VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TVORBA ROZVRHŮ POMOCÍ GENETICKÝCH ALGORITMŮ

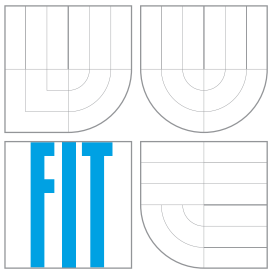
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ALEŠ HORKÝ

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TVORBA ROZVRHŮ POMOCÍ GENETICKÝCH ALGORITMŮ

CREATING TIMETABLES USING GENETIC ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ALEŠ HORKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MILOŠ MINAŘÍK

BRNO 2012

Abstrakt

Tato bakalářská práce obsahuje návrh a implementaci dvoufázového genetického algoritmu určeného pro tvorbu rozvrhů na základních školách. Algoritmus je vytvořen tak, aby bez snížení obecnosti výpočtu co nejvíce zredukoval stavový prostor řešeného problému. Implementovaný program v jazyce C++ je použitelný pro tvorbu rozvrhů na menších a středních školách.

Abstract

This bachelor thesis contains design and implementation of two-phase genetic algorithm intended for creating timetable schedules at primary schools. The algorithm is designed for maximum reduction of state space of solved problem without decrease of its universality. The implementated program in C++ language is applicable for creating timetable schedules at small and medium sized schools.

Klíčová slova

genetický algoritmus, umělá inteligence, tvorba rozvrhů, dvoufázový algoritmus, hybridní algoritmus, základní škola, c++

Keywords

genetic algorithms, artificial intelligence, timetable scheduling, two-phase algorithm, hybrid algorithm, primary school, c++

Citace

Aleš Horký: Tvorba rozvrhů pomocí genetických algoritmů, bakalářská práce, Brno, FIT VUT v Brně, 2012

Tvorba rozvrhů pomocí genetických algoritmů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Miloše Minaříka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Aleš Horký
14. května 2012

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Milošovi Minaříkovi za vřelý a trpělivý přístup při pomoci s tvorbou mojí prací. Dále bych chtěl vyjádřit poděkování paní Mgr. Věře Maříkové za předání zkušeností získaných při vytváření rozvrhů na základní škole. V neposlední řadě bych rád poděkoval rodičům a přítelkyni za projevenou podporu a trpělivost.

Obsah

Kapitola 1

Úvod

Problematika tvorby časových plánů je obecně velmi obtížná a komplexní. Problém vytváření časových plánů vyžaduje vybírat takové kombinace konečných zdrojů umístěných do časových oken, které nabídnou optimální rozdělení činností z hlediska nákladů, času, lidského pohodlí a podobně. Obtížnost tohoto úkolu spočívá především ve velmi rychlém nárůstu možných variant řešení při zvyšování počtu zdrojů a časových oken – tzv. *exploze stavového prostoru*. Ani pro relativně malé problémy tedy nelze v rozumném čase prozkoumat všechny možné varianty [7].

Pro většinu časových plánů naštěstí není nutné hledat optimální řešení. Často stačí, pokud se podaří najít dostatečně dobré řešení tzv. *sub-optimální řešení* [7]. K hledání sub-optimálního řešení lze použít různých optimalizačních metod, například stochastické metody založené na evolučních algoritmech, horolezecké algoritmy nebo specializované plánovací algoritmy [12].

1.1 Cíl práce

Cílem této práce je navrhnout a vytvořit program, který usnadní tvorbu školních rozvrhů na základních (případně středních) školách. Tento program je vytvořen objektově v jazyce C++, avšak ve výpočetně náročných částech je z důvodu optimalizace v některých případech od objektového zapouzdření upuštěno.

Před jeho spuštěním je nutno definovat základní entity (*konečné zdroje*) jako jsou vyučující, předměty, učebny nebo třídy a nastavit jejich parametry. Po spuštění se program pro tyto entity pokusí vygenerovat sub-optimální řešení rozvrhu, ve kterém bude splněno co nejvíce z nastavených parametrů. Tyto parametry jsou v práci dále nazývány *preference*[1], protože tento název lépe vystihuje jejich účel a navíc umožňuje v dalším textu rozlišit parametry základních entit a vlastní parametry výpočtu genetického algoritmu (jako příklad lze uvést parametr pravděpodobnosti mutace). Výsledné řešení musí být pro zadané entity platné[2].

Program vytvořený spolu s touto prací, by mohl vypomoci především v těch školách, které pro každoroční vytváření rozvrhů nepoužívají automatizovaný software. Zde vznikají

[1] Preference odpovídají měkkým omezením. Více v kapitole 2.

[2] Platnost rozvrhu znamená, že splňuje všechny zadaná tvrdá omezení. Viz kapitola 2.

rozvrhy, s využitím různých *papírkových metod*, po několikadenní ruční práci rozvrhářů[3]. Takto vzniklý rozvrh je sub-optimální z hlediska požadovaných (často špatně dohledatelných) preferencí. Výsledkem této práce by měl být takový program, který se takto vznikajícím rozvrhům vyrovná a dokáže nabídnout uživateli možnost výběru z více variant. Výstupem programu je rozvrh generovaný ze tří pohledů – rozvrhy pro učitele, učebny a třídy. Tyto tři pohledy jsou sémanticky ekvivalentní. Na každém z nich lze však pozorovat splnění jiných preferencí.

Protože různé časové plány mají své specifické vlastnosti, neexistuje obecný algoritmus, který by dokázal generovat libovolný časový plán a byl přitom dostatečně efektivní [7]. Z toho důvodu budou při návrhu a následné implementaci voleny takové cesty k řešení, aby výsledný program co nejlépe pracoval při generování rozvrhů pro cílovou skupinu základních škol (pokud se v této práci dále mluví pouze o *rozvrhu* příp. *školním rozvrhu*, implicitně se tím myslí *školní rozvrh na základní škole*). Neznamená to ale, že nebude použitelný například pro střední školy. Je však možné, že bude podávat horší výsledky z důvodu nutnosti rozdílného přístupu pro žáky různého věku[4].

Výstupem práce je program využívající dvoufázový hybridní genetický algoritmus pro generování rozvrhů se zaměřením na základní školy. Vlastnosti tohoto programu jsou následně porovnány s vlastnostmi podobných programů na trhu.

1.2 Struktura dokumentu

V první části práce (kapitoly 2 a 3) jsou uvedeny obecné problémy spojené s tvorbou časových plánů. Poté jsou detailně popsány požadavky na tvorbu školních rozvrhů. Požadavky jsou uváděny z pohledu školy, učitele a žáka. Dále jsou představeny metody, které umožňují řešit tyto plánovací problémy – především genetické algoritmy.

V druhé části práce (kapitola 4) je představen návrh aplikace a jsou navrženy metody pro generování rozvrhů spolu s popisem reprezentace rozvrhu v počítači. Také je navržena *účelová funkce* a její převod na tzv. *fitness funkci*.

V poslední části práce (kapitoly 5, 6 a 8) je uvedeno testování navržených řešení a jeho vyhodnocení. Podle výsledků testů je zvolena implementace aplikace, která je následně popsána.

[3] Velké množství škol využívá software, který rozvrh přímo nevytváří, ale při jeho tvorbě asistuje. Takový program umožňuje grafické znázornění rozvrhu, hlídá kolize mezi předměty a v některých případech obsahuje deterministický algoritmus pro generování rozvrhu. U nás je běžným příkladem takového softwaru program Bakaláři [21, 3]. Ten pro vytváření rozvrhu využívá algoritmus *forward-checking* bez navracení – konflikty řeší uživatel.

[4] U mladších žáků je například kladen větší důraz na dodržování pravidelných návyků a je nevhodné, aby se uprostřed výuky vyskytovala volná hodina.

Kapitola 2

Problematika tvorby časových plánů

Na problém tvorby časových plánů se dá pohlížet také jako na problém zabývající se efektivní distribucí zdrojů do časových oken. Pro tyto zdroje převážně platí, že jejich množství je konečné a pro možnost jejich použití existuje mnoho omezení. Uvažujeme omezení dvojího typu:

- tvrdá omezení *(hard-constraints)* - Všechna omezení tohoto typu musí být splněna (jejich vyhodnocením pro konkrétní rozvrh je binární hodnota), aby bylo řešení považováno za platné. V našem případě školních rozvrhů se může jednat například o omezení, že konkrétní učitel smí učit pouze v jedné třídě v daném čase. Některá z těchto omezení jsou v našem případě dána fixně a není možné, aby je uživatel měnil, jiná si může nastavit, případně vypnout. Detailní popis tvrdých omezení je uveden v kapitole 2.1.1.

- měkká omezení *(soft-constraints)* - Omezení tohoto typu ovlivňují hodnotu fitness funkce (jejich vyhodnocením pro konkrétní rozvrh je reálná hodnota). Čím více jsou měkká omezení pro konkrétní řešení splněna, tím je toto řešení kvalitnější. V případě školních rozvrhů se měkká omezení nazývají preference a uživatel je před spuštěním výpočtu může nastavit – např. učitel si přeje rozdělení hodin v blocích, aby mu nezůstávaly v průběhu vyučování volné hodiny. Detailní popis preferencí je uveden v kapitole 2.1.2.

Bylo dokázáno, že většina plánovacích problémů spadá do kategorie NP-těžkých, není možné je tedy vyřešit v polynominálním čase deterministickým algoritmem a o jejich řešení nelze (bez znalosti všech ostatních řešení) říci, zda je nejlepší [15]. Stavový prostor D možných řešení je daný všemi kombinacemi kartézského součinu zdrojů rozdělených do daných časových oken [12].

Prostor D je pro většinu těchto problémů tak rozsáhlý, že jeho prohledávání hrubou silou nenalezne optimální řešení v rozumném čase [7]. Pro tento stavový prostor platí, že obsahuje mnoho lokálních optim (říkáme, že je multimodální). U těchto lokálních optim se bez znalosti řešení nedá jednoduše určit, zda se pro ně hodnota fitness funkce blíží globálnímu optimu [17].

Pro řešení problémů tvorby časových plánů je proto nutno využívat metody, u kterých sice není zaručeno, že naleznou optimální řešení, je u nich však předpoklad, že naleznou řešení, které se bude optimálnímu blížit – tzv. sub-optimální řešení[7]. Tyto metody by

však neměly brzy uváznout v některém z lokálních optim, ale po celou dobu běhu by se měly snažit hledat vylepšení stávajícího řešení. Musí si tedy umět poradit s multimodalitou stavového prostoru [17].

Většina těchto metod vyžaduje existenci účelové funkce definované nad stavovým prostorem D, která určitým způsobem (závislým na daném problému), umí ohodnotit kvalitu každého řešení z D. Pro optimální řešení se snažíme najít takový prvek $x_{opt} \in D$, pro který má účelová funkce f minimální hodnotu, hledáme tedy:

$$x_{opt} = \underset{x \in D}{\operatorname{argmin}} f(x). \tag{2.1}$$

Z účelové funkce je odvozená tzv. *fitness funkce* se kterou již přímo pracují metody uvedené v předchozím odstavci [12].

2.1 Generování školních rozvrhů

Tvorba rozvrhů je zajímavá z hlediska složitosti, protože pro řešení těchto problémů není v dnešní době možné použít přímočaré algoritmy. Je obtížné jasně odlišit suboptimální řešení od neoptimálních, ale také i platná řešení od nepřípustných. Všech možných řešení je navíc nepřeberné množství (viz kapitola 2.1.3) [23].

Při generování rozvrhů existují tři základní entity[1], které je potřeba co nejlépe rozdělit do předem pevně daných časových bloků – jsou to třídy, učitelé a učebny. Pro všechny tyto entity existují pravidla, jak smí být umístěny (tvrdá omezení) a jak by měly být co nejvhodněji umístěny (měkká omezení). Tato omezení jsou popsána v následujících dvou podkapitolách.

Všechna dále uvedená omezení neberou v úvahu vyučovací hodiny volitelných předmětů jako jsou náboženství nebo například konverzace v cizím jazyce. Výuka těchto předmětů se většinou odehrává v ranních nebo odpoledních hodinách a jsou většinou organizovány vyučujícími individuálně podle jejich potřeb.

2.1.1 Tvrdá omezení (hard-constraints)

V uvedeném seznamu jsou vypsána všechna tvrdá omezení, která jsou relevantní pro školní rozvrhy a která vyplývají z běžných zvyků na školách nebo z vyhlášek Ministerstva školství.

- fixní (nenastavitelná) - Jsou dána řešeným problémem. Bez jejich platnosti není v praxi možné rozvrh dodržet:
  1. Každý vyučující smí v jednom časovém okně učit nejvýše v jedné učebně.
  2. V každé učebně smí v jednom časovém okně učit nejvýše jeden vyučující.
- proměnná (nastavitelná) - Jsou dána Ministerstvem školství, nebo běžnými zvyky ve školách:
  3. Všem třídám musí začínat každý den výuka v daném časovém okně (první vyučovací hodinu).
  4. V rozvrhu nemůže mít žádná ze tříd volnou hodinu, pokud by po této volné hodině následovalo ještě vyučování.

[1] Z pohledu tvorby časových plánů jsou entity jednotlivými zdroji, které mají být rozdistribuovány.

5. Každá třída musí mít každý vyučovací den minimálně X hodin a maximálně Y. Běžně se X rovná čtyřem hodinám a Y sedmi hodinám.
6. V průběhu vyučovacího týdne musí v každé třídě proběhnout právě tolik hodin daného typu, kolik zadal uživatel.
7. Dělené hodiny (cizí jazyky, tělocvik, ...) mohou vyžadovat, aby probíhaly paralelně ve více třídách.
8. Předměty s dotací více hodin týdně mohou vyžadovat, aby byly umístěny za sebou (výtvarná výchova). U jiných předmětů naopak musí výuka probíhat nejvýše jednu hodinu denně (matematika).

2.1.2 Měkká omezení (soft-constraints)

V uvedeném seznamu jsou vypsána taková měkká omezení, která ovlivňují kvalitu vytvořeného rozvrhu z pohledu školy (učeben), vyučujících a žáků. Tato omezení mohou jít často proti sobě, proto by mělo být umožněno nastavení vah podle důležitosti, jakou jim uživatel přikládá (nastavení nulové váhy umožní daný požadavek ignorovat).

- <u>škola</u> (učebny) - většina těchto omezení má ekonomický pohled:
  1. Výuku rozdělit do minima učeben (v nevyužívaných učebnách není nutný tak častý úklid ani vytápění a lze je v průběhu roku využít k jiným účelům).
- <u>vyučující</u> - tato omezení mohou být rozdílná pro každého vyučujícího a měla by být nastavitelná jak globálně, tak individuálně:
  2. Většina vyučujících si přeje mít vyučování v ucelených blocích, aby nevznikaly volné hodiny v průběhu dne (někteří, z důvodu přípravy na hodinu, však mohou chtít opak).
  3. Vyučující s menším než celým úvazkem si mohou přát, aby měli vyučování jen některé konkrétní dny.
- <u>žáci</u> - většina těchto omezení má pedagogický důvod:
  4. Rozvrh s vyrovnanou délkou vyučování (ideálně v pátek kratší výuka).
  5. Náročnější hodiny (matematika, čeština, cizí jazyk, ...) v ranních hodinách a naopak volnější hodiny (výtvarná výchova, tělocvik, ...) v odpoledních.
  6. Předměty, které jsou vyučovány vícekrát za týden, je vhodné rozdělit v celém vyučovacím týdnu (je nevhodné mít cizí jazyk s třemi vyučovacími hodinami v pondělí, úterý a ve středu).

2.1.3 Odhad velikosti stavového prostoru

Pro lepší představu o stavovém prostoru řešené úlohy byl vytvořen hypotetický příklad požadavků běžné základní školy. Pokud budeme uvažovat základní školu s V vyučujícími, U učebnami a T třídami, na které se vyučuje až v H hodinových blocích, D dnů v týdnu, je počet všech možných (i nepřípustných) řešení R dán výrazem:

$$R \;=\; (V \;\times U \;+1)^{T\times H\times D} \;\approx\; 6,95\times 10^{1761}. \tag{2.2}$$

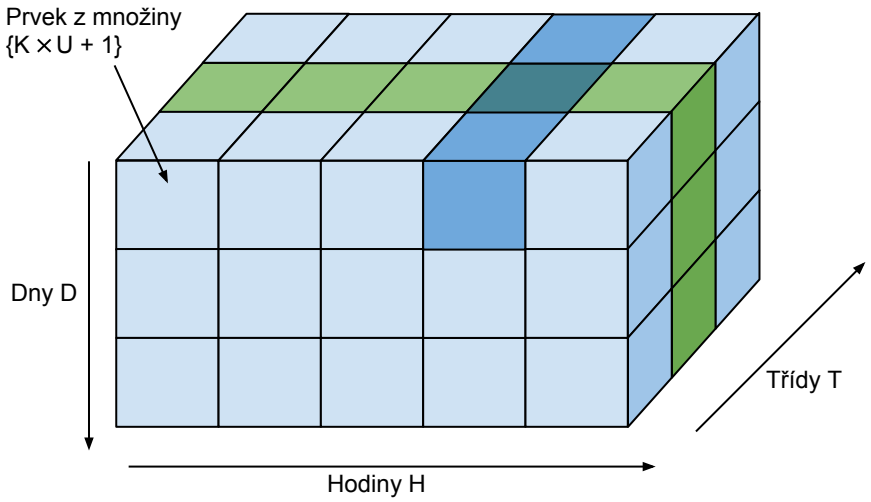
Rovnice 2.2 se skládá ze členu $(V \times U \ + 1)$, který vyjadřuje kardinalitu množiny obsahující všechny možné kombinace vyučujících a učeben. Navíc je přičtena hodnota 1, která vyjadřuje volnou hodinu. Umocnění tohoto členu hodnotou $T \times H \times D$, která udává počet uvažovaných časových oken, dostaneme horní odhad počtu možných řešení – stavový prostor uvažovaného problému.

Názornější představu o rozsahu tohoto stavového prostoru lze docílit dosazením do rovnic 2.2 a 2.3 hodnoty reprezentující běžnou základní školu, které jsou uvedeny v tabulce 2.1. Zvolené hodnoty vychází z nařízení vlády č. 75/2005 Sb. [14] a Rámcového vzdělávacího programu [16].

počet tříd	$T = 18$ (tj. 2 třídy na ročník)
počet učeben	$U = 25$
počet vyučujících	$V = 25$
maximální počet hodin denně	$H = 7$
počet vyučovacích dnů	$D = 5$

Tabulka 2.1: Hypotetické hodnoty reprezentující běžnou základní školu.

Grafické znázornění časových oken jako prvků kartézského součinu $T \times H \times D$ lze nalézt na obrázku 2.1, který zobrazuje část takto vzniklé třídimenzionální matice. Prvky v jednotlivých rovinách kolmých na osu T odpovídají rozvrhu pro konkrétní entitu (třídu). Příklad jednoho takového rozvrhu je na obrázku 2.1 znázorněn zelenou barvou.



Obrázek 2.1: Časová okna jako prvky kartézského součinu.

Přibližný horní odhad počtu platných řešení R_P, s ohledem na jedno tvrdé omezení z kapitoly 2.1.1, popisuje rovnice 2.3. Jedná se o toto tvrdé omezení:

- V daném čase smí každý vyučující učit nejvýše v jedné učebně a zároveň v jedné učebně smí probíhat také nejvýše jedna vyučovací hodina.

Stavový prostor určený rovnicí 2.3 se poněkud liší od prostoru určeného rovnicí 2.2. Z pohledu rovnice 2.3 se stavový prostor skládá z dvoudimenzionální matice časových oken $H \times D$, jejichž prvky jsou vektory složené z prvků kartézského součinu $V \times U$. Jeden z těchto vektorů je na obrázku 2.1 zvýrazněn tmavě modrou barvou.

Tyto vektory lze zkoumat z hlediska jejich obsahu. Díky tomu je možné sestavovat pouze takové vektory, které splňují výše uvedené tvrdé omezení. Skládání vektorů odpovídá v rovnici 2.3 součin $\prod_{i=0}^{T-1}(V-i)\times(U-i)+1)$[2], ten vytváří jednotlivé vektory jako dvoudimenzionální kombinace vyučujících a učeben bez opakování. Takto vytvořené vektory jsou potom umísťovány na všechny možné pozice (umocnění hodnotou $H \times D$).

$$R_P \;=\; (\prod_{i=0}^{T-1}((V-i)\times(U-i)+1))^{H\times D} \;\approx\; 3,95\times 10^{1505} \tag{2.3}$$

Přestože je v rovnici 2.3 zahrnuto pouze jedno tvrdé omezení, lze vidět, že oproti rovnici 2.2 snížila rozsah stavového prostoru o více než 250 řádů. I tak však v praxi není možné v konečném čase projít celý stavový prostor tohoto hypotetického zadání. Při výpočtu je navíc, před vytvořením konkrétního řešení, velmi obtížné určit, zda je toto řešení platné, či nikoli [23].

Řešení problému navržené v kapitole 4 vychází právě z výše uvedené úvahy.

[2]Součin z rovnice 2.3 lze pro výpočet použít pouze v případě, že celkový počet učitelů a počet učeben je vyšší nebo roven počtu tříd – tento požadavek však musí z principu splňovat každá škola.

Kapitola 3

Genetický algoritmus

Genetické algoritmy patří mezi základní stochastické optimalizační algoritmy s evolučními rysy. Jedná se o tzv. *informované metody* – řešení problému tedy neprobíhá zcela náhodně, ale velký vliv na volbu následujícího kroku má u nich znalost optimalizované funkce (*fitness funkce*). Jejich základní výhodou je, že se zakládají na optimalizaci celé populace chromozomů. Běžné optimalizační metody pracují pouze s jedním objektem – z pohledu genetických algoritmů jedním chromozomem.

Ukazuje se, že mnoho praktických problémů lze řešit na základě analogie s procesy, probíhajícími v biologických systémech[1]. Genetické algoritmy jsou převážně založeny na Darwinově teorii evoluce [6] sestávající z procesů jako je dědičnost, mutace, přirozený výběr, křížení a dalších. Podle Darwinovy teorie, probíhá evoluce na základě selekce vycházející z úspěšnosti rozmnožování jedince. Teorie vychází z předpokladu, že jedinci α v aktuální populaci P_t s nějakou dobrou vlastností budou mít více potomstva, kterému tuto vlastnost mohou předat. V následném potomstvu P_{t+1} bude tedy více jedinců α s těmito dobrými vlastnostmi a jedinci v ní tak budou minimálně stejně kvalitní. Tuto tezi popisuje rovnice 3.1;

$$\max_{\alpha \in P_{t+1}} F(\alpha) \geq \max_{\alpha \in P_t} F(\alpha) \tag{3.1}$$

Pokud posloupnost populací $P_0, P_1, \ldots, P_N$ splňuje pro fitness funkci F rovnici 3.1, potom algoritmus pro $t \to \infty$ dosáhne globální optimum α_{opt}:

$$\alpha_{opt} = \lim_{t \to \infty} (\arg \max_{\alpha \in P_t} F(\alpha)) \tag{3.2}$$

V našem případě jsou jedinci v populaci jednotlivá řešení problému. Kvalita řešení (míra přežití) je pro ně dána jejich hodnotou fitness funkce tzv. *fitness jedince/chromozomu.*

Obecný genetický algoritmus se skládá z populace obsahující jednotlivá řešení problému, která nazýváme chromozomy. Chromozom reprezentuje informační obsah jedince. V populaci probíhá mezi chromozomy reprodukce a vytváří se tak následující populace. Stará populace se zahazuje – vymírá. Tento postup můžeme chápat jako evoluci populace v každém kroku algoritmu.

Proces reprodukce chromozomů se skládá ze dvou základních operací, jedná se o operátory křížení a mutace. Při křížení se z rodičovských chromozomů (pseudonáhodně vybraných s ohledem na jejich fitness) určitým způsobem vytváří potomci. Přestože operátor křížení má často stochastický charakter, tak jenom jeho použití při vytváření následující populace

[1] Slovníček dále užívaných pojmů z oblasti genetických algoritmů, které jsou převzaty z biologie, lze nalézt v příloze A.

není vhodné. Samotné křížení chromozomů může mít tendenci uváznout s řešením v lokálním extrému optimalizované funkce. Z toho důvodu se po operaci křížení na některé[2] potomky aplikuje navíc operátor mutace. Ten spočívá v náhodné změně jednoho či více genů nově vzniklého potomka. Může tak vzniknout ještě lepší jedinec, který má možnost ovlivnit celou populaci. Častěji však vnikají horší jedinci, kteří jsou však díky výběru chromozomů ke křížení brzy eliminováni. Podle typu řešeného problému je možné navrhnout a použít i další genetické operátory, které nějakým způsobem poupraví chromozom tak, aby dosáhl lepší hodnoty fitness.

Vznik nového jedince v následující populaci začíná tím, že se ze stávající populace pseudonáhodně vyberou minimálně dva chromozomy v závislosti na jejich fitness a podle nich se za pomoci genetických operátorů vytvoří nové chromozomy v následující populaci. Nové chromozomy jsou buď totožné se starými, a nebo vzniknou křížením vybraných chromozomů. Která z variant reprodukce se použije, je opět náhodné. Pokud noví jedinci vzniknou křížením, je na ně ještě případně aplikován operátor mutace.

S malou optimalizací původního algoritmu se do nové populace mohou dostat i jedinci ze staré populace bez křížení, tzv. *elitní jedinci*. Jedná se o několik málo jedinců (často i pouze jednoho) s nejvyšší hodnotou fitness. Ti jsou přeneseni do následující populace beze změny (křížení s nimi však samozřejmě pořád probíhá). Díky tomu je zajištěno, že neztratíme nejlepší jedince, o které bychom v průběhu obecné evoluce mohli přijít. Zvláště v malých populacích může být tato ztráta velmi citelná [10, 12].

3.1 Kódování chromozomů

Vhodné zakódování chromozomu je jedním z nejtěžších úkolů při návrhu genetického algoritmu pro daný problém. Zakódování ovlivňuje podstatnou část dalšího návrhu, ať už jde o výpočet fitness funkce, nebo o implementaci genetických operátorů. Naším úkolem je tedy najít takové vhodné kódování chromozomů, které vyhovuje následujícím často protichůdným požadavkům:

1. Rychlé vyhodnocení fitness funkce a genetických operátorů pro daný chromozom.
2. Jednoduchá konverze genotypu na fenotyp.
3. Možnost jednoduše určit platnost řešení daného chromozomem.
4. Řešení chromozomů s podobným obsahem by si měla být také podobná. Naopak chromozomy s rozdílným obsahem by měly vést na rozdílná řešení[3].
5. Malá paměťová náročnost pro uložení chromozomu.

Jako běžné metody kódování chromozomů lze uvést binární, grafové (většinou stromy), permutační, kódování pomocí messy-chromozomů a další. Často se však můžeme setkat i s návrhy kódování, které jsou přímo určeny pro řešení konkrétního problému, avšak mohou vznikat úpravou obecnějších kódovacích technik. Taková kódování využívají hybridní genetické algoritmy, jež jsou navrhovány přímo pro daný problém a jejich kódování buď vychází z tradičních algoritmů pro řešení tohoto problému, nebo je navrhnuto s maximální snahou o využití znalosti daného problému. Některá z uvedených kódování a problémy spojené s nimi jsou rozebrány v následujících podkapitolách [10, 12].

[2] Četnost použití operátoru mutace silně závisí na řešeném problému. Běžně se operátor mutace aplikuje na přibližně 1% nově vzniklých potomků.

[3] Například v případě binárního kódování řeší korelaci genotypu a fenotypu tzv. *Grayův kód*.

3.1.1 Binární kódování

Nejběžnější kódování chromozomů je bezpochyby binární kódování. V nejjednodušším případě, pokud se řešení problému dá převést na hledání číselných vstupů fitness funkce, se může jednat o binární reprezentaci vektoru číselných proměnných, se kterou se pracuje bitově. Tyto číselné vstupy mohou být obecně spojité, na číslicovém počítači je však nutné je vždy diskretizovat.

Při použití klasického binárního kódování může docházet k situaci, kdy binární vektory lišící se ve všech pozicích bitových proměnných, mohou odpovídat dvěma sousedním řešením. Příkladem takových řešení mohou být čísla 15 (`01111`) a 16 (`10000`). Takovýto skok v kódování chromozomu se nazývá *Hammingova bariéra*. Z pohledu genetického algoritmu je tato vlastnost kódování nežádoucí, protože v případě, že hledané optimální řešení leží za bariérou, může být její překročení z pohledu genetických operátorů značně nepravděpodobné. Například v případě použití operátoru mutace očekáváme, že výsledkem operace bude nově vzniklý náhodně pozměněný chromozom, který se však přesto s velkou pravděpodobností bude podobat původnímu chromozomu. U klasického binárního kódování však není v žádném případě zaručeno, že každým dvěma blízkým chromozomům odpovídají také blízká řešení. Pokud by genetický algoritmus pracoval s chromozomy, mezi kterými by vznikala Hammingova bariéra, byl by jeho potenciál najít optimální řešení značně degradován.

Řešením problému Hammingovy bariéry je například použití tzv. *Grayova kódování*. Pro toto kódování platí, že má Hammingovu vzdálenost 1[4]. Platí tedy, že takto zakódovaná sousední řešení se liší právě v jednom bitu. Díky korelaci genomu a fenotypu je možné provádět genetické operace přímo s chromozomy, a není tak nutné chromozomy dekódovat a znovu kódovat.

Druhou možností, jak řešit problém Hammingovy bariéry, je s touto vlastností kódování počítat již při navrhování genetických operátorů a explicitně ji zde ošetřit. V případě binárního kódování se může jednat o zavedení nového operátoru, a to tzv. *inverzního operátoru*, který invertuje bity chromozomu od náhodně zvolené pozice dále. Toto chování přímo odpovídá překročení Hammingovy bariéry v klasickém binárním kódování. Inverzní operátor je možné v genetickém algoritmu umístit mezi operace křížení a mutace. Je volán pouze na některé nově vzniklé chromozomy. Pravděpodobnost použití operátoru je, podobně jako u operátoru mutace, dosti nízká [12].

3.1.2 Vícehodnotové kódování

Jedná se o binární kódování s rozšířenou množinou alel. Jednotlivé geny tedy mohou nabývat více hodnot, může se jednat například o desetinná čísla s teoreticky nekonečnou množinou hodnot. Toto kódování umožňuje relativně snadno definovat smysluplné a konkrétnímu problému odpovídající genetické operátory. Díky tomu tato reprezentace umožňuje vyzkoušet velkou škálu operátorů a podstatným způsobem tak ovlivnit chování celého genetického algoritmu. Oproti binární reprezentaci je nevýhodou větší výpočetní náročnost (hlavně při použití reálných čísel) [10].

Jako příklad takového kódování může být třeba výpočet vektoru vah pro klasifikátor. V tomto případě lze použít velké množství operátorů křížení (aritmetický/geometrický průměr alel, použití alely z prvního nebo druhého předka, ...) i operátorů mutace (náhodná změna hodnoty genu, prohození dvou alel, ...).

[4]Hodnota Hammingovy vzdálenosti udává maximální počet lišících se bitů dvou sousedních řešení (kódových slov).

3.1.3 Permutační kódování

Permutační kódování se často hodí právě pro plánovací úlohy – záleží zde na pořadí prvků, které se nesmí opakovat. To přímo odpovídá řešenému problému, ve kterém hledáme permutaci jednotlivých hodin v rozvrhu, viz kapitola 2.1.3. Avšak je možné je použít i pro známější problém obchodního cestujícího, kolem jehož řešení právě pomocí permutačního kódování existuje rozsáhlá teorie. Rozdílnou, avšak také kombinatorickou NP-úplnou úlohou je například problém dvou loupežníků. Ukázky řešení těchto dvou úloh lze nalézt v [12] a detailně popsaný problém obchodního cestujícího v [5].

Permutační úlohy byly jedny z prvních úloh, kde začaly genetické algoritmy slavit úspěch. Řešení těchto problémů je obvykle velmi těžké z toho důvodu, že při permutaci o velikosti N existuje $N!$ různých řešení. V důsledku toho často není možné pro větší permutace deterministicky projít všechna řešení.

Vlastní reprezentace chromozomu permutačního kódování může být shodná s chromozomem vícehodnotového kódování, avšak buď se liší genetické operátory, u kterých je zaručeno, že z permutace vytváří permutaci[5], nebo se použijí klasické operátory a z jejich výsledku je permutace dodatečně vytvořena. Algoritmus dodatečné opravy je často silně závislý na znalosti řešeného problému [12, 15].

3.2 Fitness funkce

Pod pojmem fitness funkce rozumíme kvantitativní míru schopnosti jedince přežít a vstupovat do reprodukčního procesu. Biologický ekvivalent fitness funkce je zobrazení genotypu na fenotyp, kde chromozom kóduje organismus, jehož schopnost reprodukce a přežití je dána fitness daného chromozomu [12].

Fitness funkci F lze získat transformací z účelové funkce f, která, jak již bylo řečeno v kapitole 2, umí ohodnotit kvalitu řešení vybraného chromozomu α ze stavového prostoru D problému. Účelová funkce tedy reprezentuje prostředí – náš řešený problém. Pro vztah mezi F a f obecně platí:

$$\forall \alpha_1, \alpha_2 \in P : f(\alpha_1) \leq f(\alpha_2) \Rightarrow F(\alpha_1) \geq F(\alpha_2) \geq 0 \tag{3.3}$$

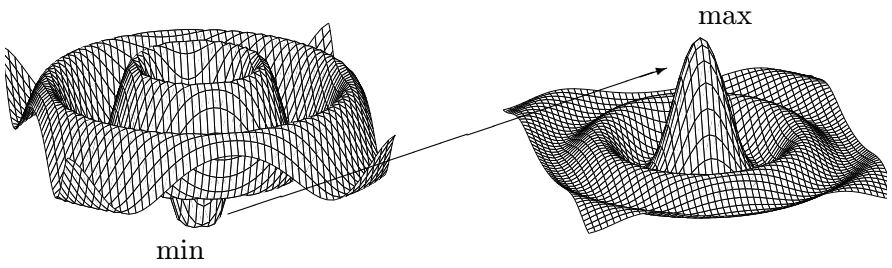
Z praktických důvodů je často navíc vhodné, aby všechny hodnoty fitness funkce F byly normalizovány do intervalu $(0; 1)$ a zároveň suma všech hodnot z populace byla rovna 1 [12]. Grafické znázornění tohoto zobrazení lze nalézt na obrázku 3.1.

Existuje mnoho metod zobrazení účelové funkce f na fitness funkci F tak, aby byla splněna podmínka 3.3. Jedním z možných přístupů je jednoduché lineární zobrazení účelových hodnot na fitness tak, že maximální (minimální) funkční hodnotě je přiřazena minimální (maximální) fitness, ostatní funkční hodnoty jsou lineárně interpolovány mezi těmito dvěma limitními body:

$$F(\alpha) = \frac{F_{max} - F_{min}}{f_{min} - f_{max}} f(\alpha) + \frac{f_{min}F_{min} - f_{max}F_{max}}{f_{min} - f_{max}} \tag{3.4}$$

Takovéto zobrazení je vhodné především v případě, kdy se hodnoty účelové funkce příliš nemění, nejsou tedy například rozdílné v jednotkách řádů. Pokud by tato podmínka nebyla dodržena, zobrazení by z pohledu genetického algoritmu přidělovalo neopodstatněně velké váhy dobrým řešením a naopak.

[5] Jako příklady permutačních operátorů lze uvést operátor křížení s částečným přiřazením (PMX), křížení se zachováním pořadí (OX) nebo mutace pomocí výměny. Uvedené operátory jsou popsány v kapitole 3.4.



Obrázek 3.1: Zobrazení účelové funkce (nalevo) na fitness funkci (vpravo) v prostoru všech možných chromozomů, které jsou přetransformovány do dvoudimenzionálního prostoru.

Řešením problému lineárního zobrazení je jiný přístup, kdy se uspořádají vzestupně chromozomy z populace podle účelové funkce. Chromozomu s hodnotou účelové funkce f_{max} (f_{min}) je poté přiřazena fitness F_{min} (F_{max}). Chromozomům, které se po seřazení vyskytují mezi okrajovými chromozomy, jsou poté rovnoměrně lineárně přiděleny fitness z intervalu $(F_{min}; F_{max})$.

Nevýhodou druhého přístupu může být, že přestože splňuje podmínku 3.3, tak přiřazuje rozdílnou fitness chromozomům se shodnou hodnotou účelové funkce. To může být problém především na konci běhu genetického algoritmu, kdy je populace značně homogenní. Je proto často vhodné modifikovat algoritmus tak, aby při přidělování fitness neokrajovým chromozomům tento fakt respektoval [12].

3.3 Strategie výběru jedinců k reprodukci

Při výběru jedinců z aktuální populace ke křížení a následnému vzniku nových jedinců do následující populace se snažíme simulovat přirozený výběr probíhající v přírodě tak, jak je popsán v Darwinově teorii o původu druhů. Přestože existuje mnoho různých strategií výběru jedinců, tak se všechny snaží dosáhnout stejného výsledku, kterým je propagace dobrých vlastností jedinců z jedné populace do následující a zároveň zachování dostatečné rozmanitosti této populace.

Pokud bychom následující populaci vytvářeli pouze z nově vytvořených jedinců, kteří vznikli křížením jedinců z předchozí populace, nebyla by úplně dodržena vlastnost fitness funkce z kapitoly 3.2. Ta by měla udávat kromě schopnosti jedince se reprodukovat, i jeho schopnost přežít do následující populace. Pokud chceme, aby fitness funkce tuto vlastnost přesto měla, lze to zařídit tak, že vybrané jedince k reprodukci s určitou mírou pravděpodobnosti nezkřížíme, ale jednoduše překopírujeme do populace následující. Tato přímá kopie běžně probíhá s pravděpodobností okolo 0,05-0,25 [10].

Nejběžnější metody výběru jsou uvedeny v následujících podkapitolách. Jedná se o ruletový a turnajový mechanismus selekce. Při podrobnějším porovnání obou dále uvedených metod bylo ukázáno, že při vhodném nastavení jejich parametrů, není ani jedna z nich výrazně lepší, avšak turnajový mechanismus vykazuje lepší časovou náročnost [9].

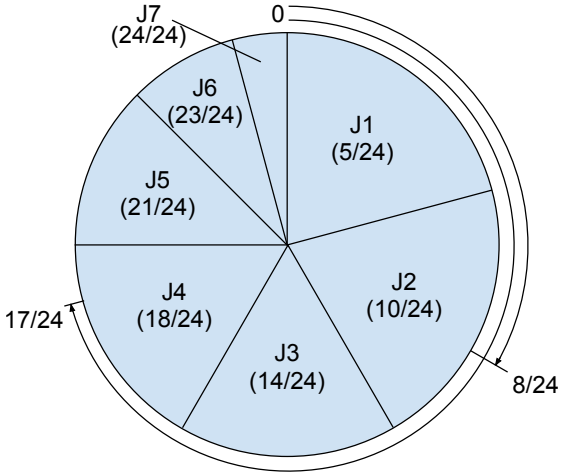
3.3.1 Ruletový mechanismus selekce

Ruletový mechanismus selekce je zřejmě nejčastěji používaný způsob implementace přirozeného výběru. Můžeme si ho představit jako běžnou ruletu s tím, že jednotlivé výseče

příslušící jedincům mají rozdílnou velikost úměrnou jejich fitness. Při roztáčení rulety jsou tak favorizováni lepší jedinci na úkor horších, a je pravděpodobné, že lepší jedinci stvoří více potomků.

Jednoduchá implementace ruletové metody je ukázána na obrázku 3.2. Při vytváření rulety se fitness sedmi jedinců (J1 až J7) populace normalizují tak, aby jejich součet byl jedna. Poté se použije součtové ohodnocení jedinců, podle nějž jsou přiřazeny výseče. Pro úplnost je vhodné poznamenat, že seřazení jedinců podle jejich fitness, které lze vidět na obrázku, není podmínkou správného fungování algoritmu.

Při spuštění selekce jsou vygenerována dvě náhodná čísla z intervalu $\langle 0; 1 \rangle$, které představují roztočení rulety. Podle výseče, do které tato dvě čísla spadnou, jsou vybráni dva jedinci ke křížení. V ukázce se jedná o náhodná čísla 8/24 a 17/24, která vyberou jedince J2 a J4. [10]



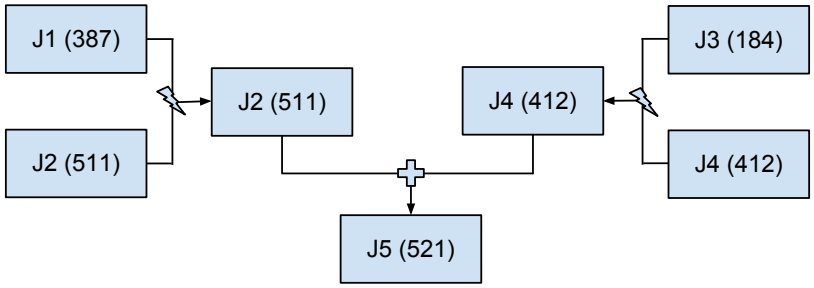
Obrázek 3.2: Selekce chromozomů pomocí ruletové metody.

3.3.2 Turnajový mechanismus

Turnajový mechanismus je selekční metoda silně inspirovaná procesy v živé přírodě, kde se jedinci v populaci musí navzájem utkávat v boji o přežití nebo v účasti na reprodukčním procesu. Metoda spočívá v tom, že se z populace náhodně vyberou minimálně dva jedinci, mezi kterými proběhne turnaj, který vyhrává jedinec s nejlepším ohodnocením. Tento jedinec se poté účastní reprodukčního procesu s druhým jedincem, který byl vybrán stejným způsobem. Čím více jedinců je do turnaje vybráno, tím vyšší vzniká selektivní tlak, proto se ve většině aplikací obvykle vybírají právě dva jedinci. Později byla navržena vylepšená verze turnajového mechanismu, při kterém automaticky nevyhrává nejsilnější z vybraných jedinců, ale je zde malá pravděpodobnost, že z turnaje může jako vítěz vyjít i slabší jedinec.

Z důvodu, že turnajový mechanismus výběru nevyužívá přímo hodnotu fitness jedince, ale spíše porovnává relativní pořadí jedinců, dá se tento způsob selekce úspěšně použít při snaze zabránit problému tzv. *předčasné konvergence*. Ten u jiných metod nastává v případě, kdy populace obsahuje jednoho či více zdatných jedinců, jejichž ohodnocení je výrazně vyšší než zbytku populace. Protože z takovéto populace budou nadprůměrní jedinci vybíráni velmi často do křížícího procesu na úkor slabších, tak se začne vytrácet rozmanitost populace a ta bude mít silnou tendenci uváznout v lokálním maximu.

Způsob selekce je názorně předveden na obrázku 3.3. Dva jedinci J1 a J2 a dva jedinci J3 a J4 jsou náhodně vybráni z populace a utkají se v souboji, který vyhrává vždy silnější z nich. Vítězům obou turnajů se dostane právo zúčastnit se reprodukčního procesu, vytvořit jedince J5 a přenést tak své geny do následující populace [10].



Obrázek 3.3: Selekce chromozomů pomocí turnaje.

3.4 Operátory křížení a mutace

Po operátorech křížení požadujeme, aby ze dvou a více řešení dokázaly vzít jejich „dobré“ geny a vytvořit řešení, které „dobrých“ genů obsahuje více. Operátory mutace mohou navíc takto vzniklé jedince náhodně upravit a zvýšit tak rozmanitost populace. Je vítané, aby tyto operátory vytvářely implicitně pouze platná řešení a my tak nemuseli výsledky těchto operátorů explicitně zahazovat, případně upravovat tak, aby z nich platná řešení vznikla.

Křížení a mutace jsou základní genetické operátory, bez kterých by genetický algoritmus postrádal své hlavní vlastnosti. Bez křížení, které vytváří z několika chromozomů nové, rodičům podobné chromozomy, bychom přišli o důležitou vlastnost genetického algoritmu, a to o optimalizaci celé populace chromozomů společně. Genetický algoritmus bez mutace, která je použita na některé nově vzniklé jedince, by měl zase velké předpoklady k uváznutí v některém z lokálních optim. Existují však i další genetické operátory, které řeší individuální problémy spojené s řešenou úlohou. Jako příklad dalších operátorů jmenujme třeba inverzní operátor zmíněný v kapitole 3.1.1.

V následujících podkapitolách jsou ukázány vybrané metody křížení a mutace, které jsou relevantní vzhledem k řešenému problému [10, 18] .

3.4.1 K-bodové křížení

V této podkapitole je popsán jednoduchý operátor křížení pro vektory pevné délky – tento operátor je tedy použitelný pro všechny 3 typy kódování uvedené v kapitole 3.1. Při použití permutačního kódování je však nutné navrhnout ještě opravný algoritmus, protože tyto operátory nezaručují dodržení platnosti permutace. Operátor funguje tak, že náhodně zvolí stejné body v obou chromozomech a části mezi těmito body vymění. Parametr k udává počet takto generovaných bodů.

Nejjednodušší varianta k-bodového křížení pro $k = 1$ se nazývá *jednobodový operátor křížení*. Jedná se o nejznámější operátor často používaný při binární reprezentaci. Algoritmus vygeneruje jeden náhodný bod křížení z intervalu $\langle 0; N - 1\rangle$ a v tomto místě rozdělí oba chromozomy délky N. Levou část z prvního a pravou část z druhé chromozomu spojí a vytvoří tak nový chromozom, druhý vytvoří obdobně ze zbylých částí:

$$\begin{array}{l}(\ 1,1,0 \mid 1,0,0,1,0\ )\\(\ 0,1,0 \mid 0,0,1,1,0\ )\end{array} \quad\Longrightarrow\quad \begin{array}{l}(\ 1,1,0 \mid 0,0,1,1,0\ )\\(\ 0,1,0 \mid 1,0,0,1,0\ )\end{array}$$

Obrázek 3.4: Jednobodové křížení binárních chromozomů.

Pro parametr k z intervalu $\langle 2; N-2\rangle$ je situace obdobná s tím rozdílem, že náhodných bodů se generuje k a geny z částí, které vzniknou mezi těmito body, jsou dvěma novým potomkům přidělovány střídavě z prvního a druhého předka:

$$\begin{array}{l}(\ 1,1,0 \mid 1,0 \mid 0,1,0\ )\\(\ 0,1,0 \mid 0,0 \mid 1,1,0\ )\end{array} \quad\Longrightarrow\quad \begin{array}{l}(\ 1,1,0 \mid 0,0 \mid 0,1,0\ )\\(\ 0,1,0 \mid 1,0 \mid 1,1,0\ )\end{array}$$

Obrázek 3.5: Dvoubodové křížení binárních chromozomů.

V případě, že parametr $k = N-1$, tedy že rozdělení rodičovských chromozomů bude probíhat mezi každými dvěma geny, mluvíme o *uniformním křížení*. V tomto případě se při vytváření potomka u každého genu rozhoduje, zdali se pro jeho hodnotu použije alela z prvního nebo druhého předka. Pravděpodobnost výběrů bývá ve většině případů 0,5 – není to však podmínkou:

$$\begin{array}{l}(\ 1 \mid 1 \mid 0 \mid 1 \mid 0 \mid 0 \mid 1 \mid 0\ )\\(\ 0 \mid 1 \mid 0 \mid 0 \mid 0 \mid 1 \mid 1 \mid 0\ )\end{array} \quad\Longrightarrow\quad \begin{array}{l}(\ 0 \mid 1 \mid 0 \mid 0 \mid 0 \mid 0 \mid 1 \mid 0\ )\\(\ 1 \mid 1 \mid 0 \mid 1 \mid 0 \mid 1 \mid 1 \mid 0\ )\end{array}$$

Obrázek 3.6: Uniformní křížení binárních chromozomů.

U předchozích variant k-bodového křížení je pravděpodobné, že budou v celistvější podobě zachovány genetické informace předků. Uniformní křížení bude mít spíše rozkladný vliv. Přesto existují aplikace, při kterých se právě tato vlastnost stává výhodou. Za zmínku také stojí varianta uniformního křížení, kde se první potomek vytváří tak, aby získal maximální ohodnocení mezi všemi variantami sourozenců, a druhý potomek tak, aby přinesl do následující populace co největší genetickou rozmanitost [18, 10].

3.4.2 Permutační křížení

Permutační operátory křížení jsou navrženy tak, aby jejich výsledkem byla znovu permutace (tedy aby se každá alela vyskytovala právě v jednom genu). Toho může být dosaženo buď dodatečnou úpravou potomka, nebo je celý operátor navržen tak, aby měl tuto vlastnost. Jednotlivé permutační křížící operátory se liší v tom, jaký způsobem zachovávají genetické informace z předků – mohou preferovat zachování absolutní pozice alel, zachování posloupnosti alel, nebo jinou vlastnost vhodnou pro řešený problém.

Jeden z nejznámějších permutačních operátorů je operátor *křížení s částečným přiřazení* (PMX - partially matched crossover). Ten je založen na dvoubodovém křížení, po kterém poté proběhne opravný algoritmus. Byl vyvinut se záměrem využít uspořádané podmnožiny permutace jednoho z rodičů a přitom zachovat pořadí a pozici co nejvíce genů z druhého rodiče. Princip fungování operátoru bude zřejmý z následujícího příkladu.

Mějme dva permutační chromozomy p_1 a p_2 s již zvolenými body křížení:

$$p_1 = (1,2,3 \,\big|\, 4,5,6,7 \,\big|\, 8,9) \qquad p_2 = (4,5,2 \,\big|\, 1,8,7,6 \,\big|\, 9,3)$$

Při vytváření potomků o_1 a o_2 se v prvním kroku vymění prostředí části chromozomů (krajové se nechají prozatím volné) a podle prvků, které se vzájemně vyměnily, se vytvoří

množina přepisovacích pravidel k takto:

$$o_1 = (\square, \square, \square \,\big|\, 1, 8, 7, 6 \,\big|\, \square, \square) \qquad o_2 = (\square, \square, \square \,\big|\, 4, 5, 6, 7 \,\big|\, \square, \square)$$

$$k = \{(1 \leftrightarrow 4), (8 \leftrightarrow 5), (7 \leftrightarrow 6), (6 \leftrightarrow 7)\}$$

V druhém kroku se z původních chromozomů vyplní ty krajové geny chromozomu, které se zatím v permutaci nevyskytují:

$$o_1 = (\square, 2, 3 \,\big|\, 1, 8, 7, 6 \,\big|\, \square, 9) \qquad o_2 = (\square, \square, 2 \,\big|\, 4, 5, 6, 7 \,\big|\, 9, 3)$$

Zbylé, prozatím nevyplněné geny se nyní doplní podle přepisovacích pravidel z množiny k tím způsobem, že kde se v původním genu vyskytovala jedna alela z přepisovacího pravidla, je tato nahrazena druhou alelou z její dvojice:

$$o_1 = (4, 2, 3 \,\big|\, 1, 8, 7, 6 \,\big|\, 5, 9) \qquad o_2 = (1, 8, 2 \,\big|\, 4, 5, 6, 7 \,\big|\, 9, 3)$$

Druhým často používaným operátorem je *operátor křížení se zachováním pořadí* (OX - order crossover), který spíše než o zachování úseků genů v permutaci se snaží o zachování pořadí genů. Tento operátor vyjme z prvního rodiče uspořádanou množinu genů. Alely, které se v této množině vyskytují, vymaže z druhého rodiče, a vybranou množinu z prvního předka do druhého vloží. Pro detailnější vysvětlení jeho činnosti uveďme znovu příklad:

Nechť dva rodičovské chromozomy jsou následující permutace p_1 a p_2:

$$p_1 = (a, b, c, d, e, f, g, h, i) \qquad p_2 = (h, g, d, i, a, b, e, f, c)$$

Zvolme dva body křížení m a n, které budou ohraničovat uspořádanou množinu genů s vyjmutou z prvního předka. Pro zvolené $m = 4$ a $n = 7$ se bude jednat o uspořádanou množinu:

$$s = (e, f, g)$$

Geny vyskytující se ve vybrané množině odstraníme z druhého rodiče:

$$p_2' = (h, d, i, a, b, c)$$

Po odstranění genů již nemůže dojít k jejich kolizi a tím i porušení permutace. Je tedy možné vybranou množinu genů z prvního rodiče vložit na příslušnou pozici do druhého rodiče. Dostaneme výsledného potomka o:

$$o = (h, d, i, a, e, f, g, b, c)$$

Jako poslední je uveden operátor *křížení založený na zachování pozice* (PBX - position based crossover). Ten se velmi podobá operátoru křížení se zachováním pořadí s tím rozdílem, že zde kopírovaná uspořádaná množina se neskládá z genů, které by nutně musely po sobě následovat. K přesnějšímu porozumění, jak operátor pracuje, je znovu uveden příklad:

Mějme dva stejné rodičovské chromozomy p_1 a p_2 jako u operátoru křížení se zachováním pořadí:

$$p_1 = (a, b, c, d, e, f, g, h, i) \qquad p_2 = (h, g, d, i, a, b, e, f, c)$$

Nyní náhodně vygenerujme binární masku l pro jednotlivé pozice v chromozomu:

$$l = (0, 1, 0, 1, 1, 0, 0, 1, 0)$$

Podle masky vybereme z prvního rodiče geny, které zůstanou na svých pozicích i v potomku o. Geny se stejnými alelami jsou naopak vyřazeny z druhého předka p_2:

$$o = (\square, b, \square, d, e, \square, \square, h, \square)$$

$$p_2' = (g, i, a, f, c)$$

Do volných pozic v potomkovi jsou nyní, podobně jako v případě operátoru křížení se zachováním pořadí, vloženy geny z druhého předka:

$$o = (g, b, i, d, e, a, f, h, c)$$

Existují ještě další permutační operátory (například cyklický operátor křížení – CX cycle crossover), které se však již z výčtu vlastností nezdají pro řešený problém rozvrhů tolik vhodné, a proto zde ani nejsou uváděny [10, 1].

3.4.3 Operátory mutace

Operátory mutace jsou již v principu v mnoha případech jednodušší než operátory křížení. Požadujeme, aby jedinec po mutaci byl blízký jedinci před mutací. Pro splnění toho požadavku často stačí lehce pozměnit genotyp jedince. Pro běžné kódování se nabízí jednoduchá možnost náhodné změny jednoho či více genů. U permutačních kódování je nejčastější náhodná výměna dvou genů v chromozomu [10].

3.5 Přechod do další populace

Důležitým aspektem při přechodu z jedné populace do následující je otázka, co udělat s předchozí populací. Volba řešení toho problému může ve výsledku ovlivnit rychlost konvergence algoritmu, rozmanitost jednotlivých populací, nebo zajistit, že nejlepší jedinci přežijí z libovolné populace do následující, a tak o nejkvalitnější řešení v průběhu evoluce nepřijdeme.

Nejjednodušší řešení je nechat předchozí populaci vymřít (zahodit). V tomto případě však ztratíme všechny informace o stavu předešlé populace (zůstává pouze poděděná genetická informace), navíc přicházíme o možnost korekce výsledku, pokud by se následující populace „nevyvedla“. Toto jednoduché řešení je použitelné v případě, že máme jistotu, že jedinci v každé následující populaci budou lepší než v té minulé. Tuto jistotu však většinou nemáme.

Možností, která se nabízí, je nechat přežít nejnadějnější jedince z aktuální populace do následující. Tato technika se nazývá *elitismus*. Před započetím křížení se vybraná elita přímo a beze změny překopíruje do následující populace, navíc se však také účastní reprodukčního procesu. Elitismus nám zajistí, že v průběhu evoluce nepřijdeme o nejlepší řešení problému. Na druhou stranu hlavně při menších populacích může zapříčinit snížení rozmanitosti populace a tím předčasnou konvergenci zbytku populace k elitám.

Jinou možností může být jedince z obou populací spojit a s pravděpodobností úměrnou fitness jedinců nechat ruletovým (příp. turnajovým) způsobem představeným v kapitole 3.3 slabší jedince vymírat tak dlouho, dokud velikost populace neklesne na požadovaný počet jedinců.

V některých případech je pro zvýšení rozmanitosti populace vhodné, pokud nově vykřížení jedinci jsou přidáni do následující populace pouze v případě, že by v této populaci byli unikátní. Touto technikou se dá také oddálit problém předčasné konvergence algoritmu k lokálnímu extrému [10, 1, 12].

3.6 Ukončení genetického algoritmu

Problém, kdy zastavit výpočet běžícího genetického algoritmu, je jednou z posledních důležitých otázek při jeho návrhu. Jako nejjednodušší způsob se nabízí výpočet ukončit ve chvíli, kdy dosáhne určitého počtu generací. Tento počet se však nastavuje před spuštěním výpočtu, a algoritmus tak nemůže upravovat dobu trvání výpočtu podle jeho stavu. Může se stát, že se algoritmus ukončí předčasně, a nedosáhne tak nejlepšího výsledku, nebo že naopak běží zbytečně dlouho, uvázlý v lokálním extrému.

Lepší variantou může být při výpočtu sledovat stav a hlavně rozmanitost populace. Pokud je převážná část jedinců v populaci velmi podobných, dá se usuzovat, že algoritmus uvázl v lokálním extrému, nebude již dále konvergovat a algoritmus je možné ukončit.

V případě použití genetických algoritmů na problém, u kterého dokážeme rychle určit, že jedinec splňuje všechny požadavky na očekávaný výsledek, je možné algoritmus ukončit v případě objevení takového jedince. Výše uvedené přístupy je možno samozřejmě kombinovat [12].

Kapitola 4

Návrh aplikace

Návrh programu pro vytvoření školního rozvrhu vychází z úvahy v kapitole 2.1.3. Zde je prezentována myšlenka na rozdělení výpočtu do dvou fází tak, aby se výrazně omezil stavový prostor problému. Podle této teze je v následujících podkapitolách navrženo takové řešení problému tvorby školních rozvrhů, aby se neomezila obecnost výpočetního algoritmu a aby se zároveň co nejvíce zredukoval zmíněný stavový prostor.

Návrh předpokládá pevné předem dané spojení učitel-třída-předmět pro všechny předměty, které mají být v rozvrhu. Pro ještě větší omezení stavového prostoru jsou z těchto vazeb vytvořeny skupiny. Ty sdružují dělené předměty, které je z důvodu splnění tvrdých omezení nutno vyučovat ve stejnou vyučovací hodinu souběžně v různých třídách. Nejčastěji se jedná o předměty výuky cizích jazyků, informatiky, tělocviku, a dalších. Takovouto skupinu již dále nelze rozdělit, a díky tomu není možné tato tvrdá omezení ve vygenerovaném rozvrhu jakkoli porušit.

Skupiny předmětů jsou v první fázi výpočtu umísťovány do navzájem nezávislých vektorů o délce dané počtem tříd. Grafické znázornění jednoho z nich lze nalézt na obrázku 2.1 tmavě modře. Počet použitých vektorů je dán násobkem počtu dnů a maximálního počtu vyučovacích hodin. Takovýto vektor reprezentuje vyučované předměty ve všech třídách v jednu vyučovací hodinu[1]. Konkrétní přiřazení vektorů k vyučovacím hodinám však probíhá právě až v druhé fázi výpočtu jako hledání nejvhodnější relace přiřazení mezi vektory a vyučovacími hodinami.

Pro obě fáze výpočtu jsou navrženy hybridní genetické algoritmy. První genetický algoritmus po dokončení první fáze předá nejlepší nalezené vektory následujícímu genetickému algoritmu do druhé fáze. Ten se je pokusí umístit do rozvrhu tak, aby byl výsledný rozvrh co nejlepší.

4.1 Formát vstupních požadavků

Program očekává vstupní data jako XML soubor, který obsahuje výčet vyučujících, tříd a učeben, které jsou navzájem provázány pomocí jednotlivých předmětů. Předměty jsou již ve vstupním souboru rozděleny do skupin. Do jedné skupiny patří předměty, které musí být vyučovány současně. Ve většině případů se jedná o půlené předměty, ale není to podmínkou. Z důvodu jednotnosti jsou i předměty, které nemají žádnou vazbu na další předměty, umísťovány do skupiny po jednom předmětu.

[1] Podle terminologii zavedené v kapitole 2 je pojem vyučovací hodina ekvivalentní pojmu časové okno.

Pro lepší představu je na obrázku 4.1 uvedena ukázková skupina pro půlený předmět angličtiny. Detailnější popis vstupního XML souboru lze nalézt v příloze B.

```
<group times="3" type="paralel" importance="1">
    <subject name="Aj" class="7" teacher="9" group="(5a1)" classroom="-1" />
    <subject name="Aj" class="7" teacher="8" group="(5a2)" classroom="16" />
</group>
```

Obrázek 4.1: Příklad skupiny předmětů ve vstupním XML souboru.

Každá skupina má tři povinné parametry, kterými jsou počet opakování (`times`), typ skupiny (`type`) a důležitost předmětů (`importance`). Počet opakování značí, kolikrát musí být daná skupina umístěna ve výsledném rozvrhu. Z logiky věci vyplývá, že platný rozvrh může být pouze takový, ve kterém jsou umístěny všechny skupiny v požadovaném počtu. Aby bylo možné vyhovět tomuto požadavku musí také platit, že v rozvrhu existuje dostatečný počet časových oken, do kterých mohou být všechny skupiny umístěny.

Parametr typ udává, jakým způsobem budou instance skupiny rozmístěny v rozvrhu[2]. Může nabývat buď hodnoty „paralelní“, nebo „sériový“. Sériová skupina obsahuje takové předměty, které musí ve výsledném rozvrhu následovat bezprostředně po sobě, tzv. „dvouhodinovky“. Naopak paralelní skupina obsahuje předměty, u kterých je vhodné, aby jejich výuka byla rozdělena rovnoměrně v celém týdnu.

Poslední parametr důležitost říká, o jak náročný předmět se jedná, a tedy zda je lepší jej umístit na začátek dne, kdy jsou děti ještě plné energie, nebo naopak ke konci dne. Parametr může nabývat tří hodnot. Hodnota 0 značí vysokou důležitost, hodnota 1 střední a hodnota 2 nízkou důležitost.

4.2 První fáze: vytváření vektorů

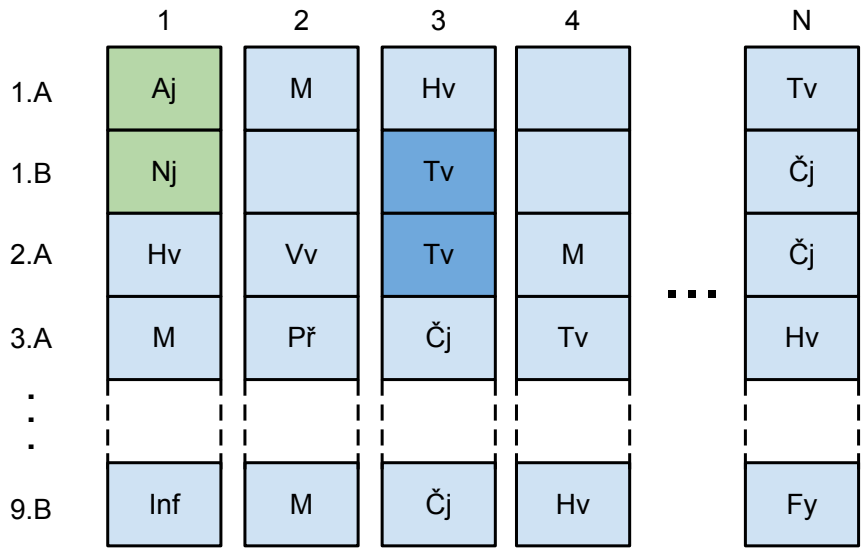
První genetický algoritmus se snaží rozmístit všechny požadované skupiny (popsané v kapitole 4.1) do vektorů tak, aby mezi skupinami v jednom vektoru nevznikaly žádné kolize a zároveň vznikly tak zaplněné vektory, aby je algoritmus v druhé fázi byl schopen úspěšně uspořádat do rozvrhu. Následující podkapitoly popisují návrh první fáze výpočtu pomocí hybridního genetického algoritmu.

4.2.1 Kódování chromozomů

Kódování jedince u prvního genetického algoritmu ukazuje obrázek 4.2, který je však z důvodu větší přehlednosti lehce zjednodušen, protože neobsahuje informace o vyučujících ani o učebnách, které správně patří do každé vyplněné buňky vektoru. Chromozomy jsou pevné délky a skládají se z vektorů, které slučují předměty vyučované v jednu hodinu. Z obrázku je zřejmé, že položky ve vektorech postupně odpovídají jednotlivým třídám. Protože množina alel odpovídá množině umísťovaných skupin, jedná se o vícehodnotové kódování chromozomů.

Platí, že předměty ve skupinách umístěných v jednom vektoru budou vyučovány souběžně. Z toho důvodu mohou mezi předměty vznikat kolize v případě, že se jeden vyučující, třída nebo učebna vyskytují ve dvou skupinách z vektoru současně. Tato situace na obrázku

[2]Parametr typ skupiny má smysl pouze u těch skupin, které mají být použity více než jednou.



Obrázek 4.2: Ukázka kódování chromozomu v první fázi algoritmu.

nastává v případě, že by škola měla pouze jednu tělocvičnu. Ve vektoru číslo 3 můžeme vidět tmavě modře zvýrazněnou vzniklou kolizi. Při běhu výpočtu jsou takovéto vektory zakázané a zde zobrazená hypotetická situace by neměla vůbec nastat.

Na obrázku můžeme také vidět zeleně zvýrazněnou skupinu skládající se z více než jednoho předmětu. Jedná se o dva dělené jazykové předměty, při kterých se třídy slučují podle toho, do které skupiny žák patří. K tomuto slučování dochází převážně z ekonomických důvodů. Není totiž běžné, aby dva učitelé učili souběžně jednu a tu samou třídu. Pokud bychom takovouto skupinu chtěli přesunout do jiného vektoru, je to bez odebrání skupiny z jiného vektoru možné pouze do vektoru číslo 4, ve kterém je místo pro oba dva předměty ze skupiny.

Každý jedinec v populaci prvního genetického algoritmu obsahuje právě takový počet N vektorů s předměty, aby při jejich přeskládání do mřížky o rozměrech *počet hodin ve dni* krát *počet dní* vznikl rozvrh pro všechny třídy.

4.2.2 Generování počáteční populace

Pro vytváření počáteční populace byl navržen algoritmus, který při každém spuštění vytvoří náhodného jedince tak, aby jeho vektory neobsahovaly žádné kolize mezi skupinami. Takto vzniklý jedinec má sice minimální předpoklady k tomu, aby mohl být okamžitě použit v druhé fázi výpočtu, ale díky náhodnosti přináší do populace rozmanitost. Dobře navržený genetický algoritmus pracující s takto vytvořenými jedinci dokáže vysledovat jejich dobré vlastnosti, které přenese do další populace a časem tak může vytvořit kvalitní jedince, kteří jsou jako výsledek první fáze již přijatelní.

Algoritmus pracuje tak, že náhodně vybírá nezařazené skupiny předmětů a postupně zkouší do každého vektoru vybranou skupinu přiřadit. Tím, že program začíná zkoušet zařadit skupinu vždy od prvního vektoru, tak počáteční vektory mírně upřednostňuje, protože ve chvíli, kdy se skupinu podaří přiřadit, algoritmus okamžitě přechází na další skupinu. Díky upřednostnění některých vektorů nejsou náhodně generovaní jedinci tak úplně náhodní, ale je u nich velká pravděpodobnost, že budou obsahovat zcela zaplněné vektory

(právě ty upřednostněné) a počáteční populace tak bude již obsahovat kvalitnější jedince[3].

V pozdější fázi generování jedince však může nastat situace, kdy vybranou skupinu nelze bez vzniku kolize vložit do žádného vektoru. Tento problém je vyřešen tak, že při pokusu o vložení jsou vytipovány skupiny, které by bylo nutno z vektoru odebrat, aby bylo možné aktuální skupinu do tohoto vektoru přiřadit. Poté je vybrán ten vektor, jehož odebírané skupiny bude v budoucnu nejjednodušší zase zařadit do jiného vektoru. Skupiny jsou z vektoru odebrány a nahrazeny problémovou skupinou. V další fázi algoritmus zase náhodně vybere skupinu a pokusí se ji podobně zařadit do vektorů.

Takto navržený způsob navracení by však mohl mít tendenci uváznout v nekonečné smyčce. V nejjednodušším případě by mohly na konci generování zůstat dvě skupiny, které by se v jednom vektoru donekonečna navzájem vyměňovaly. Byl proto navržen mechanismus, který tyto cyklické závislosti detekuje a neustálému opakování výměn zabrání.

Pro každý vektor je zapamatováno několik posledních výměn a počet nezařazených skupin v době této výměny. Ve chvíli, kdy by mělo dojít k výměně skupin ve vektoru, je zkontrolována uložená historie pro informaci, zdali k této výměně za stejného (případně nižšího) stavu zbývajících skupin nedošlo v minulosti. Pokud ano, tak tato výměna není provedena a provede se výměna na následujícím vektoru, ze kterého odebrané skupiny půjdou také jednoduše zařadit. Protože si tento mechanismus z důvodu efektivity pamatuje řádově pouze desítky posledních výměn, je teoreticky možné, že nastane situace, kdy cyklus výměn bude delší než pamatovaná historie a algoritmus tak znovu uvázne. V praxi však k takto velkým smyčkám nedochází a navržená metoda je tedy použitelná.

4.2.3 Fitness funkce

Výsledná hodnota fitness jedince se skládá ze součtu ohodnocení kvality jednotlivých vektorů. Její vzorec je uveden v rovnici 4.1.

$$\text{ohodncení jedince} = \sum_{\text{vektor} \in \text{jedinec}} (\text{ohodnocení vektoru}) \times \text{postihy} \tag{4.1}$$

Ohodnocení vektorů probíhá na bázi pravidel, která hodnotí jejich vlastnosti. Pro každé z nich platí, že podle míry jeho splnění navrací ohodnocení z intervalu $\langle 0; 1 \rangle$. Toto je nutné z důvodu, aby mohla být pravidla uživatelem váhována, a bylo tak možné ovlivnit vlastnosti generovaného rozvrhu. Hodnocení všech pravidel jsou sečtena do výsledného ohodnocení vektoru. Protože však ohodnocení kvality vektorů probíhá bez globálního pohledu na ostatní vektory v jedinci, je výsledná fitness hodnota jedince posléze snížena o postihy vzniklé vztahy mezi nimi.

Všechna pravidla mají formu čítačů, které udávají určité vlastnosti vektoru. Hodnoty těchto čítačů jsou většinou normalizovány do potřebného intervalu $\langle 0; 1 \rangle$ pomocí upravených *logistických sigmoid*[4]. Konkrétní úpravy logistických sigmoid jsou empiricky voleny tak, aby náhodně generovaní jedinci obdrželi pouze pár desítek procent z ohodnocení pravidla. Lepší jedinci však díky nim rychle získávají vyšší ohodnocení, a je tak v průběhu evoluce vytvářen dostatečný selektivní tlak mezi jedinci.

V některých případech, kdy lze zjistit celkový rozsah čítače, je přistoupeno také k jiným metodám jeho transformace. Zvolená metoda transformace je uvedena v následujícím textu vždy u konkrétního pravidla.

[3] Důvody, proč jsou jedinci se zcela zaplněnými vektory výhodnější, jsou uvedeny v kapitole 4.2.3.

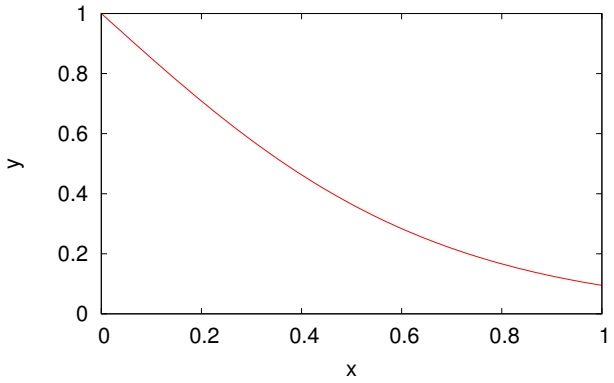
[4] Způsob převodu hodnocení do požadovaného intervalu pomocí logistické sigmoidy je popsán například v [22] na straně 35.

Ohodnocení vektorů se skládá z následujících pravidel:

- Rozptyl hodnoty důležitosti předmětů ve vektoru - každý předmět má ve vstupním XML souboru nastavenou svoji důležitost podle preferencí uživatele. Podle důležitosti jsou ve výsledném rozvrhu řazeny předměty tak, aby důležitější předměty byly vyučovány v ranních hodinách a volnější předměty v odpoledních. Tohoto však nelze dosáhnout, pokud bude jeden vektor obsahovat předměty různé obtížnosti.

  Z toho důvodu je spočítán matematický rozptyl obtížností jednotlivých předmětů, který by měl být v ideálním případě co nejnižší. Obtížnost předmětů je tříhodnotová veličina, tudíž bude nabývat hodnot již v požadovaném v intervalu $\langle 0; 1 \rangle$. Pro silnější selekční tlak je však výsledek přetransformován do podobného intervalu pomocí zvolené logistické funkce 4.2, její graf lze vidět na obrázku 4.3.

$$y = -\frac{2}{1 + e^{-3x}} + 2 \tag{4.2}$$



Obrázek 4.3: Transformační funkce pro pravidlo rozptylu důležitosti předmětů ve vektoru.

- Zaplnění vektorů - protože se v jedinci vyskytují i vektory, které nejsou plně zaplněny, je u nich vhodné, aby se zaplněné hodiny vyskytovaly převážně u starších ročníků. Například vektor, který by obsahoval vyučování pouze v první třídě, by skoro jistě nebylo možné umístit. Jeho umístění by totiž znamenalo, že výuka ve všech vyšších ročnících již skončila, což odporuje běžným zvyklostem. Z podobného důvodu je vhodné také preferovat vektory, které jsou plně nebo vůbec zaplněny. Ty lze totiž jednoduše umístit buď na konec, nebo na začátek vyučovacího dne.

  Pro vyhodnocení této vlastnosti byla zvolena metoda, která započítá vektoru bod za každou dvojici hodin, která splňuje jednu z následujících podmínek:

  - Obě dvě hodiny nejsou obsazeny.
  - Obě dvě hodiny jsou obsazeny.
  - Hodina vyšší třídy je obsazena, když hodina nižší třídy není.

  Protože počet všech dvojic hodin ve vektoru je znám, je možné takto získanou hodnotu podělením normalizovat do intervalu $\langle 0; 1 \rangle$. Do výpočtu je navíc vložena nelinearita, která penalizuje horší vektory. Díky tomu je více dosaženo chtěného chování pravidla.

Výše popsané chování pravidla bylo experimentálně ověřeno a porovnáno s ručním ohodnocením vektorů. Při porovnání bylo zjištěno, že obě křivky jsou si podobné. Toto porovnání je možno nalézt v příloze D.

- Kvalita přidělených tříd - ve vstupním XML je možné definovat výuku jednoho předmětu ve více učebnách. Z těch se při generování rozvrhu vybírají učebny podle pořadí tak, aby výuka v dříve uvedených učebnách byla priorizována (může se jednat například o specializované učebny). Protože se tato vlastnost rozvrhu týká přímo jednotlivých vektorů, je nutno ji také zahrnout do jejich ohodnocení.

  Zvolená metoda spočítá aritmetický průměr pořadí přidělených učeben pro ty předměty, u kterých existuje možnost volby mezi více učebnami. Protože je možné zjistit maximální možnou hodnotu průměru pro konkrétní vektor, je spočítaná hodnota průměru lineárně transformována do požadovaného intervalu $\langle 0; 1\rangle$. Pokud vektor neobsahuje žádný předmět s možností volby učebny, obdrží polovinu bodů z váhy pravidla.

- Rovnoměrné rozdělení předmětů v týdnu - podmínku kladenou na paralelní skupiny předmětů (viz kapitola 4.1) je při vytváření vektorů také nutné řešit. Pro každý vektor totiž musí existovat minimální počet dalších vektorů, se kterými vektor nebude v kolizi v důsledku nasazení paralelních skupin. Vektory z takovéto skupiny se stávají potenciálními následníky vektoru v konkrétním dni.

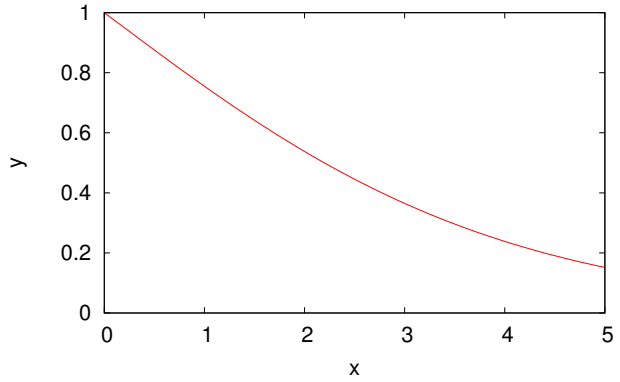
  Protože vytvořená skupina z pohledu jednoho vektoru nezaručuje nekoliznost ostatních vektorů ve skupině, je nutné, aby takto vytvořená skupina vektorů byla mírně větší než je maximální počet hodin za den. Je zde předpoklad, že pokud budou takovéto skupiny pro každý vektor dostatečně velké, bude v nich možné najít menší skupiny, ve kterých se nebudou tyto kolize vyskytovat. Tento předpoklad je nutný, protože kontrola bezkolizních skupin je obecně NP-úplný problém[5]. Přestože je kardinalita množiny vektorů v jedinci známá a relativně malá, tak by takováto kontrola významně zpomalila celý průběh evoluce.

  Pravidlo nejprve pro ohodnocovaný vektor spočítá počet kolizí paralelních skupin mezi ním a ostatními vektory. Z nich vybere několik nejmenších, které sečte. Kolik dvojic se bude sčítat, je možné nastavit jako parametr pravidla. Je však vhodné, aby tento počet byl větší než je počet vyučovaných hodin ve dni. Výsledný součet je transformován do intervalu $(0; 1\rangle$ pomocí následující logistické sigmoidy, jejíž tvar je zobrazen na obrázku 4.4.

$$y = \frac{2}{1 + e^{x/2}} \tag{4.3}$$

Předchozí uvedená pravidla se soustředila především na ohodnocení jednotlivých vektorů, pouze kvalitní vektory však nezaručují, že je půjde uspořádat tak, aby vytvořili kvalitní rozvrh. Je proto nutné zkoumat i vazby mezi nimi. O ohodnocení těchto vazeb se starají následující penalizační pravidla, jejichž výstupem je koeficient z intervalu $\langle 0; 1\rangle$, kterým je násobeno ohodnocení konkrétní entity:

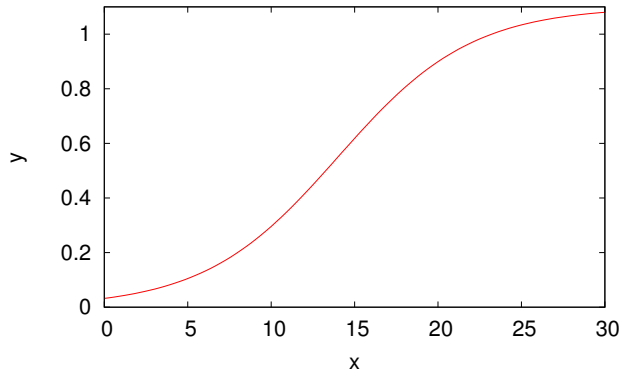
[5]Problém hledání bezkolizních skupin je převoditelný na známý NP-úplný *problém pokrytí množiny.*



Obrázek 4.4: Transformační funkce pro pravidlo rovnoměrnosti rozdělení předmětů v týdnu.

- Počet zaplněných vektorů - jedním z tvrdých požadavků na výsledný rozvrh je také minimální počet hodin v každém dni. To může být splněno pouze v případě, že existuje dostatečný počet plně zaplněných vektorů. Těch by mělo být i tak co nejvíce, protože pokud se ve vektoru nevyskytují žádné volné hodiny, je daleko jednodušší jej umístit do rozvrhu. Pravidlo tedy spočítá počet plných vektorů v jedinci a aplikuje na něj parametrizovatelnou logistickou transformační funkci uvedenou v rovnici 4.4 a zobrazenou na obrázku 4.5. Parametr funkce p odpovídá požadovanému počtu plně zaplněných vektorů. Tato funkce je velmi přísná a vytváří tak značný evoluční tlak na toto pravidlo.

$$y = \frac{1.1}{1 + e^{(-x(20/p)+14)/4}} \tag{4.4}$$



Obrázek 4.5: Transformační funkce pro penalizační pravidlo s parametrem $p = 20$ vytvářející evoluční tlak na výskyt plně zaplněných vektorů.

- Omezení počtu sériových kolizí mezi vektory - jednotlivé vektory jsou v rozvrhu řazeny za sebou, každému vektoru tak může předcházet nejvýše jeden vektor a nejvýše jeden za ním může následovat. Z toho důvodu není možné, aby jeden vektor uspokojil více než dvě sériové kolize mezi skupinami v jiných vektorech. Dokonce i dvě sériové kolize mohou představovat problém, protože vznikne skupina tří vektorů, které musí

být v rozvrhu řazeny bezprostředně za sebou. Tím se velmi omezuje manévrovací schopnost druhé fáze algoritmu.

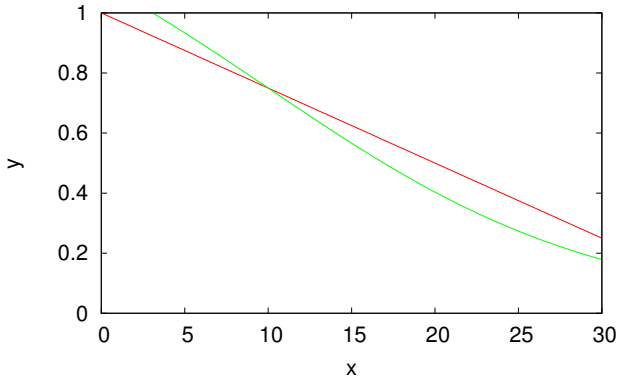
Toto pravidlo sečte pro každý vektor počet ostatních vektorů, mezi jejichž skupinami vzniká sériová vazba. Z důvodu potřeby většího tlaku na větší počet vektorů, které mají vazbu na nejvýše dva další vektory, je tato hodnota navíc umocněna na druhou. Takto získané hodnoty jsou pro všechny vektory z entity sečteny a transformovány do penalizačního koeficientu.

Protože se nepočítá síla vazby mezi vektory, ale pouze počet vazeb, je zřejmé, že je vítané, pokud jsou vazby mezi vektory co nejsilnější. Z toho důvodu je transformační funkce rozdělena do dvou funkcí. První funkce vyjádřená rovnicí 4.5 je použita pro hodnoty pravidla do počtu sériových vazeb a nevytváří již tak silný evoluční tlak. Druhá funkce vyjádřená rovnicí 4.6 je použita pro horší výsledky, s kterými by si druhá fáze algoritmu nemusela umět poradit.

Na obrázku 4.6 jsou zobrazeny obě dvě právě popsané funkce. Jejich rozdělení probíhá na hodnotě 10, která odpovídá hypotetickému počtu všech skupin se sériovou vazbou. Rovnice obou dvou funkcí jsou navrženy tak, aby obsahovaly parametr p, na jehož hodnotě se obě protínají:

$$y = \frac{-x}{4p} + 1 \tag{4.5}$$

$$y = \frac{3}{2(1 + e^{(x-p)/10})} \tag{4.6}$$



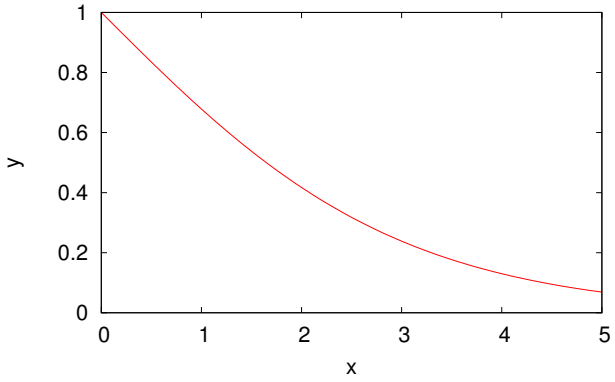
Obrázek 4.6: Transformační funkce pro penalizační pravidlo tvorby sériových vazeb s parametrem $p = 10$.

- Kontrola na souběžný výskyt paralelních a sériových kolizí - poslední penalizační pravidlo má za úkol zabránit souběžnému výskytu paralelních a sériových kolizí mezi libovolnou dvojicí vektorů. Pokud by totiž dvojice s těmito vlastnostmi existovala, nebylo by možné v žádném případě splnit požadavky dané typem skupin, protože by si navzájem odporovaly – tyto dva vektory by měly být v rozdílných dnech a zároveň následovat v rozvrhu hned za sebou.

  Podobně jako předchozí pravidlo i toto vytváří silný evoluční tlak na jedince. Funguje tak, že sečte počet výskytů dvojic vektorů s těmito vlastnostmi a tuto hodnotu transformuje pomocí přísné logistické funkce z rovnice 4.7 do intervalu $\langle 0; 1\rangle$. Výsledkem

poté vynásobí ohodnocení jedince. Zvolená funkce je zobrazena na obrázku 4.7.

$$y = \frac{2}{1 + e^{x/1,5}} \tag{4.7}$$



Obrázek 4.7: Transformační funkce pro penalizační pravidlo souběžných paralelních a sériových kolizí.

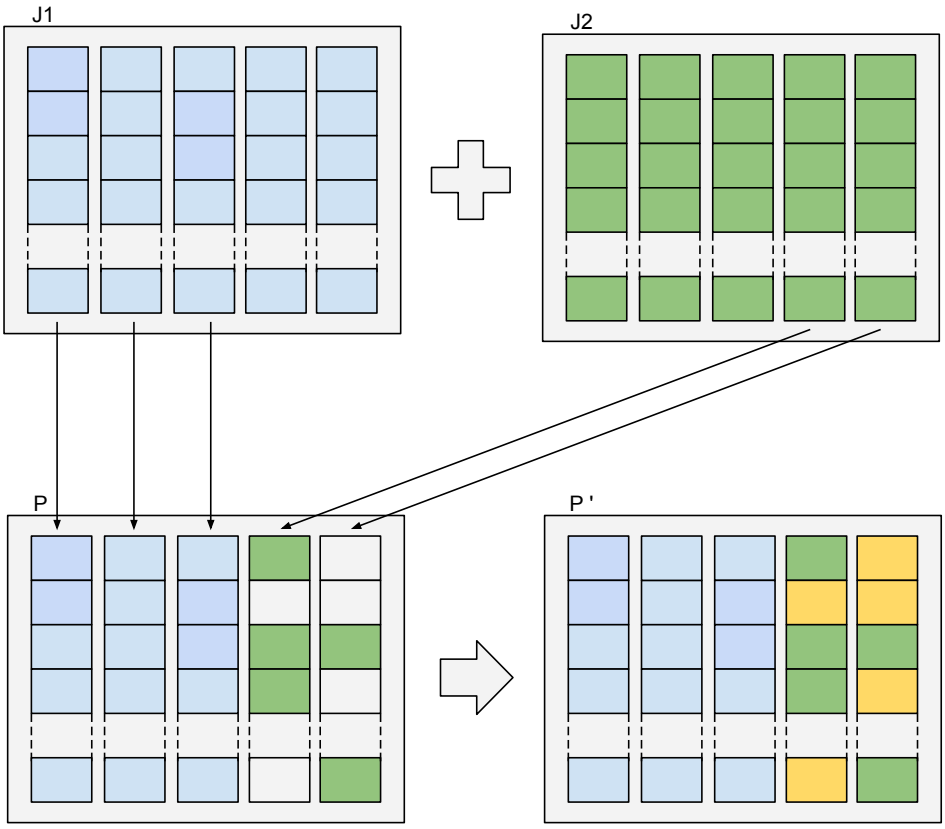
4.2.4 Genetické operátory

Výběr jedinců k reprodukci probíhá pomocí turnajové selekce. Jak bylo zmíněno v kapitole 3.3, která vychází z [9], je možné s touto metodou vhodným nastavením parametrů dosáhnout srovnatelných výsledků jako s ruletovým mechanismem selekce. Turnajová metoda je však výpočetně jednodušší.

Při průběhu reprodukce jsou použity oba základní genetické operátory křížení a mutace. Navržený binární operátor křížení vychází z jednobodového operátoru křížení dvou jedinců, který byl popsán v kapitole 3.4.1. Vytvořený operátor mutace má za úkol lehce pozměnit chromozom, teoreticky je však možné, že provede i větší zásah do jeho struktury. Pro výsledek blížící se více přírodním mechanismům jsou občas selekcí vybraní jedinci přímo přesunuti do následující populace bez křížení [10]. Z důvodu oddálení problému předčasné konvergence jsou do následující populace zařazeni pouze ti nově vzniklí jedinci, kteří jsou v ní unikátní (možnost použití této metody byla uvedena v kapitole 4.2.2).

Princip operátoru křížení dvou jedinců $J1$ a $J2$ ukazuje obrázek 4.8. Operátor náhodně zvolí bod křížení, po který jsou překopírovány vektory z prvního jedince $J1$ do potomka P. Zbylé vektory jsou doplněny do potomka z druhého jedince $J2$. Zde však vzniká problém, protože není nijak zaručeno, že ve vektorech dodaných z druhého jedince se nebudou vyskytovat předměty, které již obsahují vektory z prvního jedince. Je tak více než pravděpodobné, že některé skupiny předmětů by byly použity vícekrát, než je požadováno a naopak některé by nemusely být použity vůbec.

Skupiny, které se vyskytují ve výsledku vícekrát, než je požadováno, jsou z něj tedy odstraněny (na obrázku se jedná o nevybarvené předměty v jedinci P). O doplnění zbylých předmětů se postará stejná metoda, která generuje počáteční populaci. Tato opravná metoda se snaží doplnit do potomka P zbylé předměty tak, aby co nejméně měnila umístění předmětů z druhého jedince, a vytvořit tak výsledného potomka P'. Metoda navíc vůbec nepracuje s vektory z prvního jedince, je tak zaručeno, že potomek zdědí co nejvíce genetické informace od obou rodičů. Zbývající předměty, které bylo nutné dodatečně doplnit, jsou na obrázku ve výsledku zvýrazněny žlutě.

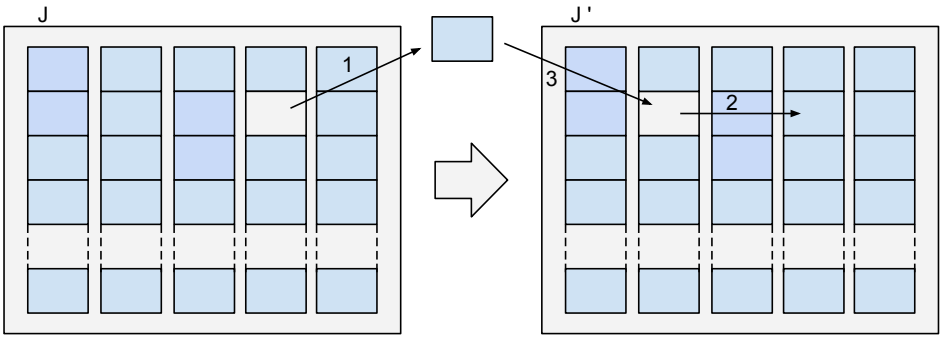


Obrázek 4.8: Princip operátoru křížení první fáze algoritmu.

Takto popsaný operátor křížení by však nikdy nepozměnil obsah prvního vektoru, ten by byl v každém případě překopírován beze změny z prvního předka do potomka. Mohlo by se tak stát, že mnoho kvalitních jedinců by bylo znevýhodněno, protože by obsahovali jeden nekvalitní vektor, který vznikl náhodným vygenerováním. Aby tato situace nenastala, operátor náhodně vybírá, zdali se jednotlivé vektory budou kopírovat z levé nebo pravé strany.

Při testech v kapitole 6.2 bylo zjištěno, že výběr křížení z pravé strany chromozomu je výpočetně o dost náročnější než výběr z levé. Tento problém vzniká z toho důvodu, že vektory v levé části chromozomů jsou ve většině případů plně zaplněné předměty, a najít tak jejich jinou platnou permutaci je obtížnější. Operátor proto kříží z pravé strany pouze v některých náhodně vybraných případech (jejich četnost je možno nastavit v parametrech výpočtu). Provedené testy prokázaly, že tento ústupek jen minimálně snižuje výpočetní kvalitu algoritmu, ale značně urychlí celý výpočet.

Operátor mutace jedince je jednodušší. Funguje tak, že náhodně zvolí skupinu předmětů, kterou odebere z jedince. Této akci odpovídá krok 1 v obrázku 4.9. Poté začne probíhat mutace jedince, která se snaží co nejjednodušeji přeskládat ostatní skupiny v jedinci, aby pro zvolenou skupinu vzniklo místo v jiném vektoru (přeskládání odpovídá krok 2 v obrázku). V poslední třetím kroku je vybraná skupina umístěna do jiného vektoru a mutace je dokončena.



Obrázek 4.9: Princip operátoru mutace první fáze algoritmu. Obrázek zachycuje stav po druhém kroku operátoru.

4.3 Druhá fáze: umístění vektorů

Úkolem druhého genetického algoritmu je uspořádat vektory do mřížky rozvrhu. Jedná se o jednodušeji uchopitelný problém než byl v první fázi, protože zde „pouze“ stačí najít vhodnou permutaci vektorů. Je tedy možné vycházet z teorie genetických algoritmů týkající se permutačních problémů. Dá se také předpokládat, že tento algoritmus bude rychlejší než algoritmus v první fázi, protože u něj nemusí docházet k žádným opravám chromozomů s využitím klasických algoritmů.

Druhá fáze předpokládá, že obdrží z první fáze dostatečně dobré vektory k tomu, aby mohla vytvořit kvalitní rozvrh. Jedním z požadavků na jeho kvalitu je i takový minimální počet plně zaplněných vektorů, aby ve výsledném rozvrhu mohl být pro všechny třídy dodržen stanovený minimální počet vyučovacích hodin. Pokud není splněna tato podmínka, tak není spuštěna ani druhá fáze algoritmu, ale je znovu restartována první fáze s pozměněnými parametry výpočtu tak, aby byla větší pravděpodobnost, že v dalším běhu tato podmínka splněna bude.

4.3.1 Kódování chromozomů

Předpokladem ke spuštění druhé fáze algoritmu je, že obsahuje daný minimální počet plných vektorů. Je tak možné s určitým množstvím plně zaplněných vektorů počítat již při návrhu kódování jeho chromozomů. Permutace je přirozeně reprezentována pomocí matice čísel. Pozice čísel v ní odpovídají pozici vektoru ve výsledném rozvrhu. Řádky matice odpovídají jednotlivým dnům a sloupce vyučovacím hodinám. Příklad popsaného chromozomu lze nalézt na obrázku 4.10.

Jednotlivé dny v chromozomu jsou rozděleny na dvě části, které jsou na obrázku zvýrazněny modře a zeleně. V první (modré) části dnů se v platném chromozomu smí vyskytovat pouze zcela zaplněné vektory. Ve druhé (zelené) části potom ostatní včetně zbylých plných[6]. Takto zvolená sémantika chromozomů zaručuje, že pro jakýkoliv vygenerovaný rozvrh bude splněno tvrdé omezení vyžadující minimální počet vyučovacích hodin v každém dni.

Výše popsaná struktura chromozomu vychází z celkové filozofie návrhu, která spočívá v co největším omezení prohledávaného stavového prostoru.

[6]Plné vektory mohou v chromozomu být umístěny do druhé části dnů v případě, že celkový počet chromozomů je vyšší než součin počtu dnů a minimálního počtu hodin.

Obrázek 4.10: Kódování chromozomu druhé fáze algoritmu pro maximální počet hodin denně 7 a minimální 4.

4.3.2 Generování počáteční populace

Vytvoření náhodného jedince je poměrně jednoduchá procedura. Vektory se skupinami jsou rozděleny na množinu plně zaplněné vektorů a množinu ostatních. Z první množiny jsou postupně náhodně vybírány vektory, jejichž čísla jsou ve vybraném pořadí umístěna do těch pozic v chromozomu, kde se musí vyskytovat plné vektory (tyto hodiny jsou na obrázku 4.10 zvýrazněny modře). Ve chvíli, kdy se všechny pozice zaplní, se obě množiny sloučí a zaplnění zbylých skupin je dokončeno obdobným způsobem.

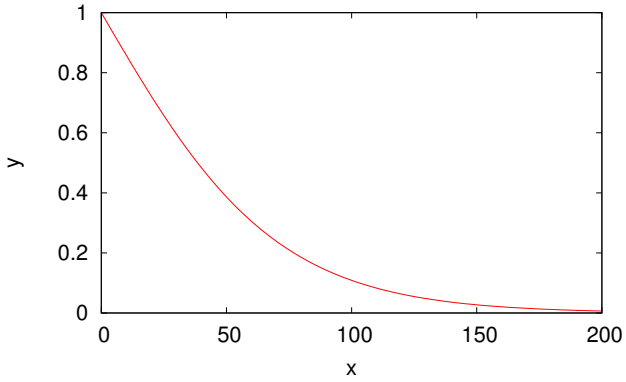
4.3.3 Fitness funkce

Podobně jako fitness funkce první fáze výpočtu je i fitness funkce v druhém genetickém algoritmu reprezentována pravidly, která v tomto případě již ohodnocují výsledný rozvrh. Celková hodnota fitness jedince se získá tak, že jsou body přidělované níže popsanými pravidly sečteny dohromady. Každé pravidlo vyjadřuje určitou vlastnost rozvrhu, která je poté transformována na jeho ohodnocení (způsob transformace pravidel byl popsán v kapitole 4.2.3). Jsou použita tato hodnotící pravidla:

- Kolize mezi paralelními a sériovými předměty ve dnech - každá paralelní skupina, kterou je nutné umístit více než jedenkrát, vyžaduje být rozmístěna v rozvrhu tak, aby se její předměty neopakovaly v jednom dni vícekrát. Naopak každá sériová skupina vyžaduje, aby její předměty byly umístěny v rozvrhu přímo za sebou. Popsané vlastnosti skupin jsou detailněji rozebrány v kapitole 4.1.

  Pravidlo nejprve projde všechny dvojice vektorů v konkrétních dnech a spočítá v nich počet kolizí k_s mezi sériovými skupinami. Poté projde všechny dvojice vektorů v rozdílných dnech a spočítá počet kolizí k_p mezi paralelními skupinami. Obě dvě získané hodnoty jsou sečteny v nastaveném poměru a. Takto získaná hodnota je transformována pomocí zvolené logistické sigmoidy z rovnice 4.8, která je zobrazena obrázku 4.11. Funkce obsahuje parametr p, který upravuje její průběh v závislosti na celkovém počtu vektorů.

$$y = \frac{2}{1 + e^{(k_p + a k_s)/p}} \tag{4.8}$$

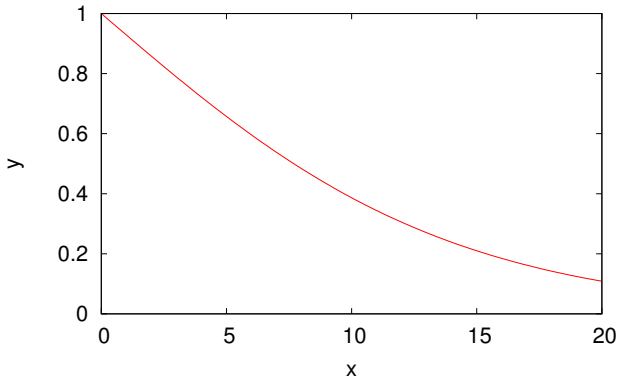


Obrázek 4.11: Transformační funkce pro pravidlo hodnotící počet vznikajících kolizí mezi skupinami ve dnech. Parametr p je zde roven hodnotě 35.

- Výskyt volných hodin v průběhu vyučování - druhým důležitým faktorem v kvalitním rozvrhu je absence volných hodin v průběhu vyučování jednotlivých tříd. Následující pravidlo spočítá, kolik těchto volných hodin ve všech rozvrzích existuje.

  Výsledek je znovu transformován pomocí zvolené logistické sigmoidy. Její parametr p udává celkový počet tříd v rozvrhu. Vzorec transformační funkce spolu s grafem lze nalézt v rovnici 4.9 a obrázku 4.12.

$$y = \frac{2}{1 + e^{2x/p}} \tag{4.9}$$



Obrázek 4.12: Transformační funkce pro pravidlo hodnotící počet vyskytujících se prázdných hodin v průběhu vyučování pro parametr $p = 14$.

- Umístění předmětů podle důležitosti - jedním z měkkých požadavků na výsledný rozvrh je také umístění předmětů v průběhu dne. Důležitější předměty by měly být zařazeny ve vyučování z rána, naopak volnější předměty je vhodné umístit až ke konci vyučování. Každá skupina předmětů má nastavenou svoji důležitost z tříhodnotové množiny.

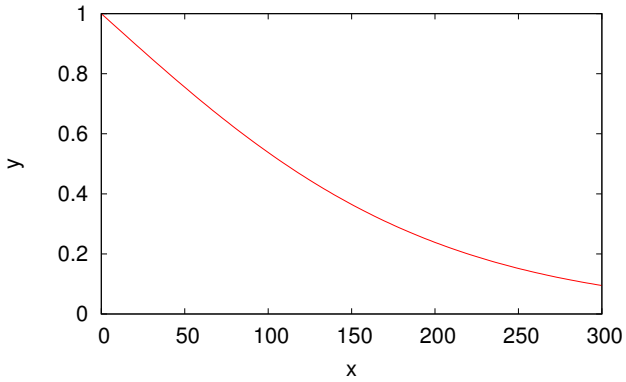
  Pro každý den pravidlo zjistí délku jeho vyučování. Nejnižší hodnotu přidělí důležitým (volným) předmětů na začátku (konci) vyučování, naopak nejvyšší hodnotu bodů

přidělí důležitým (volným) předmětů na konci (začátku) dne. Body pro předměty v průběhu vyučování jsou mezi nimi rozděleny lineárně.

Předměty, které mají nastavenou střední důležitost, nejsou hodnoceny – jejich umístění může být v rozvrhu libovolné. Dá se však předpokládat, že takovéto předměty budou v důsledku toho pravidla vytlačeny doprostřed vyučovacích dnů. Předpokládané chování bylo ověřeno testy, které jsou uvedeny v kapitole 6.4.

Body obdržené za všechny vyučovací dny všech tříd jsou sečteny a transformovány do požadovaného intervalu $\langle 0; 1\rangle$ pomocí logistické funkce s rovnicí 4.10. Jedná se opět o parametrickou funkci, kde parametr p odpovídá počtu vyučovacích hodin ve všech třídách. Její graf je zobrazen na obrázku 4.10.

$$y = \frac{2}{1 + e^{3x/p}} \tag{4.10}$$



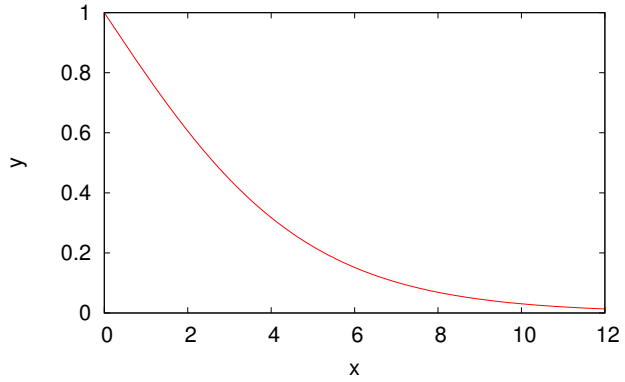
Obrázek 4.13: Transformační funkce pro pravidlo hodnotící umístění předmětů ve dnech podle jejich důležitosti pro parametr $p = 300$.

- Splnění požadavků na umístění sériových skupin - následující pravidlo kontroluje dodržení umístění sériových skupin předmětů v rozvrhu tak, aby po sobě přímo následovaly. Tyto skupiny jsou z první fáze rozmístěny v rozdílných vektorech, jejichž pořadí se pravidlo snaží vynutit.

  Pravidlo sečte počet párů sériových skupin, které nenásledují v jednotlivých dnech přímo po sobě. Ve správném rozvrhu by žádná taková dvojice předmětů neměla existovat. Výslednou hodnotu převede do intervalu $\langle 0; 1\rangle$ použitím logistické funkce 4.11. Její graf je znázorněn na obrázku 4.14. Průběh funkce je pro konkrétní zadání rozvrhu závislý na parametru p, který značí počet všech sériových skupin v rozvrhu.

$$y = \frac{2}{1 + e^{5x/p}} \tag{4.11}$$

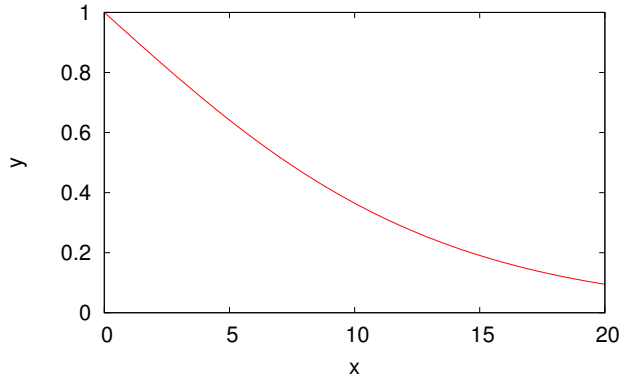
- Stejnoměrná délka rozvrhu - poslední aplikované pravidlo ve fitness funkci druhé fáze algoritmu se stará o dodržení vyrovnané délky vyučování po celý týden. Pravidlo pro každou třídu spočítá matematický rozptyl délek vyučování, který sečte přes jednotlivé třídy.



Obrázek 4.14: Transformační funkce pro pravidlo hodnotící splnění umístění dvouhodinových předmětů pro parametr $p = 12$.

Transformační funkce byla zvolena tak, aby náhodně generovaným rozvrhům přidělovala hodnocení okolo 15%. Toho bylo dosaženo úvahou, že průměrný rozptyl jedné třídy se pohybuje okolo hodnoty 1 (čtyřhodnotová veličina). Lze tak navrhnout funkci, která obsahuje parametr počtu tříd a má požadované vlastnosti. Jedná se znovu o logistickou sigmoidu s rovnicí 4.12, která je zobrazena na obrázku 4.15.

$$y = \frac{2}{1 + e^{3x/p}} \tag{4.12}$$

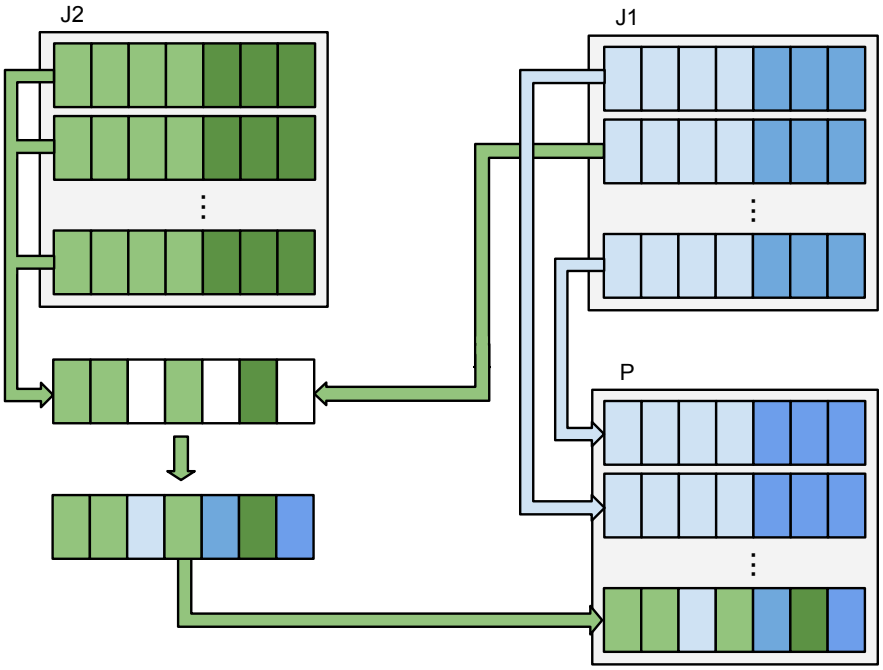


Obrázek 4.15: Transformační funkce pro pravidlo udržující stejnoměrnou délku rozvrhu v celém týdnu pro parametr $p = 20$ tříd.

4.3.4 Genetické operátory

Protože jedinci jsou kódováni jako permutace vektorů, jsou i operátory pracující s jejich chromozomy permutační. Z důvodu skryté sémantiky upřednostňování plných vektorů v chromozomech (viz kapitola 4.3.1) však není možné použít obecné permutační techniky. Navržený způsob křížení je inspirován permutačními operátory, které byly představeny v kapitole 3.4.2. Pro výběr jedinců ke křížení je zvolena metoda turnaje – důvod jejího použití je stejný jako u první fáze algoritmu (kapitola 4.2.4).

Schéma *křížení* dvou jedinců je zobrazeno na obrázku 4.16. Zde jsou pomocí turnaje vybráni dva jedinci $J1$ a $J2$ k následné reprodukci, v jejímž důsledku vznikne jedinec P. Je vybráno náhodně číslo z intervalu $\langle 1; \text{délka týdne - } 1\rangle$, které udává počet použitých dnů z prvního jedince – v našem případě byly vybrány dva náhodné dny z $J1$ a beze změny překopírovány do potomka P. Tuto operaci na obrázku reprezentují modré šipky. Je dobré si všimnout, že dny z předka jsou vybírány náhodně, ale do potomka jsou řazeny postupně. Díky této vlastnosti jsou dny mezi předky a potomkem prohozeny, a mohou tak dosáhnout vhodnějšího pořadí.



Obrázek 4.16: Schéma průběhu křížení druhé fáze algoritmu.

Zbylé dny se vytváří tak, aby byla zachována posloupnost vektorů ze zbylých dnů obou předků. Je však nutné zachovat také umístění plných vektorů v první polovině dne. Algoritmus postupně prochází plné vektory v druhém jedinci a vybírá ty, které prozatím nebyly použity. Pokud narazí na posloupnost již použitých vektorů, umístí do tohoto místa první nepoužitý vektor z jedince $J1$. Díky tomu zachovává i pořadí prvků z prvního jedince. Ve chvíli, kdy tímto způsobem vytvoří části s plnými vektory, sloučí nepoužité vektory se zbylými (neplnými) vektory, které zařadí do dnů podle stejného principu.

Popis tohoto kroku lze nalézt na obrázku 4.16 znázorněný pomocí zelených šipek z $J1$ a $J2$. Nyní je vytvořen platný den, který vznikl kombinací obou jedinců a může být vložen do potomka.

Použitý *operátor mutace* se velmi podobá klasickému permutačnímu operátoru mutace (popsaný v kapitole 3.4.3), který vymění pozici dvou genů v chromozomu. Zde je však nutné dodržet umístění plných vektorů v permutaci, a tak je před samotnou náhodnou výměnou ještě náhodně zvoleno, zdali se budou prohazovat plné vektory z počátku dnů nebo nezaplněné vektory z druhých částí dnů. Poté proběhne klasické náhodné prohození ve vybrané části chromozomu.

Kapitola 5

Implementace aplikace

Podle návrhu popsaného v kapitole 4 byla objektově vytvořena konzolová aplikace `scheduler` v jazyce C++ s dodržením standardu C++11. Z důvodu urychlení výpočetně náročných částí je však v aplikaci občas od objektového zapouzdření upuštěno. Její podrobnou dokumentaci lze spolu s diagramy spolupráce[1] vygenerovat za pomoci nástroje *doxygen* ve formátu HTML.

Aplikace při spuštění očekává dva XML soubory s požadavky na generovaný rozvrh[2] a parametry výpočtu[3]. Výstupem programu jsou tři HTML/CSS[4] soubory s rozvrhy pro jednotlivé třídy, učebny a vyučující, které jsou formátovány tak, aby při jejich prohlížení byly rozvrhy zobrazeny pod sebou, a naopak při tisku se pak tiskl každý rozvrh na zvláštní list papíru a s nevybarvenými políčky rozvrhu.

Ke zpracování vstupních souborů byla z důvodu usnadnění implementace využita externí knihovna. Byla zvolena C++ knihovna TinyXML[19]. Ta se jevila jako jednoduchý modul pro nenáročné a rychlé použití, které je pro aplikaci vhodné. Při práci s ní bylo zjištěno, že knihovna byla zvolena vhodně a dobře splnila na ni kladené požadavky.

Chyby, které mohou za dobu běhu programu vzniknout, jsou ošetřeny s využitím *výjimek*. Pokud dojde k chybě, je vygenerována chybová výjimka s příslušným kódem, která je odchycena a zpracována na konci funkce `main`. Jsou díky tomu zavolány destruktory všech vytvořených objektů a je tak zajištěno očekávané ukončení programu. Při vývoji bylo také s pomocí funkce `assert` vytvořeno mnoho kontrolních bodů pro snadnou detekci chyb v programu.

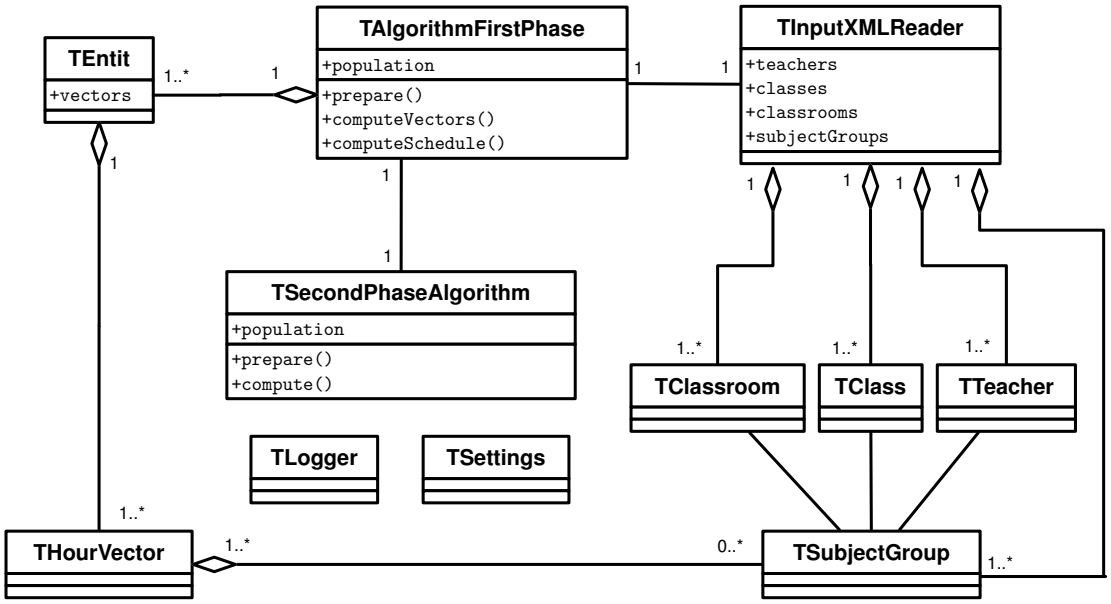
V následujících podkapitolách jsou popsány základní objekty v aplikaci a vazby mezi nimi. Jsou uvedeny problémy, které se při jejich implementaci vyskytly, a je naznačeno jejich řešení. Popisované objekty jsou zobrazeny na zjednodušeném diagramu tříd na obrázku 5.1. Pro detailnější pohled na implementaci je možno nahlédnout do automaticky generované dokumentace pomocí nástroje doxygen.

[1] Pro správné vygenerování diagramů spolupráce pomocí nástroje doxygen je nutné mít nainstalovaný volně šiřitelný balík programů Graphviz od firmy AT&T Research.

[2] Formát XML souboru s požadavky na rozvrh je popsán v kapitole 4.1 a v příloze B.

[3] Soubor s parametry výpočtu je popsán v příloze C.

[4] CSS stylopis rozvrhů byl převzat z tutorialu [13]. Ukázku rozvrhu formátovaného pomocí něj je možné nalézt v příloze F.



Obrázek 5.1: Zjednodušený diagram tříd aplikace.

5.1 Vytvoření vstupních dat

XML soubor s požadavky rozvrhu může pro větší školu obsahovat až tisíc řádků. Vytvořená aplikace však jednoduše předpokládá existenci tohoto souboru a s jeho vytvořením nijak neasistuje. Z důvodu časové náročnosti a možnosti vzniku chyb není ani vhodné, aby byl vstupní soubor celý vytvářen ručně za pomoci XML editoru. Problém by jistě vyřešilo vytvoření okenního uživatelského rozhraní. To však aplikace nenabízí.

Jako nejjednodušší se nabízela možnost získat data přímo z rozvrhů vytvářených na školách. Nejběžnější školní software pro tvorbu rozvrhů u nás je program Bakaláři [21]. V jazyce C++ byla proto vytvořena druhá konzolová aplikace `bc2xml`, která zajišťuje převod rozvrhu z výstupního HTML formátu programu Bakaláři do vstupního XML pro program `scheduler`. Aplikace `bc2xml` očekává jako parametry cesty k souborům s rozvrhy jednotlivých tříd[5] a na standardní výstup podle nich vygeneruje požadované XML. Při této transformaci jsou detekováni vyučující, třídy spolu s jejich kmenovými učebnami a paralelní skupiny předmětů. Uživatel musí dodatečně tento soubor upravit, aby doplnil specializované učebny, důležitost skupin předmětů a zvolil sériové skupiny.

Protože výstup z programu Bakalář není vytvářen pro další zpracování, je jeho čtení pomocí regulárních výrazů složitější. Navíc jeho různé verze generují lehce rozdílný HTML kód. Při použití programu `bc2xml` je tak nutné jedním z parametrů specifikovat typ souboru. Podle jeho hodnoty se aplikace přizpůsobí nuancím v HTML kódu. Před spuštěním převodu je také nutné konvertovat HTML soubory do kódování znaků pomocí UTF-8, k tomu lze využít vytvořený script `iconv.sh`.

Přestože program `bc2xml` vyžaduje od uživatele ruční úpravu souborů, umožňuje dostatečně zjednodušit tvorbu vstupních dat.

[5] Ukázky rozvrhů vytvořených pomocí aplikace Bakaláři jsou umístěny ve složce projektu `src/bakalar_charts`.

5.2 Příprava před výpočtem

Před spuštěním výpočtu jsou inicializovány instance tříd `TSettings`, `TInputXMLReader` a `TLogger` z modulů `main.cc` a `library.cc`. První jmenovaný modul obsahuje specifické objekty pro vyvíjenou aplikaci. Jedná se o již zmíněnou třídu `TSettings`, třídu pro zpracování parametrů a funkci `main()`. Modul `library.cc`[6] na druhou stranu obsahuje obecné funkce, třídy a šablony, u kterých je silný předpoklad, že by mohly být znovu použity v jiném projektu.

Instance třídy `TSettings` se postará o nastavení parametrů výpočtu podle XML souboru dodaného uživatelem. Nezadané parametry jsou inicializovány na výchozí hodnoty. Z důvodu potřeby jednoduché dostupnosti parametrů na mnoha místech se jedná o globální instanci.

Druhým a posledním globálním objektem v programu je instance třídy `TLogger`. Ta se stará o správu výstupních souborů. Program zapisuje data o vývoji kvality populace v rámci evolucí obou fází algoritmu a všeobecná logovací data vhodná především pro zpětnou analýzu běhu programu.

Objekt třídy `TInputXMLReader` zpracuje druhý vstupní XML soubor s požadavky na rozvrh. Vytvoří objektovou kopii v něm obsažených informací o entitách, která umožňuje jednodušší manipulaci s nimi v rámci zbytku aplikace. Třída sdružuje instance tříd `TClass`, `TClassroom`, `TTeacher` a `TSubjectGroup`, které jsou obsaženy ve vstupním souboru – tuto agregaci lze vidět na obrázku 5.1 vpravo. Implementaci agregovaných tříd lze nalézt v modulu `components.cc`. Obě třídy zpracovávající XML data využívají prostředků knihovny TinyXML.

5.3 První fáze výpočtu

Abstraktní část implementace první fáze výpočtu, jejíž funkce byla podrobně popsána v kapitole 4.2, obsahuje třída `TAlgorithmFirstPhase` z modulu `algorithmFirstPhase.cc`. Její metoda `prepare()` vygeneruje počáteční populaci jedinců (tříd `TEntit`), kteří obsahují zadaný počet vektorů (tříd `THourVector`) s vyučovacími hodinami. Obsah vektorů je generován náhodně tak, aby uvnitř vektorů nevznikaly žádné kolize mezi skupinami předmětů (třídami `TSubjectGroup` z modulu `components.cc`) a zároveň byl genetický kód jedinců dostatečně rozmanitý. Podrobněji je způsob generování náhodných jedinců popsán v kapitole 4.2.2.

Výpočet první fáze algoritmu spouští volání metody `computeVectors()` – zde je implementován první genetický algoritmus. Pro zajištění přežití nejlepších jedinců je využita metoda elitismu, která byla vysvětlena v kapitole 3.5.

Třída `TAlgorithmFirstPhase` implementuje výběr jedinců k reprodukci formou turnaje tak, aby byli vybráni vždy rozdílní jedinci. Pokud by toto neplatilo a byli by vybráni i shodní jedinci, nevznikl by křížením žádný nový genetický kód. Takové křížení by bylo zbytečné a navíc by snižovalo genetickou rozmanitost populace.

Z podobného důvodu je vytvořeno i omezení na výstupu křížení. Nově vzniklí jedinci, kteří mají shodnou hodnotu fitness s některým z jedinců v následující populaci, nejsou dále použiti a křížení je opakováno do té doby, dokud nevznikne unikátní jedinec. Hodnota fitness funkce jedince je zde použita jako otisk (*hash*). Tato modifikace může v případě

[6] Modul `library.cc` obsahuje mimo jiné implementaci některých tříd, které autor vytvořil v průběhu svého studia na Vysokém učení technickém v Brně, Fakultě informačních technologií.

nízkého počtu vstupních dat způsobit uváznutí programu v nekonečné smyčce, protože nebude možné vytvořit dostatečně rozmanitou populaci. Problém uváznutí není v aplikaci nijak ošetřen, protože se nepředpokládá užití aplikace k řešení triviálních rozvrhů.

Pro úplnost lze uvést, že vlastní reprodukce je implementována přímo ve třídě jedince (`TEntit`) z důvodu snazšího přístupu k jeho genetické informaci. Opravu chromozomu jedince provádí metoda `accomplishWithMinConflifts()`, kterou je nutno spustit po jeho vykřížení. Implementace křížení odpovídá návrhu z kapitoly 4.2.4.

Jednou z předpokládaných vlastností výsledného programu v kapitole 1.1 bylo, že by měl nabídnou více variant rozvrhu. Aby nebylo nutné spouštět celý výpočet vícekrát, je po dokončení první fáze spuštěna druhá fáze výpočtu pro vektory z každého elitního jedince. To umožní vygenerovat různé varianty rozvrhu a zároveň jen mírně prodloužit dobu výpočtu (výpočetní náročnost druhé fáze algoritmu je nižší než náročnost fáze první). Na druhou stranu toto vylepšení trochu omezuje možnosti programu, protože ne vždy může být závislost počtu výsledných rozvrhů na počtu elitních jedinců v první fázi výpočtu vítaná.

Jinou možností pro vygenerování více variant rozvrhu by mohlo být použít více jedinců z druhé fáze. Avšak je pravděpodobné, že takovéto rozvrhy by se lišily jen minimálně, protože se dá předpokládat, že by se jednalo o jedince z okolí jednoho lokálního optima.

Po dokončení první fáze výpočtu je volána metoda `computeSchedule()` instance třídy `TAlgorithmFirstPhase`, která pro každého elitního jedince z finální populace první fáze algoritmu spustí výpočet druhé fáze.

5.4 Druhá fáze výpočtu

Než dojde ke spuštění druhé fáze algoritmu, je nejdříve zkontrolováno, zda vektory z první fáze obsahují dostatečné množství plných vektorů, s kterými je tak možno splnit požadavek na minimální počet hodin v každém vyučovacím dni. Pokud například budeme požadovat pětidenní rozvrh o čtyřhodinové minimální délce vyučování, musí být první fáze algoritmu schopna vytvořit minimálně dvacet zaplněných vektorů, které budou umístěny právě v prvních čtyřech vyučovacích hodinách každého dne.

Pokud není předchozí podmínka při přechodu z jedné fáze do druhé splněna, je výpočet první fáze restartován s upravenými parametry výpočtu tak, aby byla zvýšena pravděpodobnost, že se takové vektory podaří vytvořit. Jsou zdvojnásobeny parametry velikosti populace a počtu iterací. Díky tomu je zvětšena rozmanitost populace a doba snahy algoritmu o nalezení lepšího řešení.

Na druhou stranu se dá předpokládat, že opakování výpočtu první fáze s takto pozměněnými parametry bude trvat až čtyřikrát delší dobu než předchozí spuštění. Mohlo by se zdát výhodné využít data z předchozího běhu a opakovaný výpočet tak urychlit. Dá se však předpokládat, že kvalitnější, evolucí přizpůsobení jedinci z předchozího běhu, by měli tendenci negativně ovlivnit populaci nově vygenerovaných náhodných jedinců a jejich výskyt v populaci rychle eliminovat.

Jak již bylo popsáno v kapitole 4.3, druhá fáze výpočtu se stará o umístění vygenerovaných vektorů do časových oken rozvrhu. Jedná se tedy vlastně o problém hledání „vhodné“ permutace vektorů. Z toho důvodu je implementace i výpočetní náročnost druhé fáze výpočtu podstatně jednodušší, a tak je celá druhá fáze algoritmu umístěna pouze v jednom modulu pojmenovaném `algorithmSecondPhase.cc`.

Spuštění druhé fáze obstarává třída `TSecondPhaseAlgorithm`. Její metoda `prepare()` analyzuje vztahy mezi vektory, které vznikly umístěním skupin do nich, za pomoci třídy `TVectorLinks`. Volání metody `randomize()` inicializuje jedince (permutace) v populaci

náhodnými hodnotami tak, aby hodnoty odpovídající plným vektorům byly umístěny na začátku dnů (tento princip byl popsán v kapitole 4.3.2).

Poté je spuštěn vlastní výpočet voláním metody `compute()`. Obecný algoritmus druhé fáze je velmi podobný algoritmu první fáze, který byl popsán v kapitole 5.3, s tím rozdílem, že zde jsou volány jiné genetické operátory (popsané v kapitole 4.3.4) pro jedince třídy `TSecondPhaseEntit`. Za zmínku však stojí použití speciálního asexuálního operátoru `magic()` třídy `TSecondPhaseAlgorithm`, který je aplikován pouze na některé nově vykřížené jedince (podobně jako operátor mutace). Tento operátor přeskládá pořadí nezaplněných vektorů v každém dni tak, aby pro všechny třídy minimalizoval výskyt počtu volných hodin v průběhu vyučování. V běžném sedmihodinovém rozvrhu, ve kterém musí být vyučovány každý den minimálně čtyři hodiny, se jedná o objevení nejlepšího pořadí tří vektorů. Pro většinu typů škol se dá předpokládat, že se bude jednat nejvýše o pět vektorů. Hledání nejvhodnějšího pořadí vektorů z hlediska výskytu volných hodin tak mohlo být (bez znatelného zpomalení zbytku výpočtu) implementováno deterministickým algoritmem.

V okamžiku, kdy je získána poslední populace druhé části algoritmu, je z ní možné vybrat nejlepšího jedince, který reprezentuje nejlepší dosažené rozmístění vektorů v rozvrhu. Podle něj jsou nyní ve formátu HTML vytvořeny finální rozvrhy pro třídy, vyučující a učebny.

Kapitola 6

Testování

Tato kapitola obsahuje popis vybraných testů implementované aplikace. Aby se zabránilo ovlivnění výsledků testování stochastickým charakterem navrženého algoritmu, byly všechny dále popsané testy opakovány nejméně třikrát. Dá se tak předpokládat, že všechny uvedené testy je možno později zopakovat se srovnatelnými výsledky. Výstupy všech testů spolu s nastavením parametrů výpočtů lze nalézt na přiloženém CD. V následujícím textu jsou pro zjednodušení uváděna pouze nejdůležitější nastavení.

Pro testování byla použita reálná data ze základních škol ZŠ Husova a ZŠ Komenského sídlících ve městě Náměšť nad Oslavou[1]. Obě jmenované školy využívají pro vytváření rozvrhů aplikaci Bakaláři, bylo tedy možné převést data z aktuálních rozvrhů do požadovaného XML formátu pomocí programu `bc2xml`. Jeho použití je popsáno v kapitole 5.1.

Ze školy ZŠ Husova byl použit rozvrh pro školní rok 2011/2012, který obsahuje požadavky na rozvrh pro 15 tříd, 23 učeben a 22 učitelů. Ze základní školy ZŠ Komenského byl získán rozvrh pro školní rok 2010/2011 pro 12 tříd, 18 učeben a 20 učitelů a také rozvrh pro školní rok 2011/2012 pro 11 tříd, 17 učeben a 20 učitelů.

Přestože se rozvrhy podle uvedených vlastností mohou zdát dosti podobné, ukázalo se, že si vyvinutý program snáze poradí s menšími rozvrhy ze školy ZŠ Komenského. Většina následujících testů byla proto spuštěna pro požadavky ze ZŠ Husova, aby mohlo být rozpoznáno více vlastností implementovaného programu. Získané výsledky byly pomocí rozvrhů ze ZŠ Komenského pouze ověřeny. Na přiloženém CD lze nalézt parametry výpočtů, pro které program s uvedenými rozvrhy dosahuje nejlepších výsledků.

6.1 Selekční mechanismus

V první i druhé fázi navrženého algoritmu bylo využito selekce jedinců pomocí turnajového mechanismu. Protože se návrh ani implementace obou selekčních mechanismů nijak zásadně neliší, je možné předpokládat, že budou vykazovat i srovnatelné vlastnosti. Z toho důvodu je v této kapitole uvedeno pouze testování selekčního mechanismu z první fáze algoritmu. Výsledky testování je však možné interpretovat pro obě fáze výpočtu.

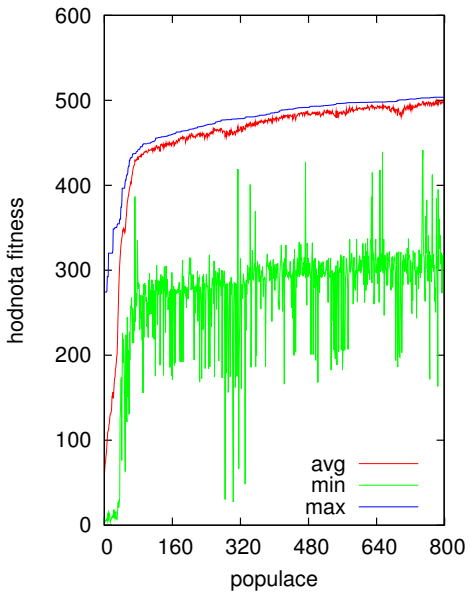
Podle návrhu turnajového mechanismu selekce, který byl popsán v kapitole 3.3.2, je ve výjimečných případech možné, aby turnaj vyhrál i slabší jedinec. I ten nejslabší jedinec v aktuální populaci má díky tomu malou šanci uspět a tak přenést své geny do následující populace. V algoritmu lze nastavit pravděpodobnost, s jakou je k reprodukci vybrán

[1] Informace o obou uvedených školách spolu s aktuálními rozvrhy je možno nalézt na jejich webových stránkách `http://zshusova.cz` a `http://zskonam.cz`.

slabší jedinec (testování předpokládá pouze dva jedince v turnaji). Pro nastavené hodnoty pravděpodobnosti v intervalu $\langle 0; 0,5)$ je díky tomu možné ovládat poměr selekčního tlaku (rychlosti konvergence) ku rozmanitosti populace. Při nastavení pravděpodobnosti přesně na hodnotu $0,5$ jsou jedinci vybíráni zcela náhodně a vlivy selekčního mechanismu jsou potlačeny. Pro pravděpodobnost vyšší než $0,5$ je dokonce funkce selekčního mechanismu obrácena a kvalita populace bude divergovat.

Implementovaný selekční mechanismus by měl být funkční v případě, že výsledky měření budou odpovídat zmíněným předpokladům v této kapitole. Pro testování byly použity požadavky na rozvrh školy ZŠ Husova z roku 2011/2012 s velikostí populace 300 jedinců a výpočtem o délce 800 evolucí.

Na obrázku 6.1 lze nalézt graf vývoje populace v čase pro hodnotu pravděpodobnosti vítězství slabšího jedince 0 (vždy tedy vyhrává silnější jedinec). Ta značí nejvyšší selekční tlak a zároveň nejnižší rozmanitost populace, kterou lze tímto parametrem ovlivnit. Graf obsahuje křivky pro vývoj ohodnocení nejlepšího jedince v populaci[2], nejhoršího jedince v populaci a průměrné ohodnocení populace. Je na nich možné vidět, že díky velkému selekčnímu tlaku průměrná kvalita populace rychle stoupala a byli také brzy eliminováni slabší jedinci. Lze také pozorovat, že kvalita nejlepšího jedince v populaci stoupá plynule. Tato vlastnost je způsobena sníženou rozmanitostí populace a ukazuje na to, že je málo pravděpodobné, že nově vznikající jedinci budou od jedinců v nynější populaci výrazněji rozdílní. Tuto vlastnost je možné interpretovat jako negativní, protože může způsobit předčasné uváznutí algoritmu v lokálním extrému optimalizované funkce.

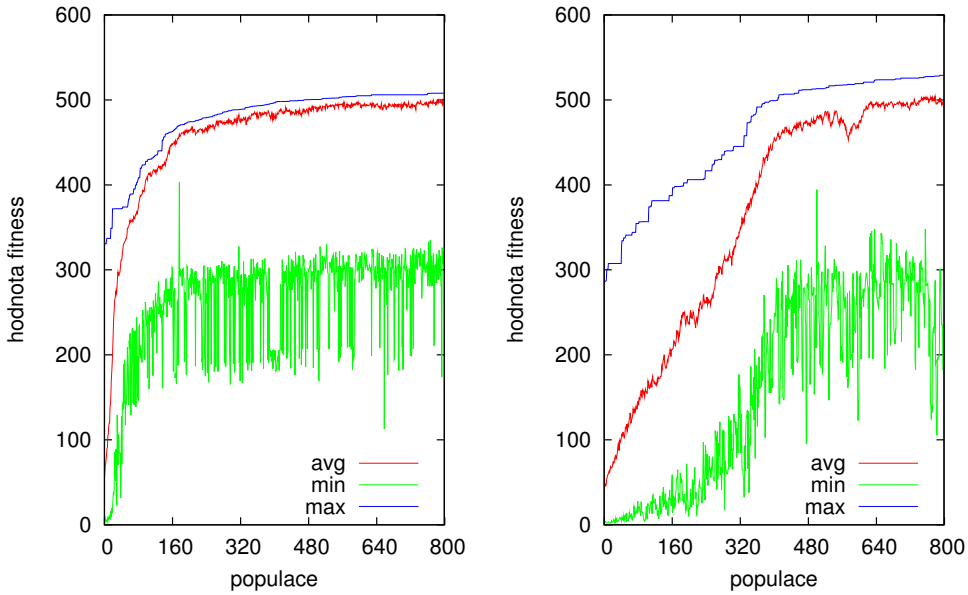


Obrázek 6.1: Graf vývoje populace v čase pro nulovou pravděpodobnost výhry slabšího jedince v turnaji.

Na obrázku 6.2 jsou uvedeny grafy vývoje populace v čase pro pravděpodobnost vítězství jedince $0,1$ a $0,35$. Oproti grafu na obrázku 6.1 na nich lze vidět vyšší rozmanitost populace (větší vzdálenosti mezi vývojovými křivkami) na úkor snížení rychlosti konvergence algoritmu. První popsaný příklad s pravděpodobností 0 vygeneruje dříve kvalitní jedince, které

[2]Ve všech grafech vývoje populace užitých v této práci je z důvodu užití elitismu křivka ohodnocení nejlepšího jedince neklesající funkcí.

v dalším průběhu jen nepatrně vylepšuje. Naopak algoritmus s rozmanitějším nastavením dosáhne kvalitních jedinců později, avšak má snahu je po celou dobu evoluce výrazněji zlepšovat. Na konci výpočtu tak druhé dva případy dosáhnou kvalitnějších jedinců.



Obrázek 6.2: Grafy vývoje populace v čase. Levý graf platí pro pravděpodobnost $0,1$ výhry slabšího jedince v turnaji, pravý pro $0,35$.

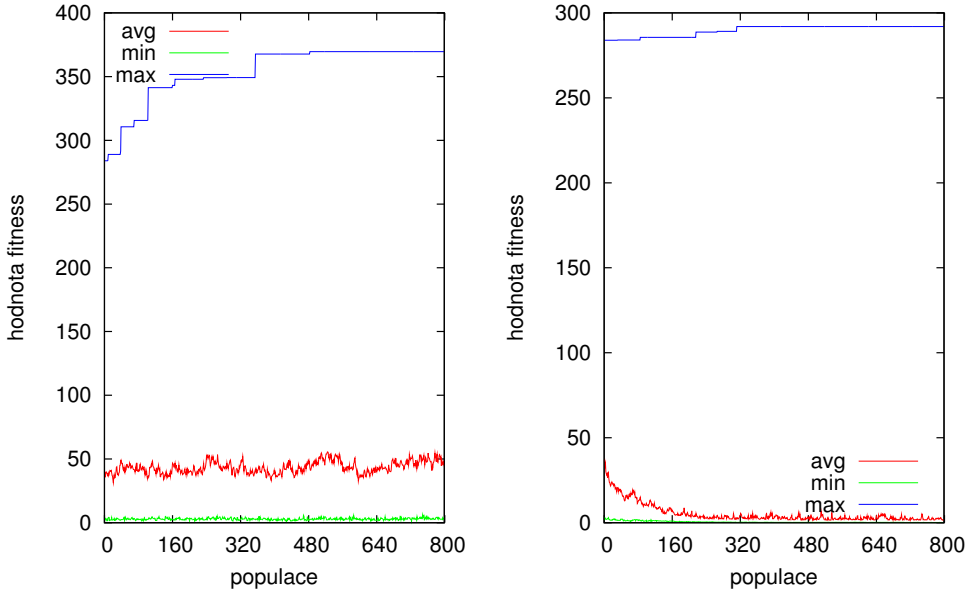
Jako poslední jsou na obrázku 6.3 uvedeny grafy pro nastavení pravděpodobnosti $0,5$ a $0,6$. První z nich ukazuje vývoj populace při potlačení selekčního mechanismu. Takováto populace se nijak nevyvíjí a ohodnocení jejích jedinců stagnuje na úrovni náhodně vygenerovaných jedinců z počáteční populace. Druhý značí obrácení funkce selekčního mechanismu. Kvalita populace s tímto nastavením podle předpokladů diverguje.

Provedené testy potvrdily předpoklady o selekčním mechanismu uvedené v první části této kapitoly. Z toho důvodu se dá považovat zvolená implementace selekčního mechanismu za funkční.

6.2 Křížení odzadu

V kapitole 4.2.4 byla vyslovena myšlenka, že v operátoru křížení první fáze algoritmu by pro urychlení evoluce mohlo být vhodné střídat směr výběru vektorů a učinit tak křížení dynamičtějším. Jak však ukázaly testy, změna směru výběru vektorů ke křížení celé křížení několikanásobně zpomalila.

Zpomalení zapříčinil ztížený výpočet v metodě `accomplishWithMinConflifts()` (viz kapitola 5.3), způsobený vyšším zaplněním počátečních vektorů v jedinci. Tyto vektory totiž ve většině případů z důvodu usměrnění výpočtu obsahují větší množství vyučovaných hodin než vektory na konci (viz kapitola 4.2.2). Pokud jsou jedinci kříženi od zadních vektorů, jsou tyto poloprázdné vektory umístěny beze změny do nového jedince a už do nich nebude přidána žádná skupina předmětů. Z toho důvodu zůstanou pro opravnou metodu takové vektory, které musí být často zcela zaplněny. Výpočet v opravné metodě se tak musí o mnoho častěji navracet a je podstatně náročnější.



Obrázek 6.3: Grafy vývoje populace v čase. Levý graf platí pro pravděpodobnost $0,5$ výhry slabšího jedince v turnaji, pravý pro $0,6$.

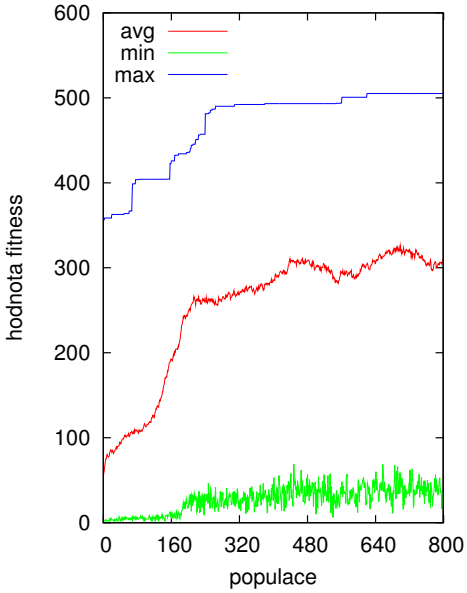
Aby doba výpočtu nebyla zbytečně prodlužována a aby zároveň zůstala možnost požadované změny prvního vektoru vykříženého jedince, bylo přistoupeno ke kompromisu. Při křížení není výběr směru úplně náhodný, ale pravděpodobnost výběru křížení jedinců odzadu je znevýhodněna. Míru znevýhodnění je možno nastavit parametrem výpočtu (viz příloha C).

K testování byla použita populace čítající 300 jedinců, pro kterou bylo vypočítáno 800 generací. Jako požadavky na rozvrh byla použita data z roku 2011/2012 ze ZŠ Husova. Byly provedeny tři testy. První z nich křížil odzadu každé druhé křížení, druhý každé čtyřicáté a poslední každé desetitisícé. Pro informaci lze uvést, že doba výpočtu prvního zmíněného byla oproti poslednímu přibližně desetinásobná.

Graf vývoje populace v čase pro každé druhé křížení odzadu je zobrazen na obrázku 6.4. Na něm je možno vidět, že změna směru křížení nemá na vývoj populace očekávaný vliv – rychlost konvergence populace je snížena a také nejsou eliminováni nekvalitní jedinci v ní. Nejenže byl tedy tento výpočet časově výrazně náročnější, ale jeho výsledek je navíc méně kvalitní. Tento fakt je způsoben tím, že opravný mechanismus aplikovaný po křížení si neumí moc dobře poradit s doplněním zcela zaplněných vektorů, musí v nich provést výrazné změny, a tak v nich zanechá jen málo obsažené genetické informace. Ve výsledku takto upravený operátor aplikovaný při každém druhém křížení působí spíše jako operátor mutace.

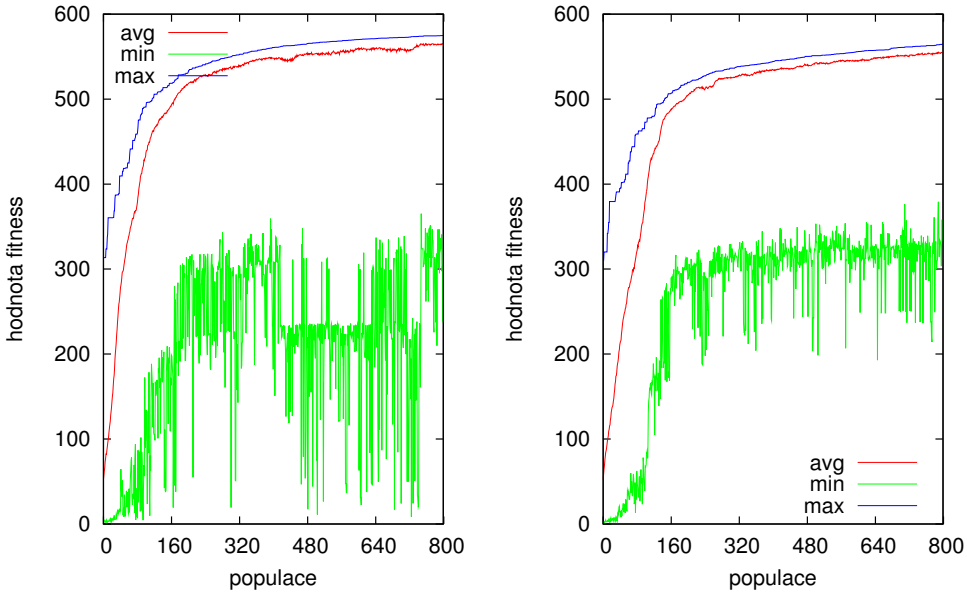
Na dalších dvou grafech z obrázku 6.5 je ukázán vývoj kvality populace pro zbylé dvě hodnoty analyzovaného parametru. Užití změny směru při každém čtyřicátém křížení se zdá jako vhodný kompromis, který umožní změnu prvního vektoru v jedinci a zároveň výrazně neovlivní celkovou dobu výpočtu. Změna směru výběru vektorů při každém desetitisícém křížení zmíněnou modifikaci operátoru výrazně potlačuje a toto měření je tak možno interpretovat shodně, jako kdyby změna směru výběru nebyla aplikována nikdy.

Maximální a průměrná křivka na obou grafech z obrázku 6.5 je velmi podobná, rozdíl lze však nalézt v křivce značící ohodnocení nejhoršího jedince v populaci. Tato křivka má



Obrázek 6.4: Graf vývoje populace první fáze výpočtu při použití zpětného výběru vektorů v každém druhém křížení.

podstatně větší rozptyl při častějším použití zpětného křížení. Tento rozptyl vzniká právě v důsledku užití zpětného křížení, kdy operátor výrazně změní počáteční vektory v jedinci. Z toho důvodu se dá předpokládat, že tento operátor zvyšuje rozmanitost jedinců v populaci s malým vlivem na rychlost konvergence algoritmu.



Obrázek 6.5: Grafy vývoje populace první fáze výpočtu při použití zpětného výběru vektorů v každém čtyřicátém křížení (vlevo) a každém desetitisícém (vpravo).

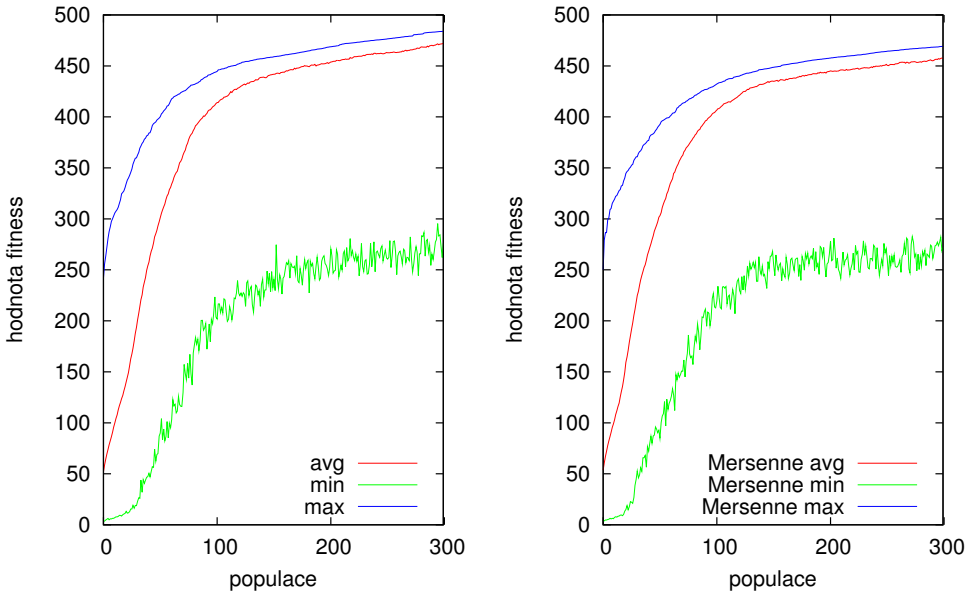
Provedený test prokázal, že operátor křížení se zpětným výběrem vektorů nemá přímo očekávané vlastnosti, avšak pokud je použit v menší míře, může vhodně přispět ke zvýšení

rozmanitosti jedinců v populaci s malým vlivem na rychlost výpočtu a rychlost konvergence algoritmu.

6.3 Pseudonáhodný generátor čísel

V průběhu výpočtu se mnoho rozhodnutí zakládá na náhodě. Dá se proto předpokládat, že použití nekvalitního pseudonáhodného generátoru čísel by mohlo negativně ovlivnit průběh celého algoritmu. V případě testování může být také vhodné, aby bylo chování generátoru pod úplnou kontrolou programátora. Z těchto důvodů je před spuštěním programu možno zvolit, zdali se použije generátor překladače jazyka C++, nebo vlastní generátor typu Mersenne Twister obsažený v programu. Implementace tohoto generátoru byla převzata z [4].

Pro testování byl použit překladač g++ ve verzi 4.4.5. Bylo spuštěno dvacet běhů programu s jedním i druhým generátorem. Poté byly jejich vývoje populací v čase zprůměrovány. Průměrné grafy pro první fázi výpočtu lze nalézt na obrázku 6.6. Protože výsledky měření pro první a druhou fázi výpočtu jsou shodné, je v následujícím textu popsán pouze výsledek z první fáze. Závěry z něj vyvozené však lze shodně interpretovat i pro fázi druhou.



Obrázek 6.6: Zprůměrované grafy vývoje populace první fáze pro dvacet běhů výpočtu. Levý graf vznikl při použití generátoru čísel překladače g++. Graf napravo vznikl při použití generátoru čísel Mersenne Twister.

Na uvedeném obrázku lze vidět, že oba grafy se liší jen minimálně. Dá se předpokládat, že vzájemné nuance v jejich průběhu jsou způsobeny nižším počtem spuštěným běhu (dvacet). Z toho lze vyvodit, že kvalita obou testovaných generátorů není natolik rozdílná, aby ovlivnila průběh evoluce.

6.4 Umístění předmětů v rozvrhu podle jejich náročnosti

V kapitole 4.3.3 byla popsána fitness funkce druhé fáze algoritmu spolu s jejími pravidly. Jedno z nich ovlivňuje umístění předmětů do rozvrhu podle jejich náročnosti. Předměty mají

nastavenou vysokou (hodnota 0 v následujících tabulkách), střední (1) nebo nízkou (2) náročnost, podle které jsou umístěny buďto na začátek, doprostřed, nebo nakonec vyučovacího dne. Předměty se střední náročností nejsou v pravidle nijak hodnoceny – předpokládá se, že hodnocení náročných a nenáročných předmětů si vynutí jejich umístění doprostřed dne.

Pro otestování chování pravidla byly znovu využity požadavky ZŠ Husova ze školního roku 2011/2012. Z důvodu přehlednějšího výstupu je dále uváděno rozmístění předmětů pouze pro nejvyšší 3 třídy, 8.B, 9.A a 9.B. Tyto třídy obsahují nejvíce předmětů, tudíž je vytvoření jejich rozvrhů nejsložitější a dá se předpokládat, že rozložení předmětů u nižších ročníků vykazuje stejné (případně lepší) vlastnosti.

Tabulka 6.1 obsahuje výstup prvního testu, který byl proveden se zvýšeným důrazem na rozmístění předmětů v rozvrhu podle jejich náročnosti. Uvedené rozvrhy ukazují, že rozmístění předmětů je uspokojivé. V některých dnech jsou předměty dokonce řazeny v neklesající posloupnosti a středně obtížné předměty jsou v mnoha případech vytlačeny podle předpokladů doprostřed vyučovacích dnů. V jiných případech jsou předměty se střední náročností umístěny mezi nenáročné (případně náročné) předměty z důvodu potřeby dodržet jiná omezení. Pouze v osmi případech dochází k přechodu důležitý-nedůležitý předmět.

8.B

0	1	2	1	1	2	
0	0	0	1	1	1	
0	0	0	1	1	1	2
0	1	0	0	2		
0	0	2	2	1	1	1

9.A

0	0	1	1	2	2	2
0	0	0	1	0	2	2
0	0	1	2	1	1	2
0	1	2	2	2	2	
0	0	2	2	1		

9.B

0	1	0	1	2	2	2
0	1	1	1	0	2	2
0	0	2	2	0	1	2
1	0	2	2	2	2	
0	0	1	2	0		

Tabulka 6.1: Rozmístění předmětů v rozvrhu s důrazem na jejich náročnost pro třídy 8.B, 9.A a 9.B.

Druhý provedený test ukazuje rozložení obtížnosti předmětů v případě, že jsou příslušná pravidla pro obtížnost předmětů ignorována (mají nastavenu váhu 0). Vygenerované rozvrhy jsou v tomto případě z pohledu řazení předmětů podle obtížnosti zcela náhodné. Popsanou situaci lze vidět v tabulce 6.2, která obsahuje vygenerované rozvrhy z druhého testu. Zatímco předchozí tři rozvrhy z prvního testu obsahovaly 8 přechodů důležitý-nedůležitý předmět, tyto rozvrhy obsahují 21 přechodů.

8.B

1	2	0	1	1	2	1
1	1	0	0	1	0	
1	0	0	0	0	1	
1	2	0	1	0	2	
1	0	0	2	2	1	

9.A

2	2	1	0	2	2	0
0	0	2	2	0	1	2
1	1	2	0	0	2	
2	0	1	2	1	1	
2	0	2	1	0	0	

9.B

2	2	1	0	2	2	0
1	0	1	2	0	0	2
0	1	2	2	0	2	
2	0	0	2	0	1	
2	1	2	1	1	0	

Tabulka 6.2: Rozmístění předmětů v rozvrhu bez ohledu na jejich náročnost pro třídy 8.B, 9.A a 9.B.

Uvedené testy prokázaly, že předpoklad o vytlačení předmětů se středí obtížností doprostřed vyučovacích dnů byl správný a že pravidla pro rozmístění předmětů podle jejich obtížnosti fungují.

Kapitola 7

Porovnání programu s existujícími aplikacemi

Kapitola obsahuje porovnání implementovaného programu s jinými aplikacemi, které jsou dostupné na trhu. Výhodou všech dále uvedených aplikací je, že obsahují okenní uživatelské rozhraní, které umožňuje jednodušší nastavení vstupních požadavků. Příloha E obsahuje tabulku s porovnáním vlastností všech uvedených programů.

V následujících podkapitolách není uveden program Bakaláři, přestože také obsahuje možnost automatické tvorby rozvrhu. Ta však probíhá pomocí deterministického algoritmu, který vyžaduje asistenci uživatele. Jedná se tak o jinou kategorii generátoru, kterou není vhodné srovnávat s ostatními dále uvedenými programy [3].

7.1 Tvůrce rozvrhů

Program pro tvorbu školních rozvrhů vytvořený jako diplomová práce na Univerzitě Pardubice v roce 2004. Od té doby prochází nepravidelným vývojem až do testované verze 3.1, která je dostupná zdarma. Aplikace funguje na principu minimalizace výskytu těžkých i lehkých konfliktů v rozvrhu. Náhodně generované rozvrhy v jeho počáteční populaci tedy obsahují i rozvrhy, které porušují tvrdá omezení. Tyto rozvrhy jsou však v průběhu evoluce brzy eliminovány a poté se algoritmus výpočtu snaží minimalizovat počet lehkých konfliktů. V rámci genetického algoritmu je použit opravný operátor mutace, který má za úkol cíleně vylepšovat jedince v populaci. Výstupní formát rozvrhů má formu HTML dokumentu.

Před spuštěním výpočtu není možné nastavit váhy jednotlivým typům konfliktů a tím ovlivnit vlastnosti generovaného rozvrhu. Je zde naopak možnost nastavit parametry genetického algoritmu, což však pro neznalého uživatele nemusí být jednoduché, a vstupní data s rozmanitou množinou požadavků na rozvrh. Ta obsahuje například možnost definovat délku rozvrhu, dobu polední přestávky a další parametry pro každou třídu zvlášť. Na druhou stranu program neobsahuje podporu pro výuku půlených hodin a není v něm možné vytvořit rozvrhy pro učebny. Vygenerovaný rozvrh také není možné dodatečně ručně upravovat.

Absenci podpory učeben a půlených hodin je v malé míře možné vyřešit ručním umístěním problematických předmětů na pevné místo v rozvrhu, avšak pro běžnou školu je takovýchto předmětů velké množství a jejich umístění je právě tím nejtěžším problém při tvorbě rozvrhu [11].

7.2 aSc Rozvrhy

Komplexní mezinárodní komerční aplikace s českou lokalizací. Premium verze dokonce umožňuje analyzovat vstupní data a tak objevit případné nesrovnalosti před spuštěním generátoru. Aplikace samozřejmě obsahuje podporu pro půlené hodiny, učebny a v nejvyšší verzi také pro jednotlivé žáky. Pro každého z nich je díky tomu možné nastavit a vygenerovat individuální rozvrh. I z důvodu běhu výpočtu na více jádrech procesoru je automatické generování rozvrhu velmi rychlé. Rozvrh je po jeho vygenerování možné ručně upravit, program přitom asistuje hlídáním kolizí a napovídáním.

Ovládání celého programu je pro neznalého uživatele značně intuitivní. Tomu napomáhá i fakt, že program neobsahuje složité tabulky pro zadání parametrů výpočtu, ale pouze tříhodnotový parametr udávající snahu, kterou algoritmus vyvine pro dosažení lepšího výsledku. Druhým a posledním parametrem výpočtu je parametr udávající, jak moc může generátor porušit požadavky rozvrhu (lehké kolize). Z důvodu zachování obchodního tajemství není zveřejněn podrobnější princip fungování generátoru. Celkový dojem z programu může lehce pokazit místy nekvalitní (chybějící) český překlad [2].

7.3 FET Free Timetabling Software

FET je open source program vytvořený v roce 2002 šířený pod GNU GPL licencí. Neobsahuje českou lokalizaci a jeho ovládání není ve srovnání s aSc Rozvrhy tak intuitivní, působí spíše programátorsky. Program obsahuje rozmanitou škálu požadavků na generovaný rozvrh. Od verze 5.0.0 (rok 2007) nevytváří rozvrhy pomocí genetického algoritmu. Ten byl totiž nahrazen rekurzivním algoritmem, který simuluje ruční vytváření rozvrhu člověkem. Tento algoritmus obsahuje jen jeden parametr výpočtu, kterým je možné ovlivnit jeho chování – jedná se o číslo inicializující počáteční hodnotu náhodného generátoru čísel.

Program obsahuje podporu pro půlené předměty, učebny a dokonce i budovy. Existují jeho varianty, které jsou komunitou upraveny pro zvláštní potřeby konkrétních škol v konkrétních státech. Jako jeho nevýhoda se může jevit rozsáhlé uživatelské rozhraní, které znalému uživateli může vyhovovat, avšak běžnému uživateli, který s programem pracuje jeden týden v roce, může připadat složité a těžkopádné. Program také neumožňuje jednoduché ruční úpravy vygenerovaného rozvrhu [8].

7.4 Untis Timetabling Programs

Rozsáhlá komerční aplikace bez české lokalizace. Podobně jako aSc Rozvrhy i ona obsahuje několik verzí, ke kterým je navíc možné přikoupit balíčky podporující například webovou agendu, dozory o přestávkách a další. Ani rozhraní tohoto programu nepůsobí tolik intuitivně jako v případě zmíněných aSc Rozvrhů. To je způsobeno hlavně velkým rozsahem možností, které uživatelské rozhraní nabízí. Na druhou stranu program obsahuje relativně přehledné rozhraní pro přidělování vah vlastnostem výsledného rozvrhu.

Z důvodu ochrany obchodního tajemství není ani v tomto případě zveřejněn přesný popis algoritmu, který se stará o generování rozvrhu. Je pouze uvedeno, že rozvrhy jsou generovány za spolupráce genetického a horolezeckého algoritmu a dalších blíže nepopsaných metod. Před spuštěním automatické tvorby je možné do rozvrhu napevno vložit vybrané předměty, jejichž umístění algoritmus zachová. Po vygenerování rozvrhu je ho možné s asistencí programu dodatečně upravit [20].

Kapitola 8

Závěr

V této práci byl proveden rozbor problému tvorby rozvrhů na základních školách. Pro automatické vytváření rozvrhů byl navržen dvoufázový hybridní genetický algoritmus založený na myšlence co nejvíce omezit stavový prostor řešeného problému. Podle jeho návrhu byla vytvořena konzolová aplikace v jazyce C++. Ta mimo jiné umožňuje uživateli zadat požadavky na půlené hodiny, obsazení učeben nebo na tzv. „dvouhodinovky“. Aby tyto požadavky nemusel zadávat uživatel ručně, byla vytvořena ještě druhá aplikace, která se stará o konverzi dat z u nás běžně používaného školního systému Bakaláři [3, 21] do vstupního XML souboru aplikace. Po dokončení výpočtu jsou vygenerovány rozvrhy v HTML formátu pro třídy, vyučující a učebny.

Implementovaná aplikace je pro menší a střední školy schopna nabídnout uživateli jeden nebo více rozvrhů s přijatelnými vlastnostmi. Myšlenka rozdělení výpočtu do dvou oddělených fází se tedy pro řešený problém jeví jako vhodná. Doba výpočtu se v případě náročného nastavení pohybuje na běžném osobním počítači v řádu několika málo hodin. Při porovnání s ostatními aplikacemi na trhu se ukázalo, že svojí rychlostí a uživatelským komfortem nemůže konkurovat placeným komerčním produktům, které jsou navíc schopny vytvořit přijatelné rozvrhy i pro větší školy.

Největší nedostatek vyvinuté aplikace se ukrývá mimo navržený algoritmus. Jedná se o absenci možnosti dodatečné ruční úpravy vygenerovaného rozvrhu uživatelem. Aplikace by tak díky tomu mohla navrhnout uživateli dostatečně dobrý rozvrh, který by si sám poupravil do pro něj ideálního stavu. Vytvořit však kvalitní a intuitivní ruční editor rozvrhu, který by hlídal kolize předmětů a navrhoval jejich nejlepší umístění, by svojí náročností vydalo na další podobnou práci.

Aplikace by mohla být v budoucnu rozšířena o grafické uživatelské rozhraní, které by umožnilo její jednoduché ovládání i běžnými uživateli. Ze stejného důvodu by bylo také vhodné navrhnout automatický kontrolní mechanismus, který by sledoval průběh výpočtu a adaptivně přizpůsoboval parametry genetického algoritmu spolu s vahami pravidel jeho aktuálnímu stavu. Nejen, že by díky tomu bylo dosaženo lepších výsledků v kratším čase, ale kontrolní mechanismus by hlavně umožnil plnohodnotné použití aplikace i těm uživatelům, kteří nemusí vědět, jak parametry genetického algoritmu nastavit.

Vlastní algoritmus by šlo rozšířit o podporu pravidel pro tvorbu rozvrhů podle preferencí učitelů, kterým by mohl být vytvořen také rozvrh dozorů o přestávkách tak, aby učitelé měli dozor pouze v okolí těch hodin, kdy mají sami vyučování.

Algoritmus by mohl být také rozšířen o podporu předmětů, které musí být umístěny na konci vyučovacího dne. V průběhu testování se totiž ukázalo, že v některých školách existují předměty, které jsou vyučovány pouze v polovině třídy (druhé polovině žáků končí

vyučování). Takový předmět může být umístěn právě pouze na konci nebo v některých případech na začátku vyučovacího dne.

V neposlední řadě by bylo v budoucnu možné rozšířit závěr obou genetických algoritmů o metodu, která by se pokusila více přiblížit nejlepší nalezené řešení nejbližšímu lokálnímu optimu. Jako vhodná metoda se nabízí například horolezecký algoritmus, zmíněný v kapitole 1.

Literatura

[1] Aguado, F.; Doncel, J.; Molinelli, J.; aj.: Certified Genetic Algorithms: Crossover Operators for Permutations. In *Computer Aided Systems Theory – EUROCAST 2007*, *Lecture Notes in Computer Science*, ročník 4739, Springer Berlin / Heidelberg, 2007, ISBN 978-3-540-75866-2, s. 282–289.

[2] aSc Rozvrhy [online]. Dostupné na: <http://www.asctimetables.com/>, 2012 [cit. 2012-05-10].

[3] Bakaláři [online]. Dostupné na: <http://www.bakalari.cz/rozvrh.aspx>, 2012 [cit. 2012-03-04].

[4] Bedaux, J.: C++ Mersenne Twister Pseudo-Random Number Generator [online]. Dostupné na: <http://www.bedaux.net/mtrand/>, 2003 [cit. 2012-05-06].

[5] Buckland, M.: *AI techniques for game programming*. Premier Press, 2002, ISBN 1-931841-08-X.

[6] Darwin, C.: *O vzniku druhů přírodním výběrem*. Academia, 2007, ISBN 978-80-200-1492-4.

[7] Fang, H.-L.: *Genetic Algorithms in Timetabling and Scheduling*. Dizertační práce, Department of Artificial Intelligence, University of Edinburgh, 1994.

[8] FET Free Timetabling Software [online]. Dostupné na: <http://www.lalescu.ro/liviu/fet/>, 2012-05-10 [cit. 2012-05-10].

[9] Goldberg, D. E.; Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, Morgan Kaufmann, 1991, s. 69–93, ISBN 1-55860-170-8.

[10] Hynek, J.: *Genetické algoritmy a genetické programování*. Grada Publishing, a.s., 2008, ISBN 978-80-247-2695-3.

[11] Jaroš, P.: Tvůrce rozvrhů [online]. Dostupné na: <http://rozvrhy.jaros.in>, 2005 [cit. 2012-05-10].

[12] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evolučné algoritmy*. STU, Bratislava, 2000, ISBN 80-227-1377-5.

[13] Lou, M.: Pimp Your Tables with CSS3 [online]. Dostupné na: <http://tympanus.net/codrops/2010/05/03/pimp-your-tables-with-css3/>, 2010-05-03 [cit. 2012-05-04].

[14] Nařízení vlády, Sbírka zákonů č. 75/2005 [online]. Dostupné na: <http://www.msmt.cz/uploads/soubory/narizeni75_2005.pdf>, 2005-01-26 [cit. 2012-01-05].

[15] Obitko, M.: Introduction to Genetic Algorithms [online]. Dostupné na: <http://www.obitko.com/tutorials/genetic-algorithms/>, 1998-08-01 [cit. 2011-12-31].

[16] Rámcový vzdělávací program pro základní vzdělávání [online]. Dostupné na: <http://www.msmt.cz/uploads/soubory/zakladni/SP_RVPZV_2007.zip>, 2007-07-01 [cit. 2012-01-05].

[17] Sigl, B.; Golub, M.; Mornar, V.: Solving Timetable Scheduling Problem by Using Genetic Algorithms. In *Information Technology Interfaces, 2003. ITI 2003*, 2003, s. 519 – 524, ISBN 953-96769-6-7.

[18] Studnička, V.: *Genetické algoritmy - multi-core CPU implementace*. Diplomová práce, Vysoké učení technické v Brně, 2010.

[19] Thomason, L.: TinyXML [online]. <http://sourceforge.net/projects/tinyxml/>, 2011-11-01 [cit. 2012-03-12].

[20] Timetable Software Untis [online]. Dostupné na: <http://www.grupet.at/>, 2012-05-10 [cit. 2012-05-10].

[21] Výsledky šetření o vybavenosti evidenčním SW na školách [online]. Dostupné na: <http://www.uiv.cz/soubor/1112>, 2004 [cit. 2012-03-24].

[22] Wang, H.; Li, S.; Oyama, S.; aj. (editoři): *WAIM'11: Proceedings of the 12th international conference on Web-age information management*, Berlin, Heidelberg: Springer-Verlag, 2011, ISBN 978-3-642-23534-4, ISSN 0302-9743.

[23] Willemen, R. J.: *School timetable constriction : Algorithms and complexity*. Technische Universiteit Eindhoven, Eindhoven, 2002, ISBN 90-386-1011-4.

Příloha A

Slovník termínů přejatých z biologie

V této příloze jsou uvedeny vybrané pojmy z oblasti evolučních algoritmů, které jsou v práci použity. Jejich popis vychází z publikací [10] a [18].

Jedinec reprezentuje jedno ohodnocené řešení zadaného problému. V našem případě se jedná o možné rozvrhy pro celou školu.

Populace genetického algoritmu je množina jedinců, kteří jsou spolu v kontaktu. Mohou spolu být kříženi a tak vytvářet jedince, kteří se umístí do následující populace.

Chromozom jedince reprezentuje způsob jeho kódování, je složen z množiny genů. Každému jedinci přísluší právě jeden chromozom.

Gen je základní jednotka informace, která kóduje vlastnosti jedince.

Alela je konkrétní forma genu, tedy jedna z hodnot, kterých může gen nabývat.

Genotyp je soubor všech genetických informací o jedinci.

Genom je soubor genů buňky jedince. V našem případě jednobuněčného jedince je ekvivalentní genotypu.

Fenotyp je soubor všech pozorovatelných vlastností jedince. Je vytvořen působením prostředí na genotyp.

Účelová funkce reprezentuje vnější prostředí (řešený problém), a tak umí ohodnotit kvalitu jedince v něm.

Fitness funkce určuje kvantitativní míru schopnosti jedince přežít a vstupovat do reprodukčního procesu. Lze ji získat transformací účelové funkce.

Selekce určuje způsob výběru jedinců ke křížení.

Křížení je proces vytváření potomka z vybraných předků.

Mutace je náhodná změna jednoho nebo více genů jedince.

Příloha B

Popis vstupního XML souboru s požadavky na rozvrh.

Ukázka jednoduchého vstupního souboru s požadavky na rozvrh. Popis jeho použití lze nalézt v kapitole 4.1. Z důvodu, že se jedná o velmi zjednodušenou ukázku, tak uvedený XML soubor (přestože je syntakticky i sémanticky validní) není možné beze změny použít jako vstup vytvořenému programu. Je to hlavně z toho důvodu, že soubor obsahuje nedostatek záznamů a nebylo by tak možné vytvořit alespoň minimální rozmanitost v populaci, kterou algoritmus pro svůj běh vyžaduje (viz kapitola 5.3). V uvedené ukázce jsou předvedeny všechny základní možnosti vstupních požadavků.

Každý dále uvedený záznam je možné po změně atributů na svém místě duplikovat, případně kombinovat atributy jednoho záznamu s ostatními. Je však nutné zachovat pořadí jednotlivých skupin definic – nejdříve musí být uvedeni učitelé, poté třídy, učebny a nakonec předměty, které vytváří vazby mezi předchozími třemi.

Definice učitelů, tříd a učeben obsahují vždy pořadové číslo daného typu záznamu. Záznamy jsou číslovány vzestupně od nuly. Podle těchto čísel se poté vytváří vazby při definici předmětů. Jistě by bylo možné tato čísla vypustit a číslovat záznamy automaticky při čtení souboru, avšak s ohledem na lepší čitelnost souboru byla zvolena varianta s povinným číslováním.

```
<?xml version="1.0" ?>
<input>
  definice učitelů
  <teachers>
    první učitel se jménem 'Nov' (Novák) a číslem 0
    <teacher id="0" name="Nov" />
    druhý učitel se jménem 'Svo' (Svobodová) a číslem 1
    <teacher id="1" name="Svo" />
  </teachers>

  definice učeben
  <classrooms>
    první učebna se jménem a číslem
    <classroom id="0" name="učebna 1" />
    druhá učebna se jménem a číslem
    <classroom id="1" name="učebna 2" />
```

```
    specializovaná učebna se jménem a číslem
    <classroom id="2" name="tělocvična" />
  </classrooms>

  definice tříd
  <classes>
    první třída se jménem '1.A' s přiřazenou kmenovou učebnou číslo 0
    <class id="0" name="1.A" rootClassroom="0" />
    první třída se jménem '2.A' s přiřazenou kmenovou učebnou číslo 1
    <class id="1" name="2.A" rootClassroom="1" />
  </classes>

  definice skupin předmětů – provazují vyučující, učebny a třídy
  <subjects>
    skupina umístěna v rozvrhu 5krát v různých dnech (type="paralel")
    má vysokou důležitost (importance="0") → umístění na začátek dne
    <group times="5" type="paralel" importance="0">
      předmět matematiky ve třídě 0 vyučovaný učitelem 0 v kmenové učebně třídy
      kmenová učebna je automaticky doplněna při zadání atributu classroom="-1"
      hodnotu -1 nelze použít v případě, že je předmět vyučován ve více třídách
      <subject name="M" class="0" teacher="0" classroom="-1" />
    </group>

    skupina umístěna v rozvrhu 3krát v různých dnech (type="paralel")
    má střední důležitost (importance="1") → libovolné umístění
    <group times="3" type="paralel" importance="1">
      předmět angličtiny ve třídách 0 a 1 vyučovaný učitelem 0 v učebně 0 nebo 1
      atribut classroom="-1"zde není možné použít, nebylo by jasné, ke které třídě patří
      <subject name="Aj" class="0|1" teacher="1" classroom="0|1" />
      předmět němčiny ve třídách 0 a 1 vyučovaný učitelem 1 v učebně 0 nebo 1
      <subject name="Nj" class="0|1" teacher="0" classroom="0|1" />
    </group>

    skupina umístěna v rozvrhu 2krát v jednom dni po sobě (type="serial")
    má nízkou důležitost (importance="2") → umístění na konci dne
    <group times="2" type="serial" importance="2">
      výtvarná výchova ve třídě 0 vyučovaná učitelem 1 v kmenové učebně třídy
      <subject name="Vv" class="0" teacher="1" classroom="-1" />
    </group>

    skupina umístěna v rozvrhu 2krát v různých dnech (type="paralel")
    má nízkou důležitost (importance="2") → umístění na konci dne
    <group times="2" type="paralel" importance="2">
      tělesná výchova ve třídě 1 vyučovaná učitelem 0 v učebně 2
      <subject name="Tv" class="1" teacher="0" classroom="2" />
    </group>
  </subjects>
</input>
```

Příloha C

Popis vstupního XML souboru s parametry výpočtu.

Ukázka vstupního XML souboru s parametry výpočtu. Žádný z níže uvedených parametrů není povinný, lze tedy dodat i soubor s prázdným obsahem tagu `<settings>`. Pro nezadané parametry budou použity takové výchozí hodnoty, které jsou nastaveny v následujícím souboru.

```
<?xml version="1.0" ?>
<settings>
  — obecná nastavení —
  počet dní ve vyučovacím týdnu
  <number_of_days value="5" />
  maximální počet hodin ve vyučovacím dni
  <number_of_hours value="7" />
  minimální počet hodin ve vyučovacím dni
  <number_of_min_hours value="4" />
  použití pseudonáhodného generátoru čísel Mersenne Twister (hodnota 'no' nebo 'yes')
  <mersenne value="no" />

  — cesty k výstupním souborům —
  cesta k logovacímu souboru
  <log_general_path value="./logs/default" />
  cesta k souborům s daty grafů obou fází algoritmu
  je vytvářeno 6 souborů s příponami _favg, _fmin, _fmax, _savg, _smin a _smax
  <log_graph_path value="./plots/default_first" />
  cesta k výstupním HTML souborům
  jsou vytvářeny 3 soubory s příponami _classes, _teachers a _classrooms
  <log_output_path value="./outputs/default" />

  — parametry první fáze výpočtu —
  pravděpodobnost vítězství slabšího jedince v turnaji (hodnota v rozsahu od 0 do 100%)
  <fp_surprise_prob value="10" />
  pravděpodobnost mutace (hodnota v rozsahu od 0 do 100%)
  <fp_mutation_prob value="15" />
  pravděpodobnost kopie jedince bez použití křížení (hodnota v rozsahu od 0 do 100%)
```

```
<fp_copy_instead_cross_prob value="25" />
```
hodnota, která značí, že každý tolikátý jedinec bude křížen odzadu
```
<fp_cross_linear_backforward_prob value="40" />
```
počet jedinců v populaci
```
<fp_pop_size value="100" />
```
počet elitních jedinců v populaci
```
<fp_number_of_elitists value="1" />
```
počet evolucí (iterací algoritmu)
```
<fp_iterations value="200" />
```

— parametry ohodnocovacích pravidel první fáze výpočtu —
váha pravidla pro podobnou důležitost předmětů ve vektoru
```
<fp_eval_vector_importance_dispersion_weight value="10" />
```
váha pravidla pro vynucení vzestupného zaplnění vektoru
```
<fp_eval_vector_ascending_full_weight value="10" />
```
váha pravidla pro ohodnocení kvality přidělených tříd
```
<fp_eval_room_aloc_quality_weight value="5" />
```
poměr, o který má být více bezkonfliktních vektorů, než je počet hodin ve dni
```
<fp_eval_min_vectors_without_parallel_conflict_multiplier value="1.5" />
```

— parametry druhé fáze výpočtu —
pravděpodobnost vítězství slabšího jedince v turnaji (hodnota v rozsahu od 0 do 100%)
```
<sp_surprise_prob value="10" />
```
pravděpodobnost mutace (hodnota v rozsahu od 0 do 100%)
```
<sp_mutation_prob value="15" />
```
pravděpodobnost užití operátoru vylepšení (hodnota v rozsahu od 0 do 100%)
```
<sp_magic_prob value="10" />
```
pravděpodobnost kopie jedince bez použití křížení (hodnota v rozsahu od 0 do 100%)
```
<sp_copy_instead_cross_prob value="50" />
```
počet rodičovských jedinců při křížení
hodnota může být rozdílná od 2 pouze při použití operátoru pravděpodobnostního křížení
```
<sp_number_of_entits_to_cross value="2" />
```
počet jedinců v populaci
```
<sp_pop_size value="100" /> <!-- population size -->
```
počet elitních jedinců v populaci
```
<sp_number_of_elitists value="1" />
```
počet evolucí (iterací algoritmu)
```
<sp_iterations value="50" />
```

— parametry ohodnocovacích pravidel druhé fáze výpočtu —
váha pravidla pro minimalizaci výskytů kolizí mezi vektory
```
<sp_eval_collision_in_days_weight value="10" />
```
poměr sériových a paralelních kolizí pravidla pro minimalizaci jejich výskytů
```
<sp_eval_collision_in_days_single_to_multiple_ratio value="5.0" />
```
váha pravidla pro minimalizaci přítomnosti volných hodin v průběhu vyučování
```
<sp_eval_empty_hours_in_days_weight value="25" />
```
váha pravidla pro umístění předmětů podle jejich důležitosti
```
<sp_eval_subj_importance_position_weight value="10" />
```

```
  váha pravidla pro správné umístění sériových skupin
  <sp_eval_serial_group_satisfaction_weight value="10" />
  váha pravidla pro kontrolu stejnoměrné délky vyučování
  <sp_eval_avg_day_len_dispersion_weight value="10" />
</settings>
```

Příloha D

Popis empirického odvození algoritmu pro ohodnocovací pravidlo stavu zaplnění vektorů

V první fázi algoritmu je nutné získat takové vektory, které při umísťování v druhé fázi nebudou vytvářet mnoho volných hodin v průběhu vyučovacích dnů. Jak tyto vektory mají vypadat, je podrobněji popsáno v kapitole 4.2.3 u pravidla *zaplnění vektorů*.

Na stav zaplnění vektoru se dá dívat jako na binární řetězec o pevné délce, která odpovídá počtu tříd v rozvrhu. Pokud v dané třídě probíhá vyučování, je příslušný bit nastaven na hodnotu `1`, jinak `0`. Bity v pravé části řetězce odpovídají vyšším ročníkům, bity v levé části odpovídají naopak nižším. V tabulce D.1 je uvedeno všech 32 možných kombinací zaplnění binárního vektoru o délce 5.

I na takto relativně jednoduchém příkladu lze vidět, že snadné rozdělení vektorů na „vhodné“ a „nevhodné“ není možné. Pro naše potřeby je navíc vítané, abychom uměli ohodnotit kvalitu každého zaplnění vektoru reálným číslem. K řešení problému bylo přistoupeno empiricky tak, že všechny permutace uvedené v tabulce D.1 byly ručně ohodnoceny reálným číslem z intervalu $\langle 0; 1 \rangle$. Toto ohodnocení se zakládá na předpokladu, že vyšší ročníky mají více hodin týdně než nižší. Ručnímu ohodnocení odpovídají hodnoty v třetím sloupci tabulky.

Poté byly navrženy různé algoritmy a jejich modifikace, které určitým způsobem také ohodnotily všechny zadané permutace. Pro konečnou implementaci pravidla byl vybrán ten algoritmus, jehož ohodnocení se co nejvíce blížilo ručně vytvořenému ohodnocení. Ohodnocení vektorů pomocí zvoleného algoritmu je možné nalézt ve čtvrtém sloupci níže uvedené tabulky. Porovnání obou ohodnocení lze nalézt také graficky znázorněné na obrázku D.1.

Implementovaný algoritmus započítá vektoru bod za každou dvojici bitů, která splňuje jednu z následujících podmínek:

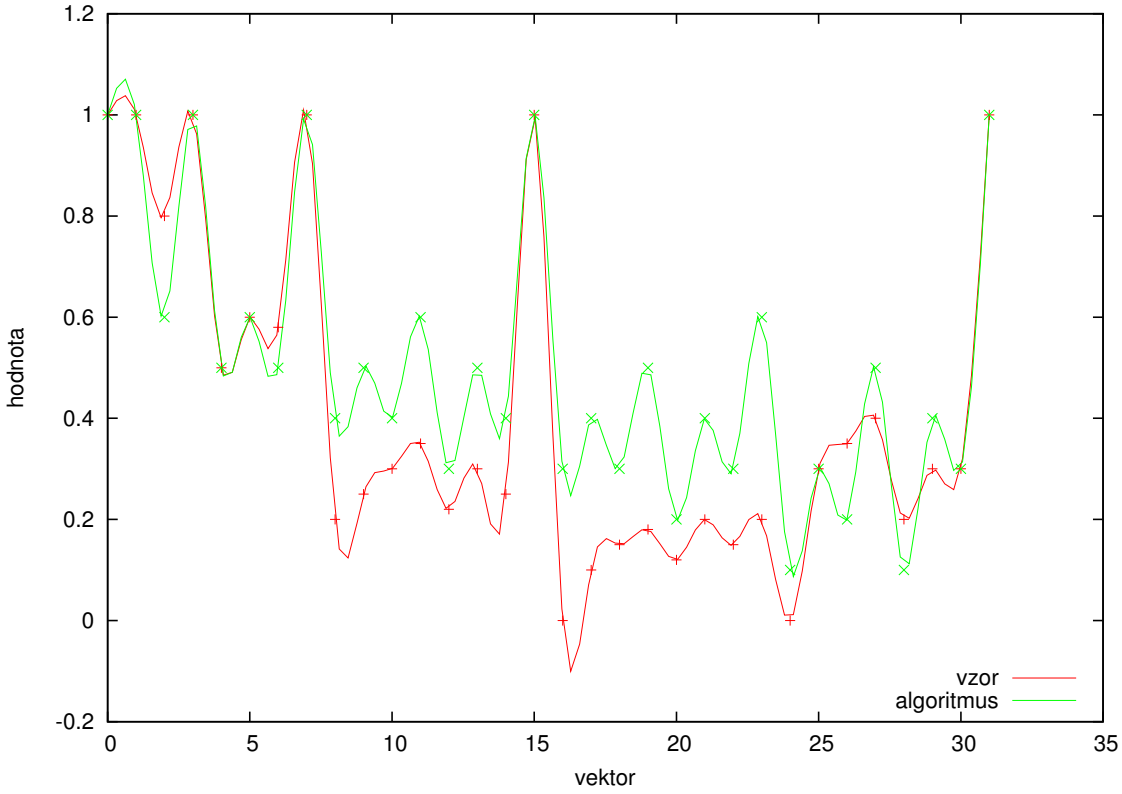
- Obě dvě hodiny nejsou obsazeny.
- Obě dvě hodiny jsou obsazeny.
- Hodina vyšší třídy je obsazena, když hodiny nižší třídy není.

Protože počet všech dvojic bitů ve vektoru je znám, je možné takto získanou hodnotu podělením normalizovat do intervalu $\langle 0; 1 \rangle$. Výsledná hodnota je navíc nelineárně upravena tak, že v případě, kdy vektor nezíská plný počet bodů, je jeho ohodnocení sníženo o 0,3

bodu. Díky tomu jsou více preferovány plné nebo prázdné vektory a výsledky algoritmu jsou bližší ručnímu ohodnocení.

pořadí	**permutace**	**ruční ohodnocení**	**ohodnocení algoritmem**
0	00000	1,0	1,0
1	00001	1,0	1,0
2	00010	0,8	0,6
3	00011	1,0	1,0
4	00100	0,5	0,5
5	00101	0,6	0,6
6	00110	0,58	0,5
7	00111	1,0	1,0
8	01000	0,2	0,4
9	01001	0,25	0,5
10	01010	0,3	0,4
11	01011	0,35	0,6
12	01100	0,22	0,3
13	01101	0,3	0,5
14	01110	0,25	0,4
15	01111	1,0	1,0
16	10000	0,0	0,3
17	10001	0,1	0,4
18	10010	0,15	0,3
19	10011	0,18	0,5
20	10100	0,12	0,2
21	10101	0,2	0,4
22	10110	0,15	0,3
23	10111	0,2	0,6
24	11000	0,0	0,1
25	11001	0,3	0,3
26	11010	0,35	0,2
27	11011	0,4	0,5
28	11100	0,2	0,1
29	11101	0,3	0,4
30	11110	0,3	0,3
31	11111	1,0	1,0

Tabulka D.1: Všechny možné stavy zaplnění vektoru s předměty o délce 5 a jejich ohodnocení.



Obrázek D.1: Všechny možné stavy zaplnění vektoru s předměty o délce 5 a jejich ohodnocení.

Příloha E

Tabulka porovnání programu s existujícími aplikacemi

Příloha obsahuje srovnávací tabulku aplikací pro tvorbu školních rozvrhů, které byly popsány v kapitole 7. Jsou porovnány následující vlastnosti:

- Uzamknutí umístění - popisuje, zda je v aplikaci možno nadefinovat pevné umístění předmětů, které algoritmus při výpočtu zachová.
- Ruční úpravy - popisuje, zda aplikace asistuje s dodatečnou ruční úpravou vygenerovaného rozvrhu.
- Půlené hodiny - udává, zda aplikace obsahuje podporu pro dělené hodiny.
- Učebny - udává, zda aplikace obsahuje podporu pro rozmístění předmětů do učeben.
- GUI - anglický termín *Graphical User Interface* udává, zda aplikace obsahuje grafické uživatelské rozhraní.
- Zdarma - udává, zda je aplikace pro školy dostupná zdarma.

	uzamknutí umístění	**ruční úpravy**	**půlené hodiny**	**učebny**	**GUI**	**zdarma**
vytvořený program	ne	ne	ano	ano	ne	ano
Tvůrce rozvrhů (verze 3.1)	ano	ne	ne	ne	ano	ano
aSc rozvrhy (verze 2012.14.6)	ano	ano	ano	ano	ano	ne
FET (verze 5.18.0)	ano	ne	ano	ano	ano	ano
Untis (verze Mar 21 2012)	ano	ano	ano	ano	ano	ne

Příloha F

HTML výstup programu.

Výstupem programu je množina rozvrhů pro třídy, učebny a vyučující v následujícím HTML/CSS formátu. U rozvrhů pro třídy je v závorce uváděno číslo učebny, ve které výuka probíhá. Rozvrh pro vyučující není z důvodu podobnosti s prvními dvěma uveden.

Třída: 6A (počet hodin: 31)

	1	2	3	4	5	6	7
Pondělí	Pp Nm (8)	Čj P (8)	M Z (8)	D Ně (8)	Inf Nm (15) Tv So (20)	Pč Nm (8) Tv So (20)	
Úterý	M Z (8)	Čj P (8)	Aj Šp (8)	Fy S (22)	Vz S (8)	Vz S (8)	
Středa	Z Ku (8)	Aj Šp (8)	Čj P (8)	Vo Ku (8)	D Ně (8)	Hv Ho (21)	Akon Ně (16)
Čtvrtek	Pp Nm (8)	Inf Nm (15) Tv So (20)	Tv So (20) Pč Nm (8)	M Z (8)	Z Ku (8)	Fy S (22)	
Pátek	Aj Šp (8)	M Z (8)	Čj P (8)	Akon Ně (17)	Vv Z (8)	Vv Z (8)	

room-TV (20)

	1	2	3	4	5	6	7
Pondělí	Tv (So) (7A)	Tv (Bu) (2A)	Tv (Fi) (4A)	Tv (So) (8A, 8B)	Tv (So) (6A, 6B)	Tv (So) (6A, 6B)	
Úterý	Tv (Hu) (1B)	Tv (Mx) (3A)	Tv (So) (7A)	Tv (Fi) (4A)		Tv (Hu) (5A)	
Středa	Tv (Č) (3B)	Tv (So) (8A, 8B)	Tv (So) (9A, 9B)	Tv (D) (1A)	Tv (Č) (3B)	Tv (So) (7A)	Tv (So) (7A)
Čtvrtek	Tv (Bu) (2A)	Tv (So) (6A, 6B)	Tv (So) (6A, 6B)	Tv (Kl) (4B)	Tv (So) (8A, 8B)	Tv (So) (9A, 9B)	Tv (So) (9A, 9B)
Pátek	Tv (Mx) (3A)	Tv (Hu) (1B)	Tv (D) (1A)	Tv (Hu) (5A)	Tv (Kl) (4B)	Tv (So) (8A, 8B)	Tv (So) (9A, 9B)

Obrázek F.1: Ukázka formátu výstupních rozvrhů.

Příloha G

Obsah přiloženého CD

Adresářová struktura přiloženého CD je následující:

- `doc` - dokumentace zdrojových textů generovaná pomocí nástroje `doxygen`
- `src` - zdrojové soubory vyvinuté aplikace
  - `bakalar_charts` - soubory s reálnými rozvrhy škol ze systému Bakalaři
  - `inputs` - převedená data ze složky `bakalar_charts` do vstupního XML formátu aplikace
  - `logs` - logovací soubory vytvářené po spuštění aplikace
  - `others` - ostatní nezařaditelné soubory projektu
  - `outputs` - soubory s vygenerovanými rozvrhy v html formátu
  - `plots` - data grafů vývoje populací vytvářené při běhu aplikace
  - `settings` - konfigurační soubory aplikace s různými typy nastavení
  - `tinyxml` - externí knihovna pro snažší zpracování XML souborů
- `tex` - zdrojové texty této bakalářské práce ve formátu LaTeX
- `bp_xhorky17.pdf` - tato bakalářská práce ve formátu pdf.