. Výsledky všech měření najdete na přiloženém CD-ROM.

Gymnázium Pacov

Výsledky výpočtů pro data z Gymnázia Pacov shrnuje tabulka 5.3. Ve všech případech se podařilo najít řešení bez kolizí po méně než jedné minutě. Další lokální prohledávání vedlo k mírnému zlepšení měkké ohodnocovací funkce, především z pohledu studentů, kdy došlo ke zmenšení průměrného počtu děr v rozvrhu. Pozorovatelný je také nárůst splňování časových preferencí učitelů. Oba algoritmy (simulované žíhání a tabu search) dosáhly podobných výsledků.

Tabulka 5.4: Ohodnocení rozvrhu – Postupická

Data	CS	Simulované žíhání		Tabu search	
		60s	240s	60s	240s
Počet kolizí	8	2,5 ± 2,14	1,63 ± 1,69	1,25 ± 2,55	0,38 ± 1,06
Začátek výuky	1,67	1,67 ± 0,00	1,68 ± 0,00	1,69 ± 0,02	1,69 ± 0,02
Rozptyl začátku výuky	2,53	2,53 ± 0,00	2,53 ± 0,01	2,58 ± 0,12	2,60 ± 0,12
Díry v rozvrhu	1,12	1,13 ± 0,01	1,13 ± 0,01	1,13 ± 0,01	1,13 ± 0,02
Rozptyl děr	1,51	1,53 ± 0,04	1,53 ± 0,04	1,53 ± 0,02	1,53 ± 0,04
Ohodnocení učitelů	1,61	1,62 ± 0,00	1,62 ± 0,00	1,61 ± 0,01	1,61 ± 0,01
Ohodnocení místností	4,96	5,03 ± 0,02	5,05 ± 0,03	5,05 ± 0,01	5,06 ± 0,01

Gymnázium Postupická

Detailní výsledky výpočtů rozvrhu pro Gymnázium Postupická obsahuje tabulka 5.4. Data z tohoto gymnázia jsou výjimečná velmi vysokým podílem hodin společných pro více tříd jednoho ročníku. Například výuka angličtiny na čtyřletém gymnáziu probíhá ve čtyřech skupinách obsahujících studenty ze všech tříd v ročníku.

Počty kolizí dosažené simulovaným žíháním a tabu search jsme testovali dvouvýběrovým t-testem proti nulové hypotéze (tj. že oba algoritmy dávají stejné výsledky). Pro výsledky obou algoritmů po 1 minutě činila dosažená hladina významnosti testu $0,31$, nulovou hypotézu nemůžeme tedy na obvyklé hladině významnosti $\alpha = 0,05$ zamítnout. Výsledky výpočtů po 4 minutách lokálního prohledávání jsme testovali proti alternativní hypotéze, že průměrný počet kolizí dosažený pomocí tabu search je menší než průměrný počet kolizí dosažených simulovaným žíháním. Takto provedený t-test dosáhl hladiny významnosti $0,05$, tedy lze říci se spolehlivostí $0,95$, že tabu search dává pro tato data lepší výsledky (co do počtu kolizí) než simulované žíhání.

Gymnázium Voděradská

Data z Gymnázia Voděradská byla co do počtu studentů, učitelů i položek v rozvrhu největší. Souhrn výsledků výpočtů podává tabulka 5.5. Rozvrh bez kolizí se simulovanému žíhání podařilo obvykle najít do čtyř minut, výsledky pro tabu search jsou méně příznivé – ač se mu obvykle podařilo najít rozvrh s minimálním počtem kolizí do 4 minut, velký rozptyl signalizuje hodnoty

Tabulka 5.5: Ohodnocení rozvrhu – Voděradská

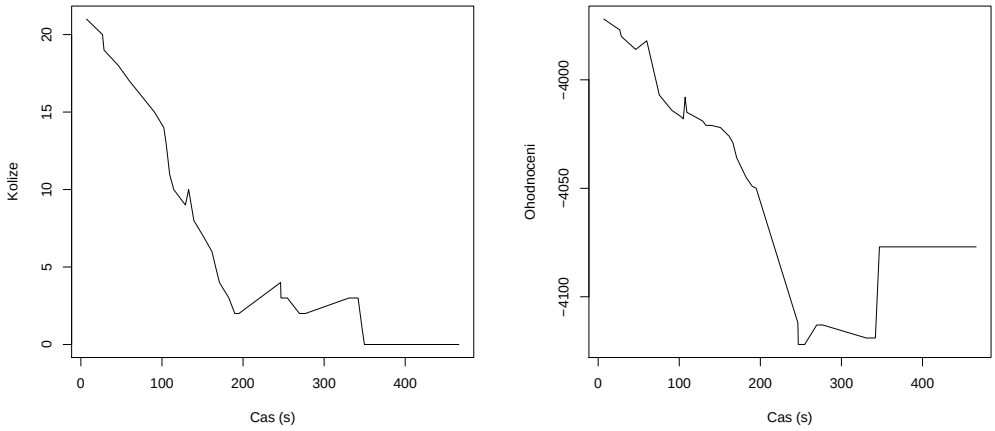
Data	CS	Simulované žíhání		Tabu search	
		60s	240s	60s	240s
Počet kolizí	19	14,75 ± 1,67	0,25 ± 0,46	14,38 ± 2,20	1,38 ± 2,56
Začátek výuky	1,11	1,11 ± 0,00	1,12 ± 0,02	1,11 ± 0,00	1,14 ± 0,03
Rozptyl začátku výuky	1,85	1,87 ± 0,01	1,89 ± 0,05	1,87 ± 0,01	1,92 ± 0,04
Díry v rozvrhu	1,41	1,41 ± 0,01	1,45 ± 0,03	1,42 ± 0,01	1,45 ± 0,03
Rozptyl děr	1,78	1,77 ± 0,01	1,84 ± 0,08	1,79 ± 0,03	1,83 ± 0,05
Ohodnocení učitelů	1,44	1,44 ± 0,00	1,44 ± 0,00	1,44 ± 0,00	1,44 ± 0,00
Ohodnocení místností	5,35	5,42 ± 0,02	5,55 ± 0,00	5,40 ± 0,02	5,54 ± 0,03

také velmi vzdálené od výsledného průměru.

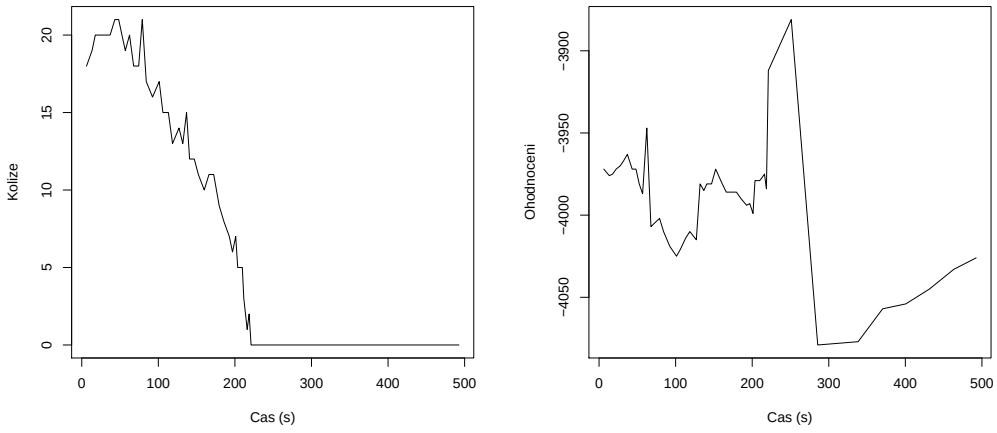
Počty kolizí při použití obou algoritmů jsme opět testovali dvouvýběrovým t-testem s předpokladem rozdílného rozptylu obou výběrů. Test jsme prováděli proti nulové hypotéze, že oba algoritmy dávají po daném čase rozvrh se stejným průměrným počtem kolizí. Pro výsledky obou algoritmů po 1 minutě činila dosažená hladina významnosti testu $0,35$, nulovou hypotézu nemůžeme tedy na obvyklé hladině významnosti $\alpha = 0,05$ zamítnout. T-test pro výsledky výpočtů po 4 minutách dosáhl hladiny významnosti $0,13$, nulovou hypotézu tedy opět nemůžeme zamítnout na dané úrovni významnosti.

Na obrázku 5.1 je znázorněn vývoj počtu kolizí a měkké ohodnocovací funkce v průběhu výpočtu simulovaného žíhání. Graf znázorňuje hodnoty pro aktuální řešení rozvrhu, nikoliv pro nejlepší dosažené řešení – proto se v grafu vyskytuje kolísání a občasný vzestup hodnot ohodnocovací funkce. Podobně obrázek 5.2 znázorňuje počet kolizí a ohodnocovací funkci v průběhu výpočtu tabu search. Lze pozorovat, že tabu search je náchylnější k přijetí zhoršujícího řešení než simulované žíhání.

Obrázek 5.1: Průběh simulovaného žíhání pro Gymnázium Voděradská – kolize (vlevo) a měkká ohodnocovací funkce (vpravo)



Obrázek 5.2: Průběh tabu search pro Gymnázium Voděradská – kolize (vlevo) a měkká ohodnocovací funkce (vpravo)



Kapitola 6

Shrnutí

V této práci jsme prezentovali systém pro sestavování rozvrhů pro střední školy. Tento systém umožňuje ruční úpravy rozvrhů a to v průběhu výpočtu i po jeho skončení. Výsledné rozvrhy lze uložit a použít jako základ pro další výpočty, nebo je lze exportovat do formátu HTML.

Model rozvrhovacího problému použitý v této práci vznikl na základě požadavků jednoho gymnázia, je však použitelný i pro další typy škol. Tento model umožňuje kompletní popis požadavků na rozvrh. Umožňuje vyjádřit časové preference učitelů, požadavky předmětů na konkrétní místnosti a také požadavky týkající se rozdělení skupin studentů na podskupiny. Data je možné do systému zadávat buď ve formátu XML (pro tuto práci byl vytvořen jeho specifický dialekt), nebo prostřednictvím kancelářského software, jako například MS Excel.

Algoritmus rozvrhování je založen na použití obecné knihovny pro řešení problémů s omezujícími podmínkami v kombinaci s technikami lokálního prohledávání. Pro rozvrhování položek rozvrhu do místností je navíc použit algoritmus maximálního váženého párování v bipartitním grafu. Z konkrétních algoritmů lokálního prohledávání jsme implementovali simulované žíhání a tabu search, systém navíc umožňuje snadnou implementaci dalších algoritmů, založených buď na lokálním prohledávání nebo i jiném přístupu.

Možnosti systému jsme ilustrovali na datech získaných ze tří gymnázií lišících se velikostí a studijními plány. Ve všech případech se podařilo dosáhnout přípustného rozvrhu v přijatelném čase. Oba implementované algoritmy vykazovaly podobné výsledky, a to jak co do počtu kolizí ve výsledných rozvrzích, tak i dle dalších kritérií. Pouze v jediném případě dosáhl tabu search lepších výsledků než simulované žíhání.

Literatura

[Ba] Barták R.: *On-line Guide to Constraint Programming*. `http://kti.mff.cuni.cz/~bartak/constraints/`

[BM02] Barták R., Müller T.: *Interactive Timetabling: Concepts, Techniques and Practical Results*. In Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT2002), Gent, 2002, pp. 58–72.

[BKJ97] Burke E., Kingston J., Jackson K., Weare R.: *Automated University Timetabling: The State of the Art* The Computer Journal 40 (9), 1997, pp. 565–571.

[BP02] Burke E., Petrovic S.: *Recent Research Directions in Automated Timetabling*. European Journal of Operational Research 140 (2), 2002, pp. 266–280.

[CK95] Cooper T.B., Kingston J.H.: *The Complexity of Timetable Construction Problems*. Technical Report No. 495, Sydney, February 1995.

[Ce85] Černý V.: *A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm*. Journal of Optimization Theory and Applications, 1985, pp. 41–45.

[Fr97] Freuder E. C.: *In Pursuit of the Holy Grail*. Constraints, April 1997, Springer Netherlands, pp. 57–61.

[Ka72] Karp R.M.: *Reducibility among combinatorial problems*. Complexity of Computer Computations 43, 1972, pp. 85–103.

[MN99] Mehlhorn K., Näher St.: *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999, pp. 132–162.

[Ni] Niemeyer G.: *Python-constraint module for solving CSP*. `http://labix.org/python-constraint`

[RBK02] Rossi-Doria O., Blum C., Knowles J., Sampels M., Socha K., Paechter B.: *A local search for the timetabling problem*. In E. Burke, P. De Causmaecker (eds.): Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), Gent, 2002, pp. 124–127.

[RSB+02] Rossi-Doria O., Sampels M., Birattari M., Chiarandini M., Dorigo M., Gambardella L. M., Knowles J., Manfrin M., Mastrolilli M., Paechter B., Paquete L., Stutzle T.: *A comparison of the performance of different metaheuristics on the timetabling problem*. In E. Burke, P. De Causmaecker (eds.): Proceedings of the 4th international conference on the Practice And Theory of Automated Timetabling (PATAT 2002), Gent, 2002, pp. 115–119.

[SKC93] Selman B., Kautz H.A., Cohen B.: *Local Search Strategies for Satisfiability Testing*. In Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability, Providence RI, 1993, pp.

Příloha A

Uživatelská dokumentace

Rozvrhovací systém jsme pojmenovali *Tsine* (název je zkratkou pro *Timetabling System with INteractive Editing*). Mezi jeho hlavní vlastnosti patří možnost sledovat průběh výpočtu rozvrhu v grafickém uživatelském rozhraní, ručně upravovat výsledek za běhu výpočtu a výsledné rozvrhy ukládat ve formátu XML a HTML. Požadavky na rozvrh mohou být zadávány pomocí tabulkových editorů jako MS Excel nebo OpenOffice Calc.

A.1 Instalace

A.1.1 Systémové požadavky

Základní systémové požadavky jsou tyto:

- Python ve verzi ≥ 2.3
- Databázový server PostgreSQL. Tento není nutné mít nainstalován, stačí k němu mít přístup.
- Knihovna GTK+ ve verzi ≥ 2.6

Tsine pro svůj běh potřebuje také některé moduly Pythonu, které nejsou obsaženy ve standardní knihovně tohoto jazyka. Jsou to:

- PyGTK a libglade ≥ 2.6 – rozhraní ke knihovně GTK+ pro Python
- PyGreSQL – rozhraní pro databázi PostgreSQL

- libxslt a libxml2 – moduly pro práci s XSL a XML
- Cheetah – systém textových šablon, nutné (pouze) pro export do XML a HTML

Tsine používá modul pro řešení omezujících podmínek *python-constraint* vyvinutý Gustavo Niemeyerem. Protože ale tento modul není dostupný v běžných distribucích, je jeho aktuální verze součástí aplikace.

Tsine byl vyvíjen a je primárně určen pro operační systém Linux, byl však testován i na MS Windows a při splnění výše uvedených systémových požadavků by měl fungovat i na jiných platformách, kde ovšem testován nebyl.

A.1.2 Instalace aplikace

Instalace na Debian Linuxu

Nejprve je třeba nainstalovat knihovny potřebné pro běh aplikace. Stručně (jako root):

```
# apt-get install python python-glade2 \
  python-pygresql python-libxslt1 python-cheetah
```

Pokud nemáte přístup k databázi Postgresql, je doporučeno nainstalovat i ji:

```
# apt-get install postgresql
```

Poté pokračujte obecnými instrukcemi pro instalaci v části *Vlastní instalace*.

Instalace na MS Windows

Na CD s programem najdete soubor `tsine-win.zip`. Tento soubor rozbalte například do adresáře `C:\Program Files` (nebo jiného, podle nastavení vašeho systému). Tsine pak budete spouštět buď z příkazové řádky (**Start** → **Programy** → **Příslušenství** → **Příkazový řádek**), nebo spuštěním programu **tsine.exe** v adresáři `C:\Program Files\Tsine`.

Výpočetní server nefunguje, pokud je na firewallu Windows zapnuto blokování portu číslo 2907. Pokud se objeví problémy s komunikací mezi klientem a serverem, zkuste firewall dočasně vypnout. Pozor! Tím značně zvyšujete nebezpečí napadení počítače. Na řešení tohoto problému se pracuje...

Pokud nemáte přístup k databázi Postgresql, nainstalujte ji, buď pomocí instalátoru na přiloženém CD, nebo jej stáhněte z www.postgresql.org. Zbývá už jen nastavení databáze, pokračujte dále podle části A.1.3.

Vlastní instalace

Instalace aplikace se provede ze zdrojového balíčku `tsine-0.1.tar.gz`. Po jeho rozbalení ve vzniklém adresáři spusťte jako root příkaz

```
# python setup.py install
```

Tím se nainstalují moduly Tsine do vaší instalace Pythonu.

Tsine funguje i nenainstalované – v tom případě spouštějte všechny skripty Tsine z adresáře `scripts`.

A.1.3 Nastavení databáze

Ověřte, že v databázi Postgresql máte vytvořen uživatelský účet a databázi. Pokud ne, požádejte o to vašeho správce systému, nebo vytvořte nový uživatelský účet sami (už jako obyčejný uživatel, ne root):

```
$ createuser --pwprompt -U postgres <vaše_uživatelské_jméno>
```

Po spuštění tohoto příkazu odpovídejte na otázky (většinou ano) a zadejte také heslo pro nového uživatele.

Teď je čas vytvořit samostatnou databázi pro Tsine:

```
$ createdb -U <vaše_uživatelské_jméno> tsine
```

Poté je třeba databázi nastavit, resp. provést instalaci tabulek do databáze. K tomu slouží příkaz **tsine-install-db**. Jeho parametry jsou přístupové údaje k databázi, především:

- adresa databázového serveru
- uživatelské jméno
- heslo
- jméno databáze (např. `tsine`)

- předpona tabulek – můžete nastavit vlastní předponu názvů tabulek, která slouží k rozlišení různých instalací Tsine ve stejné databázi

Pro více informací o volbách tohoto příkazu zadejte

```
$ tsine-install-db --help
```

A.2 Začínáme s Tsine

Tsine se skládá ze dvou hlavních součástí – klienta, který umožňuje spouštět a řídit výpočty rozvrhu a zobrazovat jejich výsledky a serveru, který výpočty provádí a umožňuje jejich sledování více klientům současně. Kromě toho Tsine obsahuje další utility, které slouží přípravě vstupních dat.

Před prvním spuštěním je dobré vytvořit vstupní data pro výpočet, o tom více v části A.3. Vzorová data o rozvrhu najdete také na přiloženém CD.

Načtení dat do databáze

Data pro Tsine uložená ve formátu Tsine XML načtete do databáze příkazem

```
$ tsine-db-loader <jméno vstupního souboru>
```

Spuštění serveru

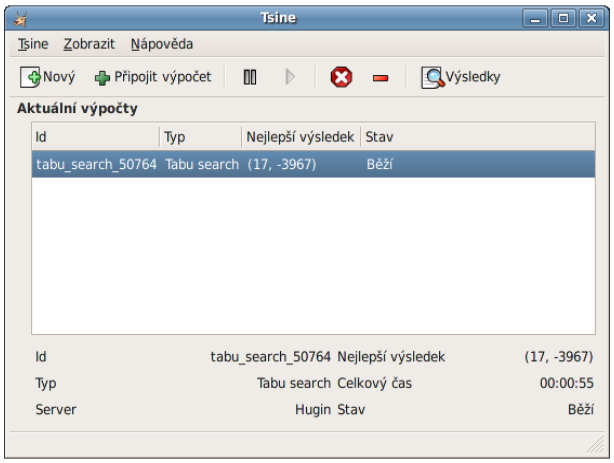
Pro spuštění serveru na portu 2907 zadejte příkaz:

```
$ tsine-server
```

Spuštění klienta

Klienta spustíte příkazem **tsine** bez parametrů. Po spuštění se objeví okno s nastavením připojovacích údajů k databázi. Pokud Tsine spouštíte pouze na jednom počítači (tj. server, klienta i databázový server), je třeba pouze nastavit uživatelské jméno a heslo, tyto údaje by měly být shodné s údaji zadávanými při instalaci tabulek v databázi.

Obrázek A.1: Hlavní okno Tsine



Nový výpočet

Poté (pokud jste již spustili na nějakém počítači server Tsine) můžete spustit nový výpočet rozvrhu – pomocí **Tsine** → **Nový výpočet** (**Ctrl**–**N**). Zobrazí se seznam serverů, který je při prvním spuštění prázdný. Přidejte nový server pomocí tlačítka **Nový** v seznamu serverů. Pokud výpočet běží jen na vašem počítači, stačí nastavit pouze jméno serveru (které stejně slouží jen pro identifikaci v programu). Pak stiskněte tlačítko **Připojit**. Zobrazí se seznam typů výpočtů, které jsou na serveru k dispozici. Vyberte jeden z nich a stiskněte **Spustit**.

Průběh výpočtu

Výpočet chvíli poběží než vyprodukuje nějaké výsledky, mezitím se ve sloupci **Nejlepší výsledek** zobrazuje (-1, -1). Až se objeví nějaký výsledek (tj. hodnota se změní na nějakou smysluplnou), můžete jej zobrazit pomocí **Zobrazit** → **Výsledky** (**Ctrl**–**R**).

Uložení výsledků

V okně Výsledky můžete zobrazovat rozvrhy nebo je uložit, buď do XML pro další použití v Tsine pomocí **Výsledky** → **Uložit** (**Ctrl**–**S**) , nebo do HTML vhodného například pro zobrazení na webových stránkách školy pomocí **Výsledky** → **Exportovat** .

Ukončení výpočtu

Výpočet skončí buď sám (po uplynutí předem nastaveného času), nebo ho můžete ukončit pomocí tlačítka **Zastavit** na panelu nástrojů v hlavním okně aplikace.

A.3 Vstupní data výpočtu

Předtím než může Tsine vůbec nějaký rozvrh spočítat, je třeba mu připravit vstupní data s popisem požadavků na rozvrh. To je možné udělat dvěma způsoby – jednodušší je určitě vytvoření souboru v tabulkovém editoru (jako například Microsoft Excel nebo OpenOffice Calc) a jeho následný převod (pomocí dodaných nástrojů) do formátu, kterému Tsine „rozumí“, nicméně pro pochopení významu jednotlivých vstupních položek není na škodu přečíst si i část o formátu Tsine XML...

A.3.1 Tsine XML

Tsine používá vlastní dialekt XML pro popis vstupních dat výpočtu. Jeho příklad najdete ve zdrojovém balíčku v souboru `data/xml/school.xml`. Pro podrobný popis jednotlivých elementů můžete použít i XML Schema vstupního souboru `data/xml/school.xsd`.

Informace o učitelích

Tag `teachers` obsahuje seznam informací o jednotlivých učitelích. Pro každého učitele je třeba uvést v tagu `teacher` jeho jméno a identifikátor, který by měl začínat 't_'. Navíc vnořené tagy `roomPref` obsahují informace o jeho časových možnostech. Ty jsou vyjádřeny hodnotou 0 až 10, přičemž 0 znamená, že učitel v danou hodinu určitě nemůže a Tsine se pak snaží na tyto hodiny nerozvrhovat tomuto učiteli žádnou výuku. Navíc, pokud pro nějaký časový úsek není uvedena žádná hodnota, je to stejné, jako kdyby tato hodnota byla 0.

Místnosti a budovy

Budovy a místnosti jsou popsány hierarchií tagů – `buildings`, `building`, `room` a `roomCategory`. Tag `buildings` ohraničuje sekci s popisem budov, v dokumentu se vyskytuje právě jednou a může obsahovat jeden nebo více

Příklad A.3.1 Definice učitele a jeho časových možností

```
<teacher id="t_ciglerova" name="Ludmila Ciglerova">
   <timePref day="1" hour="3" value="1"/>
   <timePref day="2" hourFrom="2" hourTo="6"
             value="5"/>
   <timePref dayFrom="2" dayTo="4"
             hourFrom="2" hourTo="6" value="6"/>
</teacher>
```

tagů `building` s popisem jednotlivých budov (ty kromě obvyklého atributu id obsahují také jméno a adresu).

Místnosti mají vyhrazen tag `room`, který je obsažen v některém z nadřazených tagů `building`. Místnost má tyto atributy: `id`, jméno (`name`) a kapacitu (`capacity`). Kromě toho je možné přiřadit tzv. kategorie, pomocí vnořeného tagu `roomCategory`, respektive jeho atributu `class`. Kategorie místností slouží ke snadnému a úspornému popisu požadavků jednotlivých předmětů na místnosti – předmět požaduje pouze místnost patřící do určité kategorie, nikoliv pouze jednu specifickou. Podrobnější popis viz níže v sekci věnované předmětům.

Příklad A.3.2 Definice budovy a místnosti

```
<building id="b_voderadska"
   name="Gymnazium Voderadska"
   address="Voderadska 2, 100 00, Praha 10">
   <room id="r_phy_1" name="Fyzika 1" capacity="32">
      <roomCategory class="rc_physics" />
      <roomCategory class="rc_classroom" />
      <roomCategory class="rc_video" />
      <roomCategory class="rc_darkening" />
   </room>
</building>
```

S popisem místností souvisí ještě jedna část vstupního XML souboru – sekce ohraničená tagem `roomCategories`. V této sekci jsou definovány jednotlivé kategorie místností, pomocí již zmiňovaného tagu `roomCategory`. Tentokrát se však používají jiné jeho atributy, konkrétně `id` (identifikátor,

odpovídá hodnotám zadávaným v popisu místností do atributu class), `name` a `desc` (nepovinné atributy sloužící pouze pro popis kategorie).

Předměty

Sekce popisující předměty je vymezena tagem `subjects`. Každý předmět je popsán vlastním tagem `subject`, který kromě obvyklých atributů `id` a `name` může obsahovat další vnořené tagy `roomPref`, které pomocí výše zmíněných kategorií místností popisují požadavky předmětu na místnosti.

Tag `roomPref` má dva atributy – `room` (hodnotou je identifikátor kategorie místností) a `value`. Atribut `value` může nabývat hodnot od 0 do 10, přičemž 0 znamená, že v místnosti dané kategorie předmět vůbec nemůže probíhat. Pokud pro daný předmět některá místnost nemá nastavenou vůbec žádnou hodnotu, bere se jako výchozí hodnota preference 5. Pokud je některé místnosti přiřazeno více kategorií, preference předmětu pro tuto místnost je rovna maximu preferencí těch kategorií místnosti, pro které předmět nějaké preference nastavil.

Příklad A.3.3 Definice předmětu

```
<subject id="su_matematika" name="Matematika">
   <roomPref room="rc_fyzika" value="4"/>
</subject>
```

Tělocvik je dobré rozvrhovat jen do tělocvičen, proto je ovšem nutné vytvořit speciální kategorii `rc_telocvicna`, kterou přiřadíte všem tělocvičnám. Navíc je třeba ostatním učebnám nastavit preferenci na 0, nejlépe tak, že všechny učebny zařadíte do kategorie `rc_ucebna`, a této kategorii nastavíte preferenci na 0. Pokud tedy budete chtít pro předmět zakázat, aby se rozvrhoval do nějaké místnosti, je třeba zkontrolovat preference a kategorie dané místnosti.

Studenti

Pro popis skupin studentů se používá hierarchická struktura, obsahující prvky dvou typů – skupiny studentů (popsané tagem `students`) a rozdělení skupiny studentů (tag `studentsLayout`).

Skupina studentů odpovídá například třídě, ale může to být i její část, třeba podskupina pokročilých angličtinářů, nebo naopak její nadmnožina

– studenti celého ročníku, nebo dokonce celé školy. Skupina studentů je popsána tagem `students`, s atributy `id`, `name`, `desc` a `size` (identifikátor, jméno, popis a počet studentů).

Rozdělení skupiny odpovídá disjunktnímu rozdělení skupiny na podskupiny, například rozdělení na 2 skupiny různě pokročilých angličtinářů, či rozdělení na chlapce a dívky při tělocviku. Skupiny ze stejného rozdělení mohou mít rozvrženy hodiny do stejného času, skupiny z rozdílných rozdělení ne.

Příklad A.3.4 Definice skupiny studentů a jejích podřízených rozdělení

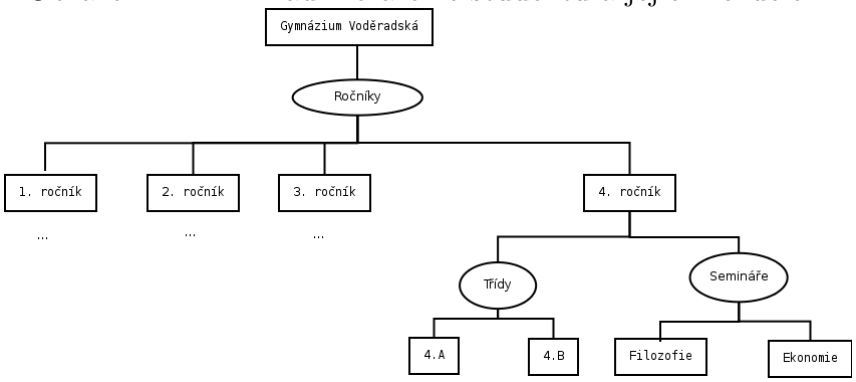
```
<students id="s_Q-A" name="Q.A" size="26">
   <studentsLayout id="sl_Q-A-aj" together="true">
      <students id="s_Q-A-aj-A" size="14" />
      <students id="s_Q-A-aj-B" size="12" />
   </studentsLayout>
   <studentsLayout id="sl_Q-A-nj" together="true">
      <students id="s_Q-A-nj" size="16" />
      <students id="s_Q-A-nj" size="13" />
   </studentsLayout>
   <studentsLayout id="sl_Q-A-tv" together="true">
      <students id="s_Q-A-tv-chlapci" size="12" />
      <students id="s_Q-A-tv-divky" size="14" />
   </studentsLayout>
</students>
```

Kořenem hierarchie studentů je skupina studentů zahrnující všechny studenty školy. Ta obsahuje alespoň jedno rozdělení (obvykle právě jedno) do disjunktních skupin – jednotlivých ročníků, nebo rovnou tříd, pokud nejsou žádné hodiny společné pro studenty z více tříd jednoho ročníku (jako jsou například volitelné semináře).

Položky rozvrhu

Položky rozvrhu jsou popsány v sekci ohraničené tagem `timetableItems`. Každá položka rozvrhu je popsána svým tagem `timetableItem`, s atributy `teacher` (obsahem je identifikátor učitele podle seznam učitelů, který danou hodinu učí), `students` (identifikátor studentů), `subject` (identifikátor předmětu) a `lessons` (počet hodin pro danou položku týdně).

Obrázek A.2: Příklad hierarchie studentů a jejich rozdělení



Příklad A.3.5 Definice položky rozvrhu

```
<timetableItems>
   <timetableItem teacher="t_ciglerova"
                  subject="su_matematika"
                  students="s_1-A"
                  lessons="4"/>
</timetableItems>
```

Uspořádání jednotlivých sekcí v konfiguračním souboru

Z technických důvodů (nastavení referenční integrity v tabulkách databáze) je třeba v konfiguračním souboru zachovat následující pořadí sekcí s daty.

1. `roomCategories` – sekce s popisem kategorií místností
2. `buildings` – popis místností a budov
3. `teachers` – seznam učitelů
4. `subjects` – seznam předmětů
5. `students` – seznam skupin studentů
6. `timetableItems` – seznam rozvrhovaných položek rozvrhu

A.3.2 MS Excel XML

Jednodušším způsobem zadávání vstupních dat o rozvrhu je použití kancelářského software schopného exportu do formátu XML programu Microsoft Excel, jako například Openoffice.org Calc.

Formát vstupních dat je podobný zadávání dat do relační databáze. Jednotlivé sešity souboru odpovídají tabulkám databáze, první řádek každého sešitu odpovídá hlavičce tabulky a další řádky obsahují data.

Struktura souboru

Soubor se vstupními daty musí obsahovat tyto sešity (přičemž záleží na jejich pojmenování, ale nikoliv na pořadí):

students Sešit obsahující popis studentů – obsahuje sloupce `id` (identifikátor studentů), `name` (jméno), `desc` (popis), `size` (počet studentů) a `parentLayout` (nadřazené rozdělení – viz výše). Skupina, která nemá vyplněnou položku `parentLayout`, se považuje za nadřazenou všem ostatním.

studentsLayouts Sešit obsahující popis rozdělení skupin studentů – obsahuje sloupce `id` (identifikátor rozdělení), `name` (jméno), `together` (zda mají být podřízené skupiny rozvrhovány společně) a `parentLayout` (nadřazená skupina studentů, tj. skupina, kterou toto rozdělení rozděluje).

teachers Sešit obsahující seznam učitelů – má sloupce `id` (identifikátor rozdělení) a `name` (jméno).

rooms Sešit obsahující seznam místností – se sloupci `id` (identifikátor rozdělení), `name` (jméno), `capacity` (kapacita místností) a `b_id` (identifikátor budovy, ve které se místnost nachází).

roomCategories Sešit obsahující seznam kategorií místností – sloupce `id` (identifikátor rozdělení), `name` (jméno), `desc` (popis).

roomCategory Sešit obsahující přiřazení místností do kategorií – má dva sloupce, `r_id` (identifikátor místnosti) a `rc_id` (identifikátor kategorie).

subjects Sešit obsahující seznam předmětů – obsahuje sloupce `id`, `name` a `lessonLength` (délka hodiny).

buildings Sešit obsahující seznam budov – sloupce `id`, `name` a `address`.

timetableItems Sešit obsahující seznam položek rozvrhu – sloupce `s_id` (identifikátor předmětu), `t_id` (identifikátor učitele), `st_id` (identifikátor studentů) a `lessons` (počet hodin týdně).

roomPref Sešit obsahující preference předmětů pro kategorie místností – sloupce `s_id` (identifikátor předmětu), `rc_id` (identifikátor kategorie místností) a value (hodnota preference, celé číslo v rozmezí 0 až 10).

teacherPref Sešit obsahující časové preference učitelů – sloupce `t_id` (identifikátor učitele), `dayFrom` (od kterého dne preference platí), `dayTo` (do kterého dne platí), `hourFrom` (počáteční hodina), `hourTo` (koncová hodina) a `value` (hodnota preference, celé číslo od 0 do 10). Pro jednoho učitele může být v seznamu více záznamů, musí však být časově disjunktní.

Zásady pro přípravu vstupních dat ve formátu MS Excel XML

Při přípravě vstupních dat je třeba dodržet několik zásad:

- Zachovat pojmenování sešitů v souboru – podle jména sešitu Tsine pozná, o jakou sekci se jedná
- První řádek každého sešitu obsahuje hlavičku – její obsah není nijak závazný, ale aspoň nějaký by tam měl být
- Snažte se vyhnout zbytečnému formátování – mohlo by Tsine zmást...
- Jednotlivé hodnoty v políčkách `id` apod. si musí navzájem odpovídat, tak aby byla data konzistentní (tj. například u položek rozvrhu je potřeba uvést správný identifikátor učitele, tedy takový, který je uveden někde v tabulce učitelů apod.)
- Záleží na pořadí sloupců v jednotlivých sešitech, tj. například je-li v prvním sloupci identifikátor a ve druhém jméno, nelze toto pořadí změnit

Po vytvoření dat soubor uložíte jako XML pomocí **Soubor** → **Uložit jako**, a následném okně vyberte typ souboru "MS Excel 2003 XML".

Převod souboru v MS Excel XML do Tsine XML

Předtím než bude moci Tsine vámi připravená data nahrát do své databáze, je třeba je převést do formátu Tsine XML. K tomu slouží příkaz

```
$ tsine-excel2tsine -o <výstupní soubor> <soubor v Excel XML>
```

A.4 Tvorba rozvrhů v Tsine

Tsine umožňuje kombinovat automatické rozvrhování s ručním prostřednictvím grafického rozhraní klienta. Tsine může počítat rozvrh od začátku, nebo začít s výpočtem s dříve uloženým rozvrhem a ten pak dále vylepšovat.

A.4.1 Výpočet rozvrhu v Tsine a jeho ruční úpravy

Tuto možnost použijte, pokud chcete rozvrh spočítat od začátku, například proto, že nemáte uložen jiný, už dříve spočítaný rozvrh. Po spuštění výpočtu je poté, co výpočet dosáhne nějakého výsledku, možné zobrazit jeho výsledky.

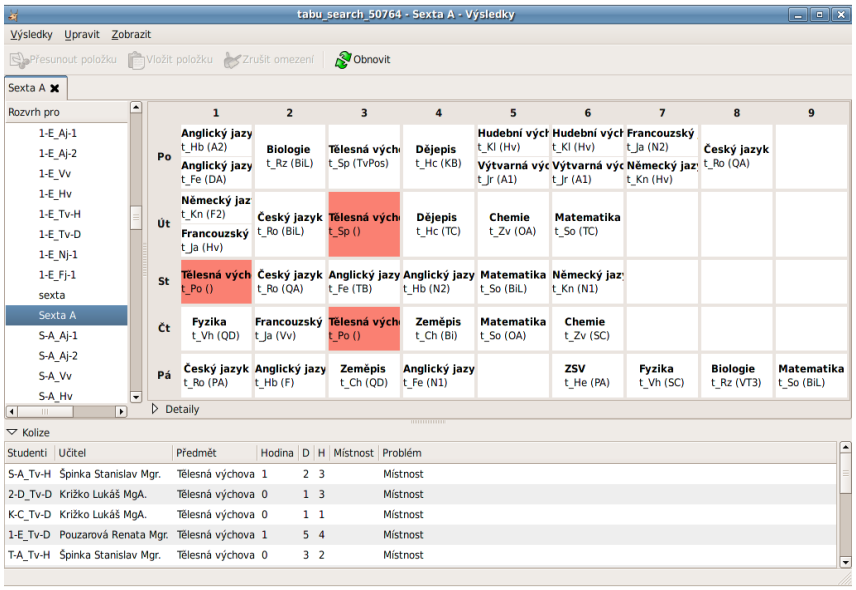
Řízení výpočtu

Průběh výpočtu je možné sledovat a řídit z hlavního okna aplikace. Výpočet můžete pozastavovat i úplně zastavit, je také možné zrušit sledování výpočtu, nebo ukončit klienta, přičemž výpočet dále poběží na serveru. Výpočet je možné přidat znovu do seznamu pomocí **Tsine** → **Připojit výpočet**. Zobrazí se seznam serverů a výběrem některého z nich (nebo přidáním nového) se po stisknutí **Připojit** zobrazí seznam aktivních výpočtů – nejen běžících, ale i těch, které již skončily.

Zobrazení rozvrhu pro studenty/učitele/místnost

V okně Výsledky lze zobrazit rozvrh pro skupinu studentů/učitele/místnost výběrem ze seznamu na levé straně. Lze zobrazit i více rozvrhů současně, pomocí tzv. pohledů (viz **Pohled** → **Nový**).

Obrázek A.3: Zobrazení výsledků a kolizí v rozvrhu



Kolize v rozvrhu

V dolní části okna výsledků je seznam kolizí v rozvrhu. U každé kolize je rovněž uvedeno, z jakého důvodu nastává. Položky způsobující kolizi jsou v rozvrhu označeny červeně. Možné důvody kolizí v rozvrhu jsou:

- Položka nemá přiřazenu žádnou místnost – takové položky zobrazíte v rozvrhu pro místnost „Žádná“ (jako důvod kolize je uvedeno „Místnost“)
- Do místnosti je v jeden čas rozvrženo více položek (důvod kolize je „Místnost“)
- Učitel má v některou hodinu učit víckrát (důvod kolize „Učitel“)
- Dvě skupiny studentů, které nejsou navzájem disjunktní, mají v nějakou hodinu více hodin (důvod kolize „Studenti“).
- Skupina studentů má některý předmět víckrát za den (důvod kolize „Předmět“)

Ruční přesouvání položek

Položky lze ručně přesouvat buď pro aktuální výpočet, nebo pro všechny výpočty (a tak vlastně ručně předrozvrhovávat položky). Pro přesouvání (a vůbec provádění všech dalších úprav) jen pro aktuální výpočet vyberte **Upravit** → **Jen pro tento výpočet**, pro úpravy pro všechny výpočty vyberte **Upravit** → **Pro všechny výpočty**. Po výběru položky stiskněte **Přesunout položku**, poté vyberte pozici v rozvrhu, kam chcete položku přesunout a stiskněte **Vložit položku**. Takto můžete i například zafixovat položku do jejího už rozvrženého času, aby jí náhodou algoritmus nepřerozvrhl jinam. Položka zafixovaná do nějakého času pro tento výpočet je zvýrazněna modře, položka zafixovaná pro všechny výpočty je zvýrazněna zeleně.

Pro přiřazení pevné místnosti položce tuto položku vyberte stejně jako v předchozím případě, následně vyberte (například v novém pohledu) zobrazení rozvrhu pro přiřazovanou místnost a stiskněte **Vložit položku**. Kromě místnosti se takto zafixuje i čas položky.

Pokud už je položka zafixována, můžete její zafixování zrušit (bez přesouvání zpět) pomocí tlačítka **Zrušit omezení**. Pokud bylo vybráno provádění změn jen pro tento výpočet, musíte ke zrušení omezení platného pro všechny výpočty zaškrtnout v menu volbu **Upravit** → **Pro všechny výpočty** (a naopak). Všechny změny můžete samozřejmě vrátit zpět.

Uložení a export rozvrhu

Rozvrh, který Tsine spočítal (a vy případně ručně upravili), si můžete uložit. První možností je uložení do formátu XML, pomocí **Výsledky** → **Uložit**. Takto uložený rozvrh pak budete moci využít při dalších výpočtech – bude sloužit jako základ výpočtu pro Tsine, více v části A.4.2.

Druhou možností je export do formátu HTML, pomocí **Výsledky** → **Exportovat** – posléze vyberte adresář, do kterého budete data exportovat (nejlépe prázdný).

A.4.2 Nahrání uloženého rozvrhu a jeho další úpravy

Dříve uložený rozvrh lze nahrát v hlavním okně aplikace pomocí **Tsine** → **Načíst rozvrh**. Při spouštění nového výpočtu je pak třeba zaškrtnout, zda má výpočet probíhat od začátku, nebo použít načtený rozvrh.

Další úpravy probíhají již stejně jako při výpočtu od začátku.

A.5 Konfigurace Tsine

Tsine načítá své nastavení z několika konfiguračních souborů. Pomocí tohoto nastavení můžete měnit například počet dnů a hodin v rozvrhu nebo parametry výpočetních algoritmů.

A.5.1 Konfigurační soubor `config/tsine.conf`

Po rozbalení zdrojového balíčku najdete v adresáři `config` soubor pojmenovaný `tsine.conf-dist`. Tento soubor obsahuje výchozí hodnoty parametrů jako je počet dní v týdnu, počet hodin denně, parametry algoritmů apod. Pokud je budete chtít změnit, zkopírujte tento soubor do nového souboru s názvem `tsine.conf`. Můžete jej také zkopírovat do svého domovského adresáře pod názvem `.tsine`. V takto zkopírovaném souboru už můžete měnit parametry libovolně – více informací o jednotlivých parametrech najdete v přímo v souboru `config/tsine.conf-dist`.

A.5.2 Adresy, ze kterých je dovoleno připojení

V souboru `config/allowed_addresses.ini` je uveden seznam adres, ze kterých je dovoleno připojovat se k výpočetnímu serveru a spouštět na něm výpočty. Na každé řádce je uvedena jedna adresa a síťová maska, ze které se lze připojit na server.

Příklad A.5.1 Příklad řádky v souboru `allowed_addresses.ini`

```
#Protokol    Adresa    Sitova maska
IPv4   127.0.0.1    255.255.255.0
```

Příloha B

Programátorská dokumentace

B.1 Architektura aplikace

Tsine je navržena jako klient/server aplikace. Základní představa fungování je naznačena na obrázku B.1. Díky této struktuře může běžet výpočet na jiném počítači (a potenciálně výkonnějším) než klient. Výpočet také může pokračovat i bez lidské asistence, například přes noc.

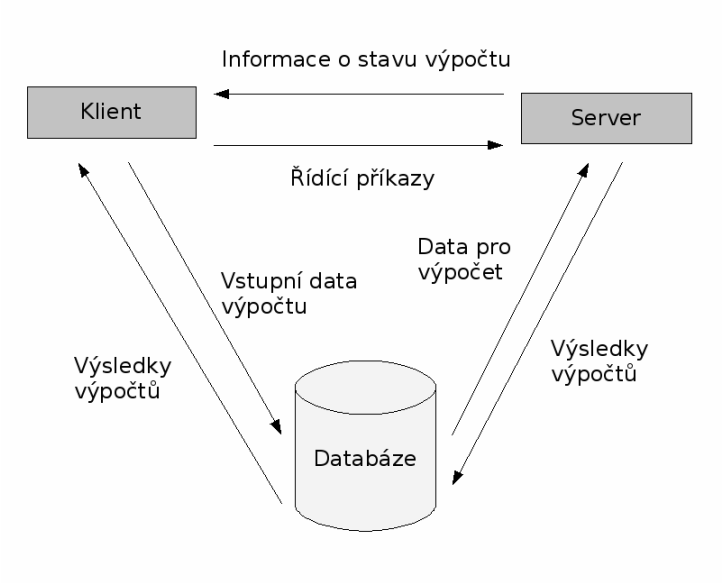
Server běží standardně na portu 2907 (což lze změnit volbou `-p port` při spuštění. Při požadavku od klienta server nejdříve zkontroluje, zda má klient povoleno se k serveru vůbec přihlásit – jednoduše porovná IP adresu klienta se seznamem povolených IP adres (a síťových masek) v souboru `config/allowed_addresses.ini`.

B.1.1 Databáze

Pro uložení vstupních dat v průběhu výpočtu a výsledků výpočtů je použit databázový server PostgreSQL. Použití databáze pro uložení dat má několik výhod:

- Perzistence dat – data jsou k dispozici i po skončení aplikace
- Databáze zajišťuje integritu vstupních dat
- Aplikace může přistupovat k datům v podstatě libovolně, omezena je pouze možnostmi SQL
- Odpadá nutnost opakovaného odesílání dat mezi klientem a serverem

Obrázek B.1: Architektura aplikace



- Podstatné zjednodušení komunikace mezi klientem a serverem

Nevýhodou databáze je určité snížení výkonu při opakovaném načítání týchž dat. To je řešeno cachováním jak na straně serveru, tak na straně klienta. Cache se uplatňuje vždy pouze v rámci výpočtu, pro každý nový výpočet jsou data z databáze načtena znovu.

Otázkou, která přímo souvisí s návrhem aplikace, je použití souborového úložiště dat, jako je například SQLite. Toto řešení by mělo smysl, pokud by obě části aplikace běžely na stejném počítači, při použití síťového řešení se však nehodí.

B.1.2 Vstupní data

Vstupní data jsou primárně uložena v XML souboru. Tento formát byl zvolen pro svou rozšiřitelnost a flexibilitu; především však proto, že umožňoval (díky svému „stromovému“ charakteru) přirozeně vyjádřit hierarchii studentů a tříd. Kromě toho je pro tento formát k dispozici řada standardních nástrojů, od knihovny Pythonu, přes XSL procesory po editory schopné vy-

tvářet soubory v tomto formátu uživatelsky přívětivějším způsobem (jako například tabulkové procesory).

O načítání dat do databáze se stará třída `DataImporter` nacházející se v modulu `XmlImport` z balíku `Tsine.client`. Ta obsahuje objekt třídy `DataXmlHandler`, SAX parser vstupního XML souboru.

Import dat do databáze lze provádět dvěma způsoby – buď pomocí skriptu `scripts/tsine-db-loader` nebo prostřednictví grafického rozhraní. V prvním případě běží import v hlavním vlákně skriptu, v druhém případě běží v samostatném vlákně.

Tsine umožňuje zadávání dat také ve formátu MS Excel XML. Tento formát nelze použít přímo pro načítání, nejprve se musí pomocí skriptu `scripts/tsine-excel2tsine` převést na formát Tsine XML. Převod je realizován XSL stylem `data/xml/excel2tsine.xsl`.

B.2 Popis implementace

B.2.1 Síťová komunikace klient – server

Komunikační protokol

Klient a server mezi sebou komunikují prostřednictvím jednoduchých textových správ. Spojení používá protokol TCP. Seznam příkazů najdete v distribuci Tsine v souboru `docs/communication_protocol.txt`. Jednotlivé příkazy jsou zabaleny do XML-like značek, viz příklad. Rozlišují se dva typy příkazů – servisní (typ *serv*) a informační (*info*).

Příklad B.2.1 Přikaz od klienta

```
<message type="serv">
run simulated_annealing
</message>
```

Rozhraní serveru na straně klienta

Každý server, ke kterému je klient připojen, je reprezentován objektem třídy `Server` z modulu `Tsine.client.TsineServer`. Tento objekt obsahuje metody pro přijímání a odesílání zpráv z/na server. Obsahuje také socket, přes který klient se serverem komunikuje.

Kromě toho je každý výpočet na straně klienta reprezentován objektem třídy `Computation` z modulu `Tsine.client.Computation`. Jeho metody slouží k zjišťování stavu a dalších informací o výpočtu.

Zprávy od serveru jsou zpracovávány metodou `dispatch_message` třídy `mainWindow`. Ta podle typu příkazu provádí příslušné akce.

Rozhraní klienta na straně serveru

Server je schopen obsluhovat více klientů současně – využívá k tomu volání funkce `socket.select`. Každý klient je reprezentován objektem třídy `Client` z modulu `Tsine.server.client`. Tento objekt obsahuje metody pro komunikaci s klientem a také obsahuje odkazy na výpočty klientem sledované.

B.2.2 Výpočet rozvrhů

Výpočet rozvrhů probíhá na straně serveru. Každý výpočet běží v samostatném vlákně a udržuje samostatné spojení s databází – najednou může probíhat více výpočtů na různými databázemi (přitom v jedné databázi mohou být pouze data pro jeden rozvrh; přesněji, pokud se použije jiná předpona názvů tabulek pro různá data, i toto omezení lze obejít...).

Každému výpočtu odpovídá jedna instance třídy `Computation` v modulu `Tsine.server.Computation`. Tento objekt si udržuje seznam klientů, kteří sledují výpočet (instance třídy `Tsine.server.Client.Client`), tak aby byl schopen je rychle informovat o změnách souvisejících s výpočtem. Výpočet má přiřazen jednoznačný identifikátor ve tvaru `<typ_výpočtu>_<číslo>`.

Objekt třídy `Computation` obsahuje odkaz na výpočetní vlákno – potomka třídy `CompThread` v balíku `Tsine.comp`. V současnosti jsou implementovány dvě odvozené třídy – `SimulatedAnnealing` a `TabuSearch`.

Spuštění výpočtu

Ještě než může klient spustit nějaký výpočet, musí od serveru získat seznam dostupných rozvrhů. Pošle mu příkaz `info avalComp` a server odpoví zprávou `info avalComp <id> <jméno>\n ... (další typy výpočtů) ...` se seznamem výpočtů, které lze na tomto serveru spustit.

Po výběru výpočtu klient pošle serveru zprávu `serv run <typ výpočtu> [<použít rozvrh>]`. Příznak `použít rozvrh` určuje, zda má výpočet použít jako počáteční řešení rozvrh uložený v tabulce `savedtimetable`. Server

vytvoří příslušný objekt typu `Computation` a spustí výpočet. Přitom vloží záznam o výpočtu do databáze do tabulky computation.

Průběh a řízení běhu výpočtu

Kdykoliv v průběhu výpočtu může klient poslat serveru zprávu `serv pause <id>`, `serv resume <id>` nebo `serv stop <id>`, kterou pozastaví, spustí nebo zastaví výpočet s daným identifikátorem `id`. Server naopak při změně stavu výpočtu (zastavení apod.) aktualizuje záznam výpočtu v tabulce databáze computation a pošle klientovi zprávu `info compState <id> <stav>`.

Rozlišují se tyto stavy výpočtu:

- running – výpočet běží
- paused – výpočet je pozastaven
- stopped – výpočet je zastaven
- finished – výpočet skončil
- error – při výpočtu došlo k chybě

Vlastní výpočet

Výpočet rozvrhu mají na starost moduly z balíku `Tsine.comp`. Probíhá v několika fázích:

1. Vytvoří se popis problému pro constraint solver (prostřednictvím metod třídy `Tsine.comp.ConstraintSolver.ConstraintSolver`)
2. Spustí se constraint solver (`Tsine.constraint`)
3. Vytvoří se instance třídy `Timetable` a do ní se uloží nalezené řešení (tuto třídu najdete v modulu `Tsine.comp.Timetable`)
4. Spočítá se počáteční ohodnocovací funkce rozvrhu a rozdělení položek do místností (`timetable.evaluate`, `timetable.assignRooms`)
5. Pomocí lokálního prohledávání je rozvrh zlepšován

Lokální prohledávání

Lokální prohledávání probíhá pomocí změn v aktuálním rozvrhu – tyto změny jsou dvou druhů:

- Přemístění položky rozvrhu do jiného času
- Prohození dvou položek rozvrhu

Změny v rozvrhu se nejprve zkouší – pomocí metod `tryMove1` nebo `tryMove2` třídy `Timetable`. Tyto metody vrací objekt třídy `Move` s popisem změny a také s výslednou ohodnocovací funkcí po (hypotetickém) provedení změny. Metoda `Timetable.applyMove` pak tento objekt bere jako parametr a změnu skutečně provede. Pro získání většího množství možných tahů lze také použít funkce `Timetable.getNeighbourhood1` a `Timetable.getNeighbourhood2`, které vrací seznam tahů přesouvajících položky, resp. prohazujících položky. Funkce `Timetable.getRandomMove` pak vrací náhodně vybraný tah.

Vlastní změny v rozvrhu provádí metody třídy `Timetable` – `moveItem` a `switchItems`. Obě mají parametr `tryOnly`, který je příznakem, zda provedené změny uložit v rozvrhu, nebo je pouze vyzkoušet a vrátit hodnotu ohodnocovací funkce po hypotetickém provedení změny.

Při zkoušení změn se provádí přepočet ohodnocovací funkce – výsledek není uložen v rozvrhu a přepočítávají se pouze změněná časová okénka. Ve změněných časových oknech je spočítáno přiřazení místností párovacím algoritmem (viz modul `Tsine.comp.Matching`).

Výpočet ohodnocovací funkce rozvrhu

Ohodnocovací funkce má dvě části – ohodnocení počtu kolizí v rozvrhu a ohodnocení dalších kritérií (měkká omezení). K výpočtu obou částí ohodnocovací funkce slouží metody `Timetable.hardConstraintEvaluate`, resp. `Timetable.softConstraintEvaluate`.

Při provádění změn v rozvrhu je třeba přepočítat hodnotu ohodnocovací funkce. Tyto výpočty probíhají uvnitř třídy `Timetable`. Jde především o funkce `evaluateChangedTimeslots` (přepočet ohodonocovacích funkcí pro změněná časová okna), `evaluateChangedStudents` (přepočet ohodnocení studentských rozvrhů) a `evaluateChangedTeachers` (přepočet ohodnocení pro učitele). Rozvrh si u každého časového okna pamatuje počet kolizí v daném časovém okně a při změnách mu tak stačí k celkovému počtu kolizí

pouze přičíst rozdíl (nová hodnota – stará hodnota). Položky rozvrhu (instance třídy `TimetableItem` z modulu `Tsine.comp.Timetable`) mají u sebe rovněž poznámku, kolik kolizí způsobují. To umožňuje rychlé vyhledávání položek, které porušují nejvíce konfliktů a jejich přesunutí.

Ruční úpravy rozvrhů

Ruční úprava rozvrhu probíhá na straně klienta. V okně `resultWindow` z balíku `Tsine.client` probíhají uživatel provádí úpravy. Při každé úpravě se:

1. Přečtou záznamy v tabulce `mandatoryitems` týkající se přesouvané položky rozvrhu a aktuálního výpočtu a uloží se (pro případ vracení změn)
2. Provede změna v rozvrhu na straně klienta
3. Klient pošle serveru zprávu `serv update <id_výpočtu>`
4. Server nastaví příznak v objektu třídy `Computation` tak, aby si výpočet při nejbližší příležitosti nahrál nové změny z databáze.
5. Výpočetní vlákno zavolá metodu `Timetable.reload_constraints`, která načte změny a provede je v rozvrhu na serveru.

Při vrácení změn se je postup stejný, pouze se do databáze nahraje předchozí záznam (který byl uložen u položky rozvrhu na straně klienta).

B.2.3 Grafické uživatelské rozhraní

Hlavní okno aplikace

Po spuštění klienta se vytvoří hlavní okno aplikace – objekt třídy `mainWindow` z modulu `Tsine.client.mainWindow`. Toto okno zobrazuje seznam aktuálně sledovaných výpočtů a umožňuje je řídit. Třída `mainWindow` rovněž zpracovává příkazy od serveru a rozděluje je jednotlivým podobjektům (prostřednictvím metody `dispatchMessage`).

Okno výsledků

Okno výsledku rozvrhu je instancí třídy `resultWindow` nacházející se v modulu `Tsine.client.resultWindow`. Obsahuje odkaz na aktuální výsledek rozvrhu – instanci třídy `Timetable` z modulu `Tsine.client.Timetable`. Tento rozvrh se mění na základě uživatelských akcí. Okno výsledků umí zobrazovat několik pohledů na rozvrh – pohledy jsou instance `resultView`)

Rozvrh obsahuje položky v obyčejném seznamu a při zobrazení je filtruje podle příslušných kritérií – učitele, místnosti, nebo studentů. Položky rozvrhu jsou instance třídy `Tsine.comp.Timetable.TimetableItem`.

Někdy je třeba položku rozvrhu barevně odlišit od ostatních – k tomu slouží zvýraznění. Rozlišuje se několik druhů zvýraznění, lišících se barvou a prioritou. Položka obsahuje seznam zvýraznění, které jí přísluší a je vždy zvýrazněna barvou toho zvýraznění, které má nejvyšší prioritu. Zvýraznění sama o sobě jsou konstanty `Tsine.client.Timetable.HIGHLIGHT_*`.

B.3 Vlastní rozvrhovací algoritmus

Tsine je navržen tak, aby do něj bylo možné relativně snadno přidávat nové rozvrhovací algoritmy a metaheuristiky. Pro algoritmy založené na lokálním prohledávání je navíc možné využívat metody třídy `Timetable` v modulu `Tsine.comp.Timetable`.

B.3.1 Vytvoření jádra algoritmu

Prvním krokem při vytváření vlastního algoritmu je vytvoření potomka třídy `CompThread` v modulu `Tsine.comp.CompThread`. Je třeba zajistit správné volání konstruktoru této třídy, nejlépe tak, že okopírujete konstruktor některého již hotového algoritmu (`TabuSearch` nebo `SimulatedAnnealing`). Budeme předpokládat, že váš nový algoritmus je uložen v modulu `Algorithm` v balíku `Tsine.comp` a potomek třídy `CompThread` se jmenuje `Algorithm`.

Každý výpočet má přiřazen jednoznačný identifikátor – je přístupný uvnitř výpočetní třídy jako `self.compId`. Tento identifikátor slouží k určení, které výsledky patří ke kterému výpočtu v databázi. Kromě toho se používá i při nastavování ručních omezení na rozvrh.

Vlastní prohledávací metoda

Hlavním úkolem je předefinování metody `run`, která se stará o výpočet rozvrhu. V této metodě můžete využívat modulů `ConstraintSolver` (rozhraní ke constraint solveru) a `Timetable` (struktura umožňující uložení rozvrhu, jeho lokální změny a výpočet ohodnocovací funkce).

Třída `Timetable` poskytuje metody usnadňující lokální prohledávání. Jsou to především `getRandomMove` a dále metody `getNeighbourhood1` a `getNeighbourhood2`. Metoda `getRandomMove` vrací objekt typu `Move` – jde o výsledek náhodného prohození nebo přesunutí položky rozvrhu. Další dvě metody vrací seznamy objektů typu `Move`, z příslušných „sousedství“ aktuálního řešení. Položka `result` tohoto objektu obsahuje hodnotu ohodnocovací funkce po provedení „tahu“. Rozvrh tím zůstane nezměněn, pro skutečné provedení tahu je třeba zavolat metodu třídy `Timetable` `applyMove` a předat jí objekt typu `Move`.

Uložení výsledků výpočtů

V okamžiku, kdy rozvrh dosáhl nového výsledku, je třeba ho uložit do databáze. K tomu slouží metoda třídy `Timetable` `save`. Tu zavoláte s parametrem `self.compId`. Posléze zavoláte metodu `self.notify_new_result()` – upozorní klienty, kteří tento výpočet sledují, že si mají načíst nový výsledek z databáze.

Řízení výpočtu

Aby fungovalo řízení výpočtu (spouštění, pozastavování, či úplné zastavení), je třeba čas od času (obvykle po každé iteraci výpočtu, pokud probíhá iterativně..) zavolat metodu třídy `CompThread` `wait`. Pokud je výpočet pozastaven, tato metoda se zablokuje a vrátí hodnotu až po znovuspuštění výpočtu. Pokud metoda `wait` vrátí `False`, je to signál, že uživatel chce výpočet zastavit.

Načtení nových uživatelských změn z databáze

Podobný princip jako při řízení výpočtu je uplatněn při načítání uživatelských změn z databáze. Čas od času je třeba zkontrolovat proměnnou `reload_state` pomocí metody `self.compState.get_reload_state()`. Pokud tato vrátí `True`, znamená to, že v databázi jsou nové uživatelské změny rozvrhu. Načteme je metodou `reload_constraints` třídy `Timetable`.

Příklad B.3.1 Kontrola a načtení uživatelských změn z databáze

```
if self.compState.get_reload_state():
    timeTable.reload_constraints(self.compId)
    self.compState.set_reload_state(False)
```

Ukončení výpočtu

Na konci výpočtu je třeba zavolat metodu `self.storeEndTime`, která uloží koncový čas výpočtu a upozorní klienty, že výpočet skončil.

Protože se může stát, že se při výpočtu vypustí nějaká výjimka, je dobré celý výpočet obalit `try` blokem, který po zachycení zavolá metodu `self.storeError` a pak může zachycenou výjimku pustit dál (a tím například ukončit vlákno s výpočtem). Metoda `self.storeError` uloží do databáze čas ukončení výpočtu a poznámku, že výpočet skončil chybou.

B.3.2 Zakomponování algoritmu do celé aplikace

Aby mohli uživatelé spouštět váš nový algoritmus, je třeba udělat několik změn v modulu `Tsine.server.Computation`.

Zařazení do seznamu dostupných výpočtů

Dále je nutno zařadit váš nový výpočet do seznamu dostupných výpočtů aby jej bylo možné zobrazit uživateli. V modulu `Tsine.server.Computation` najděte definici funkce `get_available_computations`. Přidejte do seznamu `avalComp` identifikátor vašeho výpočtu – řetězec, například `"algorithm"`.

Úprava konstruktoru třídy `Computation`

V konstruktoru třídy `Computation` v modulu `Tsine.server.comp` přidejte do sekce `if compType == ...: ... elif: ... else: ...` větev s voláním konstruktoru pro váš výpočet (viz příklad B.3.2). Identifikátor je stejný jako v předchozím odstavci.

Přidání přeloženého jména výpočtu

Poslední (nepovinnou) procedurou, kterou je třeba podstoupit pro přidání nového výpočtu, je přidání jména výpočtu pro systém `gettext`. V modulu

Příklad B.3.2 Volání konstruktoru výpočetního vlákna

```
if compType == "default":
    self.compThread = CompThread.CompThread(self.id,
       self.compState,
       dbConnection = DbAccess.DBConnection(
     host = self.dbHost,
          user = self.dbUser,
          password = self.dbPasswd,
          database = self.dbName),
          parentComputation = self)
elif compType == "simulated_annealing":
    self.compThread = SimulatedAnnealing(self.id,
       self.compState,
       dbConnection = DbAccess.DBConnection(
          host = self.dbHost,
          user = self.dbUser,
          password = self.dbPasswd,
          database = self.dbName),
          parentComputation = self,
          useTimetable = useTimetable)
```

`Tsine.server.Computation` vyhledejte funkci `get_localized_type_name`. Do té přidejte větev výpočtu vracející jméno výpočtu obalené voláním funkce `gettext.gettext`, dostupné též pod aliasem `_`.

Příloha C

Obsah CD-ROM

K této práci je přiloženo CD-ROM s její elektronickou verzí ve formátu PDF, zdrojovými kódy rozvrhovacího systému Tsine a dokumentací. Zdrojové kódy jsou šiřitelné pod licencí GNU GPL verze 2 (či pozdější).

Tabulka C.1: Obsah přiloženého CD-ROM

Soubor či adresář	Obsah
/bak_prace.pdf	Tato práce ve formátu PDF
/tsine/	Adresář s distribucí Tsine
/tsine/src/	Zdrojové kódy Tsine
/doc/user_guide/	Uživatelská příručka ve formátu HTML
/doc/prg_manual/	Programátorská dokumentace ve fomátu HTML
/doc/api/	Dokumentace k modulům Tsine vygenerovaná programem EpyDoc
/doc/tsine_xml_doc.html	Dokumentace XML Schema pro Tsine XML
/tsine-win.zip	Soubory s instalací Tsine pro Windows
/data/	Ukázková vstupní data ve formátu MS Excel XML a Tsine XML
/results/	Výsledky testů ve formátu OpenDocument
/install/	Instalátory potřebných nástrojů (OpenOffice, Postgresql) pro MS Windows