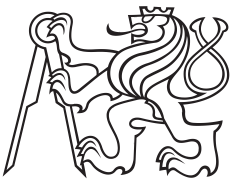
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Aplikace pro tvorbu rozvrhu

pro střední a základní školy

David Rozenberg

Duben 2014
Vedoucí práce: Půlpitel Martin Ing.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. David Rozenberg**

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: **Aplikace pro tvorbu rozvrhu**

Pokyny pro vypracování:

Nastudujte netriviální problematiku tvorby rozvrhu: http://www.unitime.org/publications.php
Prozkoumejte existující řešení: http://www.scientia.com/uk/

Navrhněte a implementujte webovou aplikaci cílenou na ZŠ a SŠ splňující zejména následující požadavky:

- profily učitelů, včetně volby časové dostupnosti
- profily učeben, včetně informace o kapacitě
- profily tříd (celá třída, skupiny dle jazyků, TV, ...)
- profily výukových jednotek (předměty, semináře, ...), včetně párování s třídou a s učebnou
- tvorba více variant návrhu celoškolního rozvrhu s pohledy na rozvrh učitele, třídy a učebny
- grafický výstup s možností přetahování jednotlivých hodin za pomoci drag and drop
- škálovatelnost (aplikace musí bez problému obsloužit řádově tisíce uživatelů v jeden okamžik)
- multilanguage
- jazyk implementace vyplyne z analýzy a návrhu řešení
- proveďte uživatelské testy a reagujte na připomínky
- testovací provoz poběží na gymnáziu: 480 studentů, 35 učitelů, 25 učeben

Seznam odborné literatury:

A multistage evolutionary algorithm for the timetable problem, Apr 1999, ISSN: 1089-778X, Author(s):Burke, E.K., Dept. of Comput. Sci., Nottingham Univ., UK , Newall, J.P.

Genetic Algorithms: A New Approach to the Timetable Problem, Print ISBN 978-3-642-77491-1, Authors: Alberto Colorni, Marco Dorigo, Vittorio Maniezzo (4)

Vedoucí: Ing. Martin Půlpitel

Platnost zadání: do konce letního semestru 2014/2015

L.S.

prof. Ing. Jiří Žára, CSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 20. 2. 2014

/ Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 05. 05. 2014

Podpis :

Abstrakt / Abstract

Tato práce se věnuje rozvrhovacímu problému, konkrétně tvorbě rozvrhu pro střední a základní školy. V první části popisuje teorii rozvrhovacího problému a v druhé části se věnuje aplikaci pro generování rozvrhu, která je hlavní bod této diplomové práce.

Klíčová slova: rozvrhovací problém, rozvrh, aplikace, silné a slabé podmínky, rekurzivní algortimus, genetický algortimus.

This work is focused on scheduling problem, specifically on creating a schedule for primary and secondary schools. First part describes theory of scheduling problem and second part is focusing on application for generation schedules, which is the main part of this master thesis.

Keywords: scheduling problem, schedule, application, soft and hard constraints, recursive algorithm, genetic algorithm.

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Generování školního rozvrhu patří do kategorie rozvrhovacích problémů a také do kategorie programování s omezujícími podmínkami. Všeobecně v rozvrhovacích problémech jde o to, že jako zadání máme konečný počet událostí a konečný počet časových oken či period. Dále máme k dispozici tzv. zdroje. Zdroje mohou být různého typu, mít určitou kapacitu, být paralelní (zvládat více událostí najednou) nebo sériové (jen jedna aktivita v určitý čas). Nakonec je dána cílová funkce, podle které se určuje kvalita řešení. Například minimální čas, rovnoměrné využití zdrojů, nejlevnější řešení a další. Rozvrhovací problém pak musí co nejlépe, vzhledem k dané cílové funkci, rozvrhnou události zadaným zdrojům. Množina počátečních časů pro události je nazývána rozvrhem.

Do množiny rozvrhovacích problémů s omezujícími podmínkami například patří Resource-Constrained Project Scheduling Problem (RCPSP). Je zadáno N úloh, M zdrojů. Úlohy mohou mít následníka nebo předka, jaký zdroj je má zpracovat a cílem algoritmu je vygenerovat rozvrh, který všechny úlohy rozmístí zdrojům tak, aby čas byl minimální a zároveň, aby toleroval předchůdce, následníky. Jako příklad uvedu objednávku ve firmě. Je potřeba skladem najít a zabalit zboží, administrativa pak vyřídí potřebné papíry a poslední úkol je předat balík poště.

Do stejné kategorie, tedy programování s omezujícími podmínkami, patří i generování školního rozvrhu. V tomto konkrétním případě figurují následující věci. Konečný počet aktivit (předmět, který má danou délku a učitele) jako události. Časová okna jsou dána výukovými dny a maximálním počtem hodin, které mohou být v jeden den učeny. Jako zdroje jsou dány časové možnosti učitelů a učeben. Cílová funkce pak zní takto: Všechny zadané aktivity musejí být přiděleny zdrojům tak, aby splňovaly zadané podmínky (pauza na oběd, maximální počet hodin denně...).

Vše bude v této práci popsáno. V první polovině se práce zabývá teorií pro rozvrhovací problémy, podmínkami a existujícími programy. A v druhé části je detailně popsána implementace takové aplikace a její testování.

Kapitola 2
Algoritmy pro generování školního rozvrhu

Při rozvrhování hodin musíme naplánovat množinu aktivit k učitelům a učebnám do časových oken tak, aby výsledný rozvrh byl realizovatelný a akceptovaný všemi lidmi, kteří jsou do rozvrhu zahrnuti. Rozvrhovací problémy jdou velmi těžce vyřešit a je známo, že tyto problémy patří do třídy NP-úplný[1]). Skoro týden práce zabere zkušenému člověku vyrobit rozvrh pro průměrnou školu (20 tříd, pro každou 30 hodin týdně) a i tak není výsledek často uspokojivý. V podstatě jde o to, že bychom museli prohledat kompletně celou množinu řešení proto, abychom našli to nejlepší z nich, které splňuje všechny zadané podmínky [2–3].

2.1 Podmínky

V úvodu bylo řečeno, že generování školního rozvrhu patří do kategorie programování s omezujícími podmínkami. Podmínka je přirozené popsání toho, jak mají věci fungovat. Je to relace mezi několika proměnnými, které mohou nabývat různých hodnot a podmínka tyto hodnoty omezuje. Uvedu jednoduchý příklad pro představu. Úkol zní: doběhni z bodu A do bodu B. Nikde není ale napsáno, kudy má dotyčný běžet nebo za jak dlouho to má zvládnout, tudíž běžec si může běžet kudy chce a jak dlouho chce. Nyní přidáme následující podmínky:

- Běž nejkratší možnou cestou přes Pražskou ulici.
- Zvládni to do pěti minut.

Nyní jsme omezili množinu hodnot tras a časových možností. Pokud nebudeme mít žádné podmínky, tak můžeme vložit jeden předmět, nějaké třídě na první hodinu v pondělí a jiný předmět jiné třídě, také na první hodinu v pondělí. Jenže tyto dva předměty mají stejné učitele. Jelikož nemáme žádné omezující podmínky, tak toto umístění je nyní možné, stejně jako u před chvílí uvedeného příkladu mohl běžet běžec kudy chtěl. Ovšem je naprosto nereálné, aby učitel učil dva předměty ve stejný čas a v různých třídách. Tato podmínka **musí** být splněna. U reálných aplikací se pak setkáváme i s podmínkami, které být splněny nemusí a i přesto dostaneme splnitelný rozvrh. Dostáváme se tak k slabým (soft) a silným (hard) podmínkám [4–5].

2.1.1 Silné podmínky

Silné podmínky (hard) jsou takové, které prostě a jednoduše musejí být splněny. Pokud není splněna jediná z nich, výsledný rozvrh není splnitelný. Jsou to podmínky, které jsou logické a všechny by v reálném světe nešly provést. Jsou to následující podmínky [6].

[1]) Úloha U je NP-úplná (NPC, NP-complete), pokud je je ve třídě NP a zároveň platí, že se na ní polynomiálně redukuje každá úloha z třídy NP. NP-úplné úlohy jsou ty „nejobtížnější“ ze všech NP úloh [1].

- Učitel nemůže učit dva předměty ve stejný čas.
- Dva různé předměty nemohou být učeny ve stejný čas a v jedné třídě (může zde být vyjímka, např. tělocvik kluci / holky).
- V jedné učebně nemohou být dvě třídy ve stejný čas.
- Třídy nemohou být v učebnách, které mají menší kapacitu než je počet žáků ve třídě.
- Každý předmět musí mít k dispozici všechny zdroje v daný čas. Musí mít přiděleného učitele, učebnu, čas a třídu.
- Předmět musí být přidělen do určitého typu učebny, pokud to vyžaduje. Např. tělocvik musí být v tělocvičně a ne v běžné učebně.
- Třída může mít maximálně N hodin denně.

2.1.2 Slabé podmínky

U slabých (soft) podmínek pouze preferujeme to, že budou splněny. Čím více je těchto podmínek splněno, tím je rozvrh lepší. Ovšem když splněny nejsou, tak to ničemu nevadí a výsledný rozvrh je správný a splnitelný. Pro představu si několik těchto podmínek opět ukážeme, ale určitě po přečtení pochopíte, že takovýchto podmínek může být celá řada a mohou být pro každou školu specifické.

- Každá třída musí mít každý den pauzu na oběd.
- V rozvrhu pro třídu může být maximálně jedna, dvě mezery denně.
- Třída by měla začínat vyučování ráno, 1. vyučovací hodinu, maximálně 2. vyučovací hodinu.
- Třídy by neměly mít stejné předměty ve stejný den.
- Třídy by měly být co nejvíce ve své vlastní „rodové“ učebně a počet přecházení z učebny do jiné učebny by měl být minimální možný.
- Učitel by měl mít pauzu na oběd.
- Učitel by měl mít nějaké pauzy mezi předměty na přípravu.

Určitě by těchto podmínek mohlo být daleko více a je z toho patrné, proč je tak těžké nějaký takový rozvrh, který by vyhovoval úplně všem, vytvořit. Pro představu vyjmenuji ještě pár takových podmínek, které již nejsou tak obvyklé, ale vykreslují to, že každá škola si může vymýšlet další a další podmínky a také to, že vytvoření takového rozvrhu dá pořádně zabrat i počítačovému programu.

- Učitel učí na dvou školách najednou. Pokud skončí na jedné v 11:00, pravděpodobně nemůže začít učit na druhé v 11:15.
- Pokud jsou dva učitelé ženatí a mají na škole dítě, je dobré zajistit, aby tito dva učitelé neučili třídu, kde toto dítě je. Navíc je dobré, aby tito dva učitelé neučili stejnou třídu.
- Pokud třídy přecházejí mezi učebnami, tak bychom měli minimalizovat počet metrů mezi nimi tak, aby stále nepřecházely z jednoho konce školy na druhý.
- Škola může mít zapůjčené laboratoře a ty mohou být k dispozici např. jen dva dny v týdnu. Proto musíme laboratorní předměty rozvrhnout do těchto dvou dnů.
- Někteří učitelé mohou učit jen některé dny, nebo preferují ráno, popřípadě odpoledne.
- V některý den chceme končit maximálně v daný čas, protože pak následuje porada.

2.2 Genetický algoritmus

Tento algoritmus není přímo určený pro rozvrhovací problém a dá se aplikovat na plno dalších optimalizačních problémů. Genetický algoritmus vychází z Darwinovy teorie o vývoji druhů. V přírodě přežívají jen ty nejsilnější a ti pak předávají svůj genetický materiál dále.

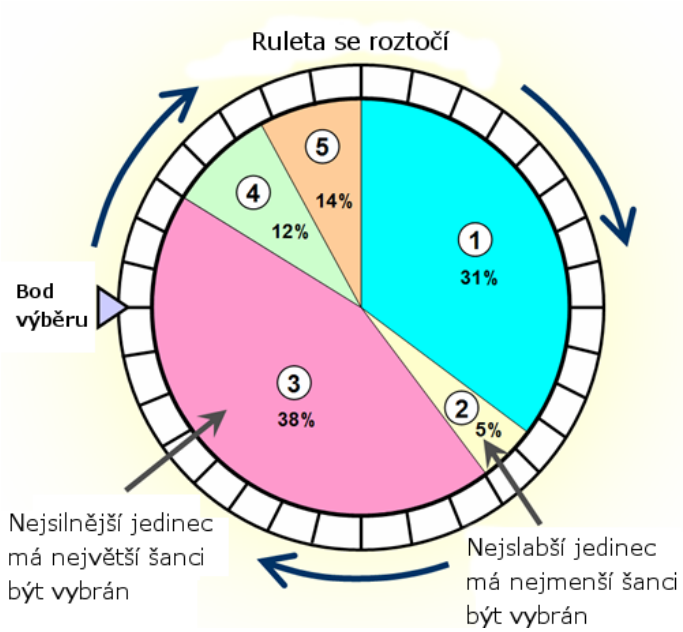
Algoritmus pracuje tak, že na začátku vygeneruje náhodnou generaci jedinců. Každý jedinec nese v sobě informaci. Tato informace je výsledek našeho algoritmu, který je často kódovaný do podoby binárního řetězce (není to konečný výsledek a v prvních generacích často špatný a nesplnitelný) a nazývá se zde i v genetice chromozom. To jak se zakóduje výsledek je samozřejmě na programátorovi a je to první věc, kterou musí vymyslet. Počet jedinců je na začátku daný a pro další vysvětlení jich je N.

Vlastnost genetického algoritmu 2.1. Fitness funkce.

Algoritmus na začátku vyhodnotí zdatnost každého jedince pomocí tzv. *fitness funkce*. Silnější jedinci (ti, kteří lépe reprezentují výsledek) budou mít lepší výsledek z fitness funkce, než ti slabší.

Vlastnost genetického algoritmu 2.2. Reprodukce.

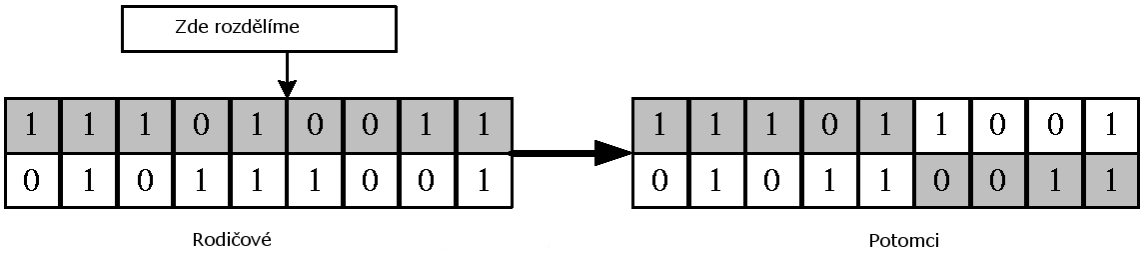
Algoritmus nyní je ve fázi **reprodukce**. Do další generace vybereme, reprodukujeme N nových jedinců tak, že vybíráme z generace nemodifikované jedince s tím, že silnější jedinci mají větší procentuální šanci být vybráni. To můžeme udělat například pomocí ruletové selekce, která je často používaná[7] viz. obr. 2.1. Momentálně tedy máme zreprodukováno N jedinců. Většina z nich by měli být ti silnější jedinci, ale mohl být vybrán i slabší, popřípadě několikrát ten samý. Na řadě jsou teď kroky *křížení* a *mutace*. Pokud bychom totiž stále opakovali kroky *reprodukce*, tak by generace postupně degenerovaly. Zbylo by v ní jen pár nejsilnějších jedinců z původní generace a naprosto nezměněných.



Obrázek 2.1. Zde máme pět jedinců. Pětkrát roztočíme ruletou a vybereme 5 členů na reprodukci, mohou se i opakovat. Jedinec číslo 3 má největší šanci být vybrán.

Vlastnost genetického algoritmu 2.3. Křížení.

Křížení je operace, která zabraňuje degeneraci. Spočívá v tom, že si jedinci vymění navzájem informace. Probíhá to tak, že s pravděpodobností c_p vybereme k jedinců. Každý z těchto vybraných jedinců se bude křížit s každým dalším vybraným. Jsou to rodičové, ze kterých vzniknou potomci. Potomek obsahuje část od každého rodiče. Funguje to tak, že vezmeme dva vybrané rodiče, neboli zakódované binární řetězce. Ty mají délku m. Máme tak *m-1* možností, kde tyto řetězce můžeme rozdělit. Náhodně vybereme místo, kde řetězce rozdělíme a dva noví potomci vzniknou tak, že první bude mít první část z rodiče jedna a druhou část z rodiče dvě a druhý potomek bude mít první část z rodiče dvě a druhou část z rodiče jedna. Dobře je to vidět na následujícím obrázku 2.2.



Obrázek 2.2. Ukázka křížení. Vybereme rodiče, bod kde rodiče rozdělíme. Tak vzniknou dva noví potomci, každý s genetickou informací od rodičů.

Existuje více technik křížení. To co jsme právě viděli se nazývá jednobodové křížení. Vybrali jsme jeden bod a v tom místě jsme vyměnili vlastnosti chromozomům a z toho udělali potomky. Existuje i vícebodové. Např. u dvoubodové vybereme 2 body v chromozomu, čímž ho rozdělíme na tři části. První potomek pak bude mít první a třetí část jednoho rodiče a druhou část dostane od druhého rodiče a druhý potomek naopak. Dalších z technik křížení je, že vybereme N určitých míst v chromozomu a ty pak navzájem překřížíme. Pro představu, máme-li chromozom délky deset, vybereme náhodně pozice jedna, pět, šest a sedm a vlastnosti na těchto pozicích mezi rodiči překřížíme.

Vlastnost genetického algoritmu 2.4. Mutace.

Nyní jsme v bodě, kdy jsme vybrali nové jedince reprodukcí, některé z nich jsme zkřížili a na řadě je poslední operace a to je **mutace**. Potomci doteď nedostávali žádné nové vlastnosti, pouze je dědily. A nové vlastnosti přidá právě mutace. Mutace probíhá tak, že bere postupně každého potomka a každou jeho část chromozomu. Pokud tedy máme zakódovaný chromozom do binárního řetězce, tak každého člena tohoto řetězce zmutuji s pravděpodobností m_p na inverzní hodnotu. Z nuly uděláme jedničku a naopak. Pravděpodobnost mutace bývá o hodně nižší než pravděpodobnost křížení. Pokud nemáme kódování do binárního řetězce, tak nemůžeme vědět na jakou hodnotu máme vybraného člena změnit. Vyřeší se to tak, že místo inverze náhodně vybereme hodnotu ze všech možných hodnot z oboru hodnot pro danou instanci problému. Orientační hodnoty pro c_p jsou vysoké, 0,7–0,9 a orientační hodnoty pro m_p jsou 0,03–0,15. Mutace je zde proto, že přidává jedincům nové vlastnosti, které by třeba nikdy nemohly vzniknout křížením, přináší do genetického algoritmu různorodost a je důležitou částí tohoto algoritmu. Představme si situaci, kdy nám reprodukcí zůstane pouze jeden silný jedinec. Křížení by nám v tomto případě vůbec nepomohlo, protože by vznikal stále stejný řetězec. A i proto je zde mutace.

1	0	0	1	1	1	0	0	1
1	0	0	1	0	1	0	0	1

Obrázek 2.3. S určitou pravděpodobností vybereme části potomka, které budou mutovat.

Toto je poslední krok v genetickém algoritmu a začínáme znovu u vyhodnocení zdatnosti pro každého člena aktuální generace. Algoritmus může skončit několika způsoby. Pokud již máme jedince, který je dostatečně zdatný, nebo splňuje zadání, nebo pokud máme konečný počet generací a uchováváme si nejlepší výsledek. Po proběhnutí všech generací, pak tento výsledek vrátíme [3, 7–9].

Zde je shrnutí všech kroků genetického algoritmu.

1. Vytvoření nulté generace.
2. Vyhodnocení zdatnosti každého jedince v generaci. Pokud máme dostatečně silného jedince, algoritmus končí. Pokud jsme dosáhli maximálního počtu generací, končíme a vracíme nejlepší dosud získaného jedince.
3. Provedeme reprodukci.
4. Provedeme křížení a mutaci.
5. Vytvoříme novou generaci.
6. Vracíme se do bodu 2).

Nevýhoda genetických algoritmů bývá často v jejich rychlosti a paměťové náročnosti, protože se starají o velké množství chromozomů, výsledků problémů, které se snažíme s tímto algoritmem vyřešit.

Naopak výhoda je v tom, že lze na ně převést velké množství optimalizačních problémů a relativně jednoduše. Nejtěžší část je vymyslet reprezentaci chromozomu pro náš problém. Pak už jen lehce modifikovat operace genetického algoritmu a máme funkční řešení.

V dalších dvou podkapitolách této kapitoly následuje jednoduchý příklad 2.2.1 a následně příklad toho, jak genetický algoritmus použít pro algoritmus pro generování školního rozvrhu 2.2.2.

2.2.1 Příklad pro genetický algoritmus

Ukážeme si jednu smyčku genetického algoritmu na jednoduchém matematickém příkladu

$$a + 2b + 3c + 4d = 30$$

Obor hodnot pro proměnné *a, b, c, d* jsou celá čísla 0–30.

První krok je vytvoření nulté generace. Generace bude obsahovat šest jedinců (to si sami zvolíme) a každý jedinec bude obsahovat chromozom o čtyřech prvcích. Pro jednoduchost nebudeme kódovat chromozom do binárního řetězce a necháme ho v celých číslech. Zakódování do binárního řetězce by mohlo probíhat tak, že pro každé číslo by nám stačilo pět míst pro převod do binárního čísla, jelikož maximální hodnota je 30. Výsledný řetězec by byl pak dlouhý 20 znaků. Pět znaků pro proměnnou a čtyři proměnné. Nyní již náhodné vygenerování nulté generace tab. 2.1.

V **druhém kroku** zjistíme zdatnost každého jedince. V tomto případě jednoduše dosadíme do rovnice

$$|a + 2b + 3c + 4d - 30|$$

Jedinec	Chromozom	Zakódovaný chromozom
1	[12;05;23;08]	01100001011011101000
2	[02;21;18;03]	00010101011001000011
3	[10;04;13;14]	01010001000110101110
4	[20;01;10;06]	10100000010101000110
5	[01;04;13;19]	00001001000110110011
6	[20;05;17;01]	10100001011000100001

Tabulka 2.1. Náhodné vygenerování nulté generace

a *výsledek* sám o sobě pak již bude vypovídat jak silný je jedinec. Čím blíže k nule bude, tím je jedinec silnější. Abychom to měli procentuálně vyjádřeno a mohli jsme udělat reprodukci, pak dosadíme výsledek do fitness funkce

$$\frac{1}{1 - výsledek}$$

Jedinec	Chromozom	Výsledek	Zdatnost
1	[12;05;23;08]	$\|12 + 2 * 05 + 3 * 23 + 4 * 08 - 30\| = 93$	$\frac{1}{1-93} = 0,0106$
2	[02;21;18;03]	80	0,0123
3	[10;04;13;14]	83	0,0119
4	[20;01;10;06]	46	0,0213
5	[01;04;13;19]	94	0,0105
6	[20;05;17;01]	55	0,0179
			Celkem : 0,0845

Tabulka 2.2. Výpočet zdatnosti jedince

Jedinec	Chromozom	Zdatnost	Procentuální	Kumulativně
1	[12;05;23;08]	0,0106	$\frac{0,0106}{0,0845} = 0,1254$	0,1254
2	[02;21;18;03]	0,0123	0,1456	0,2710
3	[10;04;13;14]	0,0119	0,1408	0,4118
4	[20;01;10;06]	0,0213	0,2521	0,6639
5	[01;04;13;19]	0,0105	0,1243	0,7882
6	[20;05;17;01]	0,0179	0,2118	1

Tabulka 2.3. Výpočet kumulativní zdatnosti pro ruletovou selekci viz. obr. 2.1

Nyní tedy máme vše připraveno pro **třetí krok** a tím je reprodukce. Využijeme již zmíněné ruletové selekce, která je znázorněna dříve v této práci na obr. 2.1. V tomto konkrétním případě roztočíme ruletu šestkrát. Jedinec číslo jedna má šanci že bude vybrán 12%, jedinec dvě má šanci 14%, atd...

Náhodně tedy vylosujeme šest reálných čísel v rozmezí 0–1 a podle toho jaké číslo padne, tak takový bude vybrán jedinec. Pokud padne číslo 0,2 bude vybrán jedinec číslo dva, dle tabulky 2.3, sloupce "Kumulativně", protože $0,1254 < 0,2 < 0,2710$.

Jedinec	Vylosované číslo	Vybraný jedinec	Chromozom
1	0,201	2	[02;21;18;03]
2	0,284	3	[10;04;13;14]
3	0,099	1	[12;05;23;08]
4	0,822	6	[20;01;10;06]
5	0,398	3	[10;04;13;14]
6	0,501	4	[20;05;17;01]

Tabulka 2.4. Vybrání jedinců k reprodukci

Analogicky když padne číslo 0,7 bude vybrán jedinec číslo pět, protože $0,6639 < 0,7 < 0,7882$.

Můžeme si všimnout, že při reprodukci nám zanikl jedinec číslo pět, jakožto nejslabší a naopak jedinec číslo tři je tam nyní dvakrát. Na řadě je nyní **krok čtvrtý**. Křížení a mutace. U křížení vybereme jedince s pravděpodobností c_p. Následně každého zkřížíme s každým vybraným. Vybereme dělící bod a vytvoříme dva nové potomky viz. obr. 2.2. Dejme tomu, že jsme zvolili $c_p = 40\%$ a algoritmus nám vybral pro křížení pouze dva jedince, jedince číslo jedna a pět a dělící bod pro ně je dva, tedy za druhým parametrem. Hned popíšeme i mutaci. U každé vlastnosti s pravděpodobností m_p např. 10% budeme mutovat. Tzn. pokud bude vlastnost vybrána, tak jí vybereme náhodnou hodnotu z oboru hodnot od 0-30. Vše je vidět v následující tabulce 2.5.

Jedinec	Chromozom	Křížení	Mutace
1	[**02;21**;18;03]	[02;21;13;14]	[02;21;13;14]
2	[10;04;13;14]		[10;04;13;14]
3	[12;05;23;08]		[12;05;**13**;08]
4	[20;01;10;06]		[20;01;10;06]
5	[10;04;**13;14**]	[10;04;18;03]	[10;04;18;03]
6	[20;05;17;01]		[**15**;05;17;01]

Tabulka 2.5. Křížení a mutace jedinců

Překřížili se tedy jedinci jedna a pět, to je vidět ve sloupci 2 tabulky 2.5 a v té samé tabulce, ve sloupci 3 je vidět, že u jedince číslo tři a šest nám zmutovaly vlastnosti chromozomu. Jedinec tři měl jako třetí vlastnost chromozomu číslo 23 a nyní má 13. U jedince 6 byla vlastnost první 20 a nyní je 15.

Právě jsme tedy udělali celou novou generaci a jednu smyčku algoritmu. Nyní se vrátíme k vyhodnocení zdatnosti každého jedince, zjistíme jestli nám nějaký nedává již správný výsledek. Pokud ne, opět uděláme reprodukci, křížení a mutaci a tak stále dokola, dokud nám nějaká generace nevrátí správný výsledek [10].

2.2.2 Genetický algoritmus pro generování školního rozvrhu

V předchozí kapitole jsme si představili konkrétní příklad toho, jak problém převést na genetický algoritmus. Pro všechny problémy, které chceme řešit pomocí genetického algoritmu je nejtěžší vymyslet to, jak bude vypadat chromozom, neboli reprezentace výsledku. Je to proto, abychom na něm pak mohli provádět všechny zmíněné operace genetického algoritmu: reprodukci, křížení a mutaci. Ukážeme si, jak by takový chromozom pro rozvrh mohl vypadat.

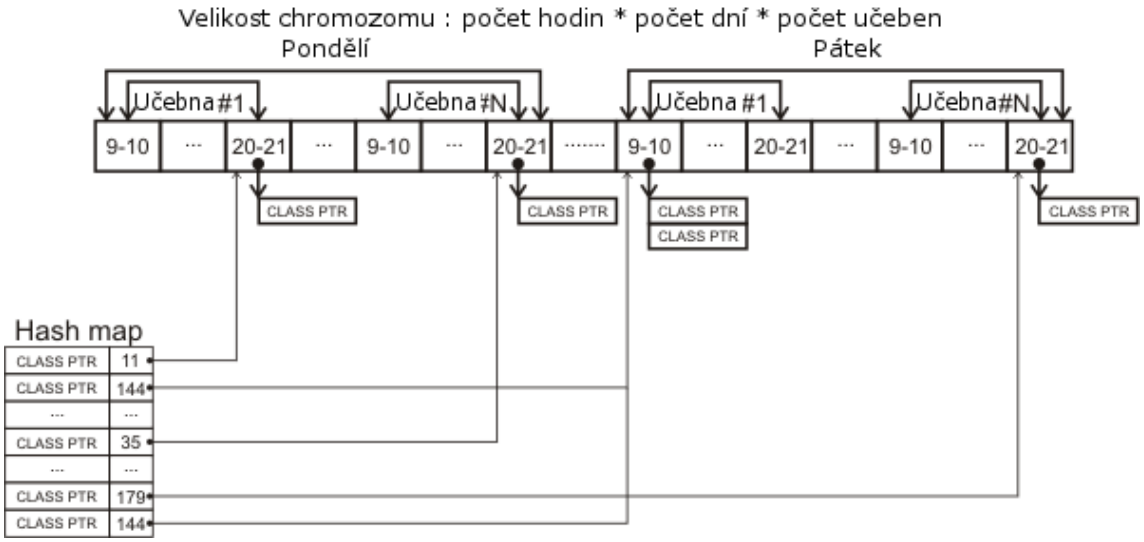
Nejdříve si představíme objekty a jejich nejdůležitější parametry, které v algoritmu vystupují.

- Učitel / Profesor:
  - Seznam předmětů, které učí.
- Třída:
  - Seznam předmětů, které mají být učeny
- Učebna:
  - Počet míst.
  - Typ učebny: počítačová, laboratoř, tělocvična... Záleží na konkrétní implementaci.
- Předmět:
  - Název předmětu.
  - V jakém typu učebny se má učit.
- Předmět / třída / profesor:
  - Odkaz na předmět, který má být učen.
  - Odkaz na třídu ve které je předmět učen.
  - Odkaz na profesora, který předmět učí.

Pro reprezentaci chromozomu, tedy reprezentaci výsledného rozvrhu potřebujeme pro určitý čas a určitou učebnu zařadit objekt předmět/třída/profesor. Tedy vezmeme počet hodin za den, počet dní a počet učeben a to nám dá dohromady velikost chromozomu a to takto:

$$počet\_hodin\_denně * počet\_dní * počet\_učeben = velikost\_chromozomu$$

Každé pole v chromozomu může odkazovat na více *předmět/třída/profesor* objektů. Ve výsledku samozřejmě nemůže být ve stejný čas a ve stejné učebně více tříd, ale v kapitole 2.2 jsme si vysvětlili, že v průběhu genetického algoritmu může algoritmus obsahovat i špatné výsledky. Lépe bude vše vidět na obrázku 2.4.



Obrázek 2.4. Na obrázku vidíme reprezentaci chromozomu pro 12 hodin denně, pět dní v týdnu a N učeben. Každé pole v chromozomu patří konkrétnímu času a konkrétní učebně a drží si informaci o tom, jaké předměty jsou zrovna na tomto poli. Obrázek převzat z [11].

Druhý problém při převádění problému na genetický algoritmus je fitness funkce. Tu jsme popsali na straně 4. Pro problém generování rozvrhu bude zdatnost chromozomu odvíjena hlavně od toho, kolik splňuje silných podmínek, viz. kapitola 2.1.1. Jakmile budou splněny silné podmínky, které je potřeba splnit všechny, jinak by rozvrh nebyl splnitelný, tak se kvalita rozvrhu bude odvíjet od toho, kolik porušuje slabých podmínek, viz. kapitola 2.1.2. Ty nemusí splnit všechny, ale čím méně jich algoritmus porušuje, tím kvalitnější je rozvrh.

Pro generování rozvrhu si tedy budeme držet dva výsledky, toho jak je rozvrh kvalitní. První výsledek bude znamenat jak moc rozvrh porušuje silné podmínky a druhý slabé. Funkce, která zjišťuje porušení silných podmínek, tedy může vypadat takto.

- Každá třída může získat 0–K bodů, kde K znamená počet silných pravidel.
- Za každé splnění pravidla, přidáme bod.
- Za nesplnění pravidla neděláme nic.
- Celkové skóre je součet všech bodů, všech tříd.
- Maximální skóre je $K * počet\_tříd$.
- Funkce vypadá pak takto: $celkové\_skóre / maximální\_skóre$.
- Výsledek bude reálné číslo mezi 0 a 1. Pokud bude rozvrh v pořádku z hlediska silných podmínek, skóre bude 1. Čím více podmínek bude porušeno, tím se bude hodnota blížit k nule a tím pádem to bude slabší rozvrh, který bude mít menší šanci se dostat do další generace.

Analogicky bude vypadat fitness funkce pro slabé podmínky. Vše je nyní připraveno k tomu, abychom využili genetický algoritmus pro generování rozvrhu. Máme problém převeden do chromozomu a máme vymyšlenou fitness funkci. Je důležité, aby výsledný rozvrh neporušoval žádnou ze silných podmínek a zároveň, aby slabých porušoval co nejméně. Algoritmus tak musí běžet minimálně do doby, kdy se v nějaké generaci vyskytne rozvrh, který žádnou silnou podmínku neporušuje. V tomto bodě můžeme algoritmus zastavit a vrátit výsledek. Další možností je, že algoritmus necháme běžet déle, dalších M generací po nalezení prvního splnitelného rozvrhu a uchováme si všechny rozvrhy s každé generace, které žádnou silnou podmínku neporušují.

Po doběhnutí těchto M generací akorát vybereme rozvrh, který porušuje nejméně slabých podmínek. Nebo můžeme klidně vrátit rozvrhů více a dát uživateli vybrat. To je určitě výhoda oproti jiným algoritmům, kterým stojí hodně času, aby vygenerovali jeden splnitelný rozvrh. Tento algoritmus jich může vygenerovat více během svého běhu.

To nejtěžší tedy máme za sebou a už akorát provádíme v každém cyklu operace křížení a mutace. Tyto operace už není těžké převést na problém generování rozvrhu.

Podíváme se ještě jednou na předchozí obrázek 2.4. Ten nám reprezentuje jeden chromozom. V algoritmu jich máme více, podle toho kolik jsme si jich na začátku vygenerovali. Jak funguje operace křížení je popsáno na straně 4. Zde to uděláme tak, že vylosujeme rodiče a například pomocí dvoubodového křížení vybereme část, kterou si rodiče vymění. Po operaci křížení budou potomci mít u některých učeben a časů jiné předměty, než měli původně. Konkrétně vylosujeme dva body. První bod je učebna sedm, čas 12:00 a druhý bod je učebna 15 čas 17:00. Jeden potomek bude mít stejné předměty v učebnách 1–6 celý den a v učebně 7 do 12:00 jako první z rodičů. Od 7. učebny, od 12:00 do učebny číslo 15, do 17. hodiny bude mít předměty druhého rodiče a dál bude mít zas stejné předměty jako první z rodičů. Druhý potomek to bude mít naopak.

Nakonec operace mutace. Mutace vybere v chromozomu náhodně předměty, které by se této operace měly zúčastnit. Tyto předměty pak prostě přemístí do náhodné učebny a náhodného času.

Takhle jsme tedy převedli genetický algoritmus, tak aby řešil generování školního rozvrhu. V předchozí kapitole jsme si zas ukázali, jak řešit jednoduchý matematický problém. Jak jsem již zmínil, genetický algoritmus se dá modifikovat celkem snadno na dost problémů. Stačí, abychom výsledek algoritmu převedli do chromozomu se kterým genetický algoritmus pracuje a abychom vymysleli odpovídající fitness funkci, která nám bude vypovídat o kvalitě aktuálního výsledku [11].

2.3 Rekurzivní algoritmus

Určitě najdeme víc algoritmů, které řeší rozvržení školního rozvrhu. Shrnul bych je ale do dvou hlavních skupin. Genetický algoritmus, který jsme si představili v minulé kapitole a rekurzivní algoritmus, kterému se věnuje tato kapitola. Tento algoritmus má na začátku prázdný rozvrh a každým jeho cyklem postupně přidává aktivitu za aktivitou do rozvrhu. Až nakonec má všechny aktivity rozmístěné.

Aktivitou v této práci rozumíme konkrétní předmět, který požaduje mít alokované všechny tyto zdroje: třídu, učitele, učebnu a čas.

Při vkládání nesmí porušit žádnou ze silných podmínek a snažit se porušit co nejméně slabých podmínek. Jeden typ rekurzivní algoritmu jsem použil při programování generátoru pro školní rozvrh a bude daleko více detailněji popsán v pozdější kapitole 4. Proto se nyní podíváme, jak rekurzivní algoritmus funguje obecně pro rozvrhovací problém a jaké jsou jeho největší problémy.

2.3.1 Rekurzivní algoritmus pro generování školního rozvrhu

Na začátku algoritmu máme prázdný rozvrh a N aktivit, které musí algoritmus do rozvrhu zařadit. Vždy vybereme jednu aktivitu a podle toho, jaké zdroje tato aktivita potřebuje, tak vybereme místo v rozvrhu, kam tuto aktivitu zařadíme. V tento okamžik máme jednu aktivitu umístěnou v rozvrhu a *N-1* aktivit k umístění. Tzn. že v každém volání máme částečné řešení rozvrhu, který je splnitelný, protože umístění nesmí porušovat žádnou ze silných podmínek. Prázdný rozvrh také považujeme za částečně správné řešení.

Problém nastává v okamžiku, kdy pro vybranou aktivitu nemůžeme vybrat žádné místo. To nastane, když všechna místa, kam bychom mohli aktivitu umístit, tak abychom ji mohli přiřadit všechny potřebné zdroje, jsou již obsazená. V tomto případě vzniká konflikt. Při konfliktu jednoduše vybereme nejlepší pozici pro aktivitu, kterou máme právě zařadit a odebereme z rozvrhu všechny ostatní aktivity, které způsobují konflikt. Tyto odebrané aktivity zařadíme zpět mezi aktivity, které ještě nejsou umístěné a pokračujeme dále v algoritmu.

V tomto algoritmu tedy musíme vyřešit dva problémy. Jakou vybrat aktivitu z těch, které ještě nejsou zařazené a kam ji umístit. Takto by mohla vypadat kostra algoritmu.

```
function vygenerovaniRozvrhu(neumisteneAktivitity, rozvrh) {
   dokuď máme neumísté aktivity:
      aktivita = vyberAktivituKRozvrhnuti(neumisteneAktivitity);
      neumisteneAktivitity -= aktivita;
      místo = vyberMistoKamAktivituUmistit(rozvrh, aktivita);
```

```
        // umísti a přidej konfliktní aktivity zpět mezi neumístěné
        neumisteneAktivity += umisti(aktivita, misto, rozvrh);
    }
```

V dalších dvou podkapitolách se zaměříme právě na tato problémová místa.

2.3.2 Výběr aktivity

Máme množinu nerozvržených aktivit. Cílem je vybrat z této množiny aktivitu, kterou budeme umisťovat jako další. Brát jednu náhodnou by nebylo moudré, protože zde mohou být aktivity, které jdou umístit lehce a které jdou umístit hůře. Pokud bychom tedy náhodně brali aktivity, které jdou umístit lehce jako první, tak je možné že by zabraly místo těm aktivitám, které jdou umístit hůře a vznikalo by daleko více konfliktů než kdybychom vybírali aktivity nějakým chytrým způsobem.

Logicky tedy budeme brát jako první aktivity, které jdou umístit hůře. Na začátku rekurzivního algoritmu si tedy můžeme seřadit aktivity od té nejtěžší po nejlehčeji umístitelnou. K tomu postačí naprogramování nějaké váhové funkce, která přiřadí aktivitě váhu. Čím těžší aktivita, tím nižší váha například (samozřejmě se dá upravit dle potřeby programátora). Když tedy budeme mít takto seřazené aktivity podle váhy, tak s výběrem není problém. V okamžiku, kdy nastane konflikt a odebereme z rozvrhu již rozvržené aktivity, tak je opět zařadíme do nezařazených podle váhy, kterou dostaly na začátku.

Toto se nazývá *statické* ohodnocení. Na začátku přiřadíme aktivitám váhu a v průběhu algoritmu s ní již nadále nehýbeme. Výhoda je v tom, že statické ohodnocení nezabírá žádný výpočetní výkon v průběhu algoritmu. Do statického ohodnocení můžeme započítat:

- Počet míst, kam aktivitu můžeme na začátku umístit. Čím více možností tím lehčeji aktivitu umístíme.
- Zda-li je hodina dvojitá. Je-li dvojitá, má méně možností na umístění, tím pádem je těžší ji umístit.
- Aktivity ve skupině, neboli když je více předmětů v jedné třídě, v tom samém čase (půlená hodina, např. německá a anglická část třídy). Aktivity ve skupině mají méně možností k umístění, jdou hůře umístit.
- Je možná kombinace dvou naposledy zmíněných bodů. Můžeme mít dvojitou aktivitu ve skupině. Tyto hodiny mají zas o něco méně možností k umístění.

V předchozím seznamu bylo prakticky uvedeno, že váhová funkce znamená to, kolik mají jednotlivé aktivity a jejich různé modifikace (dvojité, ve skupině) počet možností k umístění.

To tedy platí o statickém ohodnocení. Toto ohodnocení je plně dostačující, ale musíme se více věnovat tomu, aby nám algoritmus nezapadl do cyklu.

Dynamické ohodnocení získáme z částečně hotového rozvrhu. Je to ohodnocení, která více odpovídá aktuálnímu řešení a vybere lépe aktivitu k umístění než statické ohodnocení, ale je velice časově náročné. Po každém umístění nebo odebrání aktivit z rozvrhu se váha u aktivit změní a musíme ji přepočítávat. Proto dynamické ohodnocení pozměníme. Nebudeme po každém cyklu algoritmu přepočítávat všechny váhy všech aktivit, ale náhodně vybereme podmnožinu z neumístěných aktivit (např. 15-20% všech aktivit) a tyto aktivity ohodnotíme a vybereme tu nejhůře umístitelnou z této množiny.

Tím se částečně zbavíme časové náročnosti dynamické ohodnocení. Pro představu je zde seznam toho, co by se do dynamické váhové funkce dalo započítat.

- Kolikrát již byla aktivita odebrána z rozvrhu. Čím vyšší je hodnota tohoto kritéria, tím hůře jde aktivita umístit.
- Na kolik míst můžeme aktivitu aktuálně umístit. Místo, kam by šla umístit, ale kde je již jiná aktivita nezapočítáme. Čim méně míst, tím hůře lze aktivita umístit.
- Na kolik míst můžeme aktivitu umístit, tak aby neporušovala žádnou ze silných podmínek. Opět platí, že čím méně je takových míst, tím hůře lze aktivitu umístit.

Záměrně zde neuvádím konkrétní váhovou funkci, protože závisí od konkrétní implementace a od toho jaká kritéria chceme započítat, jestli používáme dynamické či statické ohodnocení. V druhé části této diplomové práce se navíc věnuji implementaci konkrétního algoritmu a jak jsem zde vyřešil ohodnocení aktivit pomocí váhové funkce je popsáno v kapitole 4.5.2.

2.3.3 Výběr umístění aktivity

Při výběru umístění aktivit víme, na jaká místa v rozvrhu může aktivita přijít. A to dle zdrojů, které aktivita vyžaduje (učitel, třída, učebna). Z těchto všech míst odebereme místa, kde by aktivita porušovala silné podmínky. Vznikne nám tedy množina míst, kam můžeme aktivitu umístit. Z této množiny vybereme nejlepší možné umístění metodou známou jako *best-fit*. To vybereme tak, že projdeme celou množinu a zkontrolujeme kolik by umístění aktivity na dané místo porušilo slabých podmínek. Pokud by bylo více nejlepších míst, kam by aktivita mohla být umístěna, tak vybereme náhodně jedno z nich.

Toto je také časově náročná operace, zvláště pokud mají aktivity velké množství možností k umístění. Proto zde při výběru nemusíme použít nejlepší možné místo, ale místo, které si určíme jako dostačující. Například porušuje-li jen 10% slabých podmínek nebo jiné pravidlo, které si určí programátor při konkrétní implementaci.

Pokud by byla množina prázdná, tak se dostáváme do konfliktu. Při konfliktu můžeme pokračovat ve výběru místa stejně jako v situaci, kdy žádný konflikt nemáme. Pouze s tou vyjímkou, že před umístěním odstraníme z rozvrhu aktivity, které způsobují konflikt a až teprve následně umístíme aktivitu [4, 12].

2.4 Interaktivní tvorba rozvrhu

Interaktivní tvorba rozvrhu přímo není určitý algoritmus pro generování školního rozvrhu jako byl genetický algoritmus nebo rekurzivní algoritmus. Je to způsob, jak do generování zavést uživatelskou proměnnou. Algoritmus může podle naprogramovaných podmínek vytvořit skvělý rozvrh. Ale vygenerovaný rozvrh nemusí vyhovovat všem i přes veškeré nastavení, protože např. někteří učitelé mohou mít speciální podmínky, které algoritmus nepodporuje.

Proto algoritmus může podporovat interaktivní tvorbu rozvrhu, kdy po každém cyklu algoritmu může uživatel algoritmus pozastavit a udělat nějaké změny v rozvrhu. Odstranit aktivitu a označit, že zde tuto aktivitu nechce umístit, přesunout aktivitu na jiné místo, nebo označit aktivitu jako fixní, pevnou (zamezit algoritmu v jejím přesouvání).

Tímto právě vložíme do algoritmu uživatelskou proměnnou a výsledek bude vypadat více tak, jak si ho uživatel představuje. Algoritmus v tomto případě ale musí detekovat všechny věci, které uživatel udělal a uvést svá data přesně do stavu, do kterého

ho uživatel uvedl. Tzn. odstranit aktivity z naplánovaných a dát je do množiny nenaplánovaných a naopak, označit aktivity jako fixní, nebo neodstranitelné z rozvrhu [4, 12].

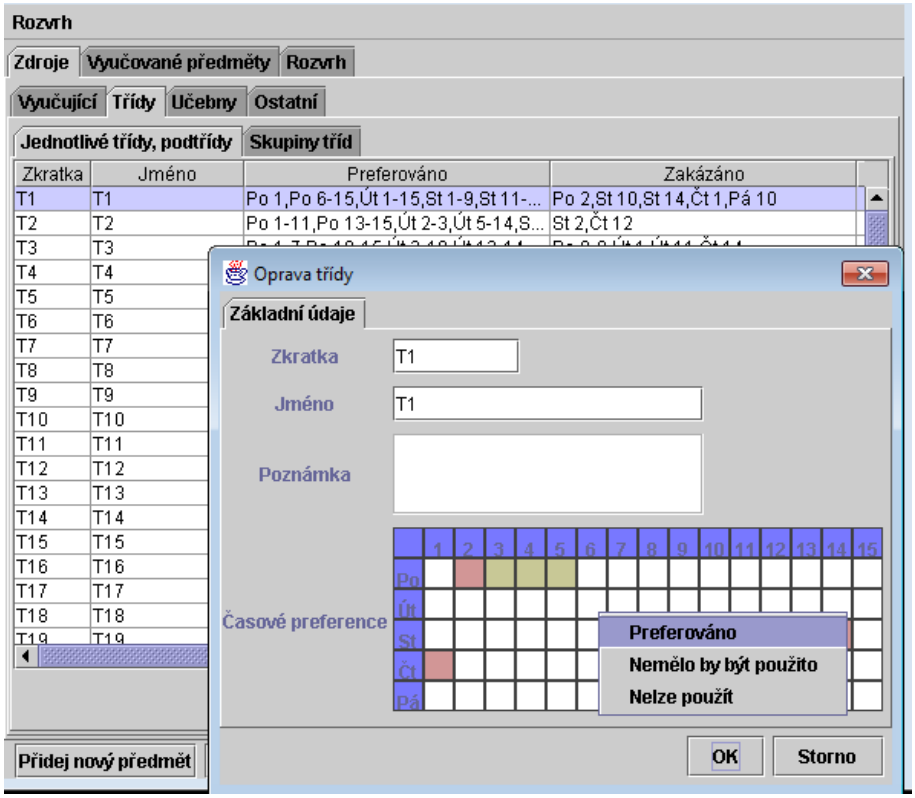
Kapitola 3
Existující programy pro generování školního rozvrhu

V této kapitole si představíme některá existující řešení a jejich klady a zápory. Tuto kapitolu sem zařazuji z toho důvodu, že navrhnutí aplikace je hlavní část této diplomové práce a budeme se o ní bavit hned v další kapitole číslo 4. Proto je zde shrnutí existujících řešení, abych do mé aplikace zahrnul to, co je správné a samozřejmě mohl přijít s něčím trochu jiným než nabízí již existující řešení.

3.1 Interaktivní tvorba rozvrhu

Diplomová práce Tomáše Müllera [4]. Funguje na bázi rekurzivního algoritmu s možností úpravy při generování uživatelem, viz. kapitola 2.4.

Program se dělí na tři hlavní části. První část obsahuje zdroje: vyučující, třídy, učebny a ostatní (např. pomůcky pro aktivity). V druhé části jsou vyučované předměty a ve třetí je generování, zobrazení rozvrhu a nastavení.



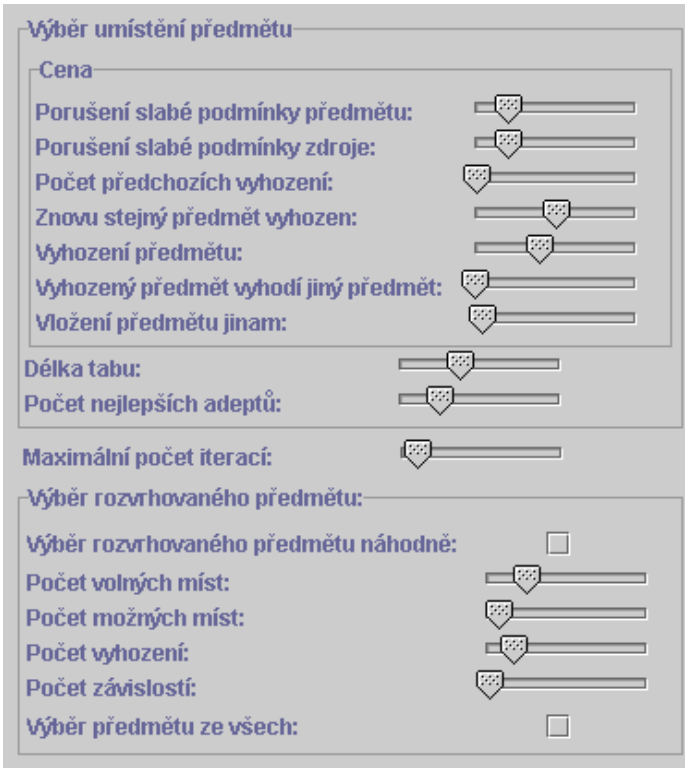
Obrázek 3.1. Ukázka program interaktivní tvorby rozvrhu a toho, jak můžeme dobře nastavovat časové preference pro jednotlivé zdroje.

První část, neboli zdroje, má stejný interface pro vložení a úpravu zdrojů. U každého zdroje je možné zvolit v jakých časech je možné zdroj použít, kdy by použit být neměl a kdy nemůže být použit, viz obr. 3.1.

To znamená, že u každého učitele, třídy a učebny můžeme celkem detailně nastavit časové možnosti jaké preferujeme.

U druhé části vytvoříme předměty a každému z nich přiřadíme všechny zdroje, které potřebuje. Tzn., že každému předmětu musíme přidat třídu, učebnu, učitele, popřípadě nějaké ostatní zdroje. U každého předmětu opět můžeme určit jaké časy preferujeme a nepreferujeme. Navíc můžeme přidat časové závislosti mezi předměty a říci, že jeden předmět může být kdykoliv před, těsně před, kdykoliv za, těsně za nebo souběžný s předmětem druhým.

V třetí části již máme vše připraveno a můžeme vygenerovat a zobrazit rozvrh, popřípadě udělat nějaká nastavení. Jak bylo zmíněno na začátku kapitoly, generátor můžeme kdykoliv zastavit. Navíc máme další možnosti a to konkrétně zafixovat již rozvržený předmět (generátor ho již nebude moci odebrat), nebo umístěný předmět přemístit na jiné místo. Zobrazit rozvrh můžeme z pohledu jakéhokoliv zdroje. Co se týká poslední části, nastavení, tak je tu pár posuvníků s popisem, bohužel trochu matoucí pro uživatele, viz obr. 3.2.



Obrázek 3.2. Nic neříkající nastavení v programu pro interaktivní tvorbu rozvrhu.

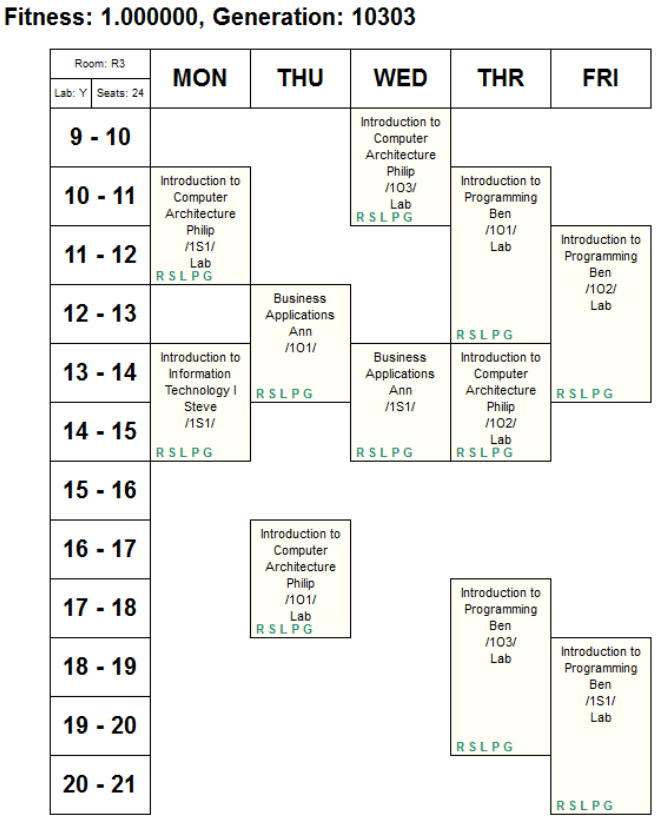
- Klady:
  - Možnost určit přesnější časové preference u každého zdroje.
  - Přidání ostatních zdrojů, např. pomůcky do předmětů.
  - Zastavení rozvrhu při generování.
  - Přesouvání předmětů v rozvrhu při i po generaci a možnost zafixování předmětů (algoritmus nemůže s daným předmětem již hýbat).

- Časové závislosti mezi předměty.
- Zdarma.

- Zápory:
  - U každého předmětu musíme zadat i množinu učeben, které chceme použít. Nedokáže přiřadit učebnu sám.
  - Pokud má třída předmět pětkrát týdně, musíme zadat pětkrát ten samý předmět a pokaždé mu přiřadit všechny zdroje.
  - Nastavení u kterého není jasné, jak nám výsledný rozvrh ovlivní.
  - Žádná možnost exportu.

3.2 GAScheduleApp

Práce Mladena Jankovice [11]. Jak již z názvu vyplývá, algoritmus funguje na bázi genetického algoritmu. Je to velice jednoduchý program, kde je pouze hlavní okno, které nám zobrazí průběh generace a na konci celý rozvrh dle učeben, viz obr. 3.3.



Obrázek 3.3. Program nám nabízí jen jednoduché okno, ve kterém můžeme vidět výsledný rozvrh pouze z pohledu učeben.

Není zde žádné složité nastavení, vše co potřebujeme pro výsledek je připravit si konfigurační soubor, který má tuto strukturu:

```
#prof
  id = 1
  name = Victor
```

```
#end
... další učitelé

#course
  id = 1
  name = Český jazyk
#end
... další předměty

#room
  name = 101
  lab = true
  size = 24
#end
.. další učebny

#group
  id = 1
  name = 1.A
  size = 19
#end
... další třídy

#class
  professor = 1
  course = 1
  duration = 2
  group = 1
  group = 2
#end
... výpis všech aktivit, které se mají naplánovat. Zde to znamená,
že učitel Victor má český jazyk, který trvá dvě hodiny vkuse a tato
aktivita je ve třídě 1.A a 1.B (group 2 zde není vidět, je to jen
ukázka, že mohu nastavit aktivitu pro více tříd).
```

Následně stačí jen nahrát konfigurační soubor, spustit generování a počkat na výsledek.

- Klady:
  - Žádné složité nastavování.
  - Jednoduché ovládání.
  - Rozlišuje dva typy učeben a přiřadí učebnu sám k hodině.
- Zápory:
  - Pokud má třída předmět pětkrát týdně, musíme zadat pětkrát ten samý předmět a pokaždé mu přiřadit všechny zdroje.
  - Žádné nastavení.
  - Žádná možnost exportu.
  - Zobrazení jen z pohledu učebny.
  - Psaní konfiguračního souboru, grafické rozhraní by bylo rozhodně lepší.
  - Žádné časové preference a ovlivnění rozvrhu učitelem.

3.3 Tvůrce rozvrhů

Další program na generování rozvrhů, který je založen na genetickém algoritmu [13]. Jako zdroje používá seznam vyučujících (bez časových možností, pouze jejich jména) a seznam tříd a jejich týdenní skladby (název hodiny, kdo ji učí, jak je dlouhá a kolikrát týdně je). A to je vše, žádné učebny zde nejsou.

Co se týká nastavení, tak můžeme třídy rozdělit do skupin, např. po ročnících a u každé nastavit nějaké slabé podmínky, jako maximální možné opakování předmětu denně, maximální počet mezer v rozvrhu atd. Takže je zde rozumné nastavení, kterému každý porozumí. Navíc je zde i nastavení pro samotný genetický algoritmus, který asi běžný uživatel nevyužije. V tomto nastavení můžeme nastavit maximální počet generací, počet jedinců, pravděpodobnost křížení a mutace a prakticky všechny věci, které lze nastavit u genetického algoritmu.

Rozvrh pak můžeme zobrazit z pohledu tříd a učitelů. Navíc ho můžeme vyexportovat do HTML souboru. Program je zdarma a nabízí kvalitní řešení. Bohužel nepracuje s učebnami.

- Klady:
  - Jednoduché vkládání dat.
  - Jednoduché ovládání.
  - Rozumné nastavení pro uživatele, kterým může ovlivnit výsledný rozvrh.
  - Export do HTML.
  - Možnost pevně svázat předmět a čas.
  - Zdarma
- Zápory:
  - Nepracuje s učebnami.
  - Není zde žádná možnost, že by byly dva předměty pro jednu třídu a v jeden čas (např. půlená hodina na angličtinu a němčinu).
  - Žádné nastavení časových preferencí.

3.4 ascTimeTables

Tento program od Applied Software Consultants [14] je placený, ale to je snad jediný zápor na něm. Pracuje se všemi zdroji: předměty, třídy, učebny, učitelé. U každého zdroje je možné zvolit časové preference. Umožňuje rozdělit třídy na skupiny a přidat předměty pouze jednotlivým skupinám. U každého zdroje jsou srozumitelná nastavení pro běžného uživatele. U vygenerovaného rozvrhu můžeme přesouvat předměty a spojit je s konkrétním časem. Dokáže exportovat do několika různých formátů a zobrazit rozvrh z pohledu každého zdroje, který je zadán. Tento program opravdu umí vše na co si vzpomenete při generování.

- Klady:
  - Jednoduché vkládání dat.
  - Intuitivní ovládání.
  - Export do více formátů.
  - Časové preference všech zdrojů.
  - Sudé, liché týdny.
  - Možnost manuálně upravit výsledný rozvrh.

Obrázek 3.4. Ukázka vygenerovaného rozvrhu od ascTimeTables.

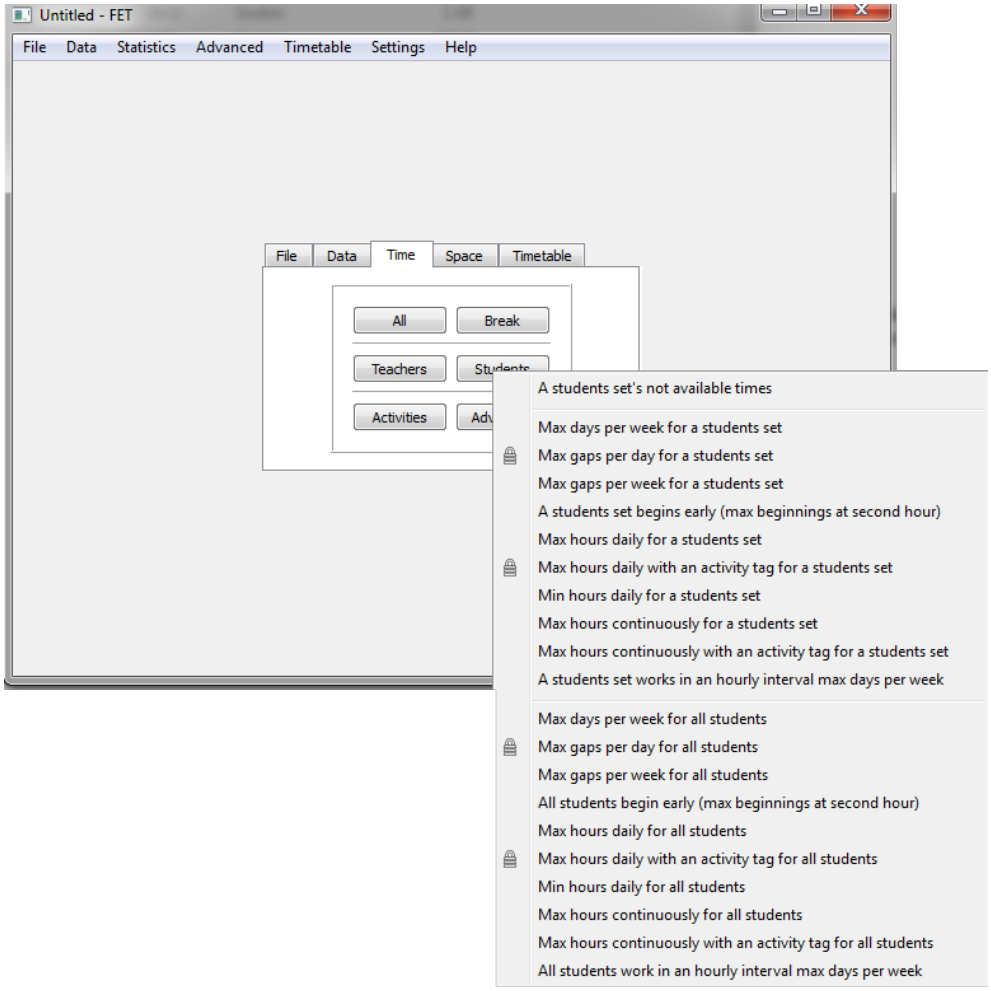
- Mnoho dalšího.

- Zápory:
  - Není zdarma. Na druhou stranu, je to opravdu kvalitní program, kterému se nevyrovná žádná alternativa, která je zdarma. Navíc zakoupením se získá licence pro všechny počítače ve škole. Cena je od 4000 Kč do 23900 Kč (od nejlevnější varianty, která by pravděpodobně většině školám stačila až po nejdražší variantu).

3.5 Free Timetabling Software

Free Timetabling Software, neboli FET [15]. Generátor rozvrhu fungující na principu rekurzivního algoritmu. V předchozí kapitole jsme se dívali na ascTimetables. Placený program, který umí prakticky vše. Toto je varianta zdarma, která umí také prakticky vše, akorát není tak intuitivní a není oblečená do tak hezkého kabátku.

Nebudu zde psát klady a zápory. Opravdu umí skoro vše co ascTimetables. Možná má až moc položek v nastavení (viz. obr. 3.5), ale za to jsou dobře popsaná a pomohou tak vygenerovat rozvrh, jaký si uživatel opravdu představuje. Najdou se i další věci. Např. neumí hýbat s výsledným rozvrhem. Ale je to opravdu dobrý program pro generování školního rozvrhu.



Obrázek 3.5. Jaké všechny časové omezující podmínky nám FET program nabízí pro třídu.

Kapitola 4
Návrh a implementace aplikace pro generování školního rozvrhu

Návrh a implementace aplikace pro generování rozvrhu je hlavní část této diplomové práce. Aplikaci jsem navrhnul na základě analýzy existujících řešení a zároveň na základě konzultací na Arcibiskupském gymnáziu v Praze.

Z toho vzniklo hlavní motto aplikace. Aplikace musí být co nejjednodušší a přitom umožnit generovat rozvrh, dle toho jak si uživatel představuje. Samozřejmě, když se odebere nastavení, tak větší váhu na rozhodování dostane samotný algoritmus a vytratí se z řešení uživatelská proměnná. Vzhledem k tomu, že rozvrh je tvořen pro lidi, tak nějaké nastavení zde být musí. Cílem bylo najít rovnováhu mezi tím, co všechno bude algoritmus dělat sám a co umožní uživateli nastavit. Myslím, že tuto rovnováhu se mi podařilo najít. Jak aplikace vypadá, co umožňuje a jaký algoritmus používá je popsáno v následujících kapitolách.

4.1 Analýza

Aplikace má být řešena pro více škol. Vzhledem k tomu, že aplikace má být co nejjednodušší, rozhodl jsem, že to bude webová aplikace a zvolil jsem klient-server řešení, kdy škole stačí pouze webový prohlížeč a přihlašovací údaje.

Jako programovací jazyk jsem zvolil PHP s pomocí frameworku Nette a databázi MySQL. V aplikaci je potřeba zadat spoustu dat pro generování od uživatele. Nette framework výrazně usnadňuje vytváření formulářů a práci s nimi, navíc implementace Nette frameworku používá Model-View-Controller (MVC) softwarovou architekturu. MVC[1]) rozděluje aplikaci na 3 vrstvy, čímž zpřehledňuje a usnadňuje vývoj aplikace.

Pro generování samotného rozvrhu jsem zvolil rekurzivní algoritmus, konkrétně jsem se inspiroval Free Timetabling Softwarem, viz. kap. 3.5, kde autor používá algoritmus, který nazval „recursive swapping“. Rekurzivní algoritmus dává větší kontrolu nad generováním rozvrhu než genetický algoritmus a bývá rychlejší. Navíc o genetických algoritmech existuje již spousta materiálů a prací. Proto jsem se rozhodl implementovat a vylepšit algoritmus „recursive swapping“. Vzhledem k výběru tohoto algoritmu, jsem se v průběhu implementace domluvil s vedoucím práce, že algoritmus nebude podporovat drag and drop metodu (přesouvání předmětu po vygenerování). Více o algoritmu bude v kap. 4.5.

[1]) MVC, neboli Model-View-Controller. Model je datová vrstva. Má přístup k databázi a tvoří funkční základ aplikací. Neví o existenci ostatních dvou vrstev. Ven nabízí pevně dané rozhraní, metody, které může volat vrstva controller. View je zobrazovací vrstva. Zobrazuje data z modelu. A nakonec controller. Controller zpracovává požadavky uživatele, dle požadavků zavolá metody modelu a vybere view, které se má uživateli zobrazit.

4.2 Zdroje

Pro výsledný rozvrh je potřeba všem aktivitám přiřadit zdroje (aktivita je v této práci považována za objekt, u kterého je řečeno jaké všechny zdroje má k dispozici). Algoritmus pro generování školního rozvrhu používá tyto zdroje: učebny, předměty, učitele a třídu. Pro generování mám tedy množinu aktivit, kterým algoritmus přiřadí časové okno, vzhledem ke zdrojům, které má přiřazené, a učebny, ve kterých mají být učeny. Pro lepší představu uvedu, jak taková aktivita může vypadat.

- Předmět: čeština
- Třída: 1.A
- Učitel: Jan Nový
- Délka: 1 hodina
- Hodin týdně: 5

Toto je aktivita (prakticky je to pět stejných aktivit), které algoritmus přiřadí časové okno, vzhledem k časovým možnostem zdrojů (např. učitel může být ve škole jen určité hodiny) a učebnu. Samozřejmě tak, aby neporušil žádné silné podmínky. Více detailů popisují následující podkapitoly.

4.2.1 Učebny

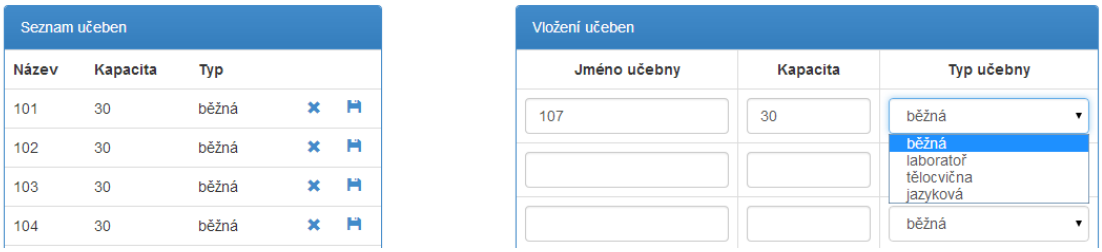
Tento zdroj není jako jediný implicitně přiřazený k aktivitě. Tzn. že algoritmus sám vybere, jakou učebnu aktivita použije. Jelikož může být požadavek na to, že potřebuji určitou učebnu pro určitý předmět, tak uživateli umožňuji nastavit typ učebny. Tělocvik musí být samozřejmě v tělocvičně a ne v obyčejné učebně. Stejně tak v laboratořích mohu mít počítače a pro výuku programování počítače potřebuji. Uživatel má tedy při vkládání možnost označit učebnu nějakým typem a u aktivity říci, že požaduje, aby byla vyučovaná v určitém typu.

Celkově u učeben mohu nastavit následující věci: název, jakou má učebna kapacitu a typ učebny. Je možné vybrat ze čtyř různých typů učebny: běžná, laboratoř, tělocvična a jazyková učebna.

A to je vše. Jednoduché, přehledné a pochopitelné. Pro celou aplikaci se snažím, aby byla jednoduchá a zároveň, aby uživateli přinesla možnost ovlivnit to, jak výsledný rozvrh bude vypadat. Jak jsem již zmínil, jde o to, najít rovnováhu mezi tím, co aplikace zvládne sama a tím co uživatel může nastavit. Proto zde budu zároveň s klady, zmiňovat i zápory, které vznikají tím, že uživatel nemá všechny možnosti nastavení.

Jedním ze záporů je, že uživatel nemá možnost přiřadit určitou učebnu k určité aktivitě. Nemůže říct, potřebuji přesně tuto učebnu pro tuto aktivitu. Řešením by bylo u aktivit dát nepovinnou možnost nastavení určité učebny, nebo umožnit uživateli vytvářet typy učeben, čímž by mohl vytvořit unikátní typ. Tento typ by pak přiřadil jedné určité učebně a u aktivity by mohl nastavit, že tento typ požaduje u této aktivity. Z aplikace jsem toto nastavení vynechal z důvodu toho, že by muselo být přidáno další nastavení u aktivit a v kapitole třídy a aktivity (kap. 4.2.4) je vidět, že je tam již relativně dost nastavení. Možnost nastavení typu učebny, tedy možnosti přiřadit aktivitu množině určitých učeben, shledávám jako dostačující.

Druhý ze záporů je nemožnost nastavit časové preference jednotlivým učebnám. Uživatel nemůže nastavit, že konkrétní učebna je k dispozici pouze v určitý den a určité hodiny. Toto nastavení jsem vynechal z toho důvodu, že při vytváření každé učebny



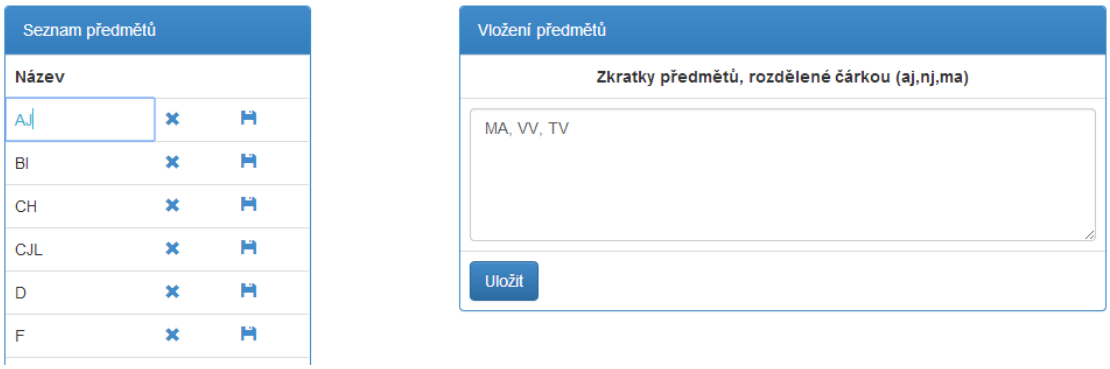
Obrázek 4.1. V pravé části obrázku je formulář pro vkládání učeben, v levé části zas uživatel může mazat a upravovat učebny.

by uživatel musel pokaždé vyplňovat časové preference této učebny, přestože by byla většina stále stejná. Střední a základní školy mají většinou svá zázemí a učebny mají k dispozici neustále.

Na obrázku 4.1 je ukázka toho, jak aplikace vypadá při zadávání učeben. Uživatel může vložit více učeben najednou. Navíc může ihned učebny upravit (funguje zde tzv. inline úprava, stačí pouze kliknout na data, která chci upravit a okamžitě v řádku upravuji), nebo vymazat.

4.2.2 Předměty

Předměty jsou další zdroj pro generování rozvrhu. Aplikaci bohatě stačí pouze názvy předmětů, popřípadě pouze zkratky předmětů. To k jaké třídě patří nebo v jakém typu učebny mají být učeny, pak přiřazuji až u jednotlivých tříd. Zde neurčuji k předmětu více dat, protože by vznikala redundantní data a uživatel by musel vkládat více stejných dat. Jestliže mám nějaký předmět, tak mi stačí jeho název a až u tříd uvedu, že první třída má tento předmět s určitým učitelem a devátá třída s jiným. Je zbytečné zde vkládat matematiku pro první třídu a matematiku pro devátou třídu.



Obrázek 4.2. Jednoduchá úprava, vkládání a mazání předmětů.

Proto zde uživatele nutím vyplnit pouze zkratky předmětů. Na obr. 4.2 je vidět obrazovka předmětů. Uživatel zde má jedno velké textové pole, takže všechny předměty může napsat rychle a najednou. Opět zde funguje rychlá inline úprava a samozřejmě možnost smazání předmětu.

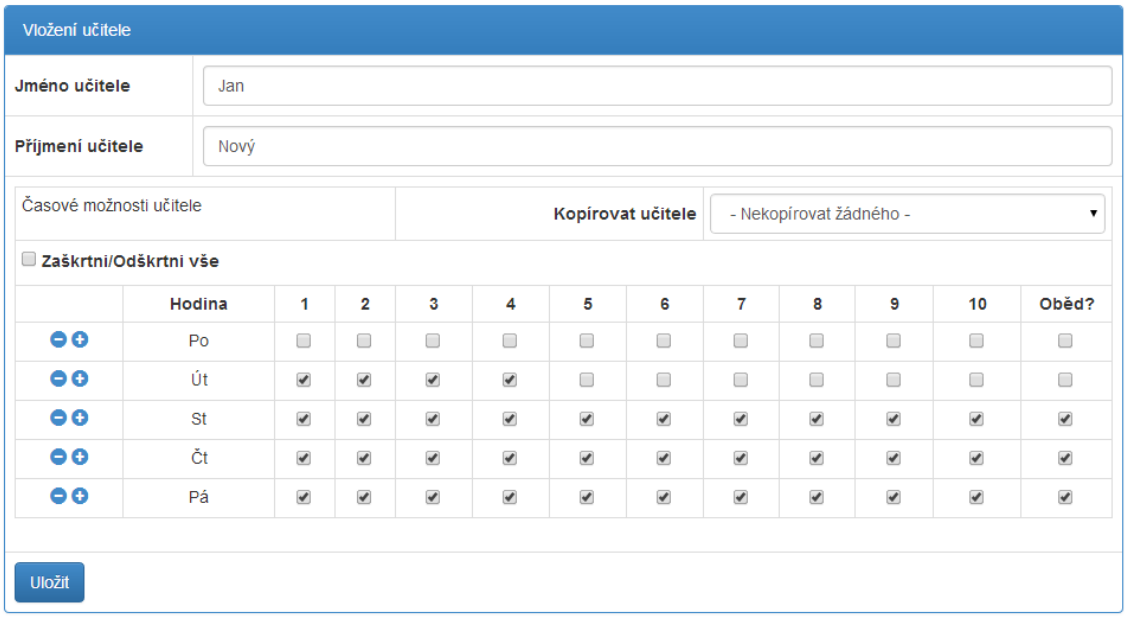
Nevýhoda je v tom, že zde není možnost nastavit časové preference předmětů. Prakticky to znamená, že uživatel nemůže říct, že by tento předmět měl raději dopoledne než odpoledne. Důvod je stejný jako u předchozího zdroje. Uživatel by musel u každého vkládaného předmětu zadávat časové preference předmětů. Rád bych uvedl, že tuto

možnost neztrácí úplně. Protože hned u dalšího zdroje (kapitola 4.2.3, učitelé) je možnost nastavit časové preference a zde uvedu, jak nastavení časových preferencí učitele může ovlivnit časové preference předmětům.

4.2.3 Učitelé

U učitelů jsem ponechal možnost nastavení časových preferencí. Je to jediný zdroj, který tuto možnost má. Mohl jsem zde také časové preference vynechat a předpokládat, že jsou učitelé k dispozici neustále. Nicméně taková situace není velmi často reálná, protože učitelé mohou být na škole například jen na částečný úvazek, nebo některé dny neučí a mají tyto dny pouze na přípravu. Proto jsem došel k závěru, že bez nastavení časových preferencí učitelů by aplikace byla nevyužitelná.

Jedná se o trochu složitější nastavení než u ostatních zmíněných zdrojů. I proto jsem se nastavení snažil zjednodušit, jak nejvíce to jde. Na obr. 4.3 je ukázka toho, jak se vkládá učitel. Uživatel může nastavit učiteli jaké dny i jaké konkrétní hodiny je učitel k dispozici pro rozvrh. Pro usnadnění je zde možnost označit/odznačit vše a označit/odznačit jeden konkrétní den. Navíc může uživatel zkopírovat jiného, existujícího učitele. Tím se zkopírují časové preference existujícího učitele do právě vytvářeného.



Obrázek 4.3. Ukázka vložení učitele. Uživatel může každému učiteli nastavit, kdy je k dispozici pro vyučování.

Nastavení časových preferencí u učitelů vnáší do aplikace dostatečnou míru uživatelské proměnné a se správným nastavením časových preferencí dokáže aplikace vygenerovat rozvrh, jaký si uživatel představuje. Uvedu zde příklady scénářů od jednoduchých po složitější, včetně toho jak přiřadit určitý předmět do dopoledních hodin, či na konkrétní hodinu (je to alternativní a složitější způsob řešení, primárním řešením by bylo nastavení časových preferencí u každého předmětu, což jsem neudělal z důvodu vysvětleném v předchozí kapitole 4.2.2).

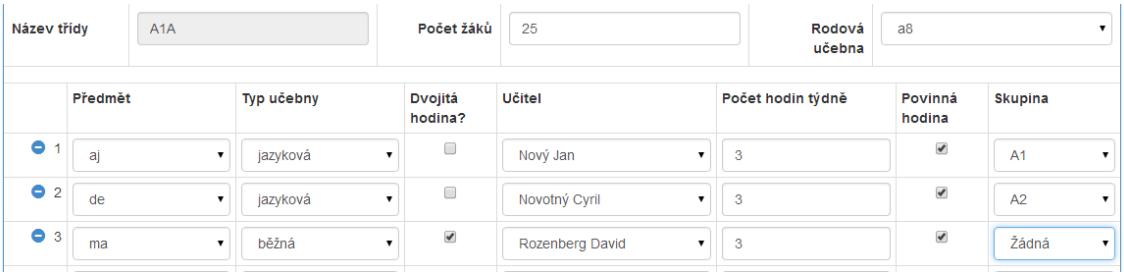
- Učitel má plný úvazek, může učit stále. Bude mít zaškrtnuté vše, pomocí tlačítka „zaškrtni vše“.

- Učitel má plný úvazek, ale pátek neučí a připravuje si materiály. Bude mít zaškrtnuté vše a pátek se odškrtne tlačítkem mínus u tohoto dne.
- Učitel má částečný úvazek a může jen určité dny a hodiny. Učitel bude mít zaškrtnuta určitá pole podle úvazku.
- Učitel potřebuje vždy pauzu po čtyřech hodinách. Stačí odškrtnutí hodin, kde nechce, aby učil. Algoritmus pak nemá žádnou možnost na takovou hodinu přiřadit vyučování.
- Každou středu od sedmé hodiny má škola schůzku. Každému učiteli, který se má schůzky účastnit se nastaví aby ve středu po sedmé hodině nemohl mít žádné vyučování.
- Učitel požaduje určitý předmět v určité hodiny, nebo dopoledních hodinách. Jde o zmíněné alternativní řešení, ale plně funkční. Stačí vytvořit nového učitele se stejným jménem a nějakým rozlišením ve jménu, abychom je od sebe mohli rozeznat. Tomuto dvojníkovi nastavíme určitou hodinu, či dopolední hodiny. Následně aktivitě, kterou si přejeme mít určitou hodinu s tímto učitelem, přiřadíme tohoto vytvořeného dvojníka a ne původního učitele. Tím algoritmu dovolíme dát předmět jen na zadaná místa.

Jedná se o velice mocný nástroj, který může do výsledného algoritmu vnést velkou míru uživatelské proměnné. Stačí tohoto nastavení využít a ne u každého učitele bezmyšlenkovitě odškrtávat všechna pole. Někde je potřeba umožnit nastavení časových preferencí, jinak by algoritmus nemohl vygenerovat uživatelsky přívětivý rozvrh. U všech zdrojů by šlo toto nastavení udělat, nicméně taková aplikace již existuje a jak jsem zmínil, motto této aplikace je snaha být co nejjednodušší. Proto, když už se někde musí nastavovat časové preference, tak u učitelů je to dle mne jediné logické místo, kde to umožnit.

4.2.4 Třídy a aktivity

Třídy jsou poslední zdroj, který aplikace využívá. U tříd mohu nastavit název, počet žáků a rodovou učebnu. Počet žáků je důležitý kvůli následnému přiřazení učeben a rodová učebna je vlastně preferovaná učebna třídy. Kdykoliv je tato učebna volná a vyhovuje požadavkům aktivity této třídy, tak tuto aktivitu algoritmus přiřadí k této aktivitě.



Obrázek 4.4. Aktivity v aplikaci.

Zároveň u tříd mohu nastavit všechny aktivity, které tato třída má. Co je aktivita již bylo zmíněno v kap. 4.2 na straně 23. Aktivitám tedy musím přiřadit všechny zdroje, které bude používat. Jak takové aktivity vypadají v aplikaci je vidět na obr. 4.4. Dle tohoto obrázku postupně popíši, jaké zdroje aktivitě mohu přiřadit a jaké vlastnosti jim mohu nastavit.

Třídu má aktivita automaticky přiřazenou, protože aktivitu vytváříme pouze u konkrétní třídy. V prvním sloupci nastavím o jaký předmět se jedná. Toto se bere ze zdroje předměty, viz. kap. 4.2.2.

V druhém sloupci přiřadím aktivitě množinu učeben, které ji může algoritmus přiřadit a to dle již zmíněných typů: běžná, laboratoř, tělocvična a jazyková učebna.

V dalším, třetím, sloupci mohu nastavit aktivitě vlastnost, zda-li je dvojitá, nebo není. Dvojitá hodina znamená, že aktivita má délku dvou po sobě jdoucích hodin.

Ve čtvrtém sloupci nastavím, který učitel má tuto aktivitu vyučovat a v jakém čase aktivita může být umístěna, jak již bylo zmíněno v kapitole 4.2.3.

Sloupec „počet hodin týdně“ si myslím, že mluví jasně sám za sebe. Nastavím zde, kolikrát týdně se tato aktivita v této třídě vyučuje. Neboli toto číslo udává, kolik takovýchto aktivit se má vytvořit.

Předposlední sloupec je nastavení, zda je aktivita brána jako povinný předmět. Je to zde z důvodu toho, že je často nastaven na školách limit pro maximální počet povinných hodin denně.

A poslední sloupec „Skupina“. U této vlastnosti pro aktivitu vlastně mohu rozdělit aktivity pro podskupiny dané třídy. Nyní jsou zde následující možnosti: žádná, A1, A2, A3, B1, B2, B3. Pokud aktivita nemá nastavenou žádnou skupinu, je tato aktivita určená pro celou třídu, pro všechny žáky. Nastavení skupiny si ukážeme na příkladu pro tělocvik. Skupina A1 může představovat např. kluky a skupina A2 holky. Algoritmu tak říkáme, že rozdělujeme třídu na půl a tyto dvě skupiny mohou být v rozvrhu ve stejný čas. Jiné skupiny žáků jsou například pro jazykové hodiny. Neplatí, že holky budou mít angličtinu a kluci němčinu (pravděpodobně). Proto můžeme využít další skupinu B. B1 mohou být angličtináři, B2 němčináři a B3 se mohou učit španělštinu. Tímto aplikaci řekneme, že dělíme třídu na třetiny a tyto skupiny mohou začínat hodinu ve stejný čas.

Zde tedy navrhnu všechny aktivity pro třídu, které má algoritmus rozvrhnout. Mohu několikrát nastavit ten samý předmět. Neznamená, že mám v jednom řádku již nastavenou matematiku, že ji již nemohu nastavit znovu. Toho se dá využít, pokud mám aktivity s různými preferencemi. Jednou mohu chtít matematiku v běžné učebně a třikrát týdně. Podruhé bych chtěl matematiku v laboratoři, jako dvojitou hodinu a pouze jedenkrát týdně. Toto mohu bez problému v aplikaci nastavit. Celkově tak budu mít matematiku čtyřikrát týdně, z toho jedna matematika bude dvojitá hodina a bude v laboratoři.

Toto řešení jsem po několika návrzích a konzultacích shledal jako dostačující. Aktivity uživatel může nastavit dle své libosti. Vyjmenoval jsem zde, co vše aplikace umí. Nyní jsou na řadě zápory. Opět zde uvedu nemožnost nastavit časové preference každé aktivity. Tato problematika byla dostatečně probrána v předchozích kapitolách. Dalším záporem je, že toto nastavení funguje na týdenní bázi rozvrhu. Proto nepodporuje sudé a liché týdny. Jako další věc, kterou rozvrh nezvládne jsou aktivity, které jsou pro více tříd najednou. Proto by byla potřeba vytvořit další stránku, kde by uživatel mohl tyto aktivity vytvářet. Další věc je, že mohu vytvořit maximálně aktivitu dlouhou dvě hodiny, ne delší. Nicméně delší hodiny zpravidla na základních a středních školách nepotřebujeme, proto jsem zde zvolil pouze možnost dvojité hodiny, kdy uživatel pouze zaškrtne pole, zda chce, aby byla aktivita dvojitá hodina, nebo to nechce. Pokud by zde bylo zadávání délky hodiny, musel by uživatel toto textové pole pokaždé vyplňovat, což je náročnější možnost než pouze zaškrtnout pole.

Jsou to vše návrhy, které nebyly v počátečním návrhu aplikace a přišlo se na ně až při závěrečné konzultaci na Arcibiskupském gymnáziu. Aplikace ale poběží ve zkušebním provozu právě pro toto gymnázium a tyto návrhy, pokud budou potřeba, budou zařazeny do seznamů věcí pro budoucí vývoj.

4.3 Ostatní nastavení

Kromě uvedených možností vkládání, úpravě a nastavení zdrojů, aplikace obsahuje jednu stránku, kde uživatel může nastavit pár dalších věcí. Toto nastavení je již předvyplněné, takže uživatel není nucen ho vyplnit. Co uživatel může nastavit a jak toto nastavení ovlivní vygenerovaný rozvrh:

- Začátek vyučování.
- Délka hodiny.
- Délka malé a velké přestávky.
- Po jakých hodinách je velká přestávka (všechny dosud uvedené nastavení neovlivňují generování rozvrhu, ale jeho následné zobrazení).
- Maximální počet povinných hodin denně.
- Od kdy, do kdy, má mít třída obědovou pauzu (tyto dvě nastavení již ovlivňují generování rozvrhu, jsou to vlastně slabé podmínky pro algoritmus).

Aplikace navíc rozlišuje dva typy uživatelů: admin, běžný uživatel.

Uživatel je přiřazen k určité škole, ke škole může být přiřazeno více uživatelů. Prakticky to znamená, že k jedné škole může mít přístup např. ředitel a zároveň člověk, který má na starost rozvrhy, každý pod svým vlastním účtem. Oba dva ale uvidí stejná data.

Admin má navíc možnost přidávání škol a uživatelů.

Opět nejjednodušší možné nastavení, které koresponduje s hlavní myšlenkou aplikace, kterou je jednoduchost.

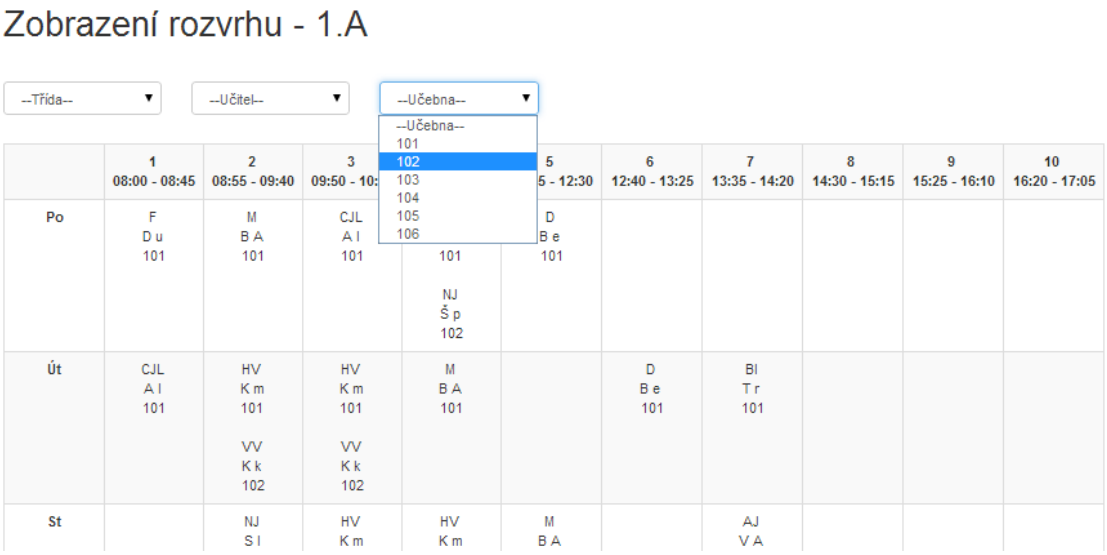
Dále aplikace podporuje více jazyků. Momentálně podporuje češtinu a angličtinu. Využívá k tomu PHP knihovnu gettext. Tato knihovna dokáže z aplikace dostat veškeré texty do *.po souboru (portable object). Tento soubor je možný editovat gettext editorem. Nejznámější je asi Poedit. S takovým editorem již může pracovat překladatel. V editoru si otevře vygenerovaný soubor a přeloží všechny texty. Editor dokáže tento překlad uložit do binárních dat, do *.mo (machine object). S takovým souborem již dokáže pracovat gettext. Aplikaci již jen řekneme s jakým *.mo souborem má pracovat (s jakým překladem) a všechny texty se dle toho přeloží [16].

Nyní ukáži jak aplikace generuje a zobrazuje rozvrh.

4.4 Generování a zobrazení rozvrhu

V aplikaci je nyní vše nastaveno a všechny zdroje vloženy. Poslední věc, kterou aplikace umí je vygenerování a zobrazení rozvrhu. Vygenerování se již spustí pouhým tlačítkem „Generovat rozvrh!“. Poté již jen uživatel počká, dokud aplikace nevygeneruje rozvrh, který si pak může zobrazit. Je možné si zobrazit rozvrh ze třech různých pohledů: z pohledu třídy, z pohledu učitele a z pohledu učebny. Ukázka je na obrázku 4.5.

Navíc aplikace umožňuje export do formátu pdf. Každý rozvrh, od každého zdroje, zobrazí na jednu stránku. Uživatel tak dostane několika stránkový pdf soubor, který je okamžitě připravený k vytisknutí a k používání.



Obrázek 4.5. Uživatel si může zobrazit rozvrh z různých zdrojů.

4.5 Algoritmus

Jak jsem již zmínil, v aplikaci jsem použil rekurzivní algoritmus, který je použit v aplikace Free Timetabling Software a jeho autor ho nazývá „recursive swapping“ [15]. Autor zveřejnil pouze kostru algoritmu. Já jsem algoritmus rozšířil o další dvě části. O validaci a o přiřazení učeben k aktivitám. Samotný algoritmus jsem se snažil vylepšit v následujících oblastech: kontrola podmínek, přiřazení aktivity do rozvrhu (výběr místa v rozvrhu) a vyhnutí se cyklení. Každé části se budu věnovat v samostatných kapitolách.

4.5.1 Validace

Validace je prvním krokem při generování rozvrhu. Snaží se zkontrolovat ze zadaných dat, zda půjde rozvrh vygenerovat. Pokud validace najde nějaký důvod, proč by rozvrh neměl jít vygenerovat, tak ho okamžitě vrátí a vůbec se nepouští do časově náročné generace.

Validace nejdříve kontroluje učitele, dále učebny a následně aktivity, vzhledem k tomu jaký typ učebny potřebují.

Validace učitele probíhá tak, že program zjistí, kolik má každý učitel aktivit a kolik z těchto aktivit jsou dvojité, neboli, které z nich mají délku dvě vyučovací hodiny. Následně každému učiteli zjistí, kolik hodin může vyučovat a kolik zvládne vyučovat dvojitých hodin.

Existuje například učitel, který může učit pátou, šestou a sedmou hodinu v pondělí a celý den v úterý (deset hodin). Tento učitel tak má celkem 13 časových oken. První validace u každého učitele spočívá v tom, jestli má vůbec kapacitu na to učit všechny předměty. To zjistíme jednoduchým výpočtem.

$$počet\_aktivit\_délky\_jedna + počet\_dvojitých\_aktivit * 2 <= počet\_časových\_oken$$

Pokud by tato rovnice neplatila, učitel by měl více předmětů než možností, kam je umístit.

Druhá validace spočívá v tom, že program zjistí, zda učitel zvládne všechny dvojité hodiny. Učitel, kterého jsem uvedl jako příklad, zvládne v pondělí jednu dvojitou hodinu (buď může být dvojitá hodina pátou a šestou, nebo šestou a sedmou, další dvojitá by se nevešla) a v úterý maximálně pět dvojitých hodin (první a druhou hodinu jedna hodina, třetí a čtvrtou druhá hodina, atd...), dohromady tak zvládne maximálně šest dvojitých hodin a pak musí tedy logicky platit následující výpočet.

$$počet\_dvojitých\_aktivit <= počet\_možností\_učit\_dvojitou\_hodinu$$

Pokud by toto neplatilo, učitel by měl více dvojitých hodin než by mohl učit.

Nyní je na řadě *validace učeben*. Ta předpokládá, že pro každou třídu existuje alespoň jedna učebna. To je z důvodu toho, aby mohly mít vyučování všechny třídy zároveň. Další kontrola u validace učeben je, že zvládne rozmístit všechny aktivity.

$$počet\_všech\_aktivit <= počet\_časových\_možností\_týdně * počet\_učeben$$

Poslední validace, *validace aktivit dle typu* je prakticky stejná jako předchozí, s tím rozdílem, že neberu všechny aktivity a všechny učebny, ale jen aktivity a učebny daného typu.

$$počet\_aktivit\_typu\_T <= počet\_časových\_možností\_týdně * počet\_učeben\_typu\_T$$

Validace proběhne vždy celá a nastanou-li nějaké problémy, tak aplikace vrátí varovné hlášky uživateli a nepokračuje v generování. Je-li vše v pořádku, tak začíná samotné generování.

4.5.2 Jádro algoritmu

Jak algoritmus funguje:

1. Vypočítá váhu jednotlivým aktivitám a srovná tyto aktivity, od nejhůře umístitelné po nejlehčeji umístitelnou, do množiny.
2. Vezme z množiny aktuálně nejhůř umístitelnou aktivitu A a zkusí najít volné pole v rozvrhu.
3. Pokud existuje jedno a více polí, vybere to nejvhodnější, vloží aktivitu do rozvrhu a vrátí se do bodu 2).
4. Pokud není žádné místo volné, vybere místo, kde aktivita způsobí nejméně problémů.
5. Odebere konfliktní aktivity např. B1, B2 a B3 a vloží aktivitu A na vybrané místo. Rekurzivně zkusí umístit konfliktní aktivity.
6. Pokud se algoritmu nepovede umístit konfliktní aktivity, přidá je zpět do původní množiny a pokračuje bodem 2). Množina teď má 3 (počet konfliktních aktivit) - 1 (aktivita A byla umístěna) = 2 (o dvě aktivity více k umístění).

Bod 1: Vypočítání váhy a seřazení aktivit. V algoritmu používám statické ohodnocení aktivit. To znamená, že na začátku vypočítám každé aktivitě váhu a pak ji již v průběhu algoritmu neměním, viz. kap. 2.3.2. Pro algoritmus toto není důležité jakou vezme zrovna aktivitu, rozvrh by dokázal i tak vytvořit. Ale to, že na začátku aktivity seřadím a beru nejtěžší možnou, tak algoritmus zrychlím až desetkrát. Samotná váhová funkce není nikterak složitá. Záleží na tom, na kolik různých míst mohu aktivitu umístit. Tedy na tom jakého má učitele a jaké on má časové možnosti a také na tom jestli

je to dvojitá aktivita nebo ne (učitel zvládne učit méně dvojitých hodin, tudíž existuje méně časových možností, kam aktivitu umístit). Váha se tedy rovná počtu časových možností. Čím menší má aktivita váhu, tím hůře půjde umístit. Může se stát, že váha je stejná u některých aktivit, pak dostane přednost ta, která je povinná.

Bod 2: Zjištění, zda existuje volné místo v rozvrhu pro aktuálně nejhůř umístitelnou aktivitu. Z množiny seřazených aktivit vezmu tu aktivitu A s nejnižší váhou a odeberu ji z této množiny. Následně projdu všechna místa, kde může být aktivita umístěna a pro všechna tato místa zkontroluji silné podmínky a pro každé takové místo si uložím kolik aktivit a které konkrétní aktivity dělají problém s případným vložením aktivity A na toto místo. Pokud existuje jedno a více míst, kam mohu aktivitu umístit, tak je vše v pořádku a mohu pokračovat bodem tři. Jestliže žádné takové místo neexistuje, dostává se algoritmus do konfliktu. V tom případě pokračuje do bodu čtyři.

Bod 3: Vložení aktivity na nejlepší volné místo. Volné místo pro aktivitu A existuje. Je-li pouze jedno, tak není co řešit, algoritmus tam aktivitu vloží a pokračuje do bodu dva. Pokud je takových míst více, nastává problém, které místo pro aktivitu vybrat. Existuje možnost, že vybereme náhodné pole. Vše by bylo v pořádku, ale rozvrh na konci by určitě nechtěla žádná škola, učitel, ani žák. Je totiž možné, že by v rozvrhu bylo spousty mezer, že by stejné předměty byly ve stejný den, třída by mohla začínat hodinu až někdy odpoledne, atd... Proto na řadu přichází slabé podmínky.

Uživatel nemá v nastavení možnosti, aby ovlivnil chování slabých podmínek algoritmu. Algoritmus se o to postará sám. Pro každé volné pole zkontroluje algoritmus všechny slabé podmínky. Každá podmínka má určitou váhu. Za splnění každé takové podmínky se přičte váha této podmínky k celkovému skóre daného volného pole. Následně se již pouze vybere pole s nejlepším skórem a tam se aktivita umístí. Po umístění algoritmus pokračuje do bodu dva.

Co se týká váhy slabých podmínek, tak například největší váhu má takové umístění, které je buď na začátku dne, nebo před tímto umístěním není žádné volné místo, nebo je tam maximálně jedno volné místo a to takové, že je to obědová pauza. Jelikož má tato podmínka největší váhu, algoritmus bude vybírat místa tak, že před nimi v daný den už budou aktivity a žádná, popřípadě jedna volná hodina. Jednoduše to znamená, že tato podmínka se snaží o to, aby nebyly v rozvrhu žádné nežádoucí mezery.

Další podmínka s menší váhou než předchozí podmínka je ta, která zajišťuje, aby stejné předměty nebyly pokud možno ve stejný den vícekrát.

Pokud bych měl v algoritmu pouze tyto dvě podmínky implementovány, vypadalo by to takto. Algoritmus vybírá ze třech různých umístění. Podmínka na nežádoucí mezery má váhu 100 a podmínka na to, aby předmět nebyl dvakrát ve stejný den má váhu 50. První umístění splňuje pouze první podmínku, tzn. nejsou v rozvrhu žádné nežádoucí mezery, ale daný předmět již daný den je vyučovaný. Skóre pro toto pole je 100. Další pole splňuje pouze druhou podmínku. Tzn. že v rozvrhu je nežádoucí mezera, ale daný den předmět ještě učen nebyl. Analogicky, když pak třetí umístění splňuje obě podmínky, má skóre 150 a algoritmus toto pole vybere jako vítěze a umístí zde aktivitu A. Jestliže by zde žádné pole, které splňuje obě podmínky nebylo, vybral by algoritmus první umístění, nebyla by zde mezera, ale předmět by byl daný den již podruhé.

Momentálně algoritmus kontroluje tyto podmínky, seřazené od největší váhy:

- Nežádoucí mezery (prakticky tato podmínka zarovnává rozvrh doleva).

- Pauza na oběd (pokud mají žáci dlouhý den, je potřeba tam pauzu mít, proto má tato podmínka druhou největší váhu).
- Předmět není dvakrát denně (jestliže vybírám mezi místy, které splňují obě předchozí podmínky, toto je podmínka, která rozliší jaké místo je lepší).
- Podmínka, která kontroluje, že se nepřekročil maximální počet předmětů v daný den a dává přednost dni, kde je menší počet aktivit, než v jiný den (vyrovnanost rozvrhu, aby jeden den nebylo 15 předmětů a druhý den jeden předmět).

Pokud skončí více míst se stejným skórem, tak se vybere náhodně jedno z nich. Tímto bodem mohu získat každou generací rozdílný rozvrh.

Bod 4: Výběr místa, které způsobí nejméně konfliktů. Algoritmus nenašel žádné volné místo pro aktivitu *A*. To znamená, že kamkoliv by šla umístit aktivita *A*, tak by způsobila nějaký problém, porušila by jednu a více silných podmínek. V bodě 2) tohoto algoritmu jsem si uložil u každého takového místa seznam aktivit, které způsobují problém. Algoritmus nyní vybere místo, které způsobí nejméně problémů. Například je to místo, kde způsobují problém aktivity *B1 a B2*. Pro lepší představu uvedu, jaké problémy mohou způsobit. Aktivita *B1* má stejného učitele jako aktivita *A*, ale ten už učí v jiné třídě ve stejný čas. Aktivita *B2* je ve stejné třídě, ale již je umístěna na místě, kam chce algoritmus umístit aktivitu *A*. Byly by tak dva předměty najednou ve stejné třídě a ve stejný čas.

V tomto bodě navíc řeším to, aby se algoritmus nedostal do cyklu. Pokaždé, když odstraním aktivitu již umístěnou v rozvrhu, uložím si do seznamu, že aktivita *C* již byla odstraněna z místa *T*. Platí tedy to, že vybírám místo, které způsobuje nejmenší počet konfliktů, ale zároveň musí platit, že všechny konfliktní aktivity ještě nikdy nebyly odebrány z míst, kde jsou právě umístěny.

Pro lepší představu uvedu opět příklad. Mám problémové aktivity *B1 a B2*. Odstraním je z rozvrhu a uložím si, že *B1* byla odstraněna z *T1* a aktivita *B2* z místa *T2*. Algoritmus běží dále a stane se to, že aktivity *B1 a B2* se opět octnou na *T1 a T2*. Opět jsou problémové tyto aktivity a je to místo s nejmenším počtem konfliktů. Pokud bych je chtěl odebrat, tak je to již náznak toho, že se dostávám do cyklu. Proto raději vyzkouším jiné místo s jinými problémovými aktivitami, které ještě nikdy nebyly odebrány z místa, kde jsou právě umístěny.

Bod 5: Odebrání konfliktních aktivit. Nyní algoritmus odebere z rozvrhu konfliktní aktivity, uloží u každé z nich odkud byly odebrány (viz. předchozí bod 4)) a již může bez problému umístit aktivitu na vybrané místo, protože již zde není žádný konflikt. Odebrané aktivity nyní tvoří novou množinu, kterou pošleme rekurzivně do algoritmu, konkrétně do bodu 2). Tzn. že momentálně budou tvořit množinu aktivity *B1 a B2*. Algoritmus vezme tu hůře umístitelnou, zkusí nalézt místo, pokud ho najde, tak ji umístí. Následně to samé udělá s druhou aktivitou. Pokud by ovšem nastal zas konflikt, algoritmus opět najde místo s nejmenším počtem konfliktů, odebere konfliktní aktivity a půjde do dalšího levelu rekurze a tak stále dokola, dokud aktivity *B1 a B2* nebudou umístěny.

Pokud se vše povede, algoritmus umístí aktivity *B1 a B2*, tak dále pokračuje do bodu 2), vezme další nejhůře umístitelnou aktivitu a hledá pro ni místo.

Bod 6: Algoritmu se nepovedlo umístit konfliktní aktivity. Samozřejmě by se mohlo stát, že by algoritmus mohl jít hlouběji a hlouběji, což je velice paměťově náročné. Proto je zde nastaven maximální level rekurze. Pokud algoritmus dosáhne tohoto maximálního

bodu rekurze, nebo již zkusil všechna místa a konflikty, tak jednoduše přidá konfliktní aktivity zpět do původní množiny. V tomto konkrétním případě, pokud by neuspěl, tak umístil aktivitu *A* a nepodařilo se mu umístit aktivity *B1 a B2*. Ty přidá do původní množiny a bude pokračovat do bodu 2). Celkově algoritmu přibude jedna aktivita navíc k rozmístění, protože jednu umístil a dvě přidal zpět do původní množiny.

Algoritmus nemůže zaručit kdy skončí, ale když skončí, tak výsledek bude správný.

4.5.3 Přiřazení učeben

V tomto okamžiku jsou již všechny aktivity rozvrženy. Jediné o co se algoritmus nestaral do této doby, je přiřazení učeben jednotlivým aktivitám. Aktivity mají daný určitý typ, dle toho v jakém typu učebny mohou být učeny. A dle toho algoritmus přiřadí učebnu k aktivitě.

Hraje zde roli nastavení rodové učebny pro třídu. Algoritmus funguje takto:

- Nejdříve vezme každou třídu a v každé třídě vezme každou jednotlivou naplánovanou aktivitu.
- Zjistí typ aktivity a pokud je rodová učebna stejného typu, tak ji přiřadí. Tzn. že algoritmus se snaží primárně přiřadit rodovou učebnu třídy pro všechny aktivity.
- Pokud se algoritmu nepovedlo primárně přiřadit rodovou učebnu, protože není stejného typu jak aktivita, nebo tuto učebnu již má v daný čas jiná aktivita, zkusí přiřadit jinou učebnu. V algoritmu existuje množina již použitých učeben pro třídu. Tuto množinu prohledává nejdříve. Efekt toho, že algoritmus se snaží nejdříve použít učebny, které již byly pro třídu použity, je ten, že se minimalizuje počet použitých učeben. To také znamená, že třídy nemusí tolik cestovat a používají učebny, které již znají.
- Pokud se ani tak nepodařilo algoritmu přiřadit učebnu s množiny již použitých učeben, tak prohledá všechny ostatní. V okamžiku, kdy najde učebnu, kterou pro aktivitu může použít, tak ji aktivitě přiřadí a zařadí tuto učebnu do zmiňované množiny použitých učeben.
- Může se stát, že se algoritmu nepodaří přiřadit učebnu (je málo učeben určitého typu, kontroluje se to sice ve validační části algoritmu, nicméně může nastat situace, že není dostatek učeben, díky aktivitám ve skupině). Toto vůbec nevadí, algoritmus pokračuje dál a rozvrh na konci zobrazí problém, že nedokázal přiřadit učebnu nějaké aktivitě. Uživatel pak může přiřadit např. jiný typ učebny k dané aktivitě, nebo jinak vyřešit tento problém, dle vlastního uvážení.

Nyní je rozvrh kompletní, algoritmus končí a vrací uživateli grafický výstup.

Kapitola 5
Testování

Tato kapitola je zaměřena na testování uživatelského rozhraní a na zátěžové testy. Aplikace je vyráběna za spolupráce s Arcibiskupským gymnáziem a zde také byla podrobena uživatelskému testování. Byly objeveny nějaké nedostatky, které byly zapracovány do této práce. Je jasné, že po testování jedné školy nebude program úplně bez problému. Ale tento projekt bude pokračovat i dále a půjde do fáze alfa testování, kdy se připojí další školy.

5.1 Testování uživatelského rozhraní

Cílovou skupinou pro toto testování jsou lidé, kteří pracují na ZŠ, nebo SŠ, mají již nějaké zkušenosti s tvorbou rozvrhu a rozvrh školy mají na starost. Předpokládá se alespoň zběžná znalost práce s počítačem a internetovým prohlížečem.

Test probíhal v nastavení:

- Notebook
- Dvou jádrový procesor, 4GHz
- 4GB RAM
- Monitor v rozlišení 1280x960
- Připojena externí myš
- Operační systém Windows 7
- Prohlížeč Google Chrome

Aplikace již byla připravena a spuštěna. V aplikaci byla založena nová škola s novým uživatelem a nebyly zde vytvořeny žádné zdroje, aktivity ani vygenerovaný rozvrh. Pro participanta (účastníka testování) byla připravena sada úkolů, které souvisely s uživatelským rozhraním. Já jsem zde byl v roli moderátora. Jako moderátor jsem kladl doplňující otázky a nebo pomáhal participantovi v případě závažného problému. Jinak jsem participanta pozoroval a nezasahoval do plnění jeho úkolů. Hlavní úkol moderátora v uživatelském testování je zapisovat si poznámky v průběhu testu, snažit se nezasahovat do průběhu testu a na konci odvodit důsledky ze zapsaných poznámek.

Úkoly pro participanta byly vytvořeny tak, aby představovaly reálné využití aplikace. Jsou to tyto úkoly.

1. Přihlašte se do aplikace a vytvořte osm učeben.
   Každá učebna bude mít kapacitu pro 30 žáků. Navíc z osmi těchto učeben budou dvě typu *běžná*, dvě typu *laboratoř*, dvě *jazykové* a poslední dvě budou *tělocvičny*.
2. Změňte jedné tělocvičně název a jedné jazykové učebně její kapacitu z 30 na 35.
3. Vytvořte následující předměty: *angličtinu, němčinu, matematiku, češtinu a tělocvik.*
4. Změňte název předmětu *matematika* na *matematika2.*
5. Vytvořte tři učitele.

Dva z těchto učitelů (U1 a U2) pracují na plný úvazek a mohou mít všechny hodiny v týdnu. Třetí učitel (U3) pracuje na částečný úvazek a je k dispozici pouze ve středu dopoledne.

6. Vytvořte dalšího učitele (U4). Tento učitel je ve škole také na částečný úvazek a má stejné časové možnosti jako naposled přidaný (U3). Tedy ve středu dopoledne.
7. Vytvořte třídu.

   Třída má 30 žáků, jako rodovou učebnu má jednu z běžných učeben. Týdenní skladba této třídy je následující.
   - Matematika 2, dvakrát týdně, vyučovaná učitelem U1 v běžné učebně.
   - Čeština, jedna dvojitá hodina týdně, vyučovaná učitelem U2 v běžné učebně.
   - Angličtina, třikrát týdně, vyučovaná učitele U3 v jazykové učebně.
   - Tělocvik, rozdělený na holky a kluky, jednou týdně, vyučovaný učitele U1 a U2 v tělocvičně.
8. Vytvořte druhou třídu, která má stejnou týdenní skladbu, akorát místo angličtiny má němčinu s učitelem U4.
9. Nastavte, že velká přestávka je po třetí a čtvrté hodině a že obědová pauza musí být mezi pátou a šestou vyučovací hodinou.
10. Vygenerujte rozvrh a zobrazte si ho nejdříve z pohledu jedné a pak druhé třídy. Nakonec se podívejte jak vypadá rozvrh pro učitele U1.

5.2 Průběh testu

V této kapitole je popsáno jak probíhal test a jak se jeho výsledek promítl do aplikace.

5.2.1 Průběh jednotlivých úkolů

Vytvoření osmi učeben proběhlo v pořádku bez sebemenšího zádrhelu, participant věděl jak nastavit jednotlivým učebnám kapacitu i jejich typ.

Další úkol byla **úprava vlastností učeben** jako jejich název a kapacita. Participantovi trvalo trochu déle, než přišel na to, jak tyto vlastnosti upravit. Nejdříve ze všeho klikl na ikonku disketky, která je zde proto, aby uložila změny, které byly provedeny. Nakonec však participant dosáhl cíle tohoto úkolu bez nápovědy od moderátora.

Nyní měl participant **vytvořit pět předmětů** a následně jeden z nich upravit. Díky nabyté znalosti z předchozího úkolu zde probíhalo vložení i úprava bez nejmenšího náznaku problému.

Následovaly dva úkoly pro **vložení čtyř učitelů**. První z těchto úkolů, kde bylo zapotřebí vložit dva učitele na plný úvazek a jednoho učitele na částečný úvazek, proběhl opět bez větších problémů. Participant se rychle zorientoval a využil rychlé možnosti zaškrtnutí všech časových polí a neodškrtával tato pole po jednom.

Druhý z těchto úkolů byl navrhnut proto, aby participant použil možnosti zkopírovat učitele. Této možnosti si všiml a správně ji použil.

Test pokračoval dalšími dvěma úkoly pro **vytvoření tříd**. Jeden z nich byl přímo na vytvoření a druhý opět na to, aby participant mohl použít možnosti kopírování třídy. Tento formulář je nejsložitější v aplikaci. Participant zaváhal ve dvou případech, ale nakonec vše splnil dle zadání. První věcí bylo špatné zadání češtiny a to konkrétně při zadávání dvojité hodiny, kdy měl zaškrtnout pole „dvojitá hodina“ a do kolonky „počet hodin týdně“ měl zadat číslo jedna. Participant ale zadal dvě dvojité hodiny týdně.

Druhý problém byl při zadávání tělocviku pro rozdělenou třídu. U tohoto bodu potřeboval nápovědu od moderátora. Následné kopírovaní třídy a změnu jednoho předmětu zvládl participant bez problému.

Poslední dva úkoly, **nastavení, vygenerování a zobrazení rozvrhu**, proběhly naprosto v pořádku. Participant udělal vše správně bez většího zaváhání.

5.2.2 Post-test dotazník

Post-test dotazník slouží ke zjištění toho, jaký měl dotazník na participanta vliv, jaké jsou pocity, zda se mu program líbil a jaké měl s programem potíže.

Porozuměl jste všem úkolům?
Ano.
Který úkol byl pro vás nejtěžší?
Nejtěžší bylo zadávání předmětu k jejich třídám. Možná je to proto, že jsem to viděl poprvé.
Ohodnoťte přehlednost programu (hodnocení jako ve škole od 1–5)?
1.
Líbil se vám design aplikace?
Ano, aplikace je hezká a velmi přehledná.
Naplnila aplikace vaše očekávání?
Spíše ano, na první pohled vypadá, že umí to, co ve škole potřebujeme.

5.2.3 Post-test rozhovor

Post-test rozhovor probíhal na základě vyplněného dotazníku a také na základě samotného testu. V rozhovoru jsem se dále doptával a vyptával na program a zároveň nechal participanta položit nějaké další otázky.

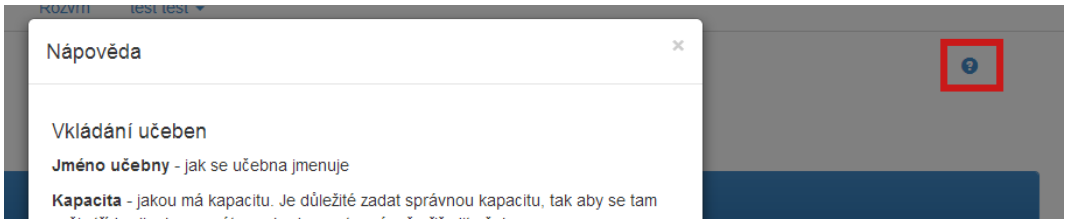
Z tohoto rozhovoru jsem se dozvěděl následující věci.

- Participant pochopil zadávání dvojitých hodin tak, že zároveň se zaškrtnutím políčka „dvojitá hodina“ musí do kolonky „počet hodin týdně“ zadat i celkový počet hodin týdně. Tzn. že pro dvě dvojité hodiny týdně by zaškrtl políčko „dvojitá hodina“ a zároveň by vyplnil, že počet hodin týdně musí být 4.
- Participantovi dělalo největší problém zadat skupinový předmět, konkrétně rozdělený předmět tělocvik. Pochopil to tak, že musí zadat pro jeden tělocvik jednu skupinu a pro druhý tělocvik druhou skupinu. Aplikace ale pracuje tak, že je potřeba vyplnit děleným hodinám stejnou skupinu. Je tedy přesně zapotřebí zadat jaké konkrétní předměty mají být spolu.
- Pro participanta nebylo dostatečně viditelné, že se dají atributy učeben, nebo název předmětu upravovat. Pole pro úpravu vypadají jako text.
- Rozvrhář pokaždé neví, jaké dva konkrétní předměty mají být spolu. Pokud mám např. dělenou angličtinu s němčinou a na oba předměty mám učitele s částečným úvazkem, kteří ve škole nejsou nikdy ve stejný čas, tak tyto dva předměty nemohu dát do stejné skupiny, protože neexistuje žádný společný čas. Nemohly by být zařazeny do rozvrhu. A i přesto je potřeba třídu na tyto dva předměty rozdělit.
- Dále se participant vyptával na možnosti sudého/lichého týdne, možnosti přiřazení určitého učitele k přesně danému času a aktivitě, nebo potřeboval třetí typ přestávky (veškeré závěry z testu a tohoto rozhovoru jsou v další kapitole 5.2.4).

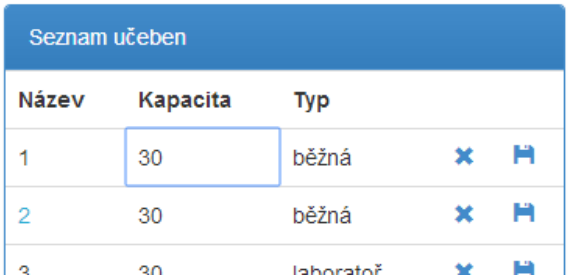
5.2.4 Vyhodnocení testu

Na základě tohoto testu jsem vyhodnotil, že aplikace je opravdu jednoduchá, přehledná a intuitivní, tak jak bylo zamýšleno. Participant navíc viděl aplikaci úplně poprvé a i přesto se dobře zorientoval a neměl žádné větší problémy splnit všechny zadané úkoly z testu. Test byl i úspěšný nejen v tom, že potvrdil jednoduchost uživatelského rozhraní, ale také objevil nějaké nedostatky, které byly po tomto testu odstraněny.

1. Na základě problémů, jako problém s vyplněním týdenní skladby pro třídu, nebo nedostatečná viditelnost možnosti editace atributů bylo upraveno uživatelské rozhraní.
   - U všech formulářů přibyly možnosti zobrazit rychlou nápovědu, viz obr 5.1.
   - Zvýraznění atributů, které se dají změnit, pokud přes ně uživatel přejede myší 5.2.



Obrázek 5.1. Nyní je možné zobrazit nápovědu u formulářů.



Obrázek 5.2. U atributů je nyní lépe poznat, že jdou změnit. Navíc je to popsané i v nápovědě.

2. Největší změna proběhla na základě rozhovoru o předmětech a jejich skupinách. Jak bylo zmíněno, před testem algoritmus pracoval tak, že jsme museli zvolit konkrétní aktivity, které měly být děleny. Např. pokud jsme chtěli půlenou angličtinu a němčinu, zadali jsme těmto aktivitám stejnou skupinu. Algoritmus pak hledal jedno místo v rozvrhu, kam by tyto dvě aktivity mohl umístit. Ovšem to nefunguje, pokud dva učitelé učící tyto dva předměty nemají žádný společný čas. Proto se tento algoritmus díky tomuto testu vylepšil. Nyní algoritmu mohu říct, které aktivity patří nějaké skupině a které algoritmy patří jiné skupině. Dopodrobna již bylo vše popsáno v kapitole 4.2.4 na straně 26.
3. Co se týká otázek od participanta.
   - Možnost přiřazení učitele k danému času a aktivity aplikace implicitně neumí. Nicméně existuje alternativní řešení popsané v kap. 4.2.3. Je možné, že pokud bude vývoj pokračovat, tak se tato možnost do aplikace doprogramuje.
   - Možnost sudého a lichého týdne aplikace bohužel neumí. Nyní pracuje pouze na týdenní bázi. Pokud by se v další části vývoje, při testování dalšími školami prokázalo, že je tato možnost potřeba, tak by to bylo další možné rozšíření, které by se

do aplikace mohlo přidat. Momentálně by se toto muselo s danou školou řešit tak, že bychom ji přidali nový účet. Do tohoto účtu bychom zkopírovali veškeré zdroje z účtu prvního a škola by si následně změnila týdenní skladbu u třídy tak, aby vyhovovala druhému týdnu. Takto by si škola mohla generovat dva různé rozvrhy.

- V aplikaci jde momentálně nastavit dva typy přestávek. U každé se dá nastavit délka. Jedna přestávka je běžná a druhá je velká. U velké přestávky se dá nastavit po jakých hodinách bude. Participant by ale na své škole potřeboval ještě třetí přestávku, protože dopoledne mají desetiminutové přestávky, pak mají jednu velkou patnáctiminutovou a odpoledne mají již jen pětiminutové. Proto se vygenerovaný rozvrh nezobrazí se správnými časy. Možnost přidat další typ přestávky je díky tomuto testu momentálně ve fázi analýzy a je možné, že se tato vlastnost v aplikaci v další verzi objeví.

5.3 Zátěžové testy

Pro aplikaci jsem provedl i zátěžové testy, abych zjistil jaké má algoritmus meze a jak dlouho bude přibližně trvat jeho průběh. Algoritmus nemá zaručený čas doběhnutí, protože záleží na tom, jak těžké je umístit všechny aktivity, proto to jsou jen orientační časy. Pokud má algoritmus dost místa pro každou aktivitu a nejsou žádné konflikty, algoritmus proběhne lineárně s počtem aktivit. Naopak pokud zde bude hodně konfliktů k řešení, čas k dokončení algoritmu se může klidně vyšplhat i k celým hodinám.

#	velikost problému	výsledný čas generování
1	1 třída, 28 aktivit	0,7 vteřin
2	2 třídy, 56 aktivit	1,36 vteřin
3	3 třídy, 84 aktivit	2,3 vteřin
4	5 tříd, 140 aktivit	4,2 vteřin
5	10 tříd, 280 aktivit	10,1 vteřin
6	15 tříd, 420 aktivit	19 vteřin
7	20 tříd, 560 aktivit	28,1 vteřin
8	30 tříd, 840 aktivit	48,9 vteřin
9	40 tříd, 1120 aktivit	78,5 vteřin

Tabulka 5.1. Zátěžové testy a jejich výsledky.

Při testech bylo v systému až 40 učitelů a 40 učeben. Aplikace zvládala všechny problémy vyřešit až do osmého případu pro 30 tříd. Při tomto testu se projevilo špatné nastavení databáze, která s touto instancí padala. Důvod byl takový, že celkový výsledek již měl větší velikost než databáze zvládla uložit. Po zvětšení limitu pro databázi již testy prošly dále a aplikace neměla problém ani se 40 třídami a 1120 aktivitami. Zátěžový test byl tedy úspěšný, protože odhalil chybu v aplikaci.

Je třeba dodat, že v tabulce 5.1 jsou průměrné časy mnoha spuštěných testů a je z ní vidět, že algoritmus zvládne velké instance a zvládne je rychle. Nicméně většina testů nemusela řešit velké množství konfliktů a jak bylo zmíněno, kdyby instance problému byla velmi těžká a bylo zde mnoho konfliktů, tak výsledný čas se může mnohonásobně zvýšit.

Kapitola 6
Závěr

V práci jsem se věnoval aplikaci pro tvorbu rozvrhu, jakožto hlavnímu tématu.

V první polovině práce je popsána do detailu problematika rozvrhování a to konkrétně programování s omezujícími podmínkami. Podrobně je zde popsáno co to podmínka je a že se v této problematice rozlišují silné a slabé podmínky.

Dále jsem popsal dva nejznámější obecné algoritmy, které je možné použít na problém rozvrhování, a to genetický algoritmus a rekurzivní algoritmus. V této části práce jsem popsal na jakém principu fungují a jaké jsou jejich výhody a nevýhody.

Protože jsem chtěl naprogramovat jedinečnou aplikaci pro generování rozvrhu, která ještě není k dispozici, tak jsem se v další části práci věnoval existujícím řešením. U každého řešení jsem popsal co je na něm dobré a co špatné.

Na základě této první poloviny práce, teorie problematiky rozvrhování a analýzy existujících řešení, jsem v druhé části práce popsal, navrhl a implementoval vlastní aplikaci pro generování rozvrhu pro střední a základní školy. Toto řešení je založeno na tom, aby bylo co nejjednodušší. Aby každý kdo s ním přijde do styku mohl pouze vložit data a okamžitě vygenerovat svůj vlastní rozvrh bez složitého nastavování.

Aplikace podporuje učebny, čtyři různé typy učeben, předměty, učitele, možnost nastavení časových možností učitele a samozřejmě třídy a nastavení toho, jaké má tato třída předměty a v jakém množství týdně. To vše v jednoduchém, hezkém a intuitivním uživatelském rozhraní.

Algoritmus, který je implementován v této aplikaci, je založen na rekurzivním algoritmu, konkrétně na existujícím řešení, které je použito v programu FET [15]. Zjednodušeně algoritmus funguje tak, že vybere aktivitu, kterou chce zařadit do rozvrhu, najde ji nejlepší místo a umístí ji zde. Pokud žádné volné místo není, tak najde místo, které způsobí nejméně konfliktů a tam tuto aktivitu umístí a rekurzivně zkusí umístit konfliktní aktivity.

Tento algoritmus jsem se snažil vylepšit. Algoritmus nyní kontroluje jiným způsobem podmínky. Nejdříve pomocí silných hledá volné místo a následně pomocí slabých se snaží najít to nejlepší umístění. Dále jsem algoritmu přidal mechanismus na vyhnutí se cyklu a nakonec jsem algoritmu přidal vlastní řešení, které přidává rozvrženým předmětům učebnu dle typu a snaží se o to, aby třídy co nejméně přecházely s jedné učebny do druhé.

Navíc tato aplikace byla vyvíjena za podpory Arcibiskupského gymnázia v Praze. Díky této podpoře jsem do aplikace mohl zapracovat i reálný pohled na generování rozvrhu a to přímo od člověka, který každoročně takový rozvrh pro toto gymnázium vytváří.

V závěru práce bylo provedeno testování uživatelského rozhraní a zátěžové testy. Výsledky s testováním uživatelského rozhraní byly do aplikace ještě zahrnuty. Díky testu například přibyly do aplikace nápovědy u formulářů nebo se změnil způsob chápání skupinových předmětů (dělení tříd na skupiny) v algoritmu. Tím aplikace může, ještě lépe než předtím, řešit reálné problémy spojené s těmito skupinovými předměty.

Na základě zátěžových testů jsem ověřil, že aplikace zvládne vyřešit i velký počet tříd. Za předpokladu, že má instance velké množství možností, kam může aktivity umístit, tak toto zvládne i ve velmi dobrém čase.

V budoucnu by se aplikace mohla dále rozšiřovat. Možností je spousta. Umožnit sudé/liché týdny, přidat další podmínky do algoritmu, umožnit uživateli přidat další typy učeben, přidat nějaké možnosti nastavení (samozřejmě tak, aby to stále korespondovalo s jednoduchostí aplikace), přidat předměty, které by měly dvě třídy společné, možnost spojit předmět u třídy s konkrétním časem v rozvrhu.

Zadání práce bylo splněno a v tomto posledním odstavci bych shrnul vše vůči tomuto zadání. Nastudoval jsem problematiku tvorby rozvrhu a prozkoumal jsem existující řešení. Díky tomu jsem mohl navrhnout a implementovat webovou aplikaci cílenou na základní a střední školy. Aplikace obsahuje profily učitelů a jejich časové dostupnosti, obsahuje profily učeben, včetně informací o jejich kapacitě. Dále obsahuje profily tříd, které lze dělit na skupiny, např. dle jazyků nebo tělesné výchovy. Lze vygenerovat více různých rozvrhů a tento rozvrh lze vidět z pohledu třídy, učitele či učebny. Navíc lze tento rozvrh vyexportovat do formátu pdf a okamžitě vytisknout. V aplikaci lze přepínat mezi dvěma jazyky: angličtinou a češtinou. Vzhledem k tomu, že aplikace je typu klient-server, tak zvládne obsloužit více klientů v jeden okamžik. Uživatelské testy byly provedeny a závěry z testů byly přidány do aplikace.

Literatura

[1] Pavel Mička. Třídy složitosti a turingovy stroje.
http://www.algoritmy.net/article/5774/Tridy-slozitosti.

[2] E. K. Burke and J. P. Newall. *A Multistage Evolutionary Algorithm for the Timetable Problem*, 1999.

[3] Sándor Györi, Zoltán Petres, and Annamária R. Várkonyi-Kóczy. Genetic algorithms in timetabling. a new approach.
http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/lecturas-heuristicos-optimizacion/TimetablingbyGAs.pdf?origin=publication_detail.

[4] Tomáš Müller. Interaktivní tvorba rozvrhu, 2001. Diplomová práce.

[5] Roman Barták. Constraint propagation and backtracking-based search.
http://ktiml.mff.cuni.cz/~bartak/downloads/CPschool05notes.pdf.

[6] Rudová Hana and Keith Murray. University course timetabling with soft constraints.
http://www.unitime.org/papers/patat03.pdf.

[7] Rakesh Kumar. Blending roulette wheel selection & rank selection in genetic algorithms, 2012.
http://www.ijmlc.org/papers/146-C00572-005.pdf.

[8] Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. *A genetic algorithm to solve the timetable problem*. Springer, softcover reprint of the original 1st ed. 1992 edition, 2011.

[9] John Dalton. Genetic algorithms, newcastle university, 2007.
http://www.edc.ncl.ac.uk/highlight/rhjanuary2007.php.

[10] Denny Hermawanto. Genetic algorithm for solving simple mathematical equality problem.
http://arxiv.org/ftp/arxiv/papers/1308/1308.4675.pdf.

[11] Mladen Jankovic. Making a class schedule using a genetic algorithm, 2008.
http://www.codeproject.com/Articles/23111/Making-a-Class-Schedule-Using-a-Genetic-Algorithm.

[12] Tomáš Müller and Roman Barták. Interactive timetabling: Concepts, techniques, and practical results.
http://www.unitime.org/papers/it02_ctp.pdf.

[13] Pavel Jaroš. Tvůrce rozvrhů.
http://rozvrhy.jaros.in.

[14] Applied Software Consultans aSc. asc rozvrhy.
http://www.asctimetables.com/.

[15] Liviu Lalescu. Free timetable software, 2002-2014.
http://lalescu.ro/liviu/fet.

[16] Abdullah Abouzekry. Localizing php applications the right way, 2011.
http://www.sitepoint.com/localizing-php-applications-1/.