VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SYSTÉM PRO POKROČILÉ PLÁNOVÁNÍ

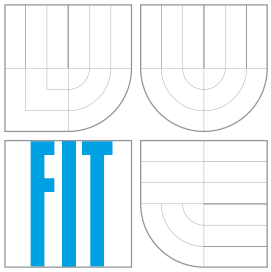
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ALEŠ HORKÝ

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SYSTÉM PRO POKROČILÉ PLÁNOVÁNÍ

SYSTEM FOR ADVANCED SCHEDULING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ALEŠ HORKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAELA ŠIKULOVÁ

BRNO 2015

Abstrakt

Tato diplomová práce se zabývá návrhem a implementací softwaru pro automatizované plánování rozvrhů zkoušek a přednášek. Návrh je přizpůsoben pro specifické potřeby Fakulty informačních technologií Vysokého učení technického v Brně. Problém je řešen kombinací genetického a heuristického algoritmu. Pomocí genetického algoritmu je získáno pořadí předmětů v jakém mají být vkládány do výsledného rozvrhu heuristickým algoritmem. Výkonnostně optimalizovaná implementace v jazyce Python 3 umožňuje tento výpočet paralelizovat, díky čemuž lze získat vygenerované rozvrhy již za dobu řádově desítek minut. Provedené experimenty vykazují ve všech sledovaných kritériích přibližně o 13 % lepší výsledky než jakých bylo dosaženo u zkouškových rozvrhů v minulosti. Vývoj byl pravidelně kozultován s osobami zodpovědnými za tvorbu rozvrhů na fakultě. Program bude použit při vytváření zkouškových rozvrhů pro akademický rok 2015/2016.

Abstract

This master thesis deals with the automatic design of examinations and courses scheduling. The design is adapted to the specific requirements of the Faculty of Information Technology of Brno University of Technology. A genetic algorithm and a heuristic algorithm are employed to solve this task. The genetic algorithm is used to specify the sequence of the examinations (or the courses) and then the heuristic algorithm spread them out into a timetable. An implementation (written in Python 3) provides a fast parallel processing calculation which can generate satisfactory schedules in tens of minutes. Performed experiments show approximately 13 % better results in all considered criteria in comparison with utilized examination schedules in the past. The development was periodically consulted with persons responsible for the schedule processing at the faculty. The program will be used while designing of examination schedules for the academic year 2015/2016.

Klíčová slova

Plánování rozvrhů, genetický algoritmus, heuristický algoritmus, kolizní matice, multikriteriální optimalizace.

Keywords

Timetable scheduling, genetic algorithm, heuristic algorithm, collision matrix, multi-objective optimization.

Citace

Aleš Horký: Systém pro pokročilé plánování, diplomová práce, Brno, FIT VUT v Brně, 2015

Systém pro pokročilé plánování

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením paní Ing. Michaely Šikulové. Dále prohlašuji, že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Aleš Horký
25. května 2015

Poděkování

Rád bych poděkoval své vedoucí Ing. Michaele Šikulové za trpělivost, cenné podněty a vřelý přístup nad rámec jejích povinností. Dále bych chtěl poděkovat Ing. Miloši Eysseltovi, CSc. za poskytnutí podkladů o postupech při tvorbě rozvrhů, Ing. Bohuslavu Křenovi, Ph.D. za poskytnutí informací ohledně plánování rozvrhů na Fakultě informačních technologií VUT v Brně a Ing. Jaroslavu Dytrychovi za diskuzi a zpětnou vazbu nad vygenerovanými rozvrhy. V neposlední řadě bych na tomto místě rád poděkoval své přítelkyni a rodině za podporu, která mi umožnila naplno se věnovat této práci.

Děkuji IT4Innovations za výpočetní prostředky poskytnuté v rámci projektů Centrum excelence IT4Innovations (CZ.1.05/1.1.00/02.0070).

Obsah

Kapitola 1

Úvod

Vývoj počítačů je od svých prvopočátků úzce spjat se snahou člověka usnadnit si práci. První počítače, lépe řečeno kalkulátory, určené pro základní matematické výpočty, byly vyrobeny již před naším letopočtem. Až v 19. století byl v Anglii navržen první *univerzální počítač*, jehož princip nám umožňuje využívat počítače tak, jak je chápeme nyní. Počítače jsou dnes všude kolem nás a často si ani neuvědomujeme, že díky nim děláme věci jinak – jednodušeji, kvalitněji nebo rychleji.

K rozmachu počítačů posledních desetiletí neodmyslitelně patří výzkum software. S masivním vývojem hardware se prudce začal rozvíjet také obor umělé inteligence. Někteří vědci věřili, že už v devadesátých letech minulého století budeme vyrábět počítače, které budou moci zastat člověka při libovolné činnosti – to se však zatím nepodařilo. Dnešní výkon osobních počítačů ale umožňuje téměř každému využívat programy, které při specifických činnostech mohou nahradit lidský mozek.

Tato práce popisuje návrh programu pro automatizované plánování rozvrhů a zkoušek na *Fakultě informačních technologií Vysokého učení technického v Brně*. Program má za cíl zjednodušit vytváření těchto rozvrhů, které na fakultě dodnes probíhá z větší části bez využití moderních technologií, a je tedy časově dosti náročné. Počítačem generovaný rozvrh by mohl nabídnout několik variant, ze kterých si uživatel pouze vybere tu nejvíce přijatelnou. Program by mohl navíc nabídnout přehledné prostředí, které by výrazně zjednodušilo případné úpravy takto získaných rozvrhů.

Tvorba rozvrhů je obecně velmi obtížná především z důvodu tzv. *exploze stavového prostoru* – ten roste exponenciálně s velikostí vstupních dat. Pro řešení daného problému tak nelze z časových důvodů použít přímočaré algoritmy. Jako vhodné může být použití různých stochastických metod, například *evolučních algoritmů*. Ty sice nezaručí zisk optimálního řešení, ale dokáží nalézt alespoň tzv. *suboptimální řešení*, jehož kvalita nám v praxi v mnoha případech dostačuje.

Pro automatizované plánování školních rozvrhů lze nalézt mnoho komerčních i volně dostupných programů. Tyto programy se snaží nabídnout obecné rozhraní, aby pokryly požadavky co nejvíce institucí. Ze studie provedené v roce 2012 v rámci bakalářské práce však vyplynulo, že žádný z nich není bez dalších úprav pro potřeby Fakulty informačních technologií přijatelný. Většina komerčních programů se totiž zaměřuje převážně na potřeby základních a středních škol, které se ale od potřeb fakulty výrazně liší.

Důvodem je mimo jiné i nedostupnost detailních (byť i anonymních) informací o studenty zapsaných předmětech. Závislosti mezi předměty jsou dostupné pouze v sumarizované formě, kterou jsou řečeny počty společných studentů pro každou dvojici předmětů. Takovéto údaje však naprostá většina rozvrhovacích programů nedokáže zpracovat.

Výhodou specificky zaměřeného programu je, že ho lze již od prvních fází vývoje přizpůsobit konkrétním požadavkům. Dá se tak předpokládat, že nabídne lepší výsledky než komerční programy. Náš návrh je založen na permutačním genetickém algoritmu, který se snaží nalézt nejlepší postup pro generování rozvrhu. Po analýze požadavků fakulty jsme navrhli heuristický algoritmus, který podle nalezeného postupu výsledný rozvrh sestrojí.

V první části práce (kapitola 2) jsou představeny základní principy a metody používané pro plánování. Jsou popsány obecné problémy spojené s tvorbou školních rozvrhů spolu s vhodnými metodami pro jejich řešení. Poté je uvedena analýza plánování rozvrhů a zkoušek na *Fakultě informačních technologií Vysokého učení technického v Brně*. Analýza je zaměřena především na popis kritérií, která vyplývají ze specifických požadavků fakulty. Nakonec jsou představeny některé existující plánovací programy a jsou popsány jejich nevýhody při použití na fakultě. Následující kapitola 3 shrnuje poznatky o evolučních a genetických algoritmech. Zaměřuje se převážně na permutační kódování chromozomů a multikriteriální optimalizaci pomocí algoritmu *NSGA-II*.

V kapitole 4 je (s ohledem na předchozí kapitoly) navržen hybridní algoritmus využívající evolučních a heuristických technik pro automatické generování rozvrhů. Jsou zde také formálně popsána kritéria určující kvalitu rozvrhu. Kapitola 5 popisuje implementaci navrženého systému v jazyce Python 3 a obsahuje uživatelský návod k jeho použití. Tato kapitola se však především zabývá různými optimalizacemi, ke kterým bylo přistoupeno, aby mohl uživatel získat výsledky v co nejkratším čase. V následující části práce (kapitola 6) jsou uvedeny experimentální výsledky a jsou v ní doporučeny vhodné hodnoty nastavení genetické části výpočtu. Zhodnocení dosažených výsledků se nachází v kapitole 7.

Kapitola 2

Plánování rozvrhů

Rozvrhování aktivit do časových oken je dnes potřeba v mnoha organizacích. U moderní průmyslové výroby je nutné dodržovat přesné výrobní postupy, což zajišťují různé výrobní plány. Instituce se směnným provozem (např. nemocnice, továrny) potřebují vytvářet rozvrhy pro své zaměstnance, aby měly v plánovaném období vždy zajištěn dostatek pracovních sil. Globální trh vytváří obrovský tlak na efektivnost výroby, proto u drahých zařízení existují plány, které zajišťují jejich maximální využití. V neposlední řadě je zde plánování školních rozvrhů, kterému se věnuje větší část této kapitoly.

2.1 Metody vytváření rozvrhů

Potřeba plánovat rozvrhy není záležitostí pouze posledních let, ale existovala zde již před masových rozšířením počítačů. Bez pomoci výpočetní techniky bylo nutné plánovat rozvrhy „ručně“. Možným zjednodušení tohoto úkolu mohlo být nanejvýše využití různých „papírkových metod[1]“. Přesný postup a údaje zobrazené na lístečcích jsou závislé na typu plánované činnosti a mohou být součástí know-how jedince nebo instituce.

Ruční plánování rozvrhů je i přes veškerý technologický pokrok často tou nejjednodušší možností. Pokud rozvrh vytváří osoba, která má širší povědomí o plánovaných aktivitách, lze u mnoha problémů dosáhnout přijatelných výsledků. Tvorba rozvrhu tímto způsobem je většinou dlouhodobější činnost – vznikají různé návrhy, ke kterým se zainteresované osoby v průběhu času vyjadřují. Ruční plánování je zvlášť vhodné v případě, že existuje podobný rozvrh, který organizace již úspěšně využila v minulosti. V takovém případě se může celý plánovací proces výrazně zkrátit v závislosti na podobnosti obou rozvrhů. Tento empirický přístup vyžaduje mnoho času, téměř vždy jej lze vylepšit a nenabízí mnoho objektivních měřítek k zjištění, jak kvalitní rozvrh jsme vytvořili [7].

S rozvojem počítačů se objevují i dva základní typy programů pro usnadnění tvorby rozvrhů. První z nich mají za cíl asistovat u procesu ručního plánování a co nejvíce ho usnadnit. Takové programy jsou založeny především na kvalitní vizualizaci, automaticky upozorňují na nežádoucí kolize a mohou uživateli doporučit nejvhodnější pozici pro zvolený předmět. Tuto funkci nabízí například český program *Bakaláři*[2], který je navržen pro administrativu na základních a středních školách. Jeho modul *rozvrh hodin* využívá při plánování algoritmus *forward-checking* bez navracení – konflikty zde řeší uživatel.

[1] Papírkovou metodou je myšlen fyzický vizualizační postup, při kterém uživatel sestavuje rozvrh pomocí přesouvání označených lístečků s plánovanými aktivitami.

[2] Informace o programu *Bakaláři* lze nalézt na http://www.bakalari.cz/rozvrh.aspx.

Druhým typem jsou většinou komplexnější programy, jejichž součástí je *autonomní* rozvrhovací algoritmus. Uživatel před spuštěním výpočtu zadá množiny zdrojů (učebny, časová okna, vyučující a podobně) a seznam předmětů, navíc nastaví kritéria a parametry algoritmu. Podle zadaných požadavků se poté vygeneruje rozvrh, který je možné v některých případech ještě dodatečně upravit. Několik příkladů programů z této kategorie lze nalézt v kapitole 2.3.2, kde jsou diskutovány jejich nedostatky při nasazení na *Fakultě informačních technologií Vysokého učení technického v Brně*.

2.2 Školní rozvrhy

Při generování školních rozvrhů je potřeba volit takové kombinace entit (přednášejících, učeben a studentů) umístěných do časových oken, které nabídnou jejich optimální rozvržení z hlediska nákladů, času, lidského pohodlí a podobně. Jedná se o NP-úplný problém, to znamená, že neexistuje deterministický algoritmus, který by uměl najít optimální řešení v polynomiálním čase. Různými heuristickými postupy se však můžeme snažit nalézt suboptimální řešení. U něj sice nedokážeme říci, zda neexistuje nějaké jiné lepší řešení, ale dosáhneme takových vlastností, že jej můžeme považovat za *dostatečně dobré* a ukončit další hledání [5].

Náročnost tohoto úkolu více přiblíží fakt, že pro rozvrh o u učebnách, p předmětech a cs časových oknech existuje $(u \cdot cs)^p$ možných variant. Uvažme hypotetický příklad menšího rozvrhu o 6 učebnách, 35 předmětech a 35 časových oknech (7 hodin, 5 dnů v týdnu), potom počet zkonstruovatelných rozvrhů je přibližně $2 \cdot 10^{81}$. To je množství srovnatelné s odhadovaným počtem atomů ve vesmíru. Ověření takového množství rozvrhů a výběr optimálního z nich je tak pro současné počítače příliš výpočetně náročná úloha.

Plánování rozvrhů spadá (podobně jako většina reálných optimalizačních problémů) do kategorie *optimalizačních úloh s omezujícími podmínkami* (angl. *constraint optimization problems*) [10]. Jsou to právě specifika omezujících podmínek, která znemožňují efektivní nasazení obecných optimalizačních algoritmů a činí tak z plánování náročnou činnost. Kvalitně vytvořený rozvrh musí v praxi splňovat mnoho často protichůdných omezení. Tato omezení lze rozdělit následovně [7, 10]:

- Tvrdá omezení (*hard-constraints*) rozdělují prohledávaný prostor na množinu přípustných a nepřípustných řešení. V mnoha případech vychází z fyzické podstaty reálného světa – klasicky jde o situace, ve kterých by se jedna osoba musela nacházet na dvou místech zároveň. Tato omezení lze pro konkrétní rozvrh vyhodnotit jako binární hodnoty. Rozvrh musí splňovat všechna tvrdá omezení, aby mohl být prakticky realizovatelný.

- Měkká omezení (*soft-constraints*) určují kvalitu přípustných řešení. Snažíme se vytvářet takové rozvrhy, které měkká omezení splňují co nejvíce. V případě školních rozvrhů se může jednat o požadavek, aby student neměl dvě zkoušky v jednom dni.

2.3 Analýza rozvrhů a zkoušek na FIT VUT v Brně

Na *Fakultě informačních technologií Vysokého učení technického v Brně* (dále jen *FIT*) se vyskytují dva typy rozvrhů – *přednáškové rozvrhy*, podle kterých probíhá vyučování v prvních třinácti týdnech semestru, a *rozvrhy zkoušek*, které trvají následujících (a také

posledních) pět týdnů semestru. Přestože jsou si v některých ohledech oba dva typy rozvrhů podobné, v mnohém se liší, a je tedy nutné je analyzovat odděleně.

První část této kapitoly popisuje konkrétní kritéria pro *přednáškové* a *zkouškové* rozvrhy na *FIT*. Pro lepší přehlednost jsou kritéria rozdělena do kategorií z pohledu studentů, předmětů, přednášejících a místností. U každého kritéria je uveden příznak podle toho, zda musí být ve výsledném rozvrhu bezpodmínečně splněno (tzv. *tvrdé* kritérium) nebo zda se snažíme tuto vlastnost minimalizovat, případně maximalizovat (tzv. *měkké* kritérium). Druhá část kapitoly shrnuje nedostatky existujících programů.

2.3.1 Kritéria rozvrhů na FIT VUT v Brně

Primární a sekundární vzdělávací instituce mají ve většině případů značně podobná kritéria na tvorbu rozvrhů. Pro jejich generování mohou využít mnoha komerčních programů (jmenujme například *ASC TimeTables* nebo *UntisExpress*), které jsou dostatečně obecné, aby tato kritéria zohlednily.

Kritéria vysokých škol a jejich fakult mohou být na druhou stranu natolik specifická, že lze jen těžko nalézt natolik obecný algoritmus, který by je pokryl a zároveň zůstal efektivní. Bakalářská práce Moniky Kubalcové z roku 2012 [12] obsahuje studii zabývající se *porovnáním programů pro plánování rozvrhů a zkoušek* na *FIT*. Závěrem práce je, že nebyl nalezen program (volně dostupný ani komerční), který by výrazněji zautomatizoval současné postupy.

Uvedené výčty kritérií byly získány z osobní konzultace s *proděkanem pro vzdělávací činnost v bakalářském studiu* na *FIT* panem Ing. Bohuslavem Křenou Ph.D, studiem současných postupů pro tvorbu rozvrhů na *FIT* od pana Ing. Miloše Eysselta CSc., Ing. Jaroslava Dytrycha a Ing. Davida Martínka a rešerší výše zmíněné studie [12].

Kolize v rozvrhu

Ještě před uvedením jednotlivých kritérií rozvrhů na *FIT* je vhodné zavést pojmy k popisu studentských kolizí, které mohou v rozvrzích nastat. Jedná se o situace, kdy jsou dvě zkoušky (nebo přednášky) naplánovány ve vzdálenosti dva a méně dnů. Pokud takovéto zkoušky (přednášky) obsahují alespoň jednoho společného studenta, dochází v závislosti na jejich vzdálenosti k následujícím typům kolizí:

- *Hodinová kolize*: zkoušky (přednášky) jsou naplánovány ve stejný den v překrývající se hodiny.
- *Denní kolize*: zkoušky (přednášky) jsou naplánovány ve stejný den.
- *Sousední kolize*: dvě zkoušky (přednášky) jsou naplánovány každá do jednoho ze dvou vzájemně následujících dnů (tj. pokud je první zkouška (přednáška) naplánována ve středu, tak druhá byla naplánována v úterý nebo čtvrtek).
- *Sousední kolize ob dva dny*: dvě zkoušky (přednášky) jsou naplánovány každá do jednoho ze dvou dnů, mezi nimiž je právě jeden volný den (tj. pokud je první zkouška (přednáška) naplánována ve středu, tak druhá byla naplánována v pondělí nebo v pátek).

Kritéria pro přednáškový rozvrh

Přednáškový rozvrh je plánován pouze jako týdenní, a aby pokryl všech třináct týdnů semestru, pravidelně se opakuje. Rozvrhy konkrétních týdnů se však mohou lehce lišit podle toho, zda je daný týden sudý nebo lichý. V mnoha případech jsou však tyto nuance symetrické (např. cvičení pro dvě skupiny studentů jednou za 14 dnů) a plánování rozvrhu tak ovlivňují jen minimálně. Pro *přednáškový rozvrh* se uvažuje vyučování od 7^{00} hodin do 20^{50} hodin, což odpovídá čtrnácti dostupným vyučovacím blokům v každé místnosti každý den. Dále je uveden výčet všech kritérií vztahujících se k *semestrálním rozvrhům*:

- kritéria z pohledu studenta:
  - tvrdá:
    1. hodinová bezkoliznost (nepřekrývání) povinných přednášek/cvičení *(pokud je přednáška opakována – dvě skupiny v jednom týdnu – stačí bezkoliznost alespoň v jednom případě)*,
  - měkká:
    2. hodinová bezkoliznost (nepřekrývání) volitelných přednášek/cvičení *(pokud je přednáška opakována – dvě skupiny v jednom týdnu – stačí bezkoliznost alespoň v jednom případě)*,
    3. obědová pauza v souvislém bloku přednášek,
    4. předcházení přednášky v daném předmětu jeho cvičením,
    5. koncentrace výuky jen do některých dnů *(vhodné především pro dojíždějící studenty)*,
- kritéria z pohledu předmětu:
  - tvrdá:
    1. nastavení pevného termínu pro předměty *(především předměty z* FEKT[3] *a* FSI[4]*, jejichž rozvrhům je třeba se přizpůsobit)*,
    2. předměty mající více přednášek v jednom týdnu je musí mít v různé dny,
    3. předmět může mít specifické požadavky na vybavení učebny,
    4. dodržení požadované délky přednášky/cvičení,
- kritéria z pohledu přednášejícího:
  - tvrdá:
    1. nemůže přednášet ve více místnostech zároveň *(výjimkou je streaming*[5]*)*,
    2. nemůže přednášet v předem zvolených dnech a hodinách *(zasedání kolegia děkana, konzultační hodiny a další činnosti)*,
    3. rozdělení výuky pro dva přednášející *(každý má například svoji polovinu semestru)*,
  - měkká:
    4. obědová pauza v souvislém bloku přednášek,

[3] FEKT - Fakulta elektrotechniky a komunikačních technologií na VUT v Brně.
[4] FSI - Fakulta strojního inženýrství na VUT v Brně.
[5] Při tzv. *streamingu* se přednáška z jedné místnosti živě promítá v dalších místnostech.

5. může navrhnout preferované termíny pro výuku,

- kritéria z pohledu místností:
  - tvrdá:
    1. hodinová bezkoliznost (nepřekrývání) přednášek,
    2. přednášková místnost musí mít kapacitu pro alespoň 70 % studentů,
    3. jedna přednáška může probíhat zároveň ve více učebnách *(streaming)*, je zvykem, že se všechny učebny nachází ve stejném komplexu,
    4. vícehodinové přednášky musí probíhat ve stejné místnosti,
  - měkká:
    5. čas na přesun mezi budovami při mezifakultní výuce.

Kritéria pro rozvrh zkoušek

Rozvrh zkoušek je plánován na období přibližně 25 dní, tedy asi pěti týdnů. Přesný počet dní se odvíjí z plánu pro daný akademický rok, z akcí konaných v příslušném období a dalších. Z důvodu snížení zátěže studentů je naprostá většina zkoušet plánována na dobu mezi 8^{00} až 17^{50}, což odpovídá deseti vyučovacím blokům v každé přednáškové místnosti každý den[6]. Velmi zjednodušeně je *rozvrh zkoušek* náročnější na splnění svých kritérií než *rozvrh přednášek*, na druhou stranu je pro jeho naplánování větší časový prostor.

Student na *FIT* má v každém předmětu nárok na řádný termín zkoušky a na dva opravné termíny. Dá se předpokládat, že účast na opravných termínech bude nižší, protože určitá část studentů bude mít zkoušku již splněnou z předešlých termínů. Kolize (všech typů) mezi opravnými termíny zkoušek tedy mohou být při generování rozvrhu považovány za méně problematické. V důsledku toho lze vyhradit větší část dnů pro řádné termíny zkoušek, ve snaze maximalizovat jejich bezkoliznost.

Centrální plánování rozvrhu se ve většině případů týká pouze předmětů s větším počtem studentů. Je zavedenou praxí, že plánování zkoušek u některých malých předmětů probíhá formou diskuze mezi studenty a přednášejícím v průběhu posledních přednášek. Dále je uveden výčet všech kritérií vztahujících se k *rozvrhu zkoušek*.

- kritéria z pohledu studenta:
  - tvrdá:
    1. hodinová bezkoliznost (nepřekrývání) zkoušek z povinných předmětů *(není nutné dodržet pro opakující studenty)*,
    2. časová rezerva pro přechody mezi fakultami,
  - měkká:
    3. hodinová bezkoliznost (nepřekrývání) zkoušek ze všech předmětů,
    4. nejvýše jedna zkouška v jednom dni (denní bezkoliznost),
    5. volné dny mezi zkouškami (sousední bezkoliznost a sousední bezkoliznost ob dva dny),

[6] Pro pondělky bylo Ing. Jaroslavem Dytrychem (s ohledem na dojíždějící studenty) doporučeno plánovat zkoušky až od 9^{00}. Pro pátky bylo naopak doporučeno plánovat nejpozději do 15^{50}.

- kritéria z pohledu zkoušky:
  - tvrdá:
    1. nastavení pevného termínu pro zkoušku *(především zkoušky z předmětů z* FEKT *a* FSI*, jejichž termínům je třeba se přizpůsobit)*,
    2. předmět může mít specifické požadavky na vybavení učebny,
    3. dodržení požadované doby zkoušky,
    4. dodržení počtu požadovaných termínů zkoušek *(může být i více než tři, student však může navštívit nejvýše právě tři)*,
- kritéria z pohledu přednášejícího:
  - tvrdá:
    1. může mít nejvýše jednu zkoušku v danou dobu *(výjimkou může být následující bod)*,
    2. může chtít naplánovat dvě zkoušky z různých předmětů do jedné dostatečně velké místnosti,
    3. nemůže být u zkoušky v předem zvolených dnech a hodinách *(účast na konferenci a další)*,
    4. na jednu zkoušku může dohlížet více přednášejících,
  - měkká:
    5. může navrhnout preferované termíny pro zkoušku,
- kritéria z pohledu místností:
  - tvrdá:
    1. hodinová bezkoliznost (nepřekrývání) zkoušek,
    2. přednášková místnost musí mít kapacitu pro všechny studenty *(často i větší, protože někteří zkoušející obsazují jen každou druhou/třetí židli v místnosti)*,
    3. jedna zkouška může zároveň probíhat ve více učebnách,
    4. vícehodinové zkoušky musí probíhat ve stejné místnosti,
  - měkká:
    5. mezi zkouškami v jedné místnosti je vhodná časová rezerva.

Protože je potřeba optimalizovat více než jedno měkké kritérium současně, je nutné považovat problematiku tvorby rozvrhů na *FIT* za *multikriteriální optimalizační úlohu.*

2.3.2 Nedostatky existujících programů

Většina běžně dostupných programů pro plánování rozvrhů je přizpůsobena požadavkům pro souvislou výuku na základních a středních školách. Existují však i programy (v mnohých případech velmi komplexní), které dokáží při tvorbě rozvrhu zohlednit také specifické univerzitní požadavky. V následujících podkapitolách budou některé z nich představeny a budou uvedeny nedostatky, které by bylo třeba vyřešit před jejich reálným nasazením na *FIT*.

Free Timetabling Software (FET)

Free Timetabling Software[7] *(FET)* je volně dostupný program šířený pod licencí GNU GPL v2. Je navržen jak pro použití na základních a středních školách, tak i na univerzitách.

Program v testované verzi FET-5.23.4 využívá algoritmus *recursive swapping (rekurzivní prohazování)*, který byl vyvinut při jeho vývoji. Algoritmus umísťuje předměty do rozvrhu, dokud jsou v něm volné pozice (tj. pozice, jejichž využití neporuší žádné kritérium). Poté se snaží rekurzivně přepočítávat, zda by se rozvrh nezlepšil záměnou s již naplánovanými předměty. Tento výpočet se opakuje, dokud existuje záměna zvyšující kvalitu rozvrhu.

FET není dobře použitelný pro nasazení na *FIT* hned z několika důvodů [12]:

- Program neumožňuje pracovat na úrovni jednotlivých studentů, ale pouze na úrovni tříd (ročníků). V prostředí *FIT* však studenti stejného ročníku mohou navštěvovat (a často také navštěvují) rozdílné předměty. Tuto skutečnost je tedy nutné při plánování zohlednit.
- Program neumí plánovat výuku jednoho předmětu do více místností. Při vytváření *přednáškového rozvrhu* tak nelze využít streaming vyučování do menších místností. *Zkouškový rozvrh* prakticky nelze naplánovat, protože separátní kapacity i největších místností na *FIT* jsou pro potřeby zkoušek často nedostatečné.
- *FET* také neumožňuje plánovat čtrnáctidenní rozvrh s rozdělením na sudé a liché týdny. Toto sice není klíčová vlastnost pro rozvrhy na *FIT*, může u nich však zhoršovat získané výsledky.

ASC Timetables

Komerční aplikaci ASC Timetables[8] využívá přibližně 150 000 škol po celém světě. Jedná se tak o jeden z nejrozšířenějších programů pro automatické generování rozvrhů. ASC Timetables je primárně určen pro základní a střední školy, přesto byl ve studii z roku 2012 [12] zvolen jako nejvhodnější dostupný kandidát pro plánování rozvrhů na *FIT*. Komerční licence vhodná pro použití na *FIT* pro rok 2015 stojí 23 900kč.

Jako největší nedostatek programu se jeví zpracování (pro *FIT* velmi typické) paralelní výuky jednoho předmětu ve více místnostech. ASC Timetables sice nabízí možnost plánovat výuku předmětu ve více místnostech, neumožňuje už ale definovat skupiny místností tak, aby výuka probíhala vždy v jedné budově (případně komplexu). Navíc program vyžaduje i explicitně uvedený počet učeben, který má pro daný předmět využít. Prakticky je tak nutné před plánováním rozvrhu přiřadit učebny ručně pro přibližně 50% největších předmětů. Tato vlastnost zmíněnou funkci významně degraduje.

Výše zmíněná studie vyhodnotila přínosy ASC Timetables při plánování rozvrhů na *FIT* jako diskutabilní. Program sice zjednodušuje některé úkony, ale jeho použití přináší i nové problémy [12].

[7] Informace o programu *Free Timetablin Software* lze nalézt na http://www.lalescu.ro/liviu/fet/.
[8] Další informace o programu *ASC Timetables* lze nalézt na http://www.asctimetables.com/.

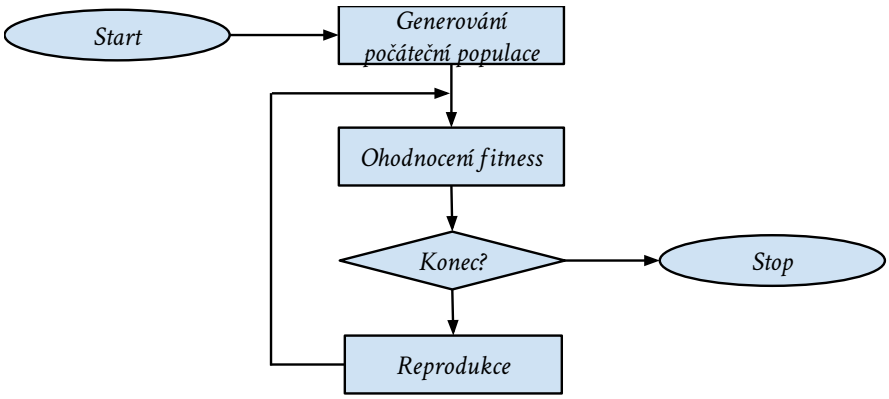
Kapitola 3

Evoluční algoritmy

Evoluční algoritmy dnes patří mezi základní stochastické postupy z oblasti umělé inteligence. Jsou inspirovány Darwinovským evolučním procesem, jež se skládá z populace jedinců, kteří se v průběhu generací vyvíjí v důsledku evolučního tlaku. Živočišné druhy se tak například mohou přizpůsobovat změnám v jejich okolním prostředí. Snahou při návrhu evolučních algoritmů je využít zmíněného principu pro řešení složitých výpočetních problémů [11, 13].

Různé *evoluční algoritmy* vyvíjely od 50. let 20. století nezávislé vědecké skupiny. Na Technické univerzitě Berlín navrhli Rechenberg a Schwefel algoritmus s názvem *evoluční strategie* pro řešení složitých optimalizačních problémů [6]. Ve stejnou dobu byla Fogelem a jeho kolegy na Kalifornské univerzitě v Los Angeles vynalezena technika pro automatický návrh konečných automatů nazvaná *evoluční programování* [14]. Nezávisle na předchozích byla Hollandem na Michiganské univerzitě vytvořena skupina technik pro návrh adaptivních systémů nazvaných *genetické algoritmy* [8]. Později bylo Kozou na Stanfordově univerzitě ještě navrženo *genetické programování* [13].

Motivace těchto skupin a jimi zvolené postupy byly rozdílné, přesto se v jejich teoriích daly najít shodné rysy. Z důvodu ulehčení spolupráce se na začátku 90. let dohodly na společném názvosloví, čímž byl celý obor zaštítěn právě pod názvem *evoluční algoritmy* – jednotlivé (a i další) vývojové větve ale dodnes existují [11].



Obrázek 3.1: Základní schéma evolučního algoritmu.

Evoluční algoritmy jsou inspirovány ději v přírodních systémech, nejsou jimi však nijak omezovány, a proto se mohou občas od biologického paradigmatu odlišovat. Základní schéma evolučního algoritmu lze vidět na obrázku 3.1. Při řešení problému pomocí evolu-

čních algoritmů se nejprve vytvoří *populace* jedinců, kteří reprezentují jeho možná řešení. Každý *jedinec* je ohodnocen *fitness funkcí* – ta určuje jeho míru schopnosti přežít a rozmnožit se [13]. Poté jsou z populace vybráni jedinci, kteří budou součástí reprodukčního procesu. Z těchto jedinců jsou vytvořeni potomci takovým způsobem, aby byli vybraným rodičům podobní, ale ne s nimi identičtí. Nakonec jsou někteří jedinci z minulé generace odebráni. Celý reprodukční cyklus se opakuje, dokud není dosaženo požadovaného řešení [11].

Dva základní způsoby, jakými může být v jedinci kódován řešený problém, vycházejí znovu z biologie. Kódování pomocí *genotypu* je velmi blízké reálným organismům – jedinec obsahuje *chromozom* (řetězec) složený z jednotlivých *genů*. Z jejich pořadí a hodnot, které nazýváme *alely*, pak v závislosti na typu problému lze dekódovat řešení a získat tak odpovídající *fenotyp*. Tento přístup je výhodný zejména z toho důvodu, že jedinci mohou být pro rozdílné problémy kódováni stejně a je tak možné při reprodukčním procesu využívat obecné postupy. Druhou možností je kódování přímo pomocí *fenotypu*. V tomto případě se po celou dobu pracuje s reprezentací, která odpovídá řešení, což umožňuje aplikovat specifické postupy pro daný problém [11].

Kvalitu konkrétního fenotypu (jeho fitness) ohodnocujeme *fitness funkcí*, tu získáme transformací z tzv. *účelové funkce*. Účelová funkce je dána řešeným problémem[1] a působí v evolučním algoritmu na místě prostředí, ve kterém se nacházejí jedinci. Pomocí ní můžeme ohodnotit kvalitu každého jedince α z prostoru přípustných řešení D_p. Bez újmy na obecnosti předpokládejme, že cílem výpočtu je účelovou funkci f minimalizovat, řešením problému je potom jedinec α_{opt}, pro kterého platí:

$$\alpha_{opt} = \arg\min_{\alpha \in D_p} f(\alpha). \tag{3.1}$$

Oborem hodnot fitness funkce jsou kladná reálná čísla, jejichž vyšší hodnoty značí kvalitnější řešení. Transformace fitness funkce F z účelové funkce f tak musí vyhovovat následující podmínce:

$$\forall \alpha_1, \alpha_2 \in D_p : f(\alpha_1) \leq f(\alpha_2) \Rightarrow F(\alpha_1) \geq F(\alpha_2) \geq 0 \tag{3.2}$$

Konkrétní způsob transformace není pevně stanoven a dá se provést mnoha způsoby. Intuitivně se nabízí lineární transformace, při níž je minimální f_{min} (maximální f_{max}) hodnota účelové funkce převedena na maximální F_{max} (minimální F_{min}) hodnotu fitness funkce, ostatní funkční hodnoty jsou lineárně interpolovány mezi nimi:

$$F(\alpha) = \frac{F_{max} - F_{min}}{f_{min} - f_{max}} f(\alpha) + \frac{f_{min}F_{min} - f_{max}F_{max}}{f_{min} - f_{max}} \tag{3.3}$$

Výhodou tohoto přístupu je především jeho jednoduchost a v mnoha případech i dobré výsledky. Lineární transformace však není vhodná pro účelové funkce, jejichž funkční hodnoty nejsou rovnoměrně rozloženy (a jsou např. v rozsahu několika řádů). Zde totiž mohou z pohledu evolučního algoritmu vznikat nepřiměřené rozdíly mezi kvalitou jedinců, které nemají reflexi v daném problému.

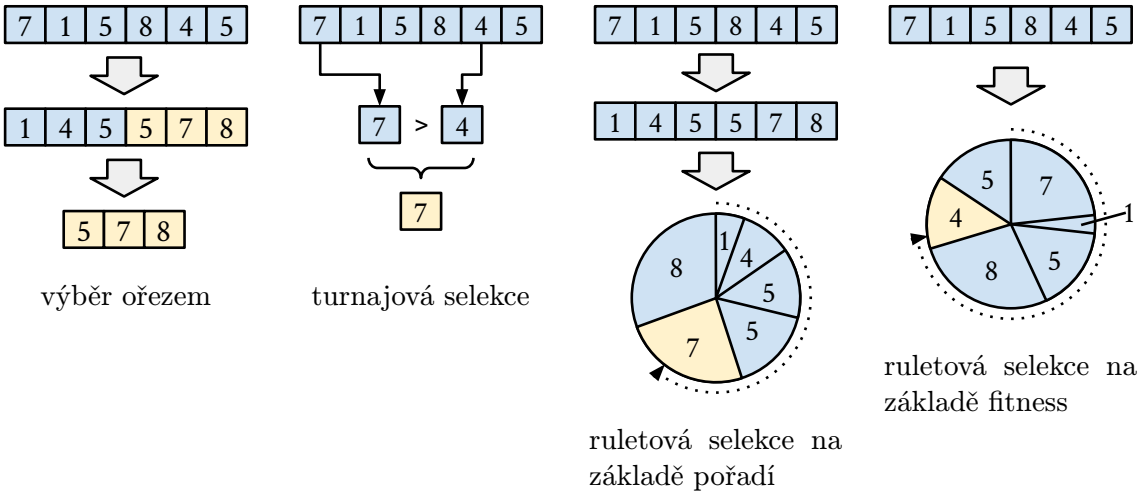
Řešením může být rovnoměrná transformace. U ní jsou maxima a minima obou funkcí přiřazeny stejně jako v předchozím případě, ale ostatní hodnoty jsou seřazeny a podle jejich pozice je jim rovnoměrně přiřazena hodnota z oboru hodnot fitness funkce. Nevýhodou může být, že dvěma a více jedincům se shodnou hodnotou účelové funkce je přiřazena rozdílná fitness [13].

[1] V případě školních rozvrhů je účelová funkce definována jedním nebo více měkkými kritérii.

Existují problémy při jejichž optimalizaci je nutné brát ohled na více nezávislých kritérií. Takovýmto problémům odpovídá více účelových funkcí (a tedy i fitness funkcí), kterým musí hledané řešení vyhovět. Problémy a postupy spojenými s tímto typem úloh se zabývá kapitola 3.3 o multikriteriální optimalizaci.

Na základě fitness funkce můžeme provádět *selekci* populace. Jak bylo zmíněno výše, v jednom cyklu evolučního algoritmu probíhá výběr jedinců dvakrát. Nejprve jsou vybráni kvalitní jedinci, kteří se budou účastnit reprodukce se záměrem zvýšit počet dobrých jedinců v populaci. Poté jsou vybráni nekvalitní jedinci, kteří nemají schopnosti, aby v konkurenčním prostředí dále přežili – tito jedinci jsou z populace odebráni. V tomto principu lze spatřit jednu z hlavních myšlenek evolučních algoritmů, která předpokládá, že celková kvalita populace se bude v průběhu generací zvyšovat [13].

Způsob selekce ovlivňuje sílu selekčního tlaku, který je na populaci vyvíjen. Pokud bude selekční tlak moc nízký, tak se jedinci nebudou vyvíjet, kvalita populace bude stagnovat a řešení nedosáhneme. V opačném extrému, kdy by byl selekční tlak moc vysoký, se po úvodním rychlém vývoji brzy vytratí rozmanitost jedinců v populaci a její kvalita bude opět stagnovat. Musíme proto volit takové selekční mechanismy, které udrží selekční tlak na vhodné úrovni.



Obrázek 3.2: Schéma průběhu čtyř selekčních mechanismů nad stejnou populací. Z důvodu zjednodušení jsou jedinci pojmenováni podle jim odpovídající hodnoty fitness funkce.

Čtyři známé způsoby selekce jsou schematicky znázorněny na obrázku 3.2 [11, 10]. Uvedené selekční mechanismy jsou seřazeny sestupně podle velikosti jimi vytvářeného selekčního tlaku. Jako nejjednodušší se jeví *výběr ořezem* (*truncation selection*). Ten seřadí jedince v populaci podle jejich kvality a vybere předem daný počet nejlepších z nich.

Turnajová selekce (*tournament selection*) je velmi podobná selekci v přírodě – z populace se náhodně vyberou dva nebo více jedinců, kteří spolu budou soupeřit. Turnaj vyhrává jedinec s největší fitness. Výběrem většího počtu účastníků turnaje je možné selekční tlak zvyšovat. Naopak je ho možné snížit tak, že s určitou malou pravděpodobností necháme vyhrát slabšího jedince. Konání turnaje se opakuje, dokud nemáme vybrán požadovaný počet jedinců.

V případě *ruletové selekce na základě pořadí* (*rank-proportional selection*) se jedinci seřadí podle fitness a podle jejich pozice je jim přiděleno pořadí v populaci. Poté je každému jedinci přidělena výseč na virtuální ruletě (interval reálných čísel) s velikostí přímo úměrnou

jeho pořadí. Následně je ruleta spuštěna (je vygenerováno náhodné číslo s rovnoměrným rozložením z rozsahu všech intervalů), čímž je vybrán jedinec. Mechanismus rulety se znovu opakuje, dokud nezískáme chtěný počet jedinců.

Ruletová selekce na základě fitness (*fitness-proportional selection*) se velmi podobá předchozí variantě s jedním rozdílem – pro výseč na ruletě je zde použita přímo hodnota fitness. Výraznější rozdíly v kvalitách jedinců se tak při selekci plně projeví.

Vybraní jedinci se účastní reprodukčního procesu, při kterém jsou na základě dědičnosti vytvářeni jejich potomci. Jedná se o klíčovou část evolučního algoritmu, protože právě zde se hledá řešení problému. Rozlišujeme dva základní způsoby reprodukce, které oba vycházejí z biologické předlohy. Při nepohlavním rozmnožování se vytváří potomci pomocí *mutace* chromozomu právě jednoho rodiče. Tento přístup se uplatnil převážně u *genetických strategií* a *genetického programování*. Druhou variantou je pohlavní rozmnožování užívané v přírodě vyššími organismy. Zde je nutné vybrat alespoň dva rodiče, ze kterých se křížením vytvoří jejich potomci. Maximální počet rodičů však není, na rozdíl od reality, ničím omezen [4]. Způsob *křížení* musí být navržen tak, aby potomci dědili vlastnosti od všech rodičů. Z důvodu zvyšování rozmanitosti populace je i zde vhodné v malé míře uplatnit mutaci. Pohlavní rozmnožování využívají například *genetické algoritmy* [11].

3.1 Genetické algoritmy

Genetické algoritmy byly první evoluční technikou, která byla široce přijata odbornou veřejností v oblasti umělé inteligence. Dnes jsou genetické algoritmy základním výpočetním prostředkem při řešení složitých optimalizačních problémů [18].

Genetický algoritmus se od ostatních evolučních algoritmů liší převážně tím, že klade důraz na součinnost *křížení*, *selekce* i *mutace*. Nepředepisuje však žádné konkrétní postupy, ale určuje pouze základní paradigma. V počátcích jeho vývoje byla mutace přehlížena na úkor křížení, v poslední době se však ukazuje její nezastupitelná pozice při udržování diverzity populace. Před ukončením běhu genetického algoritmu se navíc často využívá různých lokálních prohledávacích technik (*simulované žíhání, horolezecký algoritmus a dalších*), jejichž cílem je dodatečně zvýšit kvalitu nalezených řešení [18].

Jedinci jsou zde reprezentováni pomocí *genotypu*[2], pro který musí existovat algoritmus převodu na odpovídající *fenotyp*. Způsob reprezentace genotypu nazýváme *kódování*. Navrhnout způsob kódování u složitějších problému je jedním z nejtěžších úkolů při aplikaci genetického algoritmu. Nad ním jsou poté definovány odpovídající reprodukční operátory, účelové funkce a selekční postupy [10, 13, 18].

Nejběžnější kódování je bezesporu ve formě řetězce genů. Není však shoda v otázce, zda je vhodnější gen reprezentovat ve formě bitové hodnoty, nebo reálného čísla. V minulosti se za lepší považovalo bitové kódování, protože rozkládá problém na velké množství jednoduchých hodnot, se kterými může genetický algoritmus snáze manipulovat. Mimo jiné má také blíže k biologické předloze. Dnes jsou tyto důvody považovány za kontroverzní. Aplikační nasazení genetických algoritmů s sebou přineslo potřebu kódovat problémy reálnými čísly. Ta se dá považovat za zobecnění binárního kódování, nad kterým lze navíc definovat specifické reprodukční operátory [18].

Jako jiné způsoby kódování genotypu můžeme uvést například permutační, grafové nebo kódování pomocí messy-chromozomů. Pro tyto a další způsoby kódování existují obecné

[2]Pojem „genotyp“ je u genetických algoritmů často synonymum pro pojem „chromozom“. Na rozdíl od genotypů v přírodě se zde totiž umělý genotyp v naprosté většině případů skládá pouze z jednoho chromozomu.

reprodukční operátory, jejichž vlastnosti jsou popsány v mnoha publikacích [10, 13]. Ve složitějších případech může být výhodné navrhnout kódování, které je přímo zaměřeno na specifické rysy problému – mluvíme potom o tzv. *hybridních genetických algoritmech* – ty spojují nejlepší vlastnosti genetických algoritmů a tradičních postupů pro řešení daného problému [10].

Pro plánovací úlohy se však hodí výše zmíněné *permutační kódování*. To pracuje také s řetězcem genů, avšak každá forma genu (alela) se v něm nachází právě jednou. Zde lze spatřit analogii s rozvrhem, který je zjednodušeně vlastně pouze permutace akcí (genů). Permutační úlohy obvykle představují složitý úkol, jelikož pro permutaci o délce N existuje $N!$ eventuálních řešení – jedná se tedy o *NP-úplné problémy*. Přesto právě v této oblasti dosáhly v historii genetické algoritmy prvních úspěchů. Příkladem může být známý *problém obchodního cestujícího*, který v minulosti byl a i nyní je předmětem rozsáhlého výzkumu [13].

3.1.1 Permutační operátory pro rekombinaci

Genetický algoritmus pracuje obecně s populací jedinců, jejichž genotypy musí splňovat určitá pravidla. V případě permutačního kódování se musí všechny jejich chromozomy skládat ze stejně velké permutace jedné množiny genů. Pokud na takovéto jedince chceme uplatňovat genetické operátory, je nutné zajistit, aby byly tyto podmínky po dokončení operace vždy splněny [13].

Prvním možným přístupem je provést libovolný genetický operátor pro práci s řetězci a po něm aplikovat opravný mechanismus, který na základě znalosti problému chromozom vhodně opraví. Druhou možností je navrhnout operátory tak, aby jejich výstupem byl přímo chromozom obsahující platnou permutaci. Podle těchto dvou principů jsou navrženy permutační operátory pro rekombinaci uvedené v této kapitole [13].

Jedním z nejznámějších permutačních operátorů je *křížení s částečným přiřazením (PMX – partially matched crossover)*. Operátor vytváří z dvou rodičů jejich potomky tak, že z jednoho rodiče využije uspořádanou podmnožinu permutace, kterou doplní dosud nepoužitými geny z druhého rodiče, se snahou co nejvíce dodržet jejich původní pořadí v chromozomu. Na následujícím příkladu je funkce operátoru vysvětlena podrobně [10].

Nechť p_1 a p_2 jsou rozdílné permutační chromozomy se dvěma náhodně zvolenými body křížení, ze kterých budeme vytvářet potomky o_1 a o_2:

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

V prvním kroku jsou náhodně zvolené oblasti vyměněny a je vytvořena množina přepisovacích pravidel k, jejíž položky odpovídají alelám na odpovídajících pozicích v těchto oblastech. Značkou $\square$ jsou označena dosud neobsazená místa v chromozomech:

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

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

Poté jsou do neobsazených genů potomků přeneseny na původních pozicích ty alely druhého předka, které se v něm dosud nevyskytují (jedná se o alely, které nebyly součástí vyměněných oblastí):

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

Hodnoty zbývajících genů jsou přeneseny podobně jako v předchozím kroku, avšak aby byl zaručen vznik platné permutace, jsou jejich alely zaměňovány podle přepisovacích pravidel z množiny k:

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

Jedna alela může být zaměněna i vícenásobně v případě, že předchozí záměnou nebylo dosaženo její unikátnosti v rámci permutace[3].

Druhým uvedeným operátorem je *křížení se zachováním pořadí (OX – order crossover)*. Jak už z názvu vyplývá, je operátor navržen pro problémy, u nichž více než na absolutních pozicích alel záleží na jejich relativním uspořádání. Detailně je postup křížení vysvětlen na příkladu křížení chromozomů r_1 a r_2 [1]:

$$r_1 = (a, b, c, d, \underline{e, f, g}, h, i) \qquad r_2 = (h, g, d, i, a, b, e, f, c)$$

Z chromozomu r_1 je zvolen náhodný podřetězec k libovolné délky:

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

Geny, jejichž alely jsou obsaženy v k, jsou následně vyjmuty z r_2, čímž je vytvořena jeho zkrácená varianta r_2':

$$r_2' = (h, d, i, a, b, c)$$

Takto zkrácený chromozom r_2' již nemůže při kombinaci s k porušit platnost permutace, protože oba obsahují pouze rozdílné formy genů. Podřetězec prvního chromozomu k je tedy jednoduše vložen do r_2' na pozici odpovídající jeho původnímu umístění v r_1:

$$o = (h, d, i, a, \underline{e, f, g}, b, c)$$

Dalším operátorem je *křížení založené na zachování pozice (PBX – position based crossover)*. Ten lze považovat za zobecněné křížení se zachováním pořadí, jež přesouvá nesouvislý podřetězec genů. Operátor je znovu vysvětlen na příkladu s chromozomy r_1 a r_2 [1]:

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

Pro výběr genů, jejichž hodnota má zůstat zachována, se nejdříve náhodně vygeneruje bitové schéma l:

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

Geny z prvního rodiče p_1, které odpovídají jedničkám ve schématu l, jsou překopírovány do potomka o:

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

Chromozom druhého rodiče je následně zkrácen tak, aby neobsahoval alely, které se již vyskytují v o:

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

Geny z p_2' jsou nakonec jednoduše vloženy na volné pozice v potomkovi:

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

[3] V převzatém příkladu k operátoru *PMX* z knihy Josefa Hynka *Genetické algoritmy a genetické programování* [10] není vícenásobné užití přepisovacích pravidel nutné – v popisu algoritmu zde ani není zmíněno. V některých případech je však nutné opakovanou záměnu provést [16].

Sexuálních operátorů pro permutační kódování je velké množství, jako další můžeme zmínit například cyklický operátor křížení (CX – cycle crossover) nebo křížení založené na pořadí (OBX – order based crossover) [1]. Vlastnosti operátorů křížení detailně popsaných v této kapitole, se však jeví jako nejvhodnější pro úlohu plánování rozvrhů. Z toho důvodu nejsou tyto a další operátory podrobněji rozebírany.

Permutační asexuální operátory, které pracují pouze s jedním jedincem, jsou již z principu jednodušší. V případě operátoru *mutace* požadujeme, aby byl genotyp jedince mírně pozměněn. Toho lze dosáhnout prostou záměnou dvou nebo více genů mezi sebou. Na následující ukázce se zamění geny na pozicích 2 a 5 [10]:

$$(g, \underline{b}, i, d, \underline{e}, a, f, h, c) \quad \longrightarrow \quad (g, \underline{e}, i, d, \underline{b}, a, f, h, c)$$

Za speciální případ mutace lze považovat operátor *inverze*, který obratí pořadí genů v náhodné části chromozomu [10]:

$$(g, b, i, d, \underline{e, a, f, h}, c) \quad \longrightarrow \quad (g, b, i, d, \underline{h, f, a, e}, c)$$

3.2 Hybridní genetické algoritmy

Při navrhování genetického algoritmu se nelze spoléhat pouze na výše uvedené postupy, protože podle nich vytvořený algoritmus by zkoumal prohledávaný prostor víceméně slepě. Ve snaze dosáhnout co nejlepších výsledků, je téměř ve všech případech nutné algoritmus přizpůsobit řešenému problému. V mnoha případech se nabízí inspirace tradičními technikami jeho řešení. Takto navržené algoritmy nazýváme *hybridní* [10].

Jedním z takových přístupů může být hybridní algoritmus pro vytváření školních rozvrhů, který kombinuje genetické a heuristické metody. Díky němu je celý rozvrhovací problém převeden na hledání optimálního rozmístění akcí ve dvoudimenzionální matici. Každá akce v této matici musí být přiřazena buňkám v předem stanoveném sloupci. Pořadí, v jakém budou akce do matice umísťovány, je zjištěno právě pomocí permutačního genetického algoritmu. Tento relativně obecný postup vykazuje potenciál pro řešení různých plánovacích problémů a je více přiblížen v kapitole 4 [17].

Jiný hybridní postup pro plánování rozvrhů na základních školách jsem navrhl ve své bakalářské práci z roku 2012 [9]. Implementovaný program se skládá ze dvou po sobě následujících genetických algoritmů. V první fázi jsou vytvořeny vektory předmětů, které budou vyučovány paralelně. Druhý genetický algoritmus těmto vektorům přidělí vhodná časová okna, čímž vytvoří výsledný rozvrh.

Tento přístup není možné bez výrazných změn aplikovat na problém tvorby rozvrhů na vysoké škole, protože kritéria obou institucí jsou výrazně odlišná. Uvedený návrh byl zvolen hlavně z důvodu, že v rozvrzích žáků na základních školách se ve většině případů nesmí vyskytovat volné hodiny. Z tohoto požadavku mimo jiné vychází i kódování chromozomů. V případě rozvrhu na vysoké škole je však tato klíčová vlastnost vytvořeného algoritmu naopak nežádoucí.

3.3 Multikriteriální optimalizace

U optimalizačních problémů je naším cílem nalézt takovou formu řešení, která co nejvíce vyhovuje daným kritériím (tj. jeho účelové funkci). V případě řešení reálných problémů se ale často setkáváme s více než jedním kritériem. Takovéto problémy tedy definují i více

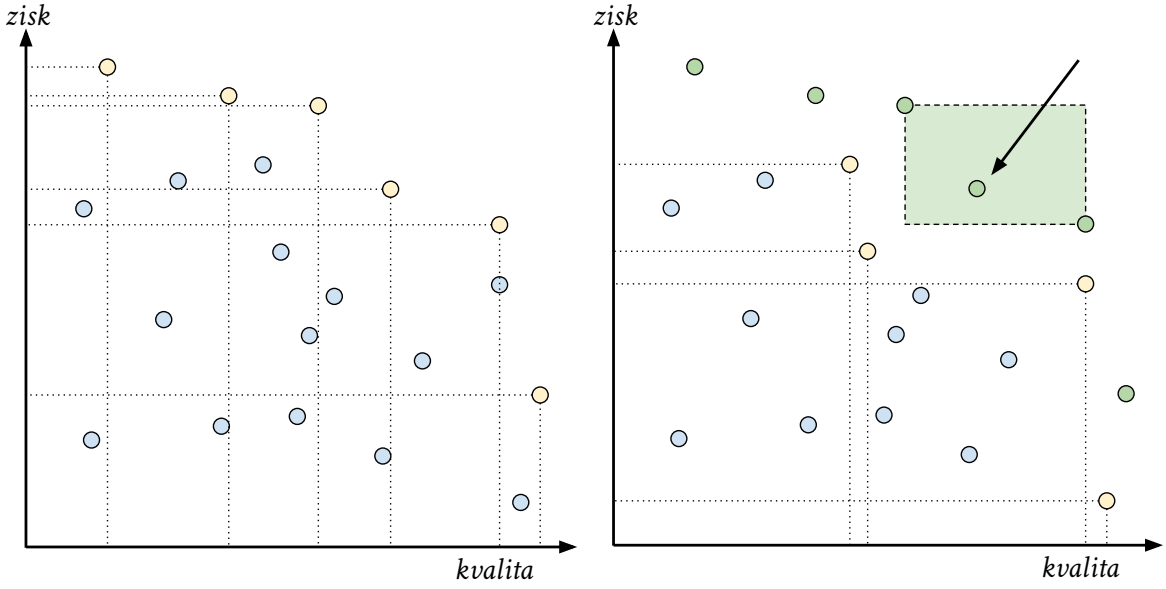
účelových funkcí, které navíc mohou být protichůdné – musíme tedy dosáhnout určitého kompromisu [19].

Pro každého jedince genetického algoritmu můžeme nezávisle vyhodnotit všechny účelové funkce a jeho kvalitu reprezentovat jako vektor reálných čísel (viz kapitola 3). Problém nastává v okamžiku, kdy potřebujeme na takto ohodnocené jedince uplatnit selekční postupy. Předpokladem klasických způsobů selekce je totiž relace lineárního uspořádání nad množinou fitness jedinců. V případě vektoru účelových hodnot však existuje pouze relace částečného uspořádání – není tedy zaručeno, že všechny prvky budou vzájemně porovnatelné. Prakticky tak někdy nemůžeme určit, které řešení je lepší než jiné. Východisko nabízí právě metody multikriteriální optimalizace, díky nimž dokážeme vektor výstupů účelových funkcí převést na jednu fitness hodnotu.

Zaveďme pojem *Pareto dominance* (značíme $\leq_{par}$) právě jako relaci částečného uspořádání nad množinou jedinců. Pro problém o n kriteriích platí, že jedinec A s vektorem fitness $f_a = (f_{a1}, \ldots, f_{an})$ dominuje jedinci B s fitness $f_b = (f_{b1}, \ldots, f_{bn})$ právě a jen tehdy, když není v žádném z kritérií horší a zároveň je alespoň v jednom lepší, tedy:

$$A \leq_{par} B \quad \Leftrightarrow \quad \forall i \in \mathbb{N}_{\leq n} \exists j \in \mathbb{N}_{\leq n} : f_{ai} \geq f_{bi} \wedge f_{aj} > f_{bj}$$

Řešení multikriteriálních problémů na základě Pareto dominance je široce uznávané, protože se jedná o přímé zobecnění tradičních metod. Na obrázku 3.3*a* je znázorněn prostor dominance od některých jedinců pomocí tečkovaných čar – všem modře označeným řešením ve vyznačených oblastech (včetně jejich okrajů) dominuje alespoň jeden žlutě označený jedinec.



(a) (b)

Obrázek 3.3: Znázornění množiny potenciálních řešení hypotetického problému, který zahrnuje dvě protichůdná kritéria – předpokládaný zisk vs. kvalitu produktu.

Pomocí Pareto dominance dále můžeme definovat tzv. *Paretovu frontu*, což je podmnožina všech nalezených řešení, jejichž prvkům žádné jiné řešení nedominuje. Tato fronta je na příkladu z obrázku 3.3a znázorněna žlutě. Všechny její prvky jsou vzájemně neporovnatelné,

tudíž nemůžeme bez doplňujících znalostí říci, který z nich je lepší. Víme ale, že se jedná o nejlepší nám známá řešení. Pomocí tohoto principu můžeme začít přiřazovat jedincům konkrétní hodnoty fitness.

Jedním z možných postupů je *řazení na základě nedominance (nondominated sorting)*, které provede rozklad na množině známých řešení do Paretových front. Postup tohoto algoritmu je znázorněn na obrázku 3.3b. Nejprve se na celé množině určí prvky Paretovy fronty podle předchozí definice (na obrázku zeleně). Jedná se o nejlepší jedince, tudíž je jim přiřazena nejvyšší hodnota fitness – protože jsou prvky neporovnatelné, je fitness pro všechny stejná. Poté jsou tyto prvky vyjmuty z množiny a je znovu nalezena Paretova fronta (žlutě). Jedná se o podmnožinu druhých nejlepších jedinců, kterým je přiřazena nižší fitness než předchozí frontě. Uvedený postup se opakuje, dokud není množina prázdná. Řazení na základě dominance sice nevytvoří na množině známých řešení lineární uspořádání, následná analýza vytvořených front je však dobrým prostředkem k jeho dosažení [15, 19].

Tímto způsobem pracuje i známý algoritmus *NSGA-II (Non-dominated Sorting Genetic Algorithm II)*. Ten jedince uvnitř fronty řadí podle diverzity, kterou jejich přítomnost přináší do populace. Jeho hlavní předností je, že k výpočtu diverzity nevyžaduje žádný předem definovaný parametr.

Výpočet probíhá na základě tzv. *shlukovací vzdálenosti (angl. crowding distance)* – pro každý prvek Paretovy fronty jsou nalezeni oba jeho sousedé a je určena jejich manhattonská vzdálenost. Tento výpočet je pro šipkou označeného jedince znázorněn na obrázku 3.3b zeleným obdélníkem (pro vícedimenzionální případy se jedná o kuboid), jehož obvod je přímo úměrný právě shlukovací vzdálenosti. Protože jsou váhy všech kritérií shodné, je před započetím výpočtu nutné znormalizovat účelové funkce tak, aby byl jejich vliv na shlukovací vzdálenost vyrovnaný.

Nechť pro jedince X značí X_f jeho fitness na základě příslušné Paretovy fronty a X_d jeho shlukovací vzdálenost. Potom je možné definovat porovnávací operátor $\prec$, který vytvoří relaci lineárního uspořádání nad množinou známých řešení jako:

$$A \prec B \quad \Leftrightarrow \quad \big(A_f < B_f\big) \vee \big((A_f = B_f) \wedge (A_d < B_d)\big)$$

Smyčka algoritmu *NSGA-II* pro populaci s kardinalitou n potom probíhá následovně. Pomocí rekombinačních operátorů je z n rodičů, kteří jsou z populace vybíráni náhodně, vytvořeno n potomků. Všichni tito jedinci jsou spojeni do jedné populace o velikosti $2n$. Ta je zmenšena na původní velikost tak, že je podle operátoru $\prec$ vybráno n nejlepších z nich. Celý cyklus se poté klasicky opakuje [3].

Kapitola 4

Návrh programu pro rozvrhování na FIT VUT v Brně

Návrh programu vychází z článku *Solving Timetabling Problem Using Genetic and Heuristic Algorithms* [17] z roku 2007. Ten navrhuje obecnou šablonu pro řešení rozvrhovacích problémů za pomoci hybridního algoritmu, který kombinuje genetické a heuristické postupy. Šablonu lze využít jak pro generování zkouškového, tak i přednáškového rozvrhu na *FIT*. Celý návrh může díky tomu být od svého prvopočátku vytvářen tak obecným způsobem, aby byl bez výraznějších změn vyhovující pro oba typy rozvrhů (podle typu rozvrhu se mění pouze deterministické rozhodování v heuristické části).

V dalších částech této kapitoly bude užíváno zjednodušení na *přednášky* – uvedené skutečnosti však platí ekvivalentně i pro *zkoušky*. V případech, kdy je tomu jinak, bude rozdíl mezi oběma typy rozvrhů explicitně uveden.

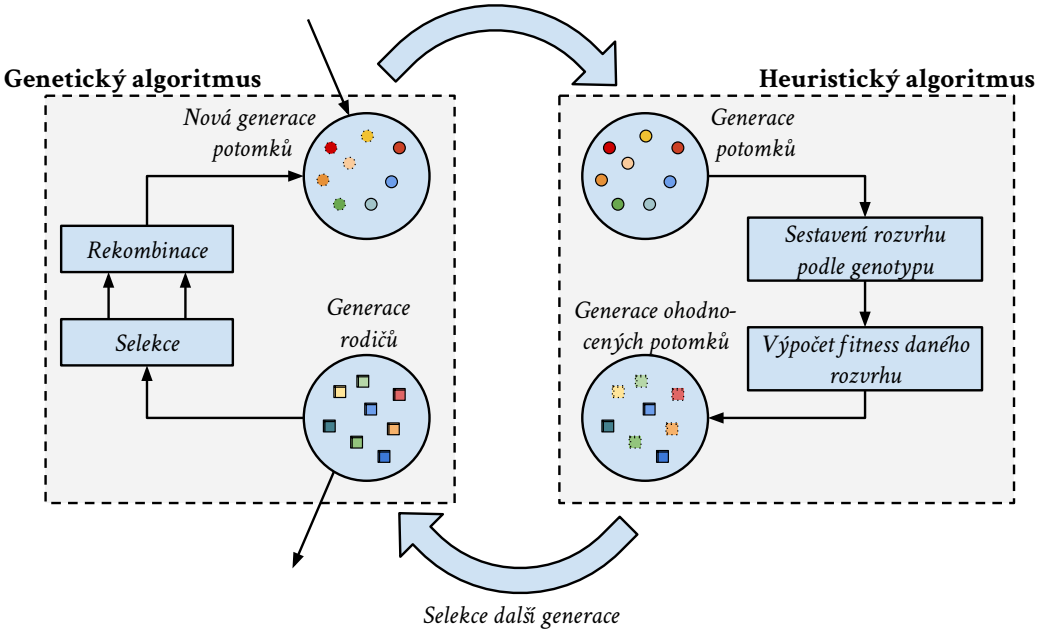
Algoritmus je řízen jeho genetickou částí, která k ohodnocení kvality jedinců využívá heuristickou část. Genotyp jedinců totiž obsahuje pouze návod k sestavení rozvrhu, ze kterého nelze přímo vyvodit jeho kvalitu. Heuristická část na základě genotypu rozvrh sestaví a určí jemu odpovídající hodnotu fitness.

Vstupem algoritmu je množina *faktorů*, které je třeba naplánovat do dostupných *zdrojů*. Pod pojmem *faktor* se míní trojice *předmět*, množina *přednášejících*, kteří ho vedou, a množina participujících *studentů*. Zaveďme pojem *časový slot* jako dvojici den a hodina. *Zdrojem* se poté míní prvek z kartézské součinu časových slotů a místností. Chromozomy jedinců jsou kódovány právě jako permutace množiny faktorů. Uvedené vztahy jsou popsány následujícími rovnicemi:

$$\begin{aligned} faktor &:= \langle předmět, \{přednášející\}, \{studenti\} \rangle \\ časový\ slot &:= \langle den, hodina \rangle \\ zdroj &\in \{časové\ sloty\} \times \{místnosti\}. \end{aligned} \tag{4.1}$$

Algoritmus vyžaduje, aby byly tyto trojice faktorů pevně zvoleny uživatelem ještě před začátkem výpočtu. To může být problém pro ty typy rozvrhů, u kterých je třeba naplánovat i přiřazení vyučujících podle jejich aprobace k předmětům. V případě *FIT* je však uvedená podmínka jednoduše splnitelná, protože tato vazba se vytváří již při studentské registraci předmětů, která plánování rozvrhů bezpodmínečně předchází.

Pseudokód hybridního algoritmu znázorňuje algoritmus 1. Na obrázku 4.1 je navíc zobrazeno schéma spolupráce mezi genetickým a heuristickým algoritmem. Pro každou generaci se provádí výpočet v obou z nich následovně. Rodičovská generace je v první fázi pomocí



Obrázek 4.1: Schéma navrženého hybridního algoritmu.

genetického algoritmu transformována do generace potomků (levá část obrázku, řádek 4 algoritmu). V případě první generace je tento krok přeskočen a na pozici potomků je přímo vložena počáteční generace (šipka vstupující do obrázku). V následné druhé fázi probíhá mapování genotypu těchto potomků na odpovídající fenotypy heuristickým algoritmem (pravá část obrázku, řádek 5 algoritmu). Takto získaný fenotyp jedince již reprezentuje konkrétní rozvrh a je možné vyhodnotit jeho fitness.

Následně pokračuje iterace genetického algoritmu. Provede se selekce, čímž se potomci stávají novými rodiči, a obě fáze se opakují. V případě, že je výpočet u konce (je dosaženo řešení požadované kvality nebo zvoleného počtu generací), jsou vybráni nejlepší jedinci, kteří budou zobrazeni uživateli (šipka vystupující z obrázku).

Rozdíl mezi generováním zkouškového nebo přednáškového rozvrhu na *FIT* se projevuje pouze změnou kritérií (umístění předmětů pouze do jednoho týdne namísto přibližně pěti týdnů, hlídání obědových pauz, koncentrování výuky jen do některých dnů namísto snahy o rovnoměrné rozložení a další), podle kterých jsou konstruovány rozvrhy v heuristické části.

4.1 Genetický algoritmus

Chromozom jedince (ve formě permutace jednotlivých faktorů) reprezentuje pořadí, ve kterém budou faktory do rozvrhu postupně vkládány heuristickou částí. Délka této permutace tedy odpovídá celkovému počtu faktorů (v praxi se jedná o desítky až stovky položek). Předpokladem je, že nejobtížněji umístitelné faktory se budou v čase dostávat na přední pozice chromozomů. Naopak faktory s nižšími nároky na umístění v rozvrhu se budou častěji objevovat na jejich zadních pozicích. Cílem genetické části algoritmu je tedy nalézt takové pořadí faktorů, jejichž „vhodným umístěním“ (viz druhá fáze) bude dosaženo co nejlepších rozvrhů.

Algoritmus 1: Pseudokód prezentovaného hybridního algoritmu. Modře jsou zvýrazněny výpočty v rámci jeho heuristické části.

```
1:  Generování počáteční populace
2:  for požadovaný počet generací do
3:      if not první iterace cyklu then
4:          Vytváření potomků                          // rekombinace
5:      end
6:      Sestavení rozvrhů                    // genotyp → fenotyp
7:      Určení fitness jedinců                 // jejich fenotypů
8:      Selekce rodičů                                  // NSGA-II
9:      if Nalezeno dostatečně kvalitní řešení then
10:         break
11:     end
12: end
13: return množina nejlepších řešení, která si vzájemně nedominují
```

Složení počáteční populace může být generováno náhodně. Ukazuje se však, že po vložení několika vhodně vytvořených permutací, může algoritmus dosáhnout řešení rychleji. Jako přímočarý postup se jeví odhadnout náročnost faktorů na umístění podle různě zvolených kritérií a následně je seřadit v permutaci tak, aby nejnáročnější faktory byly plánovány dříve [17].

Šablona [17] nepředepisuje konkrétní typy rekombinačních operátorů ani způsob selekce jedinců. V článku jsou pouze uvedena doporučení, která byla použita při jejím testování. Protože problém plánování rozvrhů na *FIT* je multikriteriální optimalizační úloha (viz kapitola 2.3), jeví se vhodné pro její řešení použít multikriteriální optimalizační algoritmus. V našem případě jsme zvolili jeden ze standardních algoritmů, konkrétně *NSGA-II* (viz kapitola 3.3).

Pro větší množství měkkých kritérií byl zvažován také *NSGA-III* [2] z důvodu, že u vybraného *NSGA-II* může dojít k významnému poklesu selekčního tlaku v závislosti na narůstajícím počtu kritérií. V průběhu řešení se však ukázalo, že jednodušší algoritmus je dostačující, protože bylo možné některá měkká kritéria agregovat do jednoho a jiná transformovat na tvrdá. Prakticky tak nebylo nutné pracovat s více než třemi kritérii.

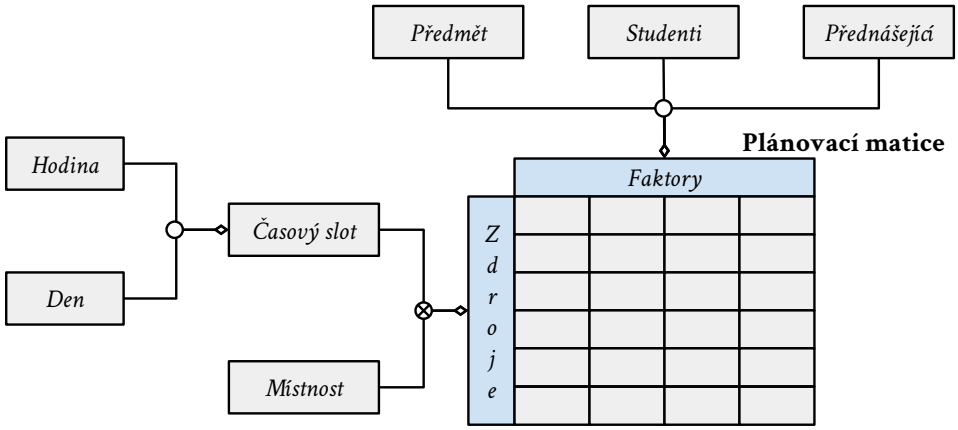
Součástí *NSGA-II* je již způsob selekce jedinců pro rekombinaci. Operátory pro mutaci a křížení však nejsou striktně určeny. Zvolený operátor mutace provede záměnu dvou náhodných genů chromozomu, což je běžný postup u permutačního kódování [10].

V případě operátorů křížení lze jen těžko dopředu předpovídat jejich vliv na průběh evoluce daného problému. Protože jejich implementace nebývá složitá, vybrali jsme rovnou tři permutační operátory, které se jeví jako vhodné. Uživatel tak bude mít možnost před spuštěním výpočtu zvolit jeden z nich, případně otestovat všechny a zvolit pro něj ten nejlepší. Konkrétně jsme vybrali *křížení s částečným přiřazením (PMX – partially matched crossover)* [10], *křížení se zachováním pořadí (OX – order crossover)* [1] a *křížení založené na zachování pozice (PBX – position based crossover)* [1]. Uvedené genetické operátory jsou detailně popsány v kapitole 3.1.1).

4.2 Heuristický algoritmus

Úkolem heuristické části algoritmu je sestrojit odpovídající fenotyp (rozvrh) na základě jedincova genotypu (permutace faktorů). To provádí formou dvoudimenzionální matice nazývané *plánovací matice* (angl. *target matrix*). Mezi plánovací maticí a výsledným rozvrhem existuje bijektivní zobrazení. Převod plánovací matice na běžný rozvrh je přímočará záležitost, avšak použití této matice při generování rozvrhu výrazně zjednodušuje celý heuristický postup.

Jak lze vidět na obrázku 4.2, plánovací matice se skládá z buněk kartézského součinu *faktorů* (sloupce) a *zdrojů* (řádky). Každá její buňka tedy reprezentuje vztah mezi jedním *faktorem* a právě jedním *zdrojem*. Rozměr této matice je v případě zkouškového rozvrhu na *FIT* přibližně 2600 řádků (zdrojů) a 125 sloupců (faktorů) – tedy asi 325 000 buněk.



Obrázek 4.2: Hierarchické zobrazení *faktorů* a *zdrojů* a jejich vztah k *plánovací matici*. Značka $\otimes$ značí kartézský součin množin prvků. Značka $\bigcirc$ podmnožinu kartézského součinu.

V průběhu generování rozvrhu jsou do buněk plánovací matice zanášeny informace tak, aby se dalo vždy jednoduše zjistit, jak přiřazení faktoru ke zdroji ovlivní jednotlivá kritéria. Buňky matice z tohoto důvodu nabývají právě jednu ze tří hodnot:

- *Prázdná:* zdroj je dostupný pro přiřazení faktoru – výchozí hodnota (na obrázcích značeno jako prázdná buňka).
- *Nedostupná:* zdroj nemůže být přiřazen faktoru, protože by takové přiřazení porušilo tvrdé kritérium (na následujících obrázcích značeno hodnotou $\times$).
- *Přiřazená:* zdroj je přiřazen faktoru (na obrázcích značeno hodnotou $\bullet$).

Pro faktory, jejichž doba trvání překračuje jeden časový slot (v případě *FIT* se jedná o naprostou většinu faktorů, tj. přednáška nebo zkouška trvá dvě a více hodin), je nutné obsadit po sobě jdoucí zdroje. To jsou zdroje, které obsahují stejnou místnost a jejichž časové sloty na sebe navazují v průběhu jednoho dne. Aby bylo možné takovéto zdroje v plánovací matici efektivně vyhledat, je vhodné, aby v ní odpovídajícím způsobem následovaly. Toho dosáhneme seřazením zdrojů primárně podle místností a sekundárně podle časových slotů. Uvažme uspořádanou množinu všech učeben U o mohutnosti $|U| = n$ a uspořádanou množinu všech časových slotů CS o mohutnosti $|CS| = m$. Pořadí zdrojů $\langle u, cs\rangle$ pro $u \in U$ a $cs \in CS$ v plánovací matici poté odpovídá následující notaci:

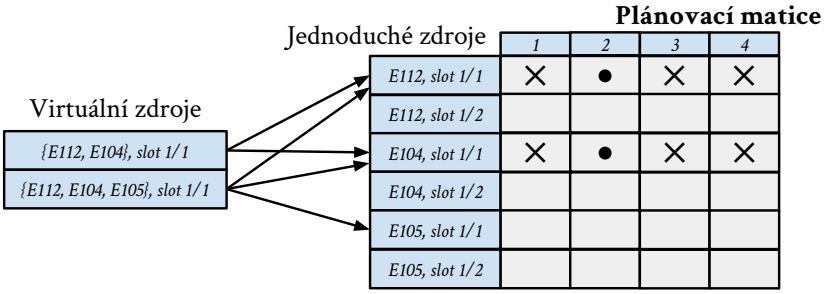
$$Zdroje := (\langle u_1, cs_1\rangle, \langle u_1, cs_2\rangle, \ldots, \langle u_1, cs_m\rangle, \langle u_2, cs_1\rangle, \ldots \langle u_2, cs_m\rangle, \ldots, \langle u_n, cs_m\rangle). \quad (4.2)$$

V důsledku toho obsahuje správně sestavená plánovací matice (taková, která má přiřazeny všechny faktory nějakým zdrojů a neporušuje přitom žádná tvrdá kritéria) v každém sloupci právě jednu souvislou sekvenci hodnot „přiřazená“. Výjimku představuje přiřazení tzv. *virtuálního zdroje*.

Jedním ze specifických požadavků *FIT* je potřeba plánovat některé přednášky paralelně do více učeben. Z toho důvodu jsme zavedli *virtuální zdroje*, o které je rozšířena množina předchozích „jednoduchých“ zdrojů. Virtuální zdroj reprezentuje množinu učeben pro jeden *časový slot*, je tak v tomto ohledu zobecněním jednoduchého zdroje. Nejsou mu však přímo přiřazeny žádné její řádky – jak je zobrazeno na obrázku 4.3. Svým významem totiž sdružuje ty jednoduché zdroje z z množiny jednoduchých zdrojů Z, které sdílí jeden časový slot cs a obsahují příslušné učebny U_s:

$$\textit{virtuální zdroj } \langle cs, U_s \rangle = \langle cs, \{z \,|\, \forall z \ \in Z : z(0) = cs \ \wedge \ z(1) \in U_s\} \rangle. \tag{4.3}$$

Díky uvedené reprezentaci virtuálních zdrojů je implicitně zajištěna synchronizace hodnot v plánovací matici – takovýto zdroj lze alokovat pouze v případě, že jsou všechny jeho podzdroje volné a naopak jeho alokace přiřadí všechny odpovídající podzdroje k faktoru (viz obrázek 4.3). Kapacita virtuálního zdroje je rovna součtu kapacit jeho podzdrojů. Přiřazení faktoru k virtuálnímu zdroji vytvoří v daném sloupci množinu posloupností (o délce odpovídající počtu hodin faktoru) s rozestupy $k \cdot$ *počet časových slotů* pro $k \in \mathbb{N}$. V případě jiné reprezentace, kdy by byly těmto zdrojům přiřazeny zvláštní řádky plánovací matice, by se v ní objevovaly duplicitní informace, které by mohly vést k nekonzistencím.



Obrázek 4.3: Ukázka způsobu mapování dvou virtuálních zdrojů na jednoduché zdroje. V plánovací matici je zobrazeno přiřazení druhého faktoru virtuálnímu zdroji „$(E112, E104)$, *slot* $1/1$“. V důsledku toho jsou bežné zdroje „$E112$, *slot* $1/1$“ a „$E104$, *slot* $1/1$“ a virtuální zdroj „$(E112, E104, E105)$, *slot* $1/1$“ nepřiraditelné žádnému dalšímu faktoru.

V případě plánování rozvrhů na *FIT* existují předem dané skupiny učeben, v rámci kterých probíhá paralelní výuka. Tyto skupiny jsou dány především fyzickým oddělením komplexů D a E. Program umožňuje označit množinu učeben U_v, ze kterých sám vytvoří virtuální zdroje jako prvky potenční množiny této množiny s mohutností větší než jedna. Navíc je možné definovat učebnu $u_{hlavní}$ ($u_{hlavní} \in U_v$), která se musí nacházet v každém takovém virtuálním zdroji (vhodné v případě, že existuje jedna hlavní učebna, ze které se výuka promítá do ostatních). Takto vytvořenou množinu virtuálních zdrojů Z_v pro všechny časové sloty CS lze formálně vyjádřit jako:

$$Z_v = \{\langle cs, U_p \rangle \,|\, \forall cs \in CS \ \ \forall U_p \in 2^{U_v} : (|U_p| > 1) \wedge (u_{hlavní} \in U_p)\} \tag{4.4}$$

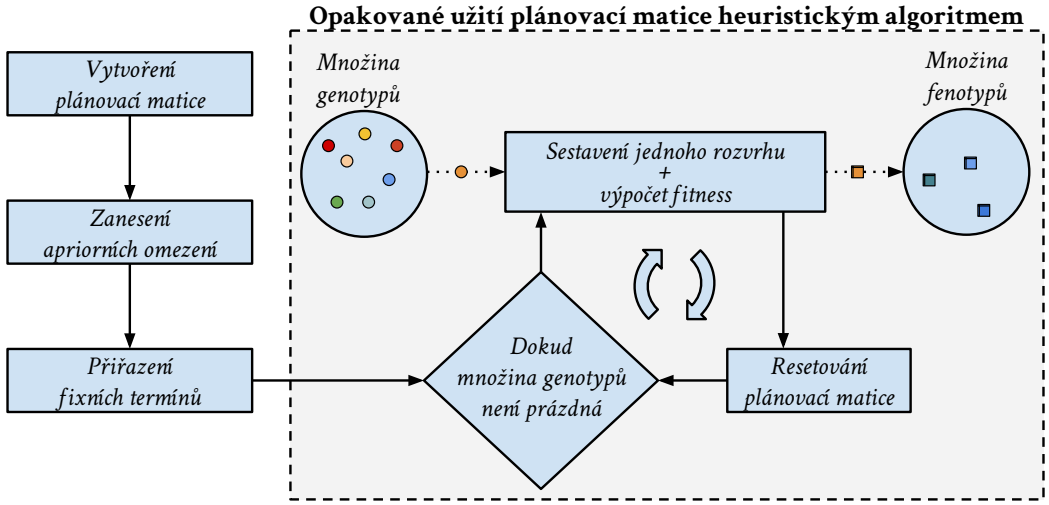
V případě zkoušek je pro největší předměty na *FIT* (především ty bakalářské) potřeba alokovat všechny dostupné učebny U. Z toho důvodu program ještě automaticky vytváří množinu virtuálních zdrojů R_{max}:

$$R_{max} = \{\langle cs, U\rangle \,|\, \forall cs \in CS\} \tag{4.5}$$

V případě zkouškového rozvrhu na *FIT* se přidáním virtuálních zdrojů zvýší celkový počet zdrojů z přibližně 2600 na 4300 (tj. asi o 65 %). Stejnou úměrou se také zvýší náročnost prohledávání plánovací matice.

4.2.1 Práce s plánovací maticí

Schéma znázorňující vývojový cyklus plánovací matice je zobrazeno na obrázku 4.4. Ta je vytvořena jako kartézská mřížka zdrojů a faktorů. Před spuštěním heuristického algoritmu jsou do plánovací matice na příslušná místa zanesena ta tvrdá kritéria, která apriori nedovolují přiřadit některé zdroje k některým faktorům. Jedná se například o situace, kdy nedostačuje kapacita místnosti ve zdroji nebo má v daném časovém slotu přednášející naplánovanou jinou činnost a další (všechna tvrdá omezení jsou uvedena v kapitole 2.3). Do plánovací matice jsou na tato místa zaneseny hodnoty „nedostupné“.



Obrázek 4.4: Vývojový cyklus *plánovací matice*, která je po vytvoření a počáteční inicializaci (levá část obrázku) opakovaně využívána pro mapování genotypu jedince na jeho fenotyp (pravá část obrázku). Při každé iteraci je sestaven právě jeden rozvrh a následně je obsah matice navrácen do stavu po „přiřazení fixních termínů“, aby mohl být sestaven další fenotyp.

V dalším kroku jsou v plánovací matici přiřazeny kombinace zdrojů a faktorů, jež zvolil napevno uživatel – jedná se například o předměty organizované FEKT (přesný způsob, jakým probíhá přiřazení faktoru zdroji, je popsán v následujících odstavcích).

Nyní může být spuštěn vlastní heuristický algoritmus. Ten vybírá faktory v pořadí, které odpovídá hodnotám genotypu daného jedince (již naplánované faktory z předchozího kroku přeskakuje). Pro každý faktor prochází příslušný sloupec matice a hledá posloupnost zdrojů v jednom dni s hodnotou „prázdné“. Velikost hledané posloupnosti odpovídá době trvání (počtu časových slotů) faktoru. Pokud algoritmus takovou posloupnost nenalezne, je

konstrukce rozvrhu ukončena a jedinci je přiřazena fitness odpovídající počtu zdrojů, které se do té doby podařilo přiřadit.

V případě, že posloupnost nalezne, označí buňky odpovídající faktoru a nalezeným zdrojů hodnotou „přiřazené“. Všem ostatním buňkám, které jsou ve stejných řádcích, přiřadí hodnotu „nedostupné“ (zdroje jsou tímto blokovány pro ostatní faktory).

Stejně tak označí hodnotou „nedostupné“ i všechny buňky, ve kterých se vyskytuje zdroj se stejným časovým slotem a faktor se stejným přednášejícím. Pokud by k tomu nedošlo, mohlo by být porušeno kritérium, kdy jeden přednášející nemůže přednášet dva různé předměty současně. Tyto buňky se mohou nacházet na libovolných pozicích v plánovací matici a není možné je při každém přiřazení slepě vyhledávat. Jejich pozice však lze pro každou buňku předpočítat, protože rozměr plánovací matice se po celou dobu výpočtu nemění.

Ve všech případech, kdy heuristický algoritmus označuje buňku hodnotou „nedostupná“, má tato buňka hodnotu „prázdná“ nebo hodnotu „nedostupná“ již nabývá. Buňka označovaná za nedostupnou nemůže nikdy mít hodnotu „přiřazená“, protože by to znamenalo, že poslední přiřazení porušilo tvrdé kritérium, a tedy že plánovací matice byla před tímto přiřazením v nekonzistentním stavu.

Analogickým způsobem lze do plánovací matice značit i měkká kritéria. V případě jejich porušení však nedochází k ukončení výpočtu. Značení měkkých kritérií slouží pouze k tomu, aby algoritmus v případě nalezení více přípustných posloupností zdrojů vybral tu, která poruší měkká kritéria nejméně. Značení měkkých kritérií pro hlídání studentských kolizí a preferencí vyučujících je popsáno v kapitole 4.2.3.

4.2.2 Příklad generování jednoduchého rozvrhu

Pro lepší pochopení, jakým způsobem probíhá konstrukce rozvrhu pomocí heuristického algoritmu, je zde popsán jednoduchý příklad. Uvažujme sedm zdrojů a tři faktory, z nichž každý vyžaduje přiřazení ke dvěma časovým slotům. Stav příslušné plánovací matice v jednotlivých krocích algoritmu je zobrazen na obrázku 4.5.

V prvním kroku jsou do matice zanesena apriorní tvrdá kritéria, která odpovídají následujícím předpokladům:

- Místnost $E112$ je z důvodu údržby obsazena v časovém slotu 1/1 (oranžově).
- Místnost $E105$ má nedostatečnou kapacitu pro předmět IZP (žlutě).
- Přednášející K je zaneprázdněn v časovém slotu 1/1 (červeně).

V druhém kroku jsou do matice umístěny faktory s pevně stanovenými zdroji. Nechť je faktor číslo 2 $\langle ROB, R, MG\rangle$ přiřazen do obou zdrojů místnosti $E105$ (oranžově). Potom je všem ostatním buňkám v matici, které odpovídají stejným zdrojům, nastavena hodnota „nedostupné“ (červeně) – je ovlivněna hodnota pouze jedné další buňky.

Nyní je plánovací matice připravena pro generování rozvrhů odpovídajících jedincům. V kroku 3*a* předpokládejme, že se má plánovat rozvrh pro jedince s genotypem *(2,1,3)*. Faktor 2 je přeskočen, protože byl již naplánován. Následující faktor 1 je umístěn do volných zdrojů místnosti $E112$. Zbývá faktor 3, který v tomto případě nelze naplánovat, protože pro něj neexistuje volná posloupnost alespoň dvou zdrojů. Parciální fitness přiřazená tomuto jedinci je tedy 4, protože se podařilo obsadit 4 zdroje. Další složky fitness jsou určeny na základě měkkých kritérií (viz kapitola 4.3).

Jako druhý je plánován jedinec s genotypem *(3,2,1)*. První faktor s číslem 3 je naplánován do místnosti $E112$, faktor 2 je znovu přeskočen a faktor 1 je umístěn do místnosti

$D105$. Podle tohoto genotypu se podařilo naplánovat celý rozvrh, jedinci je tedy přiřazena parciální fitness s hodnotou 6. Tímto způsobem algoritmus vytvoří všechny rozvrhy a určí jejich fitness, na základě které jsou vybrány genotypy jedinců k další rekombinaci.

		Faktory		
		IZP, S, BIT1	ROB, R, MG	IPP, K, BIT2
Zdroje	D105, Slot 1/1			×
	D105, Slot 1/2			
	E112, Slot 1/1	×	×	×
	E112, Slot 1/2			
	E112, Slot 1/3			
	E105, Slot 1/1	×		×
	E105, Slot 1/2	×		

Krok 1: Stav matice po zahrnutí apriorních tvrdých kritérií.

		Faktory		
		IZP, S, BIT1	ROB, R, MG	IPP, K, BIT2
Zdroje	D105, Slot 1/1			×
	D105, Slot 1/2			
	E112, Slot 1/1	×	×	×
	E112, Slot 1/2			
	E112, Slot 1/3			
	E105, Slot 1/1	×	●	×
	E105, Slot 1/2	×	●	×

Krok 2: Stav matice po vložení pevně naplánovaného faktoru číslo 2.

		Faktory		
		IZP, S, BIT1	ROB, R, MG	IPP, K, BIT2
Zdroje	D105, Slot 1/1			×
	D105, Slot 1/2			
	E112, Slot 1/1	×	×	×
	E112, Slot 1/2	●	×	×
	E112, Slot 1/3	●	×	×
	E105, Slot 1/1	×	●	×
	E105, Slot 1/2	×	●	×

Krok 3a: Stav matice po dokončení plánování pro permutaci *(2,1,3)*.

		Faktory		
		IZP, S, BIT1	ROB, R, MG	IPP, K, BIT2
Zdroje	D105, Slot 1/1	●	×	×
	D105, Slot 1/2	●	×	×
	E112, Slot 1/1	×	×	×
	E112, Slot 1/2	×	×	●
	E112, Slot 1/3	×	×	●
	E105, Slot 1/1	×	●	×
	E105, Slot 1/2	×	●	×

Krok 3b: Stav matice po dokončení plánování pro permutaci *(3,2,1)*.

Obrázek 4.5: Jednoduchý příklad práce heuristického algoritmu s plánovací maticí. Faktory jsou uváděny v pořadí ⟨*předmět, přednášející, skupina studentů*⟩ a zdroje v pořadí ⟨*učebna, časový slot*⟩.

4.2.3 Výběr zdrojů na základě měkkých kritérií

Ve chvíli, kdy algoritmus hledá posloupnost „prázdných“ zdrojů dané délky pro naplánování faktoru, může dojít k situaci, kdy přípustných posloupností existuje dvě nebo více. V tom případě je nutné jednu z nich zvolit. Protože cílem plánování není pouze přiřadit všem faktorům nějaké zdroje bez porušení tvrdých kritérií, ale i dosáhnout kvalitního uspořádání faktorů vzhledem ke kritériím měkkým, nabízí se možnost vybrat nejlepší posloupnost právě na základě měkkých kritérií. Podle nich se sice řídí selekční mechanismus, ale pokud k některým z nich přihlédneme již při sestavování rozvrhu, dá se usuzovat, že vytvořené rozvrhy budou obecně kvalitnější. Mimo jiné je díky průběžnému sledování měkkých kritérií možné po ukončení výpočtu jednoduše určit jim odpovídající části fitness.

Měkká kritéria, která je možné při plánování sledovat lokálně, jsou *preference* na umístění faktorů a studentské *kolize*. *Preference* jsou založeny na přáních vyučujícího, jimiž lze vyjádřit, ve které časové sloty by upřednostňoval který faktor. Tuto informaci lze převést na

zdroje, protože jeden časový slot odpovídá množině zdrojů. Prakticky je tak každé buňce plánovací matice přiřazena nezáporná hodnota úměrná míře preference.

Sledování *kolizí* má za cíl hlídat časové rozestupy mezi faktory z pohledu participujících studentů. V případě zkouškového rozvrhu je třeba tyto rozestupy maximalizovat z důvodu, aby měl student dostatek času na odpočinek a zopakování látky před každým zkouškovým termínem. Naopak u rozvrhu přednášek je tlak na shlukování výuky jen do některých dnů týdne takovým způsobem, aby se přednášky z pohledu studenta nepřekrývaly – paralelně plánovat přednášky se společnými studenty je možné pouze u volitelných předmětů. Díky tomu nemusí student do školy dojíždět denně a přitom má pořád možnost navštívit všechny přednášky. Ke zkouškovému nebo přednáškovému rozvrhu lze však z výpočetního hlediska přistupovat podobně, stačí pouze pozměnit význam některých *typů kolizí* (tato problematika je diskutována v kapitole 4.3).

Vraťme se zpět k situaci, kdy heuristický algoritmus plánuje faktor do sloupce *plánovací matice* a z hlediska tvrdých kritérií vyhovuje více než jedna posloupnost zdrojů. Tyto posloupnosti umístí algoritmus do množiny kandidátních posloupností K. Protože chceme v maximální míře splnit *preference* vyučujících, odeberou se z množiny K ty posloupnosti, jejichž suma preferencí je nižší než u některé jiné posloupnosti z této množiny (uvažují se tedy dále pouze posloupnosti s maximální možnou preferencí):

$$K := \arg\max_{k \in K} \Big(\sum_{zdroj \in k} zdroj_{preference} \Big). \tag{4.6}$$

Tento krok je vhodné provést za předpokladu, že jsou časové sloty ohodnoceny pouze několika málo hladinami hodnot *preference*[1]. V důsledku toho tak bude ve většině případů vybraná podmnožina víceprvková a může nad ní být proveden výběr na základě kolizí.

Pro každou ze zbývajících posloupností zdrojů lze v plánovací matici vypočíst hodnotu všech čtyř typů kolizí (viz následující kapitola 4.3), které by případným přiřazením zdrojů k faktoru vznikly (efektivní způsob výpočtu kolizí v plánovací matici je popsán v kapitole 5.2). Pro konečné přiřazení se vybere posloupnost k_{top} s kolizí minimální. Pokud je takových více, je zvolena první z nich z důvodu, aby se rozvrh plnil od začátku:

$$\begin{aligned} K &:= \arg\min_{k \in K} \Big(\sum_{zdroj \in k} zdroj_{kolize} \Big) \\ k_{top} &= \arg\min_{k \in K} \big(k_{0 časový\_slot} \big). \end{aligned} \tag{4.7}$$

4.3 Kvantifikace měkkých kritérií

K hodnocení kvality rozvrhů je potřeba, aby byla jednotlivá měkká kritéria vyjádřena takovým způsobem, který umožňuje jejich hodnoty porovnat. Potřebujeme tedy měkká kritéria vyjádřit skalárně nebo nad nimi definovat relaci lineárního uspořádání. Tato kapitola pojednává právě o zvoleném způsobu kvantifikace měkkých kritérií.

[1] Běžným požadavkem vyučujících na *FIT* je slovně vyjádřený požadavek na rozdělení množiny časových slotů na několik málo podmnožin, ve kterých jim více nebo méně vyhovuje výuka. Na základě těchto požadavků může osoba zodpovědná za tvorbu rozvrhů přiřadit podmnožinám vhodnou míru preference. Pokud by vyučující dostali explicitní možnost pro přiřazení preferencí každému časovému slotu, mohlo by z psychologického hlediska docházet k situacím, kdy by si volili pouze velmi omezené preferované podmnožiny. Ve výsledku by tak nemuselo být možné rozvrh naplánovat podle těchto preferencí, protože by si každý vyučující určil své časové sloty bez ohledu na potřeby ostatních.

Kvalitu sestaveného rozvrhu, který má přiřazeny všechny faktory zdrojům a neporušuje žádná tvrdá kritéria, je možné vyhodnotit na základě následujících měkkých kritérií:

- *Preference (F_r):* hlídá dodržení preferencí vyučujících na umístění faktorů do příhodných časových oken (požadována maximalizace).
- *Rozestupy termínů (F_s):* reflektuje vzdálenost jednotlivých termínů zkoušky, aby měli studenti dostatek času pro přípravu na opravný termín a vyučující čas na opravu písemných prací (požadována maximalizace).
- *Kolize (F_k):* sleduje počty studentů, jimž naplánované faktory jsou ve vzdálenosti dva a méně dnů (vhodná vlastnost pro přednášky, nevhodná pro zkoušky).

Předpokládejme, že existuje zaplněná *plánovací matice* PM, pro jejíž rozvrh chceme určit celkovou míru preference F_r. Jak bylo představeno v kapitole 4.2.3, obsahuje každá buňka matice PM informaci o hodnotě její *preference*. Tato hodnota je určena pro každou kombinaci faktoru a zdroje příslušným vyučujícím. Je tedy možné pro všechny zdroje z přidělené faktorům f z množiny všech faktorů F sečíst lokální míru preference buňky $PM(f, z)$ následovně:

$$F_r = \sum_{f \in F} \sum_{z \in f} PM(f, z)_{preference} \tag{4.8}$$

Rozestup termínů F_s odpovídá celkovému součtu denních vzdáleností jednotlivých termínů zkoušek vážených počtem studentů. Nechť je nad množinou faktorů definována silně antisymetrická a antitranzitivní relace $\prec$. Dva faktory jsou v této relaci právě a jen tehdy, když číslo termínu prvního je o jedna nižší než druhého (všechny první termíny jsou tedy v relaci s druhými atp.). Přesný výpočet *rozestupu termínů* je pak uveden v rovnici 4.9. Pro každý předmět p z množiny všech předmětů P a jeho naplánované faktory f_1 a f_2, které jsou v relaci $\prec$, sečti jejich hodnotu funkce $space(f_1, f_2)$. Ta je definována jako velikost rozestupu dnů, kdy jsou naplánovány, váženou počtem studentů druhého faktoru.

$$\begin{aligned} space(f_1, f_2) &: \left| f_{2_{day}} - f_{1_{day}} \right| \cdot f_{2_{students}} \\ F_s &= \sum_{p \in P} \sum_{f_1, f_2 \in p} \begin{cases} space(f_1, f_2) & f_1 \prec f_2 \\ 0 & f_1 \not\prec f_2 \end{cases} \end{aligned} \tag{4.9}$$

Míru *kolize* F_k lze určit pomocí tzv. *kolizní matice*, která obsahuje informace o počtech společných studentů mezi libovolnými dvěma faktory. Jednou z dostupných informací pro plánování rozvrhů na *FIT* jsou ale počty společných studentů mezi všemi dvojicemi předmětů (ne faktorů). V případě přednáškového rozvrhu je tato informace ekvivalentní – všichni studenti zapsaní do předmětu mají možnost navštěvovat jeho přednášky. U zkouškového rozvrhu na *FIT* je však počet studentů účastnících se termínu zkoušky obecně nižší než jejich počet zapsaný do předmětu. U řádných termínů zkoušek je toto dáno tím, že ne všichni studenti splní podmínky zápočtu. Účast na opravných termínech se navíc odvíjí od procentuální úspěšnosti u předchozích termínů. Pokud by byly pro všechny termíny jedné zkoušky použity stejné kolizní hodnoty, byly by kolize opravných termínů přeceňovány na úkor řádných termínů.

Pro rozvrh zkoušek však lze hodnoty kolizní matice vypočítat. U každého faktoru (termínu zkoušky) musí být totiž uvedena informace o horním odhadu počtu studentů, kteří se budou daného termínu účastnit, z důvodu, aby byla zajištěna dostatečná kapacita zkouškových místností. Hodnota kolizní matice KM pro faktory f_1, f_2 o horním odhadu počtu

studentů $|f_1|$, $|f_2|$, které náleží k předmětům p_1, p_2 o celkovém počtu studentů $|p_1|$, $|p_2|$ a společném počtu studentů $|p_1 \cap p_2|$ může být určena následovně:

$$KM(f_1, f_2) = KM(f_2, f_1) = \frac{|f_1|}{|p_1|} \cdot \frac{|f_2|}{|p_2|} \cdot |p_1 \cap p_2|. \tag{4.10}$$

Tento vzorec vychází z pravděpodobnostní podstaty problému a předpokladu, že úspěšnost studenta u dvou termínů různých zkoušek považujeme za nezávislé děje. Pokud tedy existují dva bezzápočtové předměty s počtem společných studentů x, jejichž opravných termínů se zúčastní právě 50 % studentů, je hodnota kolize mezi jejich opravnými termíny rovna $0,25 \cdot x$ (mezi řádným a opravným potom $0,5 \cdot x$). Hodnota vypočtená podle uvedeného vzorce odpovídá spodnímu odhadu důležitosti kolize, protože v praxi se dá spíše předpokládat, že mezi oběma ději určitá korelace existuje – méně nadaný student bude nejspíše opakovat více zkoušek než jeho nadanější spolužáci. Bylo by tedy vhodné vypočtené hodnoty zvýšit úměrně míře této korelace. Na druhou stranu vychází výpočet z hodnot horního odhadu počtu studentů, a tedy reálná důležitost kolize bude obecně nižší než ta vypočtená. Obě uvedené skutečnosti ovlivňují vypočtenou hodnotu v opačném směru, velikost jejich vlivu však nelze rozumně určit. Můžeme ale předpokládat, že se do jisté míry vzájemně anulují, vypočtenou hodnotu pak lze považovat za přijatelný odhad.

Prakticky je třeba rozlišovat mezi čtyřmi typy kolizí (seřazeno podle významnosti):

1. *Hodinové kolize (F_{kh})*: kolize v rámci jednoho časového slotu.
2. *Denní kolize (F_{kd})*: kolize v rámci jednoho dne (implicitně zahrnují *hodinové kolize*).
3. *Sousední kolize (F_{ks})*: kolize mezi konkrétním dnem a jeho včerejškem nebo zítřkem (nezahrnují denní kolize).
4. *Sousední kolize ob dva dny (F_{ks2})*: kolize mezi konkrétním dnem a jeho předvčerejškem nebo pozítřkem (nezahrnují denní ani sousední kolize)

Pokud tedy přiřadíme dva faktory f_1, f_2 pro něž platí $KM(f_1, f_2) > 0$ ke zdrojům, jejichž časové sloty se liší o dva a méně dnů, vznikne kolize alespoň jednoho z uvedených typů. V případě, že například jeden zdroj odpovídá libovolné hodině v úterý a druhý libovolné hodině ve středu, dojde k *sousední kolizi* o velikosti $KM(f_1, f_2)$. V jiném případě, kdy oba zdroje sdílí jeden časový slot, dojde k *hodinové* a zároveň *denní kolizi* (obě znovu o velikosti $KM(f_1, f_2)$). Určování obou dvou typů sousedních kolizí se řídí kalendářní vzdáleností dnů – mezi pátkem a následujícím pondělkem tedy nemůže vzniknout žádný typ kolize, přestože víkendové dny v *plánovací matici* vůbec neuvažujeme.

Formálně lze míru kolize F_k definovat jako čtveřici:

$$F_k = \langle F_{kh}, F_{kd}, F_{ks}, F_{ks2} \rangle. \tag{4.11}$$

Nechť je nad množinou faktorů definováno libovolné lineární uspořádání. To bude zajišťovat, že kolize mezi dvěma faktory se bude počítat pouze jednou. Exaktní výpočet *hodinové kolize F_{kh}* je pak následující. Pro každý časový slot cs a v něm naplánované faktory f_1 a f_2 je sečtena hodnota z kolizní matice KM v případě, že f_1 *předchází* f_2:

$$F_{kh} = \sum_{cs \in CS} \sum_{f_1, f_2 \in cs} \begin{cases} KM(f_1, f_2) & f_1 < f_2 \\ 0 & f_1 \geq f_2 \end{cases} \tag{4.12}$$

V případě *denních kolizí* F_{kd} dojde pouze k malé úpravě, při níž počítáme naplánované faktory f_1 a f_2 přes jednotlivé dny d z množiny všech plánovacích dnů D:

$$F_{kd} = \sum_{d \in D} \sum_{f_1, f_2 \in d} \begin{cases} KM(f_1, f_2) & f_1 < f_2 \\ 0 & f_1 \geq f_2 \end{cases} \tag{4.13}$$

Při počítání *sousedních kolizí* F_{ks} je nutné ke každému dni d ($d \in D$) definovat množinu $d^{\pm}$ jeho sousední dnů (tj. včerejší a zítřejší). Včerejší den ke dni d je takový den d^{-}, jehož datum je o den nižší a platí $d^{-} \in D$. Podobně zítřejší den je takový den d^{+}, jehož datum je o den vyšší a platí $d^{+} \in D$. Množina $d^{\pm}$ může být maximálně dvouprvková. V případech kdy d je okrajový den (např. první plánovací den nebo poslední den v týdnu), může být $d^{\pm}$ i jednoprvková či prázdná. Sousední kolize je nyní možné definovat jako:

$$F_{ks} = \sum_{d \in D} \sum_{d' \in d^{\pm}} \sum_{f_1 \in d} \sum_{f_2 \in d'} \begin{cases} KM(f_1, f_2) & f_1 < f_2 \\ 0 & f_1 \geq f_2 \end{cases} \tag{4.14}$$

Definujme analogicky jako množinu $d^{\pm}$ množinu sousedních dní $d^{\pm 2}$ obsahující předvčerejší a pozítřejší den s tím rozdílem, že se datum snižuje/zvyšuje o dva dny. *Sousední kolize od dva dny* F_{ks2} poté můžeme určit velmi podobně jako:

$$F_{ks2} = \sum_{d \in D} \sum_{d' \in d^{\pm 2}} \sum_{f_1 \in d} \sum_{f_2 \in d'} \begin{cases} KM(f_1, f_2) & f_1 < f_2 \\ 0 & f_1 \geq f_2 \end{cases} \tag{4.15}$$

Protože je míra kolize F_k složena z více podtypů kolizí, je nad ní nutné definovat relaci lineárního uspořádání. Prvotní návrh počítal s porovnáním definovaným hierarchicky podle významnosti typu kolize – nejdříve se porovnají nejvýznamnější *hodinové kolize*, pokud jsou shodné, porovnají se *denní kolize* atd. Tento přístup se osvědčil pro rozvrhy přednášek, u nichž se snažíme minimalizovat hodinové kolize a maximalizovat ostatní typy kolizí.

Po provedení experimentů s rozvrhy zkoušek byl však vyhodnocen jako nevyhovující, protože přikládal významnosti typů nepřiměřeně velkou váhu. Jako vhodné řešení se ukázala skalarizace tohoto strukturovaného typu a následné porovnání získaných skalárů. Jednoduchá skalarizace je možná díky tomu, že v případě rozvrhů zkoušek je potřeba všechny typy kolizí minimalizovat. Kolizní vektor $\mathbf{F_k} = \langle F_{kh}, F_{kd}, F_{ks}, F_{ks2} \rangle$ se vynásobí skalarizačním vektorem $\mathbf{s} = \langle s_h, s_d, s_s, 1 \rangle$ následovně:

$$|\mathbf{F_k}| = \mathbf{F_k} \cdot \mathbf{s}^T \tag{4.16}$$

Volba hodnot koeficientů s_h, s_d a s_s vhodných pro rozvrhování na *FIT* je diskutována v kapitole 6.1. Z důvodu dodržení významnosti kolizí dávají hodnoty koeficientů smysl pouze, pokud je splněna následující nerovnost:

$$s_h \geq s_d \geq s_s \geq 1 \tag{4.17}$$

Kapitola 5

Implementace v jazyce Python 3

Podle prezentovaného návrhu byla implementována objektově orientovaná konzolová aplikace s názvem *AEm Timetables*. Nad ní bylo postaveno odpovídající grafické uživatelské rozhraní. Jako implementační jazyk byl, především z důvodu možnosti rychlého prototypování pro ověření navrženého konceptu, zvolen Python 3. Díky němu je mimo jiné zajištěna přenositelnost mezi operačními systémy typu Unix nebo Windows. Implementace je založena na imperativním paradigmatu doplněném funkcionálními prvky.

Aplikace je vytvořena jako pythonský balíček (angl. package), který se skládá z následující struktury modulů:

```
/AEm/
    engine/
        algorithms/
        components/
        misc/
        thanh/
        tools/.
```

Implementace aplikace ve formě balíčku umožňuje využívat relativní cesty při importu jeho podmodulů. To je výhodné především proto, že vedle výpočetní části aplikace, jsou implementovány ještě podpůrné nástroje pro statistické vyhodnocení výsledků nebo přípravu konfiguračních souborů. Implementace těchto nástrojů je výrazně jednodušší v případě, kdy mohou být využity objekty z výpočetní části. Z pohledu návrhu je naopak žádoucí, aby se tyto nástroje nacházely v odděleném modulu (zde `AEm.engine.tools`). Obojího lze dosáhnout právě díky relativním importům.

Modul `algorithms` definuje obecné algoritmy, u nichž je vysoká pravděpodobnost znovupoužití (rekombinační operátory). Modul `components` obsahuje všechny komponenty, které modelují fyzický stav reality (zdroje, faktory, kolizní matici, místnosti, vyučující a další). V modulu `misc` jsou definovány použité konstanty a obálka nad modulem random, která umožňuje vnutit vlastní „náhodná“ čísla pro potřeby ladění. V modulu `thanh` se nachází výpočetní část algoritmu (genetický algoritmus včetně NSGA-II, plánovací matice a další).

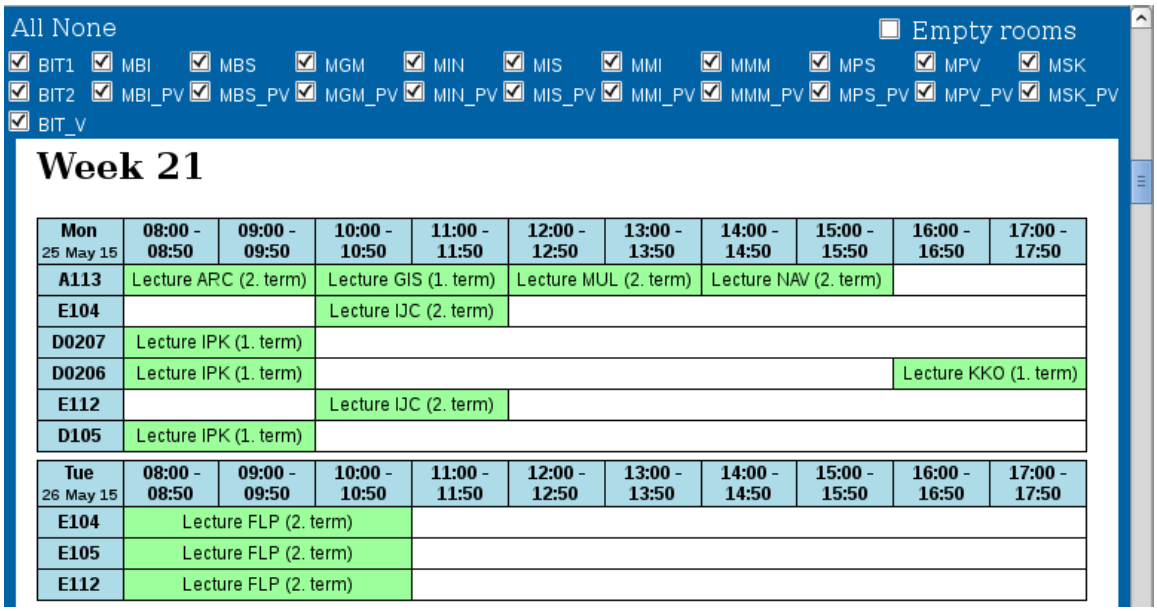
Před započetím výpočtu je nutné dodat programu dva konfigurační XML soubory. První z nich popisuje řešený *problém* (viz příloha C), druhý soubor nastavuje parametry výpočtu (viz příloha B). Výpočet lze poté spustit následujícím příkazem:

```
$ python3 -m AEm.engine problem.xml setting.xml.
```

Po spuštění je v aktuálním umístění vytvořena složka `outputs` (pokud neexistuje) a v ní podsložka s názvem složeným z data a času odpovídajících době spuštění. K jejímu názvu je možné přidat uživatelský textový řetězec pomocí parametru `--note`. Tato podsložka odpovídá jednomu běhu algoritmu – uvnitř se nachází následující program generované informace:

```
progress/
rooms/
stats/
log
problem.xml
settings.xml
```

Složka `progress` obsahuje XML soubory popisující každou generaci jedinců, která byla doposud vypočítána. Soubory obsahují serializované jedince ve formě jejich chromozomu (vektor identifikačních čísel faktorů) a příslušné hodnotě fitness. Informace z této složky jsou využity v případě potřeby restartovat výpočet z posledního dosaženého stavu pomocí parametru `--resume` (vhodné např. při neočekávaného výpadku serveru).



Obrázek 5.1: Ukázka výstupního rozvrhu ve formátu HTML. Faktory (zeleně), které jsou plánovány do více místností paralelně za pomoci *virtuálních zdrojů*, jsou uváděny opakovaně pro každou takovou místnost.

Do složky `rooms` je po dokončení výpočtu vygenerována množina nejlepších nalezených rozvrhů (Pareto fronta získaná algoritmem *NSGA-II*). Ty mají formu pro člověka čitelných HTML souborů (viz obrázek 5.1). Rozvrhy jsou v nich řazeny podle kalendářních týdnů a jejich dnů. Každý den obsahuje množinu učeben, ve kterých jsou zobrazeny jim plánované faktory. Pomocí jazyku JavaScript je v horní části obrazovky implementován filtr, pomocí kterého je možné zobrazit předměty pouze pro vybrané studijní obory a ročníky. Obdobným způsobem je možné zapnout zobrazení učeben, ve kterých není pro daný den plánována výuka (pravý horní roh). Sémanticky ekvivalentní výstup je generován navíc

ve formě CSV souboru, který umí zpracovat fakultou používaný program pro manuální plánování rozvrhů[1]. Díky ní je tak možné provádět případné uživatelské změny.

Složka `stats` obsahuje statistické informace o průběhu výpočtu, které mohou být vyčísleny z dat ve složce `progress` a obsahu logovacího souboru `log`. Její obsah je pro složku `složka_s_výpočtem` možné vytvořit spuštěním příkazu:

```
$ python3 -m AEm.engine.tools.progress_graph složka_s_výpočtem.
```

Soubor `log` obsahuje běžné logovací informace vhodné pro ladění aplikace. Soubory `problem.xml` a `settings.xml` jsou kopiemi vstupních konfiguračních souborů (vždy jsou přejmenovány na uvedené názvy). Uživatel do nich může nahlédnout, když potřebuje zjistit přesné parametry výpočtu. Aplikace je jimi inicializována v případě použití parametru `--resume`.

5.1 Optimalizace výpočtu

Užití vysokoúrovňového interpretovaného jazyka, jakým je Python 3, se může pro výpočetně náročný problém jevit jako ne příliš dobrá varianta. Volba jazyka probíhala na základě dvou kritérií:

- rychlý vývoj a snadná správa kódu,
- efektivita výpočtu.

První kritérium je základem filosofie Pythonu. Díky tomu byl mimo jiné již v listopadu 2014 vytvořen prototyp, na kterém byl ověřen potenciál prezentovaného návrhu. Tento prototyp byl následně transformován do podoby výsledné aplikace. Předtím však byly pro jeho moduly implementovány unit testy, které je možné spustit příkazem:

```
AEm/engine$ python3 unit_test.py.
```

V případě druhého kritéria a výkonnostního porovnání s jazyky C nebo C++, které bývají klasickou volbou při implementaci genetických algoritmů, je Python zcela jistě pomalejší. Je pro něj však možné vybrané moduly implementovat právě v jazycích C/C++[2], pokud by byla jejich rychlost v jazyce Python nedostatečná. Tímto způsobem lze výpočetní náročnost přiblížit na úroveň nativního programu v C/C++. Jako vhodný kandidát pro tuto optimalizaci se jeví třída pro obsluhu plánovací matice (modul `thanh.target_matrix`), která tvoří jádro aplikace a spotřebovává asi 90 % výpočetních prostředků. Uvedený postup však nebyl při implementaci nakonec uplatněn, protože zrychlení dosažené pomocí dále popsaných optimalizací se ukázalo být dostačujícím.

Jak již bylo řečeno, je nejvíce výpočetně náročnou částí algoritmu sestavení rozvrhů za pomoci *plánovací matice*. Vyhodnocení této části heuristického algoritmu je pro všechny genotypy jedinců v rámci jedné generace nezávislé. Nabízí se tak možnost tento výpočet paralelizovat. K tomu byla použita třída `Pool` z knihovny `multiprocessing`, která umožňuje paralelní výpočet funkce vyššího řádu `map`. Uživatel může při spuštění výpočtu zvolit libovolný počet procesů, do kterých je tato úloha rozdělena, případně může paralelizaci úplně

[1] Program pro manuální tvorbu rozvrhů vytvořit Ing. Jaroslav Dytrych jako náhradu v minulosti používané „papírkové metody“. Jedná se o množinu skriptů hodnotících stav přepisovatelné HTML tabulky.

[2] Způsob implementace C/C++ modulu pro Python 3 je popsána na https://docs.python.org/3/extending/extending.html.

vypnout. S počtem běžících podprocesů rostou lineárně i paměťové nároky aplikace – pro testované úlohy paměťové požadavky nepřekročily 200MB na podproces. Nejedná se tedy o velké omezení, protože na výpočetních serverech, kde je možnost uplatnění paralelizace nejvyšší, se běžně operační paměť pohybuje řádově v jednotkách GB na výpočetní jádro.

Jinou jednoduchou možností, jak zkrátit dobu potřebnou pro výpočet, je využití rychlejšího interpretu jazyka Python jménem `PyPy`[3] (namísto klasického `CPythonu`). Ten využívá tzv. *Just-in-Time* kompilátor, který za běhu programu provádí překlad, analýzu a optimalizaci programu. Díky tomu lze celý výpočet urychlit přibližně 2,5krát bez ohledu na stupeň paralelizace.

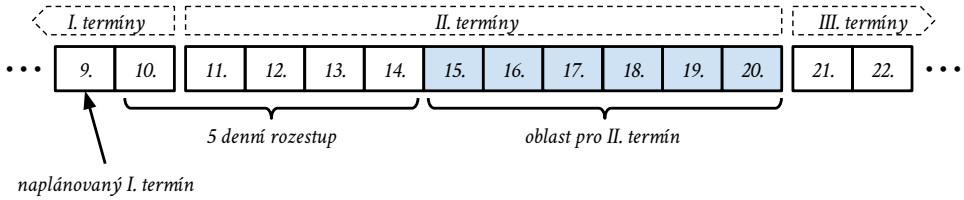
Další provedené optimalizace se již týkají způsobu práce s *plánovací maticí*. Jak již bylo zobrazeno na obrázku 4.4, není třída `thanh.target_matrix` instanciována před každým sestavením rozvrhu, ale je pokaždé pouze vynulována. Předpočítávané informace (virtuální zdroje, apriorní omezení, ...) je tak možné vyčíslit pouze jednou, protože mohou být sdíleny při jednotlivých sestaveních. Aby mohla být *plánovací matice* efektivně vynulována, bylo potřeba rozdílně reprezentovat hodnoty matice vzniklé v důsledku její přípravy a hodnoty vložené vlastním plánováním. Průběžné resetování *plánovací matice* urychlilo celý výpočet přibližně dvakrát.

V okamžicích, kdy *plánovací matice* hledá vhodné zdroje pro přidělení k faktoru (viz kapitola 4.2.3), není vždy nutné vyčíslovat všechny možné posloupnosti zdrojů. Algoritmus totiž nemusí udržovat v paměti množinu všech kandidátních posloupností, stačí když ji lineárně projde a nalezne její nejvhodnější prvek – tedy posloupnost s nejvyšší sumou preferencí, která má nejméně kolizí. Maximální hodnotu preference lze pro každý sloupec předpočítat. Pokud je tedy nalezena posloupnost s touto maximální hodnotou preference, která nezpůsobuje žádnou kolizi, je možné vyhledávání ukončit. Nemůže totiž existovat jiná posloupnost s vyšší hodnotou preference, nebo nižší hodnotou kolize. Tato optimalizace se uplatní především na počátku sestavování rozvrhu, protože v úvodu vzniká jen minimum kolizí (pokud nebudeme uvažovat fixní termíny, nemůže dokonce při umisťování prvního faktoru žádný typ kolize vůbec vzniknout). Díky předčasnému ukončení vyhledávání posloupnosti zdrojů lze algoritmus přibližně 1,5krát urychlit.

V případě zkouškové rozvrhu také není nutné prohledávat celý sloupec *plánovací matice*, ale stačí prozkoumat pouze jeho určitou oblast. U jednotlivých termínů zkoušky je totiž potřeba dodržet jejich následnost. Nedává tedy například smysl umístit třetí termín zkoušky do prvního týdne zkouškového období. Uživatel může sám zvolit časové rozmezí, které se může libovolně překrývat, pro jednotlivé termíny. Tím tak vlastně určí rozsah prohledávaných oblastí.

Prohledávanou oblast lze navíc ještě zmenšit díky tomu, že mezi termíny jedné zkoušky je potřeba dodržet jisté minimální rozestupy. Implicitní hodnota pro toto tvrdé kritérium vychází z *Rozhodnutí děkana FIT č. 54/2012, dodatek k článku 12, bod 5*, které říká, že *"výsledek termínu zkoušky musí být dostupný nejméně dva dny před následujícím termínem zkoušky"*. Aby bylo možné stihnout i reklamace písemných prací před touto lhůtou, je počet dnů zvýšen ještě o jeden – implicitně se tedy počítá se třemi dny mezi termíny zkoušky každého předmětu. Tuto hodnotu pak může uživatel pro vybrané termíny zkoušek navýšit. Pro testovací účely byla proděkanem pro vzdělávací činnost v bakalářském studiu Ing. Bohuslavem Křenou, Ph.D. doporučena hraniční hodnota 80 studentů na jeden den opravování. V případě prvního termínu o 130 studentech je tedy minimální rozestup určen jako $2 + 1 + 130/80 = 5$. Druhý termín zkoušky může v tomto případě být od prvního nejblíže ve vzdálenosti 5 dnů. Příklad takovéto situace je zobrazen na obrázku 5.2. Vyhle-

[3] Informace o interpretu `PyPy` lze nalézt na http://pypy.org/



Obrázek 5.2: Příklad prohledávané podmnožiny dnů (modře) při plánování termínu zkoušky. Druhý termín může být plánován ode dne 15 po den 20. Oblast je zleva omezena z důvodu dodržení 5denního rozestupu od předchozího termínu téže zkoušky a zprava je omezena posledním možným dnem pro druhé termíny.

dáváním zdrojů jen v omezené oblasti *plánovací matice* je celková doba výpočtu urychlena přibližně 3krát.

Výpočet je také možné urychlit deaktivováním příkazů `assert`, které na vhodných místech programového kódu hlídají správný tok výpočtu. Je tak možné urychlit výpočet přibližně o 15 %. Tento kontrolní mechanismus lze vypnout pomocí parametru `-OO`. Příkaz pro spuštění výpočtu pak pro interpret `pypy3` vypadá následovně:

```
$ pypy3 -OO -m AEm.engine.tools.progress_graph složka_s_výpočtem.
```

Díky uvedeným optimalizacím bylo dosaženo celkového přibližně 25násobného zrychlení bez ohledu na stupeň paralelizace. Takovéto zrychlení je dostatečné k tomu, aby byl i na běžném počítači získán rozvrh do fakultou požadovaných 24 hodin s dostatečnou časovou rezervou – v případě paralelizovaného běhu již za dobu desítek minut.

Hodnoty zrychlení uváděné v této kapitole byly měřeny pro výpočet zkouškového rozvrhu zimního semestru 2014/2015 na *FIT*. Nastavení bylo zvoleno tak, aby napodobovalo běžné použití aplikace bez využití paralelizace. Konkrétně byla použita populace o 100 jedincích při 30 generacích, operátor křížení byl zvolen typu *PBX* s 10 % pravděpodobností mutace.

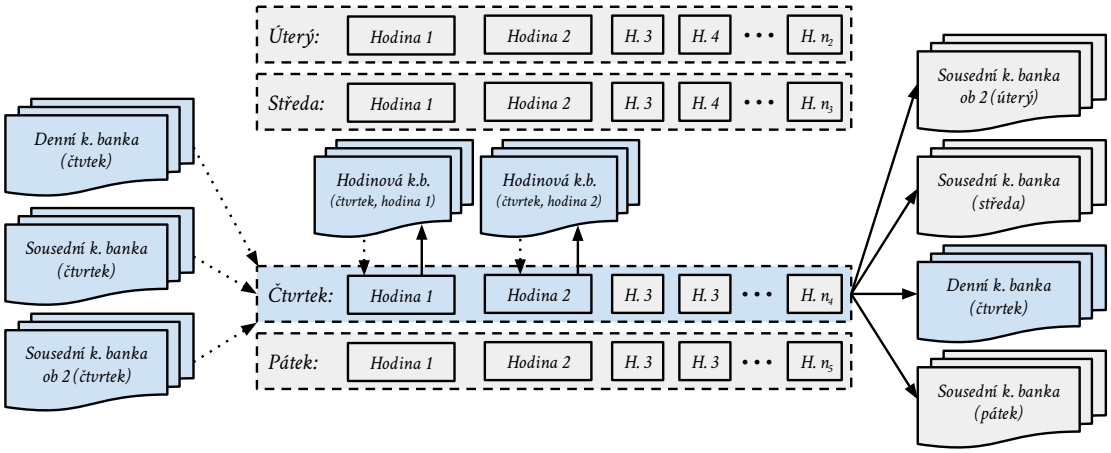
5.2 Sledování průběžného stavu kolizí v plánovací matici

V kapitole 4.2.3 byl představen postup, podle kterého jsou přiřazovány *posloupnosti zdrojů* k *faktorům* na základě měkkých kritérií. Podle něj se řídí *plánovací matice* při sestavování rozvrhu v případě, že existuje více kandidátních posloupností zdrojů, které neporušují tvrdá kritéria. Prezentovaný postup mimo jiné vyžaduje efektivní způsob výpočtu všech typů kolizí, které by způsobilo potenciální přiřazení jedné z kandidátních posloupností k faktoru. V této kapitole je popsána implementace takového výpočtu.

Prakticky se kolize nevypočítává mezi faktorem a zdrojem, ale mezi faktorem a časovým slotem, který je zdroji přidělen. Tuto hodnotu nelze nijak předpočítat, protože se může měnit v závislosti na již přidělených kombinacích faktorů a zdrojů. Dá se však předpokládat, že počet požadavků na určení kolize bude násobně větší než počet změn rozložení kolizí v plánovací matici. To je dáno faktem, že pro jedno naplánování faktoru (a tím vyvolanou parciální změnu kolizí) je potřeba vyčíslit kolize všech kandidátních posloupností. Jeví se tedy vhodné navrhnout strukturu pro průběžné sledování stavu kolizí v plánovací matici, která nabídne jednoduchý způsob, jak potenciální kolize vyčíslit.

Takovouto strukturu s názvem *kolizní banka*[4] lze implementovat ve formě vektoru ce-

[4]Implementaci *kolizní banky* lze nalézt v modulu `AEm.engine.components.collision_bank`.



Obrázek 5.3: Užití *kolizních bank* v případě přiřazení zdrojů s časovými sloty *čtvrtek hodina 1* a *čtvrtek hodina 2*. Modře jsou zvýrazněny kolizní banky odpovídající těmto zdrojům. Tečkovanými šipkami jsou znázorněny kolizní banky mající vliv na jejich výběr. Plnými šipkami jsou označeny banky, do kterých budou po přiřazení zaneseny změny. Kolizní banka *denní kolize (čtvrtek)* je z důvodu zvýšení přehlednosti zobrazena dvakrát, jedná se však o tentýž objekt.

lých čísel o délce odpovídající celkovému počtu faktorů. Každá jeho položka je přidělena právě jednomu faktoru a její hodnota značí míru kolize, kterou by přiřazení tohoto faktoru způsobilo. Výchozí hodnota všech položek vektoru je tedy nula. Každému časovému slotu je přiřazena právě jedna kolizní banka značící jeho potenciální *hodinové kolize*. Obdobně jsou každému dni přiřazeny kolizní banky pro *denní*, *sousední* a *sousední kolize ob dva dny*. Protože se časové sloty skládají z dnů, jsou fakticky každému časovému slotu přiřazeny právě čtyři kolizní banky – jedna pro každý typ kolize. Na obrázku 5.3 je modře zvýrazněno pět kolizních bank odpovídajících dvěma časovým slotům *čtvrtek hodina 1* a *čtvrtek hodina 2* – tři denní kolizní banky jsou pro oba tyto sloty společné.

Existuje velmi silný vztah mezi *kolizní bankou* a *kolizní maticí* (viz kapitola 4.3). Hodnoty vektoru kolizních bank totiž nabývají pouze hodnot odpovídajících součtu některých řádků kolizní matice. Obsah kolizních bank se mění následovně. Ve chvíli, kdy je v plánovací matici přiřazen faktor k posloupnosti zdrojů, je zjištěna množina kolizních bank, které byly ovlivněny (jejichž obsah je nutné přepočítat). Jedná se o:

- *hodinové kolizní banky*[5] přiřazené časovým slotům těchto zdrojů,
- *denní kolizní banka* přiřazená odpovídajícímu dni této posloupnosti zdrojů,
- *sousední kolizní banky* včerejšího a zítřejšího dne,
- *sousední kolizní banky ob dva dny* předvčerejšího a pozítřejšího dne.

K hodnotám těchto kolizních bank je vektorově přičten řádek kolizní matice, který odpovídá právě přiřazenému faktoru. Položky vektorů kolizních bank tedy udržují informaci o budoucí kolizi se všemi ostatními faktory.

[5] Ve speciálních případech, kdy se přiřazuje faktor ke zdroji s učebnou, která se nenachází v prostorách *FIT* (tzv. *vzdálená učebna*), jsou ovlivněny i *hodinové kolizní banky* předchozího a následujícího zdroje v daném dni (pokud existují). Je tak zajištěna hodinová pauza pro přesun na případnou předcházející/následující výuku.

Pokud je v dalších fázích sestavování rozvrhu potřeba určit potenciální míru kolize, stačí k tomu pouze čtyři kolizní banky. Každá z nich odpovídá právě jednomu typu kolize. Z nich jsou jednoduše přečteny hodnoty vektoru na pozicích příslušících přiřazovanému faktoru.

Pro větší názornost popisu kolizních bank je dále uveden příklad přiřazení faktoru k posloupnosti dvou zdrojů, který je zobrazen na obrázku 5.3. Nechť je zde plánován faktor s pořadovým číslem x do prvních dvou čtvrtečních časových slotů. Hodinová kolize, která tímto vznikne, je zjištěna jako součet x-tých složek vektorů obou příslušných hodinových kolizních bank. Denní, sousední a sousední kolize ob dva dny je analogicky zjištěna ze čtvrtečních kolizních bank (tečkované šipky). Na základě těchto hodnot je rozhodnuto o naplánování faktoru k těmto zdrojům. Následně je nutné přičíst k oběma hodinovým kolizním bankám x-tý řádek kolizní matice. Obdobně je také tuto změnu nutné zanést do čtvrteční *denní kolizní banky*, *sousedních bank* středy a pátku a *sousední banky ob dva dny* příslušící úterku (plné šipky).

V okamžiku, kdy je určena míra potenciální kolize a je rozhodnuto o přiřazení, je navíc možné takto získanou hodnotu postupně sumovat. Po dokončení plánování pak tato suma odpovídá celkové míře kolize rozvrhu definovanou pomocí vzorců 4.12, 4.13, 4.14 a 4.15. Ty tak není nutné vyčíslovat dodatečně.

5.3 Grafické uživatelské rozhraní

K aplikaci bylo vytvořeno grafické uživatelské rozhraní (dále GUI), které lze spustit ve webovém prohlížeči. K jeho implementaci byl využit framework *web.py*[6]. Jedná se o webový framework pro jazyk Python 2 obsahující vlastní webový server[7]. Aplikaci s GUI lze spustit pomocí následujícího příkazu:

```
$ python -m AEm.
```

Do příkazového řádku je následně vypsána localhostová adresa s portem, na kterém byl webový server spuštěn. Po zadání této adresy do webového prohlížeče se zobrazí úvodní okno aplikace (viz obrázek 5.4).

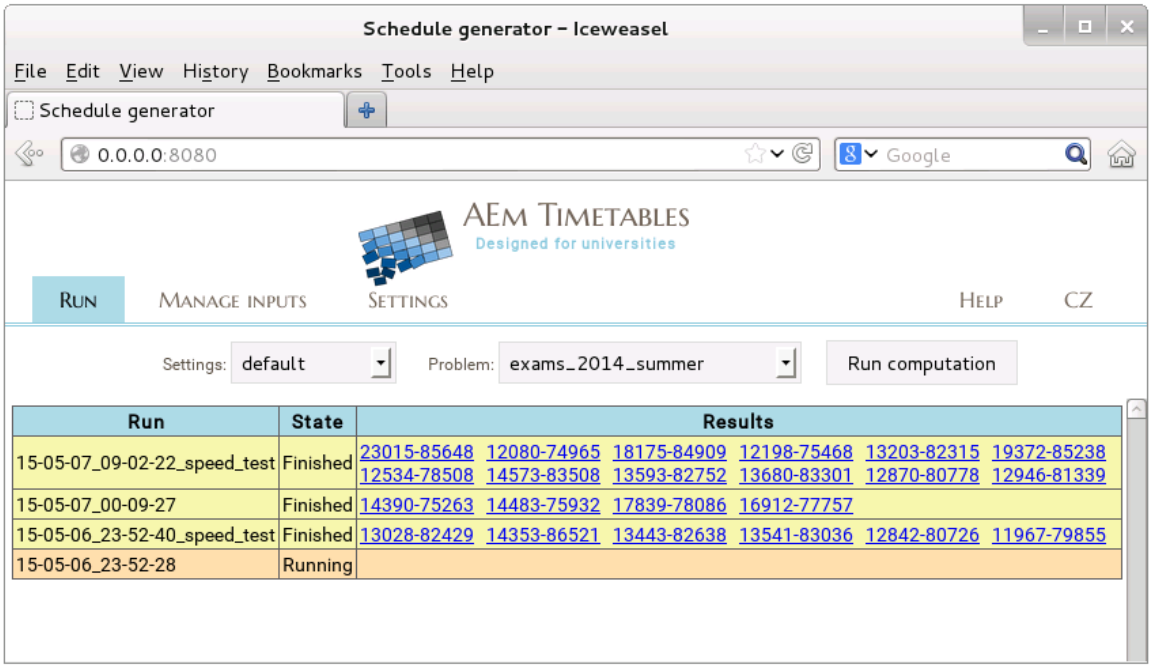
Uživatelské rozhraní je dvojjazyčné, lze volit mezi češtinou a angličtinou. Na úvodní stránce s názvem „run“ má uživatel možnost spouštět a spravovat jednotlivé výpočetní běhy. Nad tabulkou dokončených výpočtů lze ve formuláři vybrat XML soubory s nastavením a definicí problému, se kterými bude výpočet spuštěn. Nedokončený výpočet je v tabulce zvýrazněn oranžově a označen stavem „running“. Po jeho dokončení přejde do stavu „finished“ a v pravé části tabulky se u něj objeví odkazy na nejlepší nalezené rozvrhy. Jejich jména jsou odvozena od příslušných hodnot měkkých kritérií – nejdříve je uvedena skalarizovaná míra kolizí, poté je uvedena míra rozestupu termínů.

Stránky „manage inputs“ a „settings“ slouží ke správě vstupních XML souborů (viz obrázek 5.5). Soubory popisující řešený problém není možné upravovat, protože jsou velmi rozsáhlé (tisíce řádků). Předpokladem je, že tyto soubory budou generovány externími skripty. Po kliknutí na záložku „manage inputs“ se tedy pouze zobrazí jejich obsah.

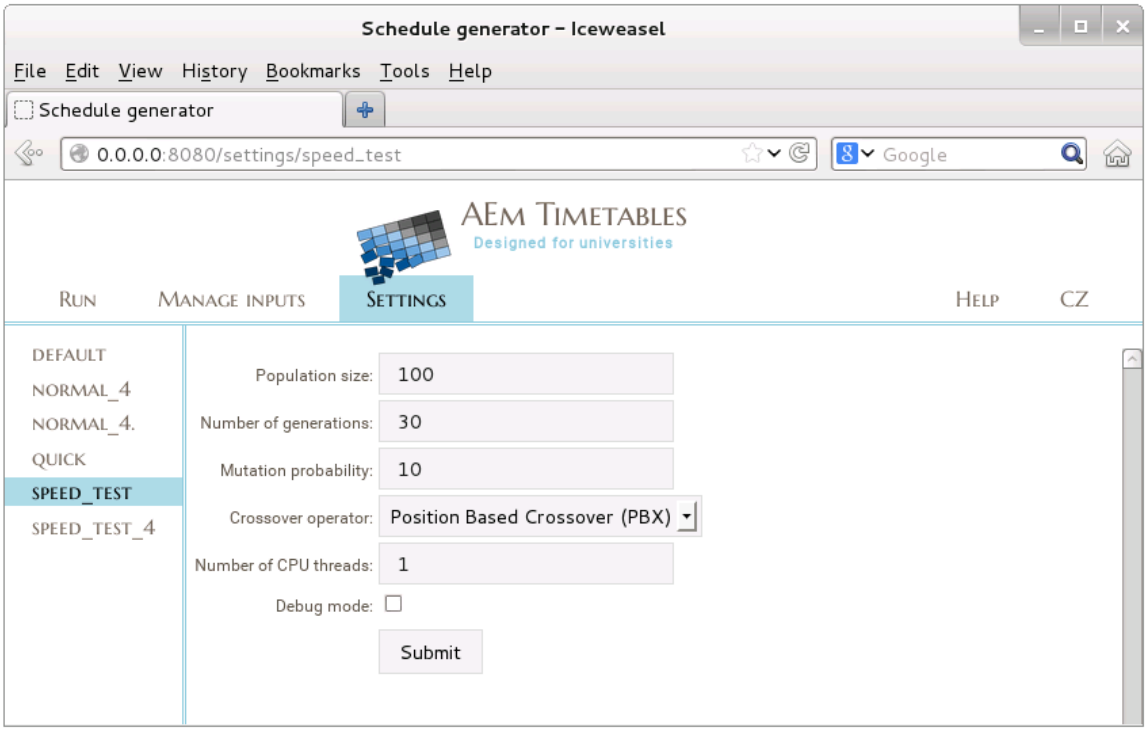
Webový server spouští výpočty na pozadí pomocí interpretu `pypy3`, je tedy potřeba nastavit jeho umístění do proměnné `$PATH`. Grafické rozhraní nenabízí všechny možnosti, které lze vykonat pomocí příkazové řádky (obnovení výpočtu, statistické výpočty a další) a neumí zobrazit případná chybová hlášení. Jedná se tedy spíše o jednoduchý doplněk použitelný pro méně zkušené uživatele.

[6] Informace ohledně frameworku web.py lze nalézt na http://webpy.org/.

[7] Framework web.py je určen pro jazyk Python 2, na rozdíl od výpočetní části aplikace vytvořené v jazyce Python 3. Pro úspěšné spuštění je tedy nutné použít správný interpret.



Obrázek 5.4: Výchozí okno GUI pro spouštění a správu výpočtů.



Obrázek 5.5: Okno GUI pro správu souborů s nastavením.

Kapitola 6

Experimentální výsledky pro zkouškové rozvrhy FIT 2014/2015

Experimenty byly provedeny na základě reálných podkladů pro tvorbu zkoušek na *FIT* pro akademický rok 2014/2015. Na zkouškové rozvrhy byl při konzultacích s osobami zodpovědnými za tvorbu rozvrhů na *FIT* kladen hlavní důraz, protože se na ně každoročně objevují nespokojené ohlasy ze strany studentů. Z toho důvodu jsou tyto požadavky také zveřejňovány spolu s hotovými rozvrhy. Uživatelům rozvrhu se bez znalosti konkrétních požadavků totiž mohou zdát některé volby nelogické. Jejich neveřejnou součástí jsou také odhady počtu studentů pro opravné termíny zkoušek na základě úspěšnosti v minulém roce. Veřejné podklady, které lze spolu s neveřejnými nalézt na přiloženém CD ve složce `/official_schedules/`, obsahují:

- textově definované požadavky vyučujících *(časová omezení, počet míst v učebnách*[1]*, požadované minimální rozestupy mezi termíny zkoušky a podobně)*,
- pevně zadané termíny pro zkoušky na ostatních fakultách,
- časová omezení učeben,
- rozsah přípustných dnů a hodin,
- počet a doba trvání jednotlivých termínů zkoušek,
- počty společných studentů pro všechny dvojice předmětů,
- dostupné místnosti a jejich kapacity.

Na základě těchto požadavků byly vytvořeny dvě experimentální úlohy (jedna pro každé zkouškové období), které lze na CD nalézt ve složce `/src/inputs/problems/` pod názvy `exams_2015_winter.xml` a `exams_2015_summer.xml`. Jedná se o textové soubory o velikosti přes 100kB. Seznam přednášek a jejich kolizní matice (zabírající přes 50 % velikosti souboru) byla získána pomocí nástroje `AEm.engine.tools.problem_html2xml`, který umí tyto informace extrahovat ze souborů s požadavky[2].

[1] Vyučující musí uvést požadovanou velikost učeben v případě, že požaduje pro zkoušku rozesazení s volnými místy mezi studenty (tj. ob jednoho případně ob dva studenty).

[2] Oba soubory s podklady mají rozdílnou hierarchii HTML značek, při jejich analýze je tedy nutné mírně pozměnit zdrojový kód modulu `problem_html2xml` (příslušná místa jsou označena poznámkami).

Ostatní informace byly manuálně transformovány pomocí textového editoru Sublime Text 2, který umožňuje velmi efektivní způsob selekce a modifikace textu obdobný skriptům s regulárními výrazy. Nejprve byly vytvořeny dny a jim odpovídající časové sloty. Oblasti časových slotů přípustné pro jednotlivé zkouškové termíny (viz kapitola 5.1) byly zvoleny na základě rozsahů překrývání termínů v minulých letech.

Poté byly definovány dostupné učebny s *virtuálními zdroji* seskupenými po komplexech E a D (viz kapitola 4.2). Byly také vytvořeny dvě speciální učebny s nulovou kapacitou[3]. První pro fixně naplánované zkoušky na *FEKT* (tzv. vzdálená učebna, viz kapitola 5.2) a druhá pro ostatní studentskou činnost (např. obhajoby semestrálních projektů) na *FIT*. Naplánováním faktorů do těchto učeben se upraví odpovídajícím způsobem kolizní banky plánovací matice.

Následně byl vytvořen seznam vyučujících jednotlivých předmětů. Tato informace není obsažena v podkladech pro tvorbu zkoušek a bylo nutné ji získat z webového informačního systému fakulty. Správným přiřazením vyučujících k faktorům je zajištěno, že se žádnému z nich nebudou časově překrývat termíny zkoušek různých předmětů.

Dále byl definován seznam s jednotlivými faktory pro naplánování. K většině předmětů jsou přiřazeny tři faktory – jeden pro řádný a dva pro opravné termíny zkoušek. V některých předmětech (např. předmět ISA), ve kterých je termínů více a studenti se na jejich době konání sami domlouvají s vyučujícím, byl naplánován pouze první termín, aby pro něj mohla být zajištěna dostatečně velká učebna ve vhodný den. Množství studentů na jednotlivých termínech vycházelo z odhadů neúspěšnosti z předchozích let. Tyto hodnoty byly zvětšeny až dvojnásobně[4] proto, aby mohla být pro každý termín zajištěna postačující kapacita učeben. Termíny zkoušek s pevně stanoveným datem byly fixně naplánovány do učeben s požadovanou kapacitou. Na základě odhadovaného množství studentů byly u termínů také stanoveny minimální rozestupy mezi jednotlivými termíny téže zkoušky (viz kapitola 5.1, ve které je diskutována hodnota 80 studentů pro přidání jednoho dne na opravu písemných prací).

Jako poslední byla k přednášejícím a učebnám dodána požadovaná časová omezení. K některým faktorům byly (podle požadavků vyučujících) navíc přidány množiny preferovaných časových slotů. Maximální hodnota preferencí, které lze dosáhnout pro úlohu `exams_2015_summer.xml` je 850. V případě úlohy `exams_2015_winter.xml` se jedná o hodnotu 750.

Vstupem všech experimentů je XML soubor s takto sestaveným zadáním a druhý XML soubor s nastavením výpočtu. Následující podkapitoly popisují provedené experimenty, podle jejichž výsledků bylo určeno doporučené nastavení výpočtů. Na závěr je porovnána kvalita takto získaných rozvrhů s reálně použitými rozvrhy na *FIT*.

6.1 Experimentální nastavení parametrů evoluce

Před spuštěním jakýchkoli výpočtů je potřeba zvolit koeficienty, pomocí kterých bude u jedinců vyčíslována parciální hodnota fitness postihující studentské kolize (viz konec kapitoly 4.3). Touto volbou dává uživatel najevo své představy o budoucím rozvrhu – usměrňuje selekční tlak evoluce na požadované vlastnosti. Konkrétně se jedná o hodnoty skalarizačního

[3] Učebna s nulovou kapacitou nemůže být alokována heuristickým algoritmem, ale pouze pomocí fixního přiřazení uživatelem.

[4] U nejkritičtějších předmětů (např. předmět IOS, nebo některé matematické předměty) se například na opravných termínech zkoušek počítá se stejnou účastí studentů jako na řádných.

vektoru $\mathbf{s} = \langle s_h, s_d, s_s, 1 \rangle$, kde s_h odpovídá *váze hodinové kolize*, s_d odpovídá *váze denní kolize* a s_s odpovídá *váze sousední kolize*.

U zkouškových rozvrhů se chceme hodinovým kolizím úplně vyhnout, protože odebírají možnost účasti studentů na některém z termínů zkoušek. V případě *FIT* se dokonce jedná o tvrdé kritérium, protože jsou zde kolize mezi zkouškami z povinných předmětů úplně vyloučeny (viz kapitole 2.3.1). Koeficient s_h tedy stačí zvolit dostatečně vysoký tak, aby rozvrh mající nenulové hodinové kolize byl významně horší než rozvrhy, které je mají nulové. Při experimentech se ukázalo, že hodnota $s_h = 100$ je nedostatečná, protože některé z výsledných rozvrhů obsahovaly přibližně do pěti hodinových kolizí u obou experimentálních úloh. Zvolili jsme proto $s_h = 500$, se kterým se tento problém již nevyskytoval. Koeficient by mohl být jistě i vyšší, avšak níže zvolená hodnota může teoreticky zvýšit rozmanitost populace. Při nahlédnutí do průběhu většiny výpočtů se skutečně v generacích vyskytují jedinci s jednotkami hodinových kolizí, kteří však nejsou dostatečně kvalitní, aby byli součástí Pareto fronty.

Koeficienty s_d a s_s vyjadřují o kolik jsou jim odpovídající kolize více nežádoucí než *sousední kolize ob dva dny*, kterým je ve vektoru $\mathbf{s}$ přiřazena implicitně hodnota 1. V našem případě jsme rozhodli, že *denní kolize* je dvakrát tolik významná než *sousední kolize*. Při té má sice student minimálně o 12 hodin více času na přípravu, přibližně polovinu této doby však stráví spánkem. Analogickou úvahou jsme došli k dvojnásobné významnosti *sousední kolize* oproti *sousední kolizi ob dva dny*. Pro koeficient sousední kolize jsme tedy zvolili $s_s = 2$ a pro koeficient denní kolize $s_d = 4$. K obdobným hodnotám jsme dospěli také na základě uživatelské fitness ohodnocené Ing. Jaroslavem Dytrychem, jehož názor vycházel především ze zkušenosti s reakcí studentů na tyto typy kolizí v minulosti. Pro následující experimenty byl tedy zvolen následující skalarizační vektor:

$$\mathbf{s} = \langle s_h = 500, s_d = 4, s_s = 2,\ 1 \rangle \tag{6.1}$$

Zbývající parametry, které je třeba určit, se týkají nastavení genetického algoritmu. Jedná se „klasicky“ o *velikost populace*, *počet generací*, *pravděpodobnost mutace* a *typ genetického operátoru*. Vliv těchto parametrů již lze vyhodnotit statisticky – pokud zvolíme vlastnosti *dostatečně dobrého řešení*[5], lze sledovat počet sestavení plánovací matice potřebných k získání takovéhoto, případně v některých ohledech lepšího, rozvrhu. Vzhledem k tomu, že výpočty plánovací matice vyžadují více než 90 % celkového výpočetního výkonu, může být tato veličina považována za dobrý ukazatel náročnosti výpočtu. Vhodná volba testovaného parametru je potom taková, se kterou bylo potřeba vykonat co nejméně sestavení a u které tedy vygenerování *dostatečně dobrého* rozvrhu vyžadovalo co nejméně výpočetních prostředků. S takovýmto nastavením můžeme například za stejný čas provést výpočet vícekrát a získat tak možnost výběru z větší množiny výsledků, nebo nechat výpočet běžet pro úměrně větší počet generací s vyhlídkou, že nalezne lepší řešení, než při kratším běhu.

Pro experimenty byla zvolena úloha `exams_2015_summer.xml`. Jí odpovídající reálně nasazený rozvrh, který byl vytvořen na fakultě, lze nalézt na přiloženém CD v souboru `official_schedules/zkousky2014L.htm.cs.html`. Ten samozřejmě splňuje všechna tvrdá kritéria. Pokud chceme jeho kvalitu porovnat s rozvrhy vytvořenými pomocí prezentovaného plánovacího systému, je potřeba pro něj určit hodnoty měkkých kritérií tak, jak byly

[5] Za *dostatečně dobré řešení* lze považovat takové, jehož kritéria splňují všechny podmínky nutné k tomu, aby mohlo být prakticky nasazeno. Jsou u něj tedy splněna všechna tvrdá kritéria a všechna měkká nabývají přijatelných hodnot.

formálně definovány v kapitole 4.3. K určení míry kolizí z uvedeného HTML souboru lze využít nástroje[6] `collision_counter` a `spacing_counter` z modulu `AEm.engine.tools`, které soubor analyzují a vypočtou hodnoty měkkých kritérií právě podle vzorců z kapitoly 4.3. Pomocí nich byly pro oficiální rozvrh letního semestru 2015 určeny následující hodnoty měkkých kritérií:

- *rozestupy termínů* $F_s = 74\,496$,
- *studentské kolize* $\mathbf{F_k} = \langle 348, 4145, 3362 \rangle$ a tedy $|\mathbf{F_k}| = \mathbf{F_k} \cdot \mathbf{s}^T = 13\,044$,
- *preference* $F_k = 850$ (tj. plné splnění preferencí).

Na základě toho byly určeny *dostatečně dobré* hodnoty měkkých kritérií, ve kterých musí být následující experimentálně hledané rozvrhy lepší, jako:

- *rozestupy termínů* $F_s = 78\,000$ (tj. alespoň o přibližně 5 % lepší),
- *studentské kolize* $|\mathbf{F_k}| = 11\,000$ (tj. alespoň o přibližně 15 % lepší),
- *preference* $F_k = 850$ (tj. stejně kvalitní).

Pro každý z evolučních parametrů byla provedena série 120 běhů výpočtu s jeho rozdílným nastavením. Tyto experimenty byly prováděny na superpočítači Anselm, jehož každý výpočetní uzel obsahuje dva osmijádrové procesory Intel Sandy Bridge E5-2665 (2,4-3,1 GHz) nebo Intel Sandy Bridge E5-2470 (2.3-3,1 GHz)[7].

Za nejdůležitější parametr pro efektivitu evoluce lze považovat velikost populace. Při malém počtu jedinců nemusí být v populaci dostatečná diverzita mezi jejich genotypy a evoluční vývoj začne stagnovat. Velký počet jedinců naopak nejspíše zajistí dostatečnou rozmanitost populace, avšak sestavení rozvrhů pro jednu populaci bude dosti časově náročné a prakticky tak nemusíme dosáhnout řešení v požadovaném čase.

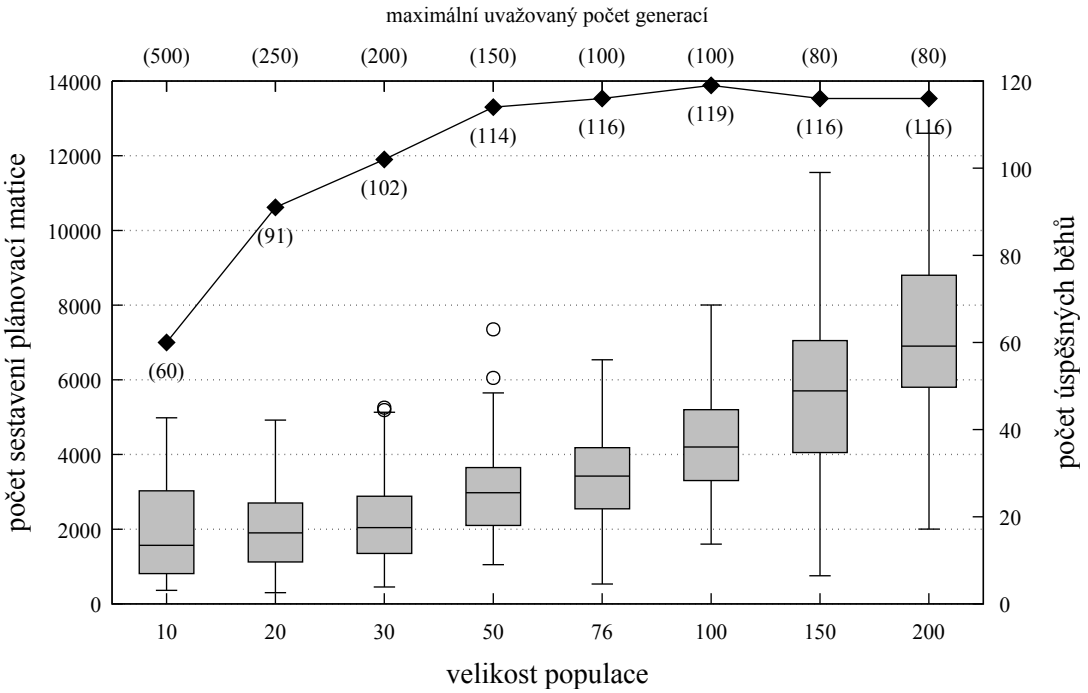
Na obrázku 6.1 je zobrazen graf náročnosti výpočtů pro velikosti populací v rozsahu od 10 po 200 jedinců. Na něm lze vidět, že při použití malých populací (10, 20 nebo 30 jedinců) je ve většině úspěšných běhů dosaženo *dostatečně dobré řešení* již po 2 000 sestavení plánovací matice. Na druhou stranu u nich však existuje vysoký počet běhů, které řešení těchto kvalit nedosáhnou ani po 5 000 až 6 000 sestavení (až 50 % běhu při 10 jedincích v populaci). Dá se předpokládat, že toto chování je způsobeno právě zdegenerováním populace – algoritmus v tomto případě uvázne v lokálním extrému optimalizovaného problému.

Pro ostatní vybrané velikosti populací je již pravděpodobnost úspěšného běhu alespoň 95 %. Se zvětšující se populací ale významně roste potřebný počet sestavení plánovací matice. Například pro populaci o 200 jedincích je ve většině běhů potřeba více než 6 500 sestavení.

Zvolené střední velikosti populací (50, 76 a případně 100 jedinců) vychází z uvedeného grafu jako nejvhodnější pro danou úlohu. Ve většině případů je u nich nalezeno *dostatečně dobré řešení* po méně než 4 500 ohodnocení. Pro další testy jsme zvolili nejmenší z nich, tedy velikost populace 50 jedinců. Jde o nejméně výpočetně náročnou variantu, která však pořád zaručuje nalezení *dostatečně dobrého řešení* při 95 % běhů. Díky její nízké výpočetní

[6] Nástroje pro výpočet měkkých kritérií z HTML souborů byly otestovány i nad rozvrhy vygenerovanými pomocí prezentovaného systému se známou hodnotou fitness. Bylo tím ověřeno, že hodnota jejich výstupu se shoduje s hodnotou určenou v průběhu evoluce.

[7] Technické parametry superpočítače Anselm lze nalézt na http://www.it4i.cz/wp-content/uploads/2013/09/Technick%C3%A9-parametry-Anselma.pdf.

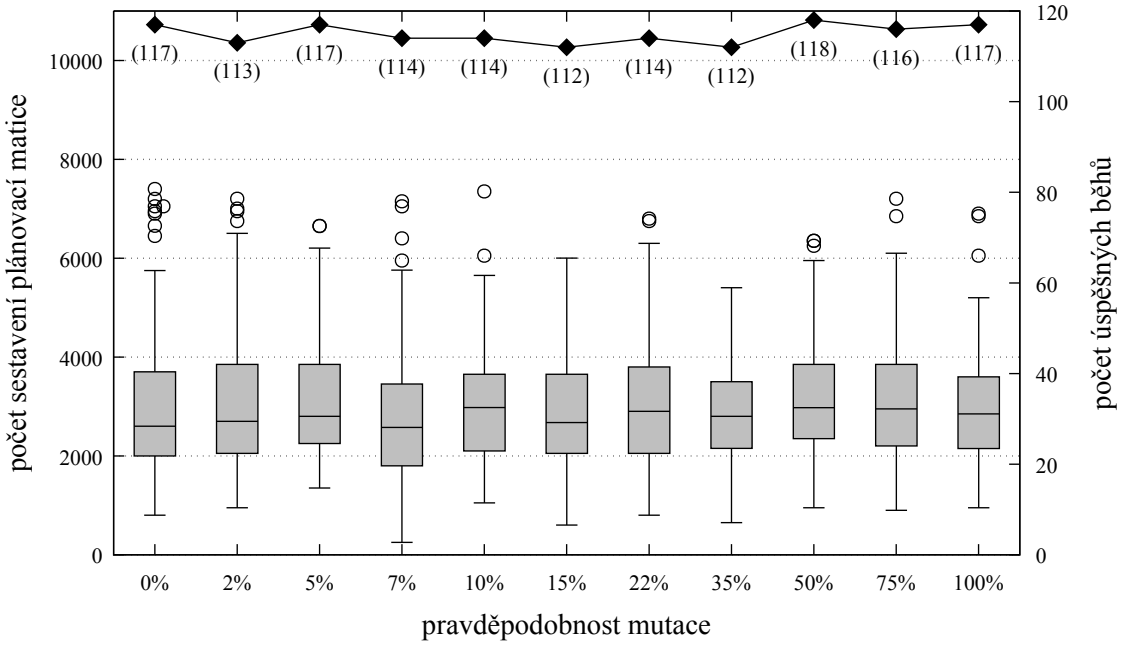


Obrázek 6.1: Graf zachycující výpočetní náročnost programu v závislosti na *velikosti populace* genetického algoritmu. Pro všechny běhy byla nastavena 10 % pravděpodobnost mutace a operátor křížení typu *PBX*. Křivka v horní části grafu zobrazuje počet běhů, při kterých bylo nalezeno *dostatečně dobré řešení* dříve, než bylo dosaženo *maximálního počtu generací*. Neúspěšné běhy nejsou při výpočtu statistických hodnot uvažovány.

náročnosti může uživatel spustit více nezávislých běhů a mít tak téměř jistotu dosažení pro něj přijatelného výsledku.

Dalším parametrem, který je třeba před spuštěním výpočtu zvolit je pravděpodobnost mutace jedince při rekombinaci. Na obrázku 6.2 je zobrazen graf náročnosti výpočtu pro různé pravděpodobnosti mutace v rozsahu od 0 po 100 %. V grafu však není zřejmá žádná míra korelace mezi těmito dvěma veličinami. Sledované statistické vlastnosti všech běhů, kterých bylo znovu provedeno 120 pro každé nastavení, se jeví velmi podobné. Pravděpodobnost úspěšného běhu se pohybuje v rozmezí 93-98 %. Dostatečně kvalitní řešení bylo u všech typů běhů ve většině případů nalezeno mezi 2 000. a 4 000. sestavením plánovací matice. Jedinou výraznější výjimku zde tvoří pravděpodobnost 7 %, u které bylo ve většině případů toto řešení nalezeno mezi 1800. a 3600. sestavením. S ohledem na okolní hodnoty, lze ale tuto nuanci přiřadit spíše statistické chybě, než vhodnosti 7 % pravděpodobnosti mutace.

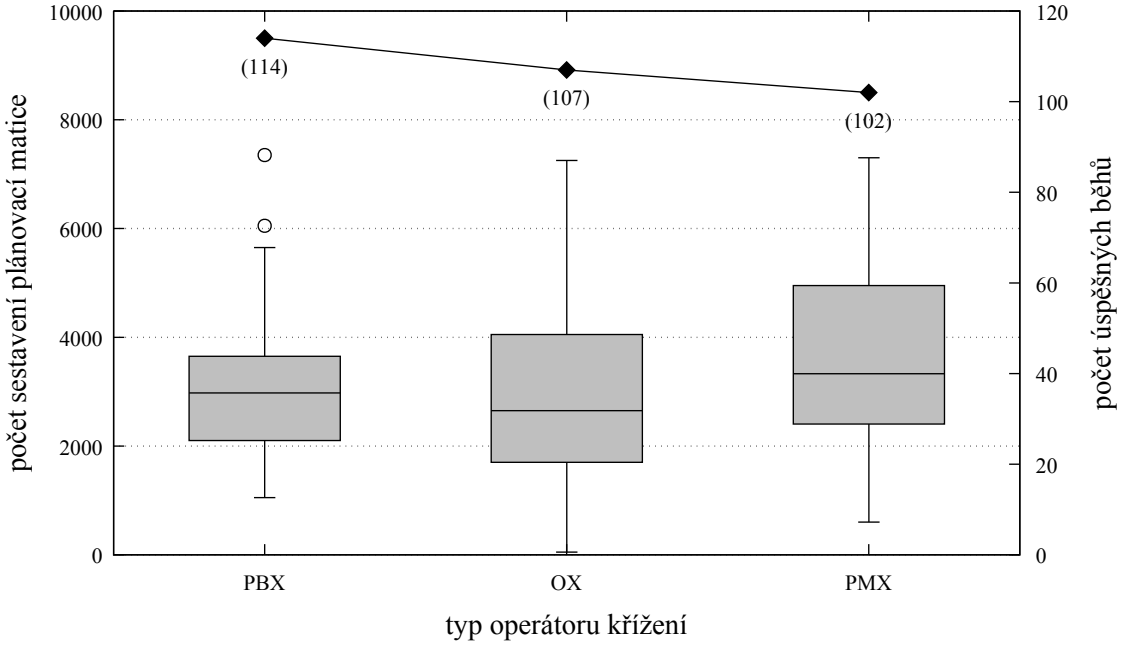
Toto chování může být způsobeno vysokou mírou elitismu, kterou přináší zvolený selekční mechanismus *NSGA-II*. Naprostá většina zmutovaných jedinců totiž bude mít zhoršenou hodnotu fitness a nemusí se tak vůbec do následující populace dostat. Jiným důvodem může také být dostatečná rozmanitost populace, která je apriori zajištěna její velikostí (viz předchozí experiment). V průběhu uvažovaných 150 generací totiž nemusí být prostor k tomu, aby dokázala silná řešení populaci zdegenerovat. Operátor mutace se pak nemusí projevit, protože do výpočtu přinese jen minimum nové informace. Mutace by se však mohla výrazněji projevit u výpočtů s menší velikostí populace, které snáze uváznou v lokálním extrému prohledávaného prostoru.



Obrázek 6.2: Graf zachycující výpočetní náročnost programu v závislosti na *pravděpodobnosti mutace* jedince při rekombinaci. Pro všechny běhy byla zvolena velikost populace 50 jedinců, maximální počet 150 generací a operátor křížení typu *PBX*. Křivka v horní části grafu zobrazuje počet běhů, při kterých bylo nalezno *dostatečně dobré řešení* dříve, než bylo dosaženo *maximálního počtu generací*. Neúspěšné běhy nejsou při výpočtu statistických hodnot uvažovány.

Posledním evolučním parametrem je typ operátoru křížení – program nabízí výběr mezi *PBX*, *PMX* a *OX*. Všechny tři operátory byly popsány v kapitole 3.1.1. Na obrázku 6.3 je zobrazen graf náročnosti výpočtu při použití každého z nich. Nejhorších výsledků bylo dosaženo při použití operátoru *PMX*. Ten měl nejnižší úspěšnost běhů a zároveň jeho úspěšné běhy byly nejvíce výpočetně náročné. Při porovnání operátorů *PBX* a *OX* lze u *PBX* pozorovat výrazně nižší rozptyl, díky němuž má *PBX* vyšší počet úspěšných běhů než *OX*, přestože byly jeho běhy mírně výpočetně náročnější. Jako vhodnější jsme zvolili právě *PBX*, protože dosahuje stálejších výsledků – uživatel tak má větší jistotu, že za určitý čas dosáhne pro něj přijatelného řešení.

Při všech experimentech provedených v této kapitole se za úspěšné běhy považovaly pouze ty, jejichž výsledky splnily v plné míře požadované preference vyučujících. Z tohoto pohledu tak bylo měkké kritérium *preferencí* transformováno na tvrdé kritérium. Prakticky však jen velmi málo běhů dosáhlo ostatních požadovaných hodnot dříve, než byly plně splněny preference. Toto chování lze považovat za správné, protože pokud uživatel preference zadá, tak předpokládá, že jim bude vyhověno. Dalšími experimenty se ukázalo, že v případě zvýšení náročnosti plánování (vyšší minimální rozestupy, méně dostupných dnů a podobně), začne heuristický algoritmus plánovat i do časových slotů, které mají nižší hodnoty preferencí. Rozvrhy s celkovou nižší mírou preference jsou tedy generovány až ve chvíli, kdy algoritmus nemá jinou možnost jak dokončit sestavení rozvrhu (toto chování koresponduje právě se způsobem výběru zdrojů, viz kapitola 4.2.3). Navržený způsob nakládání s preferencemi je tak řadí na pomezí měkkých a tvrdých kritérií.



Obrázek 6.3: Graf zachycující výpočetní náročnost programu v závislosti na zvoleném typu *operátoru křížení* genetického algoritmu. Pro všechny běhy byla zvolena velikost populace 50 jedinců, maximální počet 150 generací a pravděpodobnost mutace 10 %. Křivka v horní části grafu zobrazuje počet běhů, při kterých bylo nalezeno *dostatečně dobré řešení* dříve, než bylo dosaženo *maximálního počtu generací*. Neúspěšné běhy nejsou při výpočtu statistických hodnot uvažovány.

6.1.1 Efektivita paralelizace

V kapitole 5.1 byl popsán způsob paralelizace výpočtu do libovolného počtu podprocesů. Do nich je možné delegovat sestavení plánovací matice na základě jednotlivých chromozomů jedinců. Rodičovský proces pouze tyto chromozomy rovnoměrně rozdělí mezi podprocesy. Ty po dokončení všech sestavení navrátí odpovídající hodnoty fitness.

Na grafu z obrázku 6.4 jsou zobrazeny reálné doby výpočtu (tzv. wall time) při různém stupni paralelizace. Data byla získána při výpočtech na základních uzlech superpočítače Anselm s vypnutými technologií Intel Turbo Boost a Intel Hyper Threading. Bez uvedených technologií tyto uzly nabízí dvakrát 8 nativních výpočetních jader při konstantní frekvenci 2,4 GHz (jedná se o dvojici procesorů Intel E5-2665). Při výpočtech byl zajištěn unikátní přístup k těmto uzlům. Testované počty podprocesů (2, 4, 8 a 16) byly zvoleny proto, aby mohl být výpočetní uzel vždy plně vytížen. Navíc byly provedeny testy i bez využití paralelního zpracování (první sloupec grafu) – v tomto případě se fyzicky nevytváří žádný další podproces, ale plánovací matice jsou sestaveny přímo hlavním procesem.

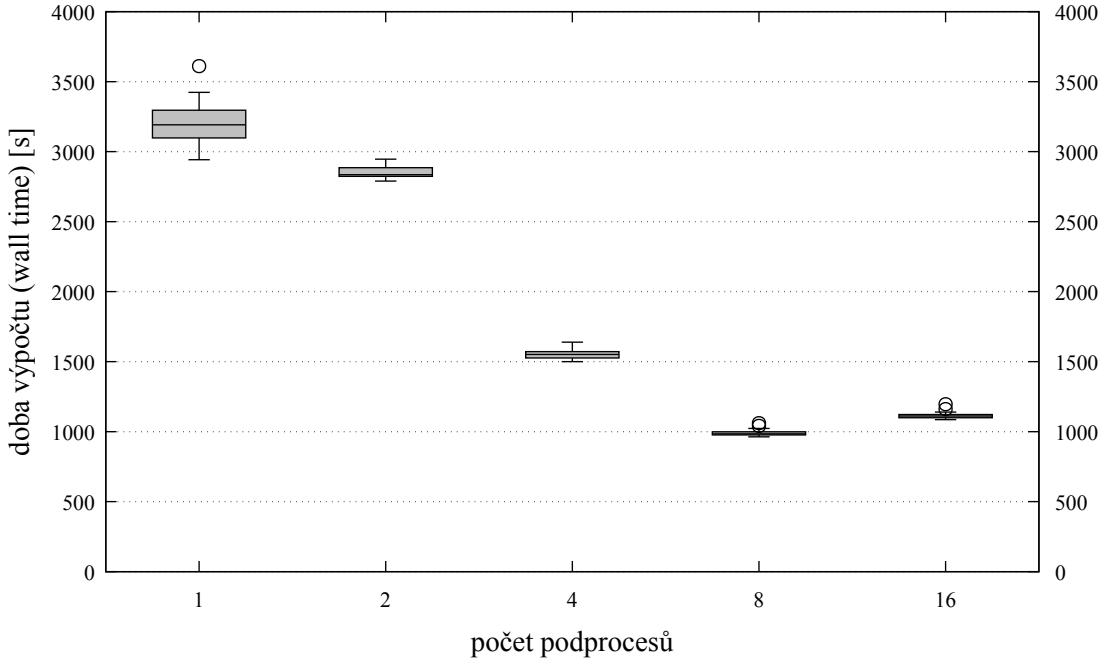
Z grafu je patrné, že náročnost výpočtů není výrazně závislá na konkrétním vývoji evoluce. Rozptyl jednotlivých měření je, především pro paralelní běhy, totiž dosti malý. Také se ukázalo, že paralelizace pouze do dvou podprocesů nepřináší příliš velký užitek. To je dáno nejspíše skutečností, že zrychlení získané rozdělením pouze do dvou podprocesů se vyrovná režii spojenou s paralelním přístupem.

Velký výkonnostní skok však lze pozorovat při přechodu k paralelizaci čtyřmi pod-

procesy. Oproti dvěma podprocesům zde byla naměřena téměř poloviční doba výpočtu. Režie spojená s paralelizací se tedy přidáním dalších dvou podprocesů nijak významně nezvýšila. Přesto je v tomto případě poměr zrychlení k využitým výpočetním prostředkům pouze mírně přes 0,5. Přidáním dalších čtyř podprocesů na celkových osm dojde již pouze k asi 30 % zrychlení. Překvapivá je doba výpočtu při využití šestnácti podprocesů, v tomto případě dojde dokonce k celkovému zpomalení oproti výpočtu s osmi podprocesy.

Horší efektivita při osmi a šestnácti podprocesech je pravděpodobně způsobena malou velikostí populace pro takto velké stupně paralelizace. Při osmi podprocesech totiž každý z nich sestavuje nejvýše sedm plánovacích matic, při šestnácti jsou dokonce každému podprocesu přiděleny nejvýše čtyři matice. Při vysoké míře paralelizace se také výrazněji projevuje tzv. *Amdahlův zákon* – výkonnostně kritickými částmi aplikace se stávají její neparalelizované moduly. V případě užití šestnácti podprocesů také přibude zpomalení v důsledku fyzické achitektury výpočetního uzlu. Jak již bylo řečeno, ten se skládá ze dvou procesorů po osmi jádrech. Při výpočtech do osmi podprocesů je zajištěno, že každý výpočet bude probíhat vždy jen na prvním nebo druhém procesoru. Při užití šestnácti podprocesů je však jedním výpočtem vytížen celý uzel a je tak nutné, aby oba procesory průběžně spolupracovaly. Komunikační linka mezi nimi je však jistě pomalejší, než komunikační linky mezi jádry pouze jednoho procesoru.

Pro zvolené nastavení se tedy při využití paralelizace jeví nejlépe rozdělení do čtyř podprocesů. V případě větší velikosti populace lze uvažovat i o více podprocesech, je však nutné vzít v ohled architekturu výpočetního serveru.



Obrázek 6.4: Graf zachycující reálnou dobu výpočtu v závislosti na míře paralelizace. Všechny výpočty byly provedeny 32krát pro populaci o velikosti 50 jedinců po 150 generací. Pravděpodobnost mutace byla zvolena 10 %, operátor křížení byl vybrán *PBX*.

6.1.2 Reálné použití

Pro obě experimentální úlohy `exams_2015_winter.xml` a `exams_2015_summer.xml` byly spuštěny i experimenty s cílem dosáhnout co nejlepších řešení. Pro každou úlohu byly spuštěny dva běhy s následujícím nastavením:

- kolizní skalarizační vektor $\mathbf{s} = \langle 500, 4, 2, 1 \rangle$,
- velikost populace 350 jedinců,
- počet generací 500,
- pravděpodobnost mutace 10 %,
- operátor křížení typu *PBX*.

Zvolené hodnoty parametrů vychází z experimentů v kapitole 6.1. Velikost populace a počet generací jsou oproti těmto experimentům výrazně navýšeny ve snaze analyzovat co největší oblast prohledávaného prostoru. Při takto velké populaci je totiž výrazně nižší pravděpodobnost, že algoritmus uvázne v lokálním optimu. Uskutečněné běhy vlastně napodobují reálné užití programu, při kterém uživatel vyžaduje co nejlepší výsledek „na první pokus“. Každý běh byl proveden dvakrát právě z důvodu zvýšení jistoty nalezení kvalitního výsledku – všechny čtyři běhy v tomto uspěly, jejich rozvrhy dosáhly velmi podobných hodnot. Znovu nejsou dále diskutovány míry preferencí vyučujících, protože jsou ve všech uváděných řešeních splněny v maximální míře.

Na obrázcích 6.5 a 6.6 jsou graficky zobrazeny míry kvality nejlepších dosažených rozvrhů pro příslušné zimní a letní zkouškové období. V pravé spodní části těchto grafů je pro porovnání vyznačena kvalita, která byla stejným způsobem vyčíslena pro oficiální rozvrhy.

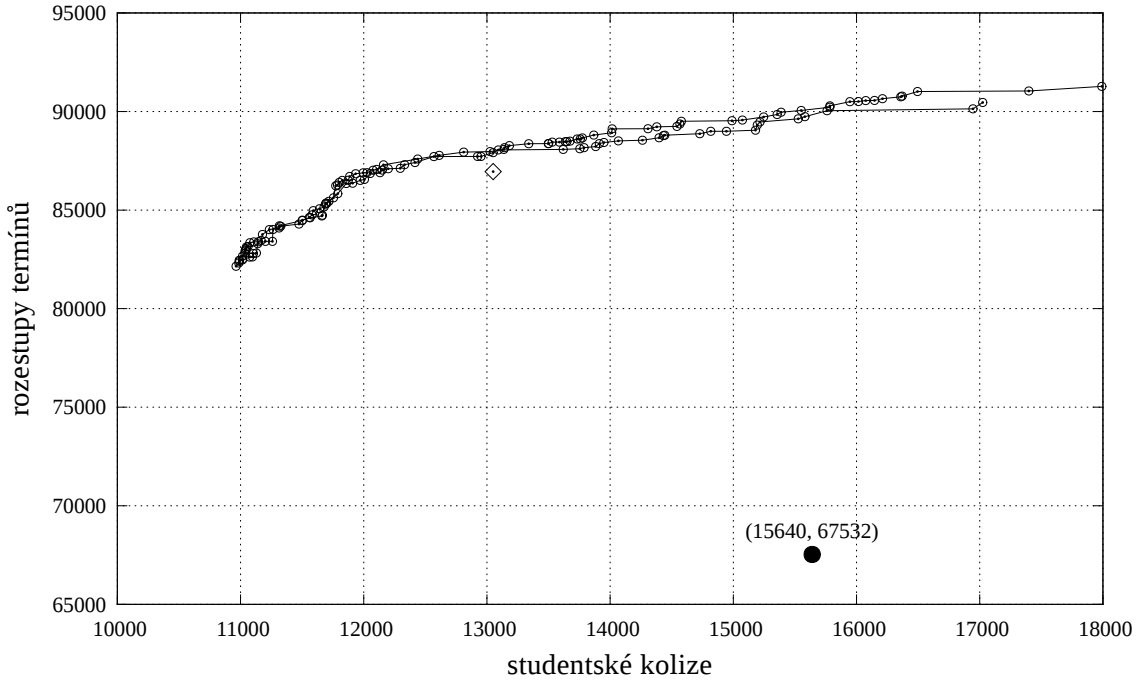
Všechna řešení v rámci jednoho běhu, jejichž kvality jsou zobrazeny, si vzájemně nedominují. Je tedy na uživateli jaký z těchto rozvrhů si na základě jeho vlastních představ vybere. Při tom může zahrnout i další kritéria, která dokáže nejlépe expertně ohodnotit právě uživatel sám.

Průměrná kvalita těchto řešení je v obou grafech zobrazena znakem ⟐ (diamant s tečkou). Pro zimní semestr byla průměrná hodnota *studentských kolizí* určena jako 13 051, což je asi o 16 % lepší než u oficiálního rozvrhu, a průměrná hodnota *rozestupů termínů* 86 954, což je asi o 29 % lepší než u oficiálního zimního rozvrhu. V případě letního semestru byla průměrná hodnota *studentských kolizí* určena jako 11 001, což je znovu asi o 16 % lepší než u oficiálního rozvrhu, a průměrná hodnota *rozestupů termínů* 84 327, což je asi o 13 % lepší hodnota než u oficiálního letního semestru.

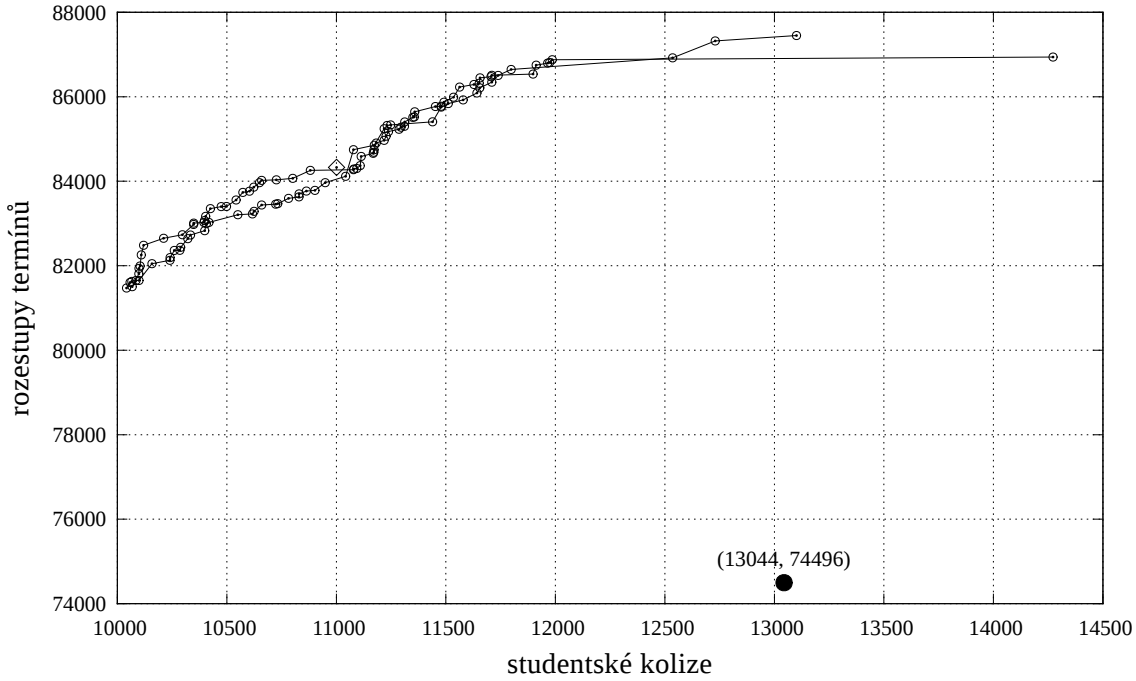
Náročnost jednoho běhu těchto experimentů je vyšší než u experimentů z kapitoly 6.1.1, které zkoumaly dobu výpočtu v závislosti na stupni paralelizace. U nich byla měřena dobou výpočtu (wall time) pro populaci o 50 jedincích po 150 generací, což odpovídá 7 500 sestavení plánovací matice. Bez využití paralelizace zde byla změřena průměrná doba výpočtu přibližně 53 minut (resp. 26 minut při rozložení do 4 podprocesů) – viz graf na obrázku 6.4.

Pro získání výše uvedených výsledků byla použita populace o 350 jedincích po 500 generací, což vyžaduje 175 000 sestavení plánovací matice. Běhy experimentů provedené v této kapitole jsou tedy přibližně 23krát náročnější. Předpokládanou dobu výpočtu na procesoru Intel E5-2665 je tak možné odhadnout[8] jako $23 \cdot 53 = 1\ 219$ minut ≈ 20 hodin (resp. přibližně 10 hodin při rozložení do 4 podprocesů).

[8] Dobu výpočtu (wall time) u těchto experimentů nešlo přímo změřit tak, aby měly vypovídající hodnotu, protože tyto experimenty probíhaly na serveru bez unikátního přístupu, jehož průměrná zátěž (load) přesahovala počet procesorových jader.



Obrázek 6.5: Dvě Pareto fronty reprezentující 65 + 76 nejlepších nalezených řešení pro zkouškový rozvrh *zimního semestru* akademického roku 2014/2015. Plná tečka v pravé spodní části značí kvalitu oficiálního rozvrhu, který byl toto období použit v minulosti.



Obrázek 6.6: Dvě Pareto fronty reprezentující 46 + 60 nejlepších nalezených řešení pro zkouškový rozvrh *letního semestru* akademického roku 2014/2015. Plná tečka v pravé spodní části značí kvalitu oficiálního rozvrhu, který byl toto období použit v minulosti.

Kapitola 7

Závěr

V této diplomové práci byl navržen a implementován systém pro automatizované plánování rozvrhů zkoušek a přednášek na *Fakultě informačních technologií Vysokého učení technického v Brně*. Byla provedena analýza potřeb fakulty, jejíž výstupem byly mimo jiné seznamy kritérií, které zde musí být pro oba typy rozvrhů splněny. S ohledem na její výsledky byl navržen obecný hybridní algoritmus pro automatizované generování rozvrhů zkoušek i přednášek.

Navržený algoritmus se skládá z evoluční a heuristické části. Genotyp jedince v evoluční části obsahuje postup, podle jakého heuristická část sestrojí odpovídající fenotyp. Následně může být ohodnocena jeho kvalita. Jedinci jsou v evolučním algoritmu kódováni permutačně. Konkrétní permutace reprezentuje pořadí přednášek (resp. zkoušek), v jakém budou heuristickým algoritmem umístěny do rozvrhu v průběhu jeho konstrukce. Rozvrh je konstruován tak, aby nebylo porušeno žádné *tvrdé kritérium*. V průběhu sestavování je sledováno porušení *měkkých kritérií*, kterým se snaží heuristický algoritmus v maximální míře vyhýbat. Díky průběžnému sledování měkkých kritérií je po dokončení sestavení možné jednoduše určit odpovídající kvalitu jedince (jeho fitness).

Na základě tohoto návrhu byl v listopadu 2014 implementován prototyp, na kterém mohl být v předstihu ověřen potenciál navrženého řešení. Z tohoto prototypu byl postupným přidáváním funkcionality vytvořen prezentovaný plánovací systém, který se skládá z konzolové aplikace v jazyce Python 3 a odpovídajícího grafického rozhraní. V průběhu implementace byly prováděny různé výkonnostní optimalizace (včetně paralelizace nejnáročnější části výpočtu), díky kterým byla náročnost výpočtu udržena na úrovni původního prototypu. Při použití výkonného procesoru je prakticky doba trvání výpočtu v řádech desítek minut. Vývoj byl průběžně konzultován s osobami zodpovědnými za plánování rozvrhů na fakultě, aby vytvořený plánovací systém co nejvíce vyhovoval jejich představám – a to jak z pohledu získaných výsledků, tak uživatelské přívětivosti.

Implementovaný systém byl testován na reálných datech pro zkouškové období zimního a letního semestru 2014/2015. Byly provedeny experimenty za účelem zjistit doporučená nastavení genetické části algoritmu a také pro porovnání generovaných rozvrhů s reálně nasazenými rozvrhy v minulosti. Získané výsledky potvrzují funkčnost zvoleného řešení – program dokáže generovat rozvrh, který oproti reálným (ručně vytvářeným) rozvrhům dosáhnul alespoň o 13 % lepších vlastností v rámci každého ze sledovaných kritérií. Po dokončení výpočtu je uživateli nabídnuto více vygenerovaných rozvrhů, ze kterých si může vybrat ten, který se mu jeví jako nejlepší. Výstup programu je kompatibilní s vizualizačním programem pro manuální vytváření rozvrhů, který se nyní na fakultě používá. Do něj je tak možné vygenerované rozvrhy importovat a případně je dodatečně upravit.

Vytvořený plánovací systém bude příští akademický rok (2015/2016) použit při vytvoření zkouškových rozvrhů na fakultě. V budoucnu by jej bylo vhodné integrovat s webovým informačním systémem fakulty. Jeho uživatelské rozhraní by umožnilo vyučujícím jednoduše zadat všechny jimi kladené požadavky na rozvrhy. Informační systém by následně automaticky vytvořil soubor se vstupními požadavky, který se předává vytvořenému plánovacímu systému při jeho spuštění.

Při řešení této diplomové práce jsem si zopakoval přednášené a nastudoval nové modely evolucí inspirovaných výpočetních metod určených pro plánovaní procesů, multikriteriální optimalizaci a podobně. S některými z nich jsem se pak blíže seznámil při vlastní implementaci v jazyce Python 3. Vyzkoušel jsem si různé optimalizace na úrovni návrhu algoritmu a využil jsem možnost experimenty provést na ostravském superpočítaci Anselm.

Tato práce byla publikována ve sborníku studentské konference Excel@FIT 2015.

Literatura

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

[2] Deb, K.; Jain, H.: An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. *IEEE Trans. Evolutionary Computation*, ročník 18, č. 4, 2014, s. 577-601, ISSN 1089-778X.

[3] Deb, K.; Pratap, A.; Agarwal, S.; aj.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, ročník 6, č. 2, 2002, s. 182–197, ISSN 1089-778X.

[4] Eiben, A.; Raué, P.-E.; Ruttkay, Z.: Genetic algorithms with multi-parent recombination. In *Parallel Problem Solving from Nature – PPSN III*, Springer Berlin Heidelberg, 1994, 78-87 s., ISBN 978-3-540-58484-1.

[5] Fang, H.-L.: *Genetic Algorithms in Timetabling and Scheduling*. Dizertační práce, Department of Artificial Intelligence, University of Edinburgh, 1994, 227 s.

[6] Fogel, D. B.: *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, 1995, 296 s., ISBN 0-7803-1038-1.

[7] Green, C. D.: The Generalisation and Solving of Timetable Scheduling Problems. In *Practical Handbook of Genetic Algorithms: Complex Coding Systems*, CRC Press, 1998, s. 17-64, ISBN 0-8493-2539-0.

[8] Holland, J. H.: *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, 1992, 211 s., ISBN 978-0262581110.

[9] Horký, A.: *Tvorba rozvrhů pomocí genetických algoritmů*. Bakalářská práce, Vysoké učení technické v Brně, 2012, 54 s.

[10] Hynek, J.: *Genetické algoritmy a genetické programování*. Grada Publishing, a.s., 2008, 182 s., ISBN 978-80-247-2695-3.

[11] Jong, K. D.: Generalized Evolutionary Algorithms. In *Handbook of Natural Computing*, Springer Publishing Company, Incorporated, 2012, s. 626-635, ISBN 978-3-540-92909-3.

[12] Kubalcová, M.: *Porovnání programů pro plánování rozvrhů a zkoušek*. Bakalářská práce, Vysoké učení technické v Brně, 2012, 50 s.

[13] Kvasnička, V.; Pospíchal, J.; Tiňo, P.: *Evolučné algoritmy.* STU, Bratislava, 2000, 215 s., ISBN 80-227-1377-5.

[14] Schwefel, H.-P. P.: *Evolution and Optimum Seeking: The Sixth Generation.* John Wiley & Sons, Inc., 1993, 456 s., ISBN 978-0471571483.

[15] Shi, C.; Chen, M.; Shi, Z.: A Fast Nondominated Sorting Algorithm. In *International Conference on Neural Networks and Brain, 2005. ICNN B '05.*, 2005, s. 1605-1610, ISBN 0-7803-9422-4.

[16] Singh, V.; Choudhary, S.: Genetic algorithm for Traveling Salesman Problem: Using modified Partially-Mapped Crossover operator. In *IMPACT '09. International Multimedia, Signal Processing and Communication Technologies, 2009.*, 2009, s. 20-23, ISBN 978-1-4244-3604-0.

[17] Thanh, N. D.: Solving Timetabling Problem Using Genetic and Heuristic Algorithms. In *Eigth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007.*, 2007, s. 472-477, ISBN 978-0-7695-2909-7.

[18] Whitley, D.; Sutton, A. M.: Genetic Algorithms – A Survey of Models and Methods. In *Handbook of Natural Computing*, Springer Publishing Company, Incorporated, 2012, s. 638-671, ISBN 978-3-540-92909-3.

[19] Zitzler, E.: Evolutionary Multiobjective Optimization. In *Handbook of Natural Computing*, Springer Publishing Company, Incorporated, 2012, s. 872-904, ISBN 978-3-540-92909-3.

Příloha A

Obsah přiloženého DVD

Adresářová struktura přiloženého DVD je následující:

- `official_schedules/` - Oficiální zkouškové rozvrhy a podklady pro jejich tvorbu na *FIT* v akademickém roce 2014/2015.
- `src/`
  - `AEm/` - Implementace plánovacího systému.
  - `inputs/` - Složka se vstupními soubory.
  - `outputs/` - Složka pro vygenerované rozvrhy.
- `tex/` - Zdrojové soubory pro tuto technickou zprávu ve formátu LaTeX.
- `dip_xhorky17.pdf` - Tato technická zpráva.
- `experiments.zip` - Výsledky a logovací soubory všech experimentů uváděných v této práci. Varování, po rozbalení zabírá asi 40 GB a obsahuje 500 000 souborů!

Příloha B

Popis konfigurační XML souboru s parametry výpočtu

Ukázka konfigurační XML souboru, který obsahuje uživatelsky měnitelné parametry výpočtu. Cesta k tomuto souboru musí být zadána při jeho spuštění (viz kapitola 5). Předpokládané umístění tohoto souboru je ve složce `/src/inputs/settings/`. Žádný z uvedených parametrů není povinný (příslušnou XML značku lze vynechat). V případě jeho absence je použita výchozí hodnota ze souboru `/src/inputs/settings/default.xml`. Hodnoty v následujícím příkladu odpovídají právě hodnotám z výchozího souboru.

```
<?xml version="1.0" encoding="UTF-8"?>
<settings>
  <!-- Velikost populace (kladné sudé číslo). -->
  <pop_size value="50"/>

  <!-- Počet generací (kladné celé číslo). -->
  <iterations value="150"/>

  <!-- Pravděpodobnost mutace (celé číslo v intervalu ⟨0-100 %⟩). -->
  <mutation_prob value="10"/>

  <!-- Operátor křížení (jeden z „PBX“, „PMX“, „OX“). -->
  <crossover_op value="PBX"/>

  <!-- Počet paralelních podprocesů užitých pro výpočet.
       (kladné celé číslo – hodnota 1 vypne paralelní zpracování) -->
  <threads_count value="4"/>

  <!-- Spuštění v testovacm módu (1 nebo 0),
       který hlídá validitu souborů při užití parametru --resume. -->
  <debug value="1"/>

  <!-- Prvky skalarizačního vektoru pro výpočet kolizí (kladná čísla). -->
  <ts_importance    value="500"/>
  <day_importance   value="4"/>
  <neigh_importance value="2"/>
</settings>
```

Příloha C

Popis vstupního XML souboru s popisem problému

Ukázka vstupního XML souboru popisujícího všechny požadavky na rozvrh, které může uživatel zadat. Cesta k tomuto souboru musí být předána při spuštění výpočtu (viz kapitola 5). Předpokládané umístění souboru je ve složce `/src/inputs/problems/`. V této složce lze nalézt soubory s požadavky na rozvrhy odpovídající akademickému roku 2014/2015 na *FIT*.

Dále uvedená ukázka není validním vstupním souborem, protože na některých místech obsahuje znak „· · · “ (tři tečky), kterým je vyjádřeno libovolné opakování značky v analogii s ostatními značkami na stejné úrovni. Při úpravách je nutné zachovat uspořádání značek na druhé úrovni (`days`, `timeslots` atd.), které obsahují odpovídající komponenty. Při jednoprůchodovém čtení tohoto souboru totiž probíhá validace vazeb mezi níže umístěnými komponentami k některým výše definovaným komponentám.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Kořenová značka s povinným atributem model, který nabývá hodnoty
     exams pro zkouškový rozvrh nebo lectures pro rozvrh přednášek. -->
<problem model="exams">

  <!-- Seznam dnů, ve kterých má probíhat plánování. Dny je nutné číslovat
       od nuly pomocí atributu id, aby na ně bylo možné jednoduše odkazovat. -->
  <days>
    <day id="0" day="5" month="1" year="2015"/>
    <day id="1" day="6" month="1" year="2015"/>
    <day ··· />
  </days>

  <!-- Seznam časových slotů. Atributem real_hour_offset je možné nastavit,
       od jaké hodiny budou časové sloty číslovány ve vytvořené tabulce s rozvrhem.
       Při nastavení real_hour_offset=8 tak bude hodina s číslem 0 začínat v 8:00
       a končit v 8:50. Tento parametr nijak neovlivňuje průběh výpočtu.

       V případě zkouškového rozvrhu musí být časové sloty rozděleny do oblastí
       pro jednotlivé termíny zkoušek (viz kap. 5.1). K vytyčení těchto oblastí
```

```
      slouží volitelné parametry order_start a order_end. -->
<timeslots real_hour_offset="8">
  <!-- V prvním časovém slotu musí začínat i oblast pro první termíny.  -->
  <timeslot day="0" hour="0" order_start="0"/>
  <timeslot day="0" hour="1"/>
  <timeslot ··· />
  <timeslot day="6" hour="10"/>

  <!-- Oblasti se mohou libovolně překrývat. Zde například začíná oblast
       pro druhé termíny a první přitom ještě nebyla ukončena. -->
  <timeslot day="7" hour="0" order_start="1"/>
  <timeslot day="7" hour="1"/>
  <timeslot ··· />
  <timeslot day="10" hour="6"/>

  <!-- Ukončení oblasti pro první termíny zkoušek. -->
  <timeslot day="10" hour="7" order_end="0"/>
  <timeslot day="11" hour="0"/>
  <timeslot ··· />
  <!-- Počet časových slotů v každém dni může být libovolný. -->
  <timeslot day="17" hour="9"/>

  <!-- V poslední časovém slotu musí být ukončena i oblast pro poslední termíny. -->
  <timeslot day="17" hour="10" order_end="1"/>
</timeslots>


<!-- Seznam místností, do kterých bude plánována výuka. Každá učebna může
     obsahovat podznačky bussy, kterými lze definovat rozsahy časových slotů,
     ve kterých není možné učebnu využít. -->
<rooms>
  <!-- Učebna G202 s kapacitou pro 80 studentů. -->
  <room name="G202" capacity="80">
  </room>

  <!-- Učebna D105 spadající do skupiny učeben číslo 1, ze které budou vytvořeny
       tzv. virtuální zdroje (viz kap. 4.2). Tato učebna je navíc hlavní, musí se tedy
       v každém takto vytvořeném virtuálním zdroji vyskytovat.

       Tuto učebnu nelze využít prvních 8 hodin ve dni 24 a 10 hodin ve dni 9. -->
  <room name="D105" capacity="300" group="1" group_type="master">
    <bussy from_day="24" to_day="24" from_hour="0" to_hour="7"/>
    <bussy from_day="2" to_day="2" from_hour="0" to_hour="9"/>
  </room>

  <!-- Učebna D0206 spadající do skupiny učeben číslo 1. Tato učebna se může vysky-
       tovat ve virtuálním zdroji pouze s nějakou hlavní učebnou ze stejné skupiny.
```

```
    Tuto učebnu nelze využít v intervalu od nulté hodiny dne s id dvě po devátou
    hodinu dne s id čtyři. -->
  <room name="D0206" capacity="154" group="1" group_type="slave">
    <bussy from_day="2" to_day="4" from_hour="0" to_hour="9"/>
  </room>

  <!-- Učebna D0207 (bez časových omezení) spadající do skupiny učeben číslo 1,
    ve které se nachází laser. -->
  <room name="D0207" capacity="90" group="1" group_type="slave">
    <provide what="laser"/>
  </room>

  <!-- Učebna FEKT s nulovou kapacitou, do které je umožněno pouze fixní plánování
    faktorů. Učebna je navíc vzdálená (viz kap. 5.2) – při plánování do ní jsou
    tedy v rozvrhu vytvořeny před i po faktoru volné hodiny, sloužící pro případný
    přesun studentů zpět na FIT. -->
  <room name="FEKT" capacity="0" distant="yes">
  </room>

  <room ··· >
    ...
  </room>
</rooms>


<!-- Seznam vyučujících, kteří vedou výuku. Každý vyučující může obsahovat
    podznačky bussy, kterými lze definovat rozsahy časových slotů,
    ve kterých nemůže vyučovat (je zaneprázdněn). -->
<teachers>
  <teacher name="Grmela Lubomír">
  </teacher>

  <teacher name="Kotásek Zdeněk">
    <bussy from_day="08" to_day="08" from_hour="0" to_hour="5"/>
    <bussy from_day="18" to_day="18" from_hour="0" to_hour="5"/>
  </teacher>

  <teacher ··· >
    ...
  </teacher>
</teachers>


<!-- Seznam předmětů, na které se dále odkazují faktory. Každý předmět může obsa-
    hovat podznačky group, kterými lze definovat skupiny studentů, kteří jsou v něm
    zapsáni. Hodnoty značky group nemají žádný vliv na průběh výpočtu, ve vygene-
    rovaném rozvrhu však podle nich lze filtrovat zobrazené faktory. -->
<lectures>
```

```
  <lecture name="IZG">
    <group type="BIT2"/>
  </lecture>

  <lecture name="MPR">
    <group type="MIS"/>
    <group type="MMI"/>
  </lecture>

  <lecture ··· >
    ...
  </lecture>
</lectures>


<!-- Seznam faktorů, které se mají plánovat. Každý faktor musí obsahovat pořadí
     termínu (pro rozvrh přednášek vždy hodnota 0), předmět, pro který je faktor
     plánován, maximální odhad počtu studentů a dobu trvání (vyjádřenou počtem
     časových slotů). Uvnitř faktoru se musí nacházet alespoň jedna značka teacher,
     kterou lze faktoru přiřadit vyučující. Volitelně může faktor obsahovat značku
     extra_days, kterou lze zvětšit minimální rozestup mezi termíny (má smysl pouze
     u zkouškového rozvrh – viz kap. 5.1), značku prefer pro vyjádření preferencí
     vyučujících a značku fixed pro fixní přiřazení faktoru ke zdrojům. -->
<factors>
  <!-- Řádný termín zkoušky (nebo přednáška) předmětu IZP. Faktor vyžaduje zdroje
       s učebnami s celkovou kapacitou alespoň 800 studentů po dobu 3 časových
       slotů. K tomuto faktoru jsou přiřazeni dva vyučující. -->
  <factor order="0" lecture="IZP" students="800" duration="3">
    <teacher name="Kreslíková Jitka"/>
    <teacher name="Smrčka Aleš"/>
  </factor>

  <!-- Řádný termín zkoušky předmětu IJC (nemůže se jednat o přednášku, protože
       faktor obsahuje značku extra_days). U faktoru je vyjádřena preference
       vyučujícího na konání zkoušky v prvních čtyřech dnech rozvrhu a požadavek
       na navýšení minimálního rozestupu od následujícího termínu o 5 dnů. -->
  <factor order="0" lecture="IJC" students="256" duration="2">
    <extra_days value="5"/>
    <teacher name="Peringer Petr"/>
    <prefer value="50" from_day="0" to_day="3" from_hour="0" to_hour="9"/>
  </factor>

  <!-- Druhý termín zkoušky s navýšeným minimálním rozestupem o 3 dny. -->
  <factor order="1" lecture="IJC" students="200" duration="2">
    <extra_days value="3"/>
    <teacher name="Peringer Petr"/>
  </factor>
```

```
  <!-- Fixně naplánovaný druhý termín zkoušky z předmětu FYO. Zkouška je přiřazena
       zdrojům s časovými sloty „den 13 hodina 1“ a „den 13 hodina 2“ a učebnami
       D0206 a D0207. Rozsah časových slotů musí odpovídat požadované době trvání. -->
  <factor order="1" lecture="FYO" students="40" duration="2">
    <teacher name="Sedlák Petr"/>
    <fixed day="13" from_hour="1" to_hour="2">
      <room name="D0206"/>
      <room name="D0207"/>
    </fixed>
  </factor>

  <!-- Fixně naplánovaný řádný termín zkoušky z předmětu FLP. Minimální rozestup
       od následujícího termínu je navýšen o 6 dnů. Při fixně naplánovaném termínu
       nedává smysl simultánní použití značky prefer, protože u takovýchto faktorů
       plánovavání vůbec neprobíhá (viz kap. 4.2.1) -->
  <factor order="0" lecture="FLP" students="339" duration="3">
    <extra_days value="6"/>
    <teacher name="Kolář Dušan"/>
    <fixed day="4" from_hour="0" to_hour="2">
      <room name="D105"/>
      <room name="D0207"/>
    </fixed>
  </factor>

  <!-- Druhý termín zkoušky z předmětu FLP vyžadující místnost s laserem. -->
  <factor order="1" lecture="FLP" students="300" duration="3">
    <extra_days value="4"/>
    <teacher name="Kolář Dušan"/>
    <need what="laser"/>
  </factor>

  <!-- Dvě přednášky z jednoho předmětu IDA, které musí být plánovány v rozdílné
       dny. Takovéto faktory musí mít společnou hodnotu atributu lecture. Hodnotu
       atributu order lze v případě přednáškového rozvrhu vynechat – je implicitně 0. -->
  <factor lecture="IDA" students="305" duration="2" another_day="1">
    <teacher name="Hliněná Dana"/>
  </factor>
  <factor lecture="IDA" students="305" duration="2" another_day="1">
    <teacher name="Hliněná Dana"/>
  </factor>

  <factor ··· >
    ...
  </factor>
</factors>
```

```
  <!-- Hodnoty studentských kolizí mezi předměty uvedenými ve značce lectures.
       Pro každou dvojici předmětů A,B jsou uváděny obě dvojice AB a BA, jejichž
       hodnoty musí být shodné. Hodnota kolize předmětu sama se sebou je vždy 0. -->
  <collisions>
    <!-- Kolize předmětu ACH se všemi ostatními předměty. -->
    <row lecture="ACH">
      <cell lecture="ACH" collision="0">
      <cell lecture="AIS" collision="2">
      <cell lecture="BIO" collision="5">
      <cell ··· >
    </row>

    <!-- Kolize předmětu AIS se všemi ostatními předměty. -->
    <row lecture="AIS">
      <cell lecture="ACH" collision="2">
      <cell lecture="AIS" collision="0">
      <cell lecture="BIO" collision="8">
      <cell ··· >
    </row>

    <row ··· >
      ···
    </row>

  </collisions>
</problem>
```