Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studiijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Diplomová práce

Algoritmy pro rozvrhování směn

Bc. Roman Václavík

Vedoucí práce: Ing. Zdeněk Bäumelt

Studijní program: Otevřená informatika, strukturovaný, Navazující magisterský

Obor: Softwarové inženýrství

13. května 2011

Poděkování

Na tomto místě bych rád poděkoval hlavně Ing. Zdeňku Bäumeltovi, vedoucímu této diplomové práce, za jeho čas, ochotu a cenné rady, které mě nasměrovaly k zdárnému dokončení této práce. Rovněž bych chtěl poděkovat své rodině za jejich podporu během celého studia.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 12. 5. 2011 .....

Abstract

This diploma thesis deals with the creation of algorithms for the rostering problems. It is a well known NP-hard problem, where the solution lies in the assignment of all requested shifts to employees with respect to the large variety of restrictions, e.g. an elimination of the unsuitable combinations of the consecutive shifts, balance of the employees' workload, the minimum spacing between consecutive shifts, etc.

An integral part of the work is a review of possible approaches to the rostering problems, from which resulted the following selection and development of algorithms. The standard algorithms like Simulated Annealing, Hill Climbing and Tabu Search were adapted for our problem. Furthermore, A Time Pre-Defined Variable Depth Search and Scatter Search based on the literature review were implemented. Moreover, another method solving this problem called Chain Algorithm was proposed, implemented and compared with all other algorithms on standard benchmark instances. A thorough analysis of the problem and a design of an application for the problem solution preceded the implementation of all algorithms. Consequently, all algorithms were evaluated in the experimental part of the work and summarized in the discussion. The result of this thesis is a system running under Microsoft Excel, allowing one to control and create rosters based on given input data, including a choice of the algorithm and its settings.

Abstrakt

Tato diplomová práce se zabývá tvorbou a porovnáním algoritmů pro rozvrhování směn zaměstnanců. Jedná se o známý NP-obtížný problém, kdy je nutné přiřadit všechny požadované směny zaměstnancům v souladu s velkým množstvím různých omezení, např. eliminace nevhodných kombinací směn, vyváženost pracovního vytížení zaměstnanců, minimální rozestupy mezi směnami následujícími po sobě atd.

Nedílnou součástí práce je rešerše možných přístupů k problému rozvrhování, od které se odvíjel následný výběr a vývoj algoritmů. Pro řešení tohoto problému byly adaptovány známé algoritmy (Hill Climbing, Simulated Annealing a Tabu Search). Dále byly naimplementovány následující algoritmy inspirované literaturou – A Time Pre-defined Variable Depth Search a Scatter Search. V neposlední řadě byl navržen a vyvinut vlastní algoritmus, pojmenovaný jako „Chain Algorithm“, který byl s ostatními algoritmy porovnán na benchmarkových instancích. Po důkladné analýze problému a návrhu aplikace pro řešení problému následuje samotná implementace algoritmů zakončená provedením experimentů a diskuzi nad nimi. Výsledkem je systém běžící v rámci programu Microsoft Excel dovolující uživateli kontrolu a tvorbu rozvrhů na základě jím zadaných vstupních údajů včetně volby rozvrhovacího algoritmu a nastavení jeho parametrů.

Obsah

Úvod **1**

1 Popis práce **3**
1.1 Cíle práce . . . 3
1.2 Popis řešeného problému . . . 3
1.2.1 Rozvrhování . . . 3
1.2.2 Přerozvrhování . . . 5
1.2.3 Tvrdá omezení . . . 6
1.2.4 Měkká omezení . . . 6
1.2.5 Vymezení pojmů . . . 6
1.2.5.1 Scheduling . . . 7
1.2.5.2 Timetabling . . . 7
1.2.5.3 Sequencing . . . 7
1.2.5.4 Rostering . . . 7
1.3 Motivace . . . 8
1.3.1 Příklad z praxe . . . 8
1.4 Katalog požadavků . . . 9
1.4.1 Funkční požadavky . . . 9
1.4.2 Obecné požadavky . . . 10
1.5 Požadavky na software . . . 10
1.6 Požadavky na hardware . . . 11

2 Rešerše **13**
2.1 Vysvětlení pojmů . . . 14
2.1.1 Úplné metody . . . 14
2.1.2 Heuristiky . . . 14
2.1.3 Metaheuristiky . . . 14
2.1.4 Hyperheuristiky . . . 15
2.2 Úplné metody . . . 15
2.2.1 Celočíselné lineární programování . . . 15
2.2.2 Splňování booleovských formulí . . . 15
2.2.3 Programování s omezujícími podmínkami . . . 16
2.3 (Meta)heuristiky . . . 16
2.3.1 Lokální prohledávání . . . 17
2.3.1.1 Simulated Annealing (Simulované žíhání) . . . 17

2.3.1.2 Hill Climbing (Horolezecký algoritmus) . . . 17
2.3.1.3 Tabu Search . . . 17
2.3.2 Evoluční algoritmy . . . 17
2.3.2.1 Genetické algoritmy . . . 17
2.3.2.2 Memetické algoritmy . . . 18
2.3.3 Algoritmy rojové inteligence . . . 18
2.3.3.1 Ant Colony Optimization (Mravenčí kolonie) . . . 18
2.3.3.2 Particle Swarm Optimization (Rojení částic) . . . 19
2.4 Shrnutí . . . 19

3 Analýza a návrh práce **21**
3.1 Analýza . . . 21
3.1.1 Funkcionalita aplikace . . . 21
3.1.2 Technologie pro naprogramování aplikace . . . 22
3.1.2.1 C# . . . 23
3.1.2.2 Visual Basic .NET . . . 24
3.1.2.3 Visual Studio Tools for Office (VSTO) . . . 24
3.1.3 Postupy implementace . . . 25
3.1.3.1 Sekvenční metodiky . . . 26
3.1.3.2 Iterační metodiky . . . 26
3.1.4 Použití knihoven . . . 28
3.2 Návrh . . . 28
3.2.1 Technologie pro naprogramování aplikace . . . 28
3.2.2 Konvence . . . 29
3.2.3 Postup implementace . . . 29
3.2.3.1 Naplánování iterací . . . 30
3.2.4 Použití knihoven . . . 30
3.2.5 Časový odhad . . . 31
3.2.6 Diagram tříd (Class Diagram) . . . 31
3.2.7 Datový model (Data Model) . . . 31
3.2.8 Diagram případů užití (Use Case Diagram) . . . 32
3.2.9 Sekvenční diagram (Sequence Diagram) . . . 32
3.2.10 Grafické uživatelské rozhraní . . . 32

4 Popis řešení **33**
4.1 Omezení kladená na rozvrh . . . 33
4.1.1 Tvrdá omezení . . . 33
4.1.2 Měkká omezení . . . 34
4.1.2.1 Základní verze . . . 35
4.1.2.2 Vzory . . . 36
4.1.3 Tvrdá omezení modelovaná jako měkká . . . 37
4.2 Počáteční rozvrh . . . 38
4.2.1 Celočíselné lineární programování . . . 38
4.2.1.1 Definice problému . . . 38
4.2.1.2 Tvorba matematického modelu . . . 38
4.2.2 Greedy Algorithm (Hladový algoritmus) . . . 40

4.3 Algoritmy pro rozvrhování . . . 41
4.3.1 Hill Climbing . . . 41
4.3.2 Simulated Annealing . . . 42
4.3.3 Tabu Search . . . 44
4.3.4 A Time Pre-defined Variable Depth Search . . . 45
4.3.5 Scatter Search . . . 46
4.3.6 Chain Algorithm . . . 48

5 Experimenty **51**
5.1 Průběh testování . . . 51
5.2 Popis testovaných instancí . . . 51
5.3 Experimenty s lokálními parametry algoritmů . . . 52
5.3.1 Délka bloku směn pro prohazování (HC, SA, TS, CHAIN) . . . 52
5.3.2 Pravděpodobnost přijmutí stejně ohodnoceného řešení (HC, TS, CHAIN) 54
5.3.3 Restarty (HC) . . . 55
5.3.4 Chain Algorithm . . . 58
5.3.5 Simulated Annealing . . . 60
5.3.6 Tabu Search . . . 63
5.3.7 Time Pre-defined Variable Depth Search . . . 64
5.4 Porovnání algoritmů na benchmarkových instancích . . . 65
5.4.1 Diskuze nad dosáhnutými výsledky . . . 67

6 Testování **69**
6.1 Testování funkcionality . . . 69
6.1.1 Jednotkové testy . . . 69
6.1.2 Benchmarky (srovnávací testy) . . . 70
6.2 Testování výkonnosti . . . 70
6.3 Testování uživatelského rozhraní . . . 71
6.3.1 Kognitivní průchod . . . 72
6.3.2 Test použitelnosti . . . 73

Závěr **77**

Literatura **79**

A Seznam použitých zkratek **85**

B Seznam použitého matematického značení **87**

C Pseudokódy **89**

D Nastavení aplikace **95**
D.1 Formát a vzhled . . . 95
D.2 Omezení . . . 95
D.3 Volba algoritmů . . . 96
D.4 Nastavení algoritmů . . . 97

E Tabulkové podklady k experimentům **99**

F Datový model **113**

G UML **115**

H Instalační manuál **119**

I Uživatelský manuál **121**

J Obsah přiloženého CD **127**

Seznam obrázků

1.1 Ukázka rozvrhu 4
1.2 Vztah mezi třídami složitosti 5

2.1 Vztah mezi heuristikou a metaheuristikou 14

3.1 Architektura Application-level add-ins (nalevo) a Document-level customizations (napravo) 25
3.2 Ukázka použití návrhového vzoru Factory 32

4.1 Prohození bloku směn 41
4.2 Chování algoritmu Hill Climbing v lokálním extrému 42
4.3 Možnost pohybu algoritmu Simulated Annealing v závislosti na teplotě 43
4.4 Typický vývoj penalizace v čase u algoritmu Simulated Annealing 44
4.5 Dekompozice algoritmu Scatter Search 47
4.6 Práce s řetězy v Chain Algorithm 48

5.1 Vliv délky bloku směn a směru procházení na penalizaci rozvrhu u algoritmu Simulated Annealing 53
5.2 Vliv délky bloku směn a směru procházení na penalizaci rozvrhu u algoritmu Hill Climbing 53
5.3 Vliv délky bloku směn a směru procházení na penalizaci rozvrhu u algoritmu Tabu Search 54
5.4 Vliv délky bloku směn a směru procházení na penalizaci rozvrhu u Chain Algorithm 54
5.5 Vliv velikosti pravděpodobnosti přijmutí stejně ohodnoceného řešení na penalizaci rozvrhu u algoritmů Hill Climbing (nalevo) a Chain Algorithm(napravo) 55
5.6 Vliv velikosti pravděpodobnosti přijmutí stejně ohodnoceného řešení na penalizaci rozvrhu u algoritmu Tabu Search 55
5.7 Vliv velikosti tolerance pro určování podobných řešení na penalizaci a čas rozvrhu u algoritmu Hill Climbing 56
5.8 Vliv počtu výskytů podobných řešení na penalizaci a čas rozvrhu u algoritmu Hill Climbing 57
5.9 Vliv počtu restartů na penalizaci a čas rozvrhu u algoritmu Hill Climbing . . 57
5.10 Vliv počáteční teploty na penalizaci a čas rozvrhu u Chain Algorithm 58
5.11 Vliv snižování teploty na penalizaci a čas rozvrhu u Chain Algorithm 59

5.12 Srovnání metod pro vyhledávání kolizí vzhledem k penalizaci a času rozvrhu u Chain Algorithm . . . 59
5.13 Srovnání metod pro procházení sousedství vzhledem k penalizaci a času rozvrhu u algoritmu Simulated Annealing . . . 60
5.14 Vliv počáteční teploty na penalizaci rozvrhu u algoritmu Simulated Annealing 61
5.15 Vliv snižování teploty na penalizaci rozvrhu u algoritmu Simulated Annealing 61
5.16 Vliv počtu kroků na penalizaci a čas rozvrhu u algoritmu Simulated Annealing 62
5.17 Srovnání metod pro modelaci tvrdých omezení vzhledem k penalizaci a času rozvrhu u algoritmu Simulated Annealing . . . 62
5.18 Vliv velikosti Tabu Listu na penalizaci a čas rozvrhu u algoritmu Tabu Search 63
5.19 Srovnání metod pro vyhledávání kolizí vzhledem k penalizaci a času rozvrhu u algoritmu Tabu Search . . . 64
5.20 Srovnání variant algoritmu Hill Climbing vzhledem k penalizaci a času rozvrhu u algoritmu TPVDS . . . 65
5.21 Srovnání algoritmů na benchmarkových instancích vzhledem k penalizaci I . . 66
5.22 Srovnání algoritmů na benchmarkových instancích vzhledem k času I 66
5.23 Srovnání algoritmů na benchmarkových instancích vzhledem k penalizaci II . 67
5.24 Srovnání algoritmů na benchmarkových instancích vzhledem k času II 67

6.1 Výstup z Profileru – kritická cesta . . . 71
6.2 Výstup z Profileru – nejvíce pracují funkce . . . 71

F.1 Datový model systému . . . 113

G.1 Případy užití – interakce se systémem . . . 115
G.2 Diagram tříd . . . 116
G.3 Sekvenční diagram – tvorba rozvrhu . . . 117

H.1 Postup instalace – krok 1 . . . 120
H.2 Postup instalace – krok 2 . . . 120
H.3 Postup instalace – krok 3 . . . 120
H.4 Postup instalace – krok 4 . . . 120

I.1 Vzhled aplikace . . . 121
I.2 List se zaměstnanci . . . 122
I.3 List se směnami . . . 122
I.4 List s obdobím pro rozvrhování . . . 122
I.5 List s kvalifikací zaměstnanců . . . 123
I.6 List s měkkými omezeními . . . 123
I.7 List s požadavky na obsazenost směn . . . 125
I.8 List s rozvrhem . . . 125
I.9 Informace o rozvrhu . . . 126

Seznam tabulek

1.1 Porovnání ruční a automatizované tvorby rozvrhu 9
1.2 Požadavky na hardware 11

3.1 Plán iterací 30
3.2 Porovnání uvažovaných ILP řešitelů pro C# 31

4.1 Transformace základních měkkých omezení do vzorů 37
4.2 Mapování jednoduchého příkladu do vektoru 39
4.3 Ukázkový příklad pro vyhledávání kolizí v algoritmu Tabu Search 45

5.1 HW konfigurace použitých strojů 51
5.2 Popis testovaných instancí 52

6.1 Nedostatky v UI objevené během kognitivního průchodu 72
6.2 Odpovědi na závěrečný dotazník 75

B.1 Použité matematické značení 87

E.1 Chain Algorithm – srovnání metod pro vyhledávání kolizí 99
E.2 Tabu Search – srovnání metod pro vyhledávání kolizí 100
E.3 TPVDS – srovnání variant algoritmu Hill Climbing 100
E.4 Simulated Annealing – srovnání metod pro prohledávání sousedních řešení . . 101
E.5 Simulated Annealing – srovnání modelování tvrdých omezení 101
E.6 Simulated Annealing – vliv délky bloku směn pro prohazování v sestupném směru procházení na penalizaci rozvrhu 102
E.7 Simulated Annealing – vliv délky bloku směn pro prohazování ve vzestupném směru procházení na penalizaci rozvrhu 102
E.8 Hill climbing – vliv délky bloku směn pro prohazování v sestupném směru procházení na penalizaci rozvrhu 103
E.9 Hill Climbing – vliv délky bloku směn pro prohazování ve vzestupném směru procházení na penalizaci rozvrhu 103
E.10 Tabu Search – vliv délky bloku směn pro prohazování v sestupném směru procházení na penalizaci rozvrhu 104
E.11 Tabu Search – vliv délky bloku směn pro prohazování ve vzestupném směru procházení na penalizaci rozvrhu 104
E.12 Chain algorithm – vliv délky bloku směn pro prohazování v sestupném směru procházení na penalizaci rozvrhu 105

E.13 Chain Algorithm – vliv délky bloku směn pro prohazování ve vzestupném směru procházení na penalizaci rozvrhu . . . 105
E.14 Hill Climbing – vliv tolerance pro podobná řešení na penalizaci rozvrhu . . . 106
E.15 Hill Climbing – vliv počtu výskytů podobných řešení na penalizaci rozvrhu . 106
E.16 Hill Climbing – vliv počtu restartů na penalizaci rozvrhu . . . 107
E.17 Chain Algorithm – vliv snižování teploty na penalizaci rozvrhu . . . 107
E.18 Chain Algorithm – vliv počáteční teploty na penalizaci rozvrhu . . . 108
E.19 Hill Climbing – vliv velikosti pravděpodobnosti přijmutí stejně ohodnoceného řešení na penalizaci . . . 108
E.20 Chain Algorithm – vliv velikosti pravděpodobnosti přijmutí stejně ohodnoceného řešení na penalizaci . . . 109
E.21 Tabu Search – vliv velikosti pravděpodobnosti přijmutí stejně ohodnoceného řešení na penalizaci . . . 109
E.22 Simulated Annealing – vliv počáteční teploty na penalizaci rozvrhu . . . 110
E.23 Simulated Annealing – vliv snižování teploty na penalizaci rozvrhu . . . 110
E.24 Simulated Annealing – vliv počtu kroků na penalizaci rozvrhu . . . 111
E.25 Tabu Search – vliv velikosti Tabu Listu na penalizaci rozvrhu . . . 111
E.26 Porovnání algoritmů na benchmarkových instancích I . . . 112
E.27 Porovnání algoritmů na benchmarkových instancích II . . . 112

I.1 Ukázka zadávání vzorů . . . 124

Seznam pseudokódů

C.1 GreedyAlgorithm 89
C.2 FindSimilarity 89
C.3 HillClimbing 90
C.4 SimulatedAnnealing – varianta náhodného výběru dnů i zaměstnanců 90
C.5 TabuSearch 91
C.6 CheckAndUpdate 91
C.7 TPVDS – varianta bez časové restrikce 92
C.8 ScatterSearch 92
C.9 SubsetCombination 93
C.10 ChainAlgorithm 93
C.11 UpdateChain 94

Úvod

V dnešní době, kdy jsou počítače prakticky ve všech domácnostech, větších či menších firmách a dosahují dříve nepředstavitelných rychlostí, je vhodné tohoto faktu využít a snažit se odstranit zbytečnou, často velmi časově náročnou, ruční práci pomocí různých programů. Ty totiž nejen že dokážou radikálně ušetřit čas, ale také mnohdy poskytují daleko kvalitnější řešení.

Cílem této práce je návrh a implementace aplikace, která bude schopna zastoupit obtížnou ruční tvorbu rozvrhu zaměstnanců. Tím je přiřazení potřebných směn do daných dnů konkrétním zaměstnancům tak, aby bylo vyhověno různým nárokům (požadavkům) kladeným na výsledný rozvrh. Motivací je tedy úspora času osoby zodpovědné za přípravu rozvrhů, která tímto úkolem často stráví pár dnů v měsíci. To sebou přináší zkvalitnění služeb poskytovaných danou institucí, protože jednak odpovědná osoba bude mít více času na svou hlavní činnost, ale také ostatní zaměstnanci budou mít možnost pracovat mnohem efektivněji díky lepšímu naplánování směn a volných dnů.

Hlavním přínosem práce je implementace a porovnání metod, které tento problém co nejefektivněji řeší. K tomuto účelu bylo vybráno a naimplementováno několik algoritmů včetně jednoho vlastního, které byly také řádně otestovány na benchmarkových instancích.

První kapitola se věnuje popisu řešeného problému včetně souvisejících věcí a nadefinováním všech kladených požadavků na systém, jež budou základním stavebním kamenem pro další kroky v procesu vývoje softwaru. Ve druhé kapitole je představen náhled na možné postupy pro plánování směn zaměstnanců s ohledem na dostupnou literaturu, přičemž dojde ke zhodnocení všech řešení a vytyčení směru, podle kterého budou voleny algoritmy pro rozvrhování v této práci. Analýze a návrhu aplikace se věnuje následující kapitola, která čtenáře seznámí mimo jiné s výběrem technologie pro programování, postupem implementace a několika UML diagramy popisujících strukturu a chování aplikace. Samotný popis implementace jak aplikace, tak i jednotlivých algoritmů řeší další kapitola, jejíž výsledky jsou uvedeny a porovnány v navazující kapitole s experimenty (uvažují se jak lokální parametry, tak i benchmarkové instance). V poslední kapitole je popsáno testování funkčnosti aplikace a uživatelského rozhraní.

Kapitola 1

Popis práce

1.1 Cíle práce

Cílem práce je naimplementovat a porovnat různé algoritmy pro rozvrhování směn zaměstnanců. Tyto algoritmy budou naprogramovány podle následujících zdrojů:

- odborná literatura,
- odborná literatura s vlastní invencí,
- nabyté znalosti a zkušenosti (vlastní algoritmus).

Aby mohly být algoritmy používány, bude pro tyto účely potřeba vytvořit aplikaci. Pro její tvorbu bude použit takový programovací jazyk, který umožňuje efektivně pracovat s tabulkovým procesorem Microsoft Excel (MS Excel) [61], jež je součástí kancelářského balíku Microsoft Office [62]. MS Excel tudíž bude grafickým uživatelským rozhraním (GUI).

Systém bude během implementace průběžně testován. Po dokončení implementačních prací bude otestován ještě jednou a navíc se provedou všechny potřebné srovnávací testy (benchmarky), které určitým způsobem také důkladně testují aplikaci. Aby byla pro uživatele zaručena příjemná a přehledná práce se systémem, budou provedeny rovněž testy uživatelského rozhraní.

Po dokončení výše uvedených činností bude napsána k této práci dokumentace, která bude mapovat vše od zvolení tématu přes naprogramování algoritmů až po diskuzi nad dosaženými výsledky a možným budoucím rozšířením.

Výstupem tedy bude funkční aplikace, umožňující plánování směn zaměstnanců pomocí různých algoritmů, spolu s dokumentací popisující tvorbu této práce.

1.2 Popis řešeného problému

1.2.1 Rozvrhování

Rozvrhování je disciplína, ve které se snažíme přiřadit zdroje a čas k aktivitám takovým způsobem, abychom dosáhli požadovaných cílů a zároveň došlo ke splnění všech nutných podmínek kladených na výsledný rozvrh.

Rozvrh je tedy množina aktivit, kde každá aktivita trvá předem určenou dobu a je vykonávána na nějakém zdroji či jej využívá.

V případě rozvrhování směn zaměstnanců jsou aktivitami směny, jež mají definován začátek, trvání, konec a relace následnosti (definice zakázaných po sobě jdoucích posloupností směn). Zdrojem jsou zaměstnanci, kterým pro každý den přiřazujeme maximálně jednu směnu, na kterou mají potřebnou kvalifikaci.

	1							2						
1993 Listopad	01	02	03	04	05	06	07	08	09	10	11	12	13	14
	Po	Ut	St	Ct	Pa	So	Ne	Po	Ut	St	Ct	Pa	So	Ne
Josef Novak				TD	TD				TD	TD		TN	TN	TN
Marie Nova	TN	TN			TD	TN	TN			TD	TD			
Oldrich Stary	TD	TD	TN			TD	TD			TN	TN			
Karel Dobrota	TD	TD	TD			TN	TN			TN	TN			

Obrázek 1.1: Ukázka rozvrhu

Během postupné alokace aktivit do rozvrhu musíme brát v potaz omezující podmínky. Mohou být různorodého charakteru a mohou se týkat kterékoliv dimenze rozvrhu (zdroje a času) či samotných aktivit. Přitom rozlišujeme tvrdá omezení (hard constraints) a měkká omezení (soft constraints). Detailněji se těmito omezeními zabývají podkapitoly 1.2.3, respektive 1.2.4.

Abychom mohli porovnávat různé rozvrhy mezi sebou, musíme zavést tzv. ohodnocení rozvrhu. Toto vyčíslení většinou představuje splnění či nesplnění preferencí vycházejících z měkkých omezení. Pro výpočet této hodnoty slouží kriteriální (ohodnocovací) funkce. Rozvrhování potom řeší optimalizační úlohu, kde se hledá buď minimální, nebo maximální hodnota kriteriální funkce.

Zásadní úskalí rozvrhování je jeho složitost, která spadá do NP-obtížných problémů. Při navyšujícím se počtu zdrojů a času nastává totiž velká kombinatorická exploze stavového prostoru, který musíme prohledat, abychom nalezli optimální řešení. Tento úkol lze vyřešit za dlouhou dobu nebo se taky vůbec nemusíme výsledku během svého života dočkat. Proto se v praxi většinou vždy spokojíme se sub-optimálním, dostatečně kvalitním, řešením (lokální extrém), o kterém můžeme říci, že v rámci prozkoumané částí stavového prostoru je optimální. Vzhledem k celému stavovému prostoru ovšem nemůžeme vynášet žádné závěry, protože jsme neprošli celý stavový prostor a tudíž nalezené sub-optimální řešení může (ale také nemusí) být optimální (globální extrém).

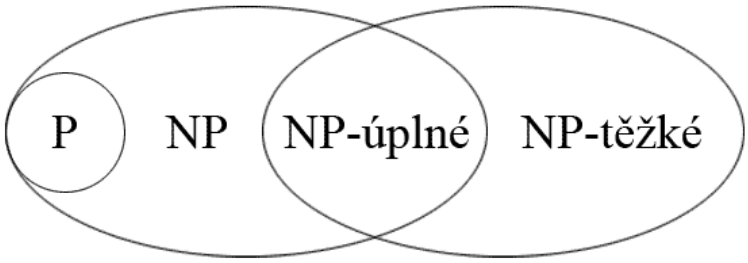
Při tvorbě rozvrhu zaměstnanců tedy jde o nalezení určitého kompromisu mezi dobou běhu algoritmu a jeho kvalitou. V zásadě platí, že nejvýraznější změny kvality k lepšímu nastávají v počáteční fázi algoritmu a postupem času se kvalita zlepšuje už jen lehce. Otázkou je, zda je pro nás důležitější každé malé zlepšení během dlouhého trvání nebo se spokojíme s rozumnou kvalitou (lišící se od dlouhého běhu řádově v jednotkách procent) za poměrně krátkou dobu.

Z hlediska složitosti se jedná o NP-obtížný problém, který bude v následujících odstavcích krátce rozveden. Problémy patřící do třídy NP můžeme rozhodnout pomocí nedeterministického Turingova stroje v polynomiálním čase, což neznamená nic jiného, než že jsme schopni ověřit jejich řešení (nikoliv vyřešit!) v polynomiálním čase.

NP-úplná úloha je taková úloha U, která patří do třídy NP a zároveň se na ni polynomiálně redukuje každá NP úloha. Dá se říci, že NP-úplné úlohy jsou nejtěžší v třídě NP.

O úloze U řekneme, že je polynomiálně redukovatelná na úlohu V, jestliže existuje polynomiální algoritmus, který převede vstupní data úlohy U na vstupní data úlohy V tak, že odpovědi na obě instance budou identické. To, že se U polynomiálně redukuje na V lze chápat tak, že U není obtížnější než V.

NP-obtížná úloha je taková úloha U, na kterou lze polynomiálně redukovat některou z NP-úplných úloh. Z předchozí věty vyplývá, že U je minimálně tak těžká jako všechny NP-úplné úlohy [29, 30].



Obrázek 1.2: Vztah mezi třídami složitosti

1.2.2 Přerozvrhování

Doteď se řešila pouze kompletní tvorba rozvrhu „od podlahy“. Existuje ovšem i tzv. přerozvrhování, kdy už máme nějaký existující rozvrh a je potřeba v něm udělat menší změny. Jenže tyto úpravy nemohou být jen tak ledabylé a musí co nejšetrněji udělat takové obměny, aby vznikl co možná nejkvalitnější nový rozvrh s minimem změn oproti původnímu řešení. Maximální počet těchto úprav bývá obvykle jedním ze vstupů algoritmu [25, 47].

Typickým příkladem, kdy se používá přerozvrhování, může být náhlé onemocnění zaměstnance. V této chvíli je potřeba všechny jeho směny v době nemoci přerozdělit ostatním zaměstnancům takovým způsobem, aby jim to způsobilo co nejmenší změny ve stávajícím rozvrhu a aby byla dosažena co nejlepší kvalita celého rozvrhu.

Při přerozvrhování je potřeba také respektovat zákoník práce §85 (3) [83], který říká: „Zaměstnavatel je povinen vypracovat písemný rozvrh stanovené týdenní pracovní doby a seznámit s ním zaměstnance **nejpozději dva týdny** před začátkem období, na něž je pracovní doba nerovnoměrně rozvržena, pokud se nedohodne se zaměstnancem jinak.“. Pracovní doba (často i uváděno jako vyrovnávací období) uvedená v citaci nemusí být jen týdenní, standardně se používá 4, 26 a 52 týdenní.

Tato práce se nicméně zabývá pouze kompletní tvorbou rozvrhu.

1.2.3 Tvrdá omezení

Tvrdá omezení kladená na rozvrh nesmí být nikdy porušena, přičemž nemají žádný vliv na hodnotu kriteriální funkce. Pokud rozvrh splňuje všechna tvrdá omezení, říkáme o něm, že je proveditelný (feasible). V opačném případě, při porušení byť jen jediného tvrdého omezení na jednom místě, nazýváme rozvrh neproveditelným (infeasible).

Tvrdá omezení mají za cíl stanovit jasná pravidla pro výsledný rozvrh, která musí vždy a za každých okolností platit a neexistuje žádná výjimka.

Konkrétní popis jednotlivých tvrdých omezení je uveden v kapitole 4.1.1.

1.2.4 Měkká omezení

Na rozdíl od tvrdých omezení mohou být měkká omezení porušována. Naší snahou ovšem je docílit co nejmenšího počtu porušení s tím, že je potřeba brát v úvahu i cenu (váhu) porušení. Někdy totiž může nastat situace, kdy bude porušeno pouze jedno omezení s příspěvkem do kriteriální funkce 1000 a na druhé straně 20 porušení s penalizací 900. Čili kvalitnější řešení je to s výrazně vyšším počtem porušení, ale celkovým nižším ohodnocením.

Měkká omezení mají za cíl vyjádřit vážené preference jak ze strany zaměstnanců, tak i zaměstnavatele, potažmo organizace. Z výše uvedeného lze vidět, že nastavit správně váhy všech měkkých omezení není zrovna jednoduché, ale je to stěžejní věc, jak donutit aplikaci vracet nejkvalitnější možný rozvrh.

Čím vyšší váhu dáme nějakému omezení, tím je pro nás důležitější a chceme, aby prakticky vůbec nedošlo k jeho porušení (ale je nutné počítat s tím, že k porušení může dojít). Naopak, pokud váhu nastavíme na minimum, říkáme, že dané omezení je vlastně taková „třešnička na dortu“ (bonus) a pokud dojde k porušení, tak se vůbec nic neděje.

V praxi se používá velké množství měkkých omezení a je samozřejmě skoro nemožné vždy všechny splnit na 100 procent.

Konkrétní popis jednotlivých měkkých omezení a jejich nastavení lze nalézt v kapitole 4.1.2.

1.2.5 Vymezení pojmů

V literatuře se často nerozlišují pojmy jako rozvrhování a plánování (anglické ekvivalenty – scheduling, timetabling, rostering), jelikož význam těchto slov plyne z kontextu a zaměření práce. Abychom ale byli přesní, je potřeba uvést, že rozdíly zde jsou a některá literatura s nimi pracuje [82]. Pro úplnost bude popsána i technika seřazení (sequencing).

Obecná definice (cíle) v nejširším slova smyslu pro všechny pojmy je následující: jedná se o řešení praktických problémů týkajících se přidělování zdrojů objektům vzhledem ke kladeným omezením do časoprostorové matice za použití nebo vyvinutí jakéhokoliv vhodného nástroje. Tyto problémy se často vážou na splnění určitých cílů.

Pokud ovšem podrobíme pojmy restriktivnímu pohledu, dostaneme se k definicím uvedených v následujících podkapitolách.

V této práci budou všechny pojmy chápány jako přiřazování směn do rozvrhu, jehož dimenzemi jsou zaměstnanci (svislá osa) a časová perioda (vodorovná osa).

1.2.5.1 Scheduling

Je přidělování zdrojů objektům vzhledem ke kladeným omezením do časoprostorové matice a to takovým způsobem, aby byly minimalizovány celkové náklady nějaké množiny použitých zdrojů.

Příklady:

- Dopravní plánování – minimalizace dopravních nákladů, vozidel, řidičů atd.
- Plánování výroby – minimalizace časových prodlev, počtu strojů atd.

1.2.5.2 Timetabling

Je přidělování daných zdrojů objektům vzhledem ke kladeným omezením do časoprostorové matice a to takovým způsobem, aby byla co možná nejpřesněji splněna množina požadovaných cílů.

Příklady:

- Školní rozvrh.
- Určité druhy personálního přidělování – obsazení mýtných bran vzhledem k danému počtu personálu.

1.2.5.3 Sequencing

Je konstrukce pořadí, vzhledem ke kladeným omezením, v jakém mají být aktivity prováděny nebo objekty umístěny v nějaké reprezentaci problému.

Příklady:

- Rozvrhování proudové výroby (flow shop scheduling).
- Problém obchodního cestujícího.

1.2.5.4 Rostering

Je umisťování zdrojů vzhledem ke kladeným omezením do slotů ve vzoru. Snahou je buď něco minimalizovat, nebo prostě jen získat proveditelné přidělení. Zdroje se často točí (opakují) v rámci rozvrhu.

Příklad:

- Rozpis sester v nemocničních zařízeních.

1.3 Motivace

Je zcela zřejmé, že ruční plánování směn zaměstnanců je netriviální záležitost (pokud se bavíme o institucích s počtem zaměstnanců větším než menším nebo s velkým počtem omezením) a osobě, která má na starosti tuto práci, to zabere řádově hodiny. Navíc není zaručeno, že výsledný rozvrh bude přípustný (tvrdá omezení – např. malý počet zaměstnanců pro pokrytí všech směn v určitém dni) nebo z hlediska cílové funkce kvalitní (měkká omezení – např. naplánovaná práce v šesti a více po sobě následujících dnech).

Oproti tomu automatizované rozvrhování zvládne vytvořit splnitelný (pokud je to možné, protože uživatel může zadat taková vstupní data, ze kterých nelze za žádných okolností vytvořit splnitelný rozvrh) a co možná nejkvalitnější rozvrh řádově během minut.

Výhody jsou nasnadě:

- redukování manuálních činností (zůstává potřeba zadat vstupní data),
- snížení provozních nákladů,
- zlepšení produktivity práce,
- spravedlivé rozvrhy pro všechny zaměstnance,
- vyšší spokojenost zaměstnanců díky dobrému rozvržení práce a volna.

1.3.1 Příklad z praxe

V jistém zdravotním zařízení pracuje vrchní sestra (dále VS), která má na starosti právě tvorbu rozvrhů pro zdravotní sestřičky (dále ZS) jejího oddělení. Zdravotní zařízení jsou naprosto typické velkým počtem různorodých omezení, takže není snadné vždy plně vyhovět všem stranám.

VS dříve plánovala směny s měsíční periodou pro 20 zaměstnanců ručně, což vyšlo zhruba na 16 hodin/měsíc. Do tohoto času je započítáno jednak samotné vytvoření rozvrhu, tak i postupné vylepšování a předělávání na základě stížností či dodatečných požadavků od ZS. Ve finále se nikdy nepodařilo vyrobit dostatečně kvalitní a spravedlivý rozvrh. Prakticky vždy se našla nějaká ZS, která byla velmi nespokojená.

Od doby, kdy VS dostala k dispozici aplikaci pro generování rozvrhů, se její čas potřebný pro plánování směn zredukoval řádově do desítek minut/měsíc. ZS jsou znatelně spokojenější, oddělení pracuje z hlediska produktivity prokazatelně lépe a opravdu jen sporadicky se musejí řešit nějaké větší problémy s rozvrhem.

Shrnutí tohoto příkladu je uvedeno v tabulce 1.1. Jak je vidět, díky automatizovanému řešení ušetří VS zhruba 9% ze svého měsíčního zatížení (160 hodin), které může věnovat užitečnějším činnostem než je plánování směn [67].

	Ruční tvorba	Automatizovaná tvorba
Potřebný čas [počet hodin]	16	1
Měsíční zátěž [%]	10	0.6
Kvalitní a spravedlivý rozvrh	Těžko docílitelné	Ano

Tabulka 1.1: Porovnání ruční a automatizované tvorby rozvrhu

1.4 Katalog požadavků

Sdružuje požadavky, které určují, co má systém dělat, ale už neříkají, jak toho docílit. Mají za cíl:

- získat přehled o tom, čeho je potřeba dosáhnout,
- zjistit jak a proč uživatelé daný systém potřebují.

Požadavky získáváme od všech zúčastněných stran tzv. stakeholderů (nemusí se jednat pouze o zákazníka/y a lidi v jeho firmě) [81].

1.4.1 Funkční požadavky

Definují, co se musí udělat.

- Systém slouží pro plánování směn zaměstnanců a má tendenci nahradit ruční tvorbu rozvrhů.
- Zadání zaměstnanců (jméno a poznámka).
- Nastavení směn (název, zkratka, začátek a konec směny, trvání směny, barva a poznámky).
- Určení časové periody pro plánování (počet dnů z historie a do budoucnosti, datum začátku a konce periody).
- Správa měkkých omezení (přidat, upravit a vymazat).
- Vyplnění kvalifikací zaměstnanců.
- Definice potřebných počtů směn na každý den z časové periody.
- Možnost zmrazit směny (daná směna se nemůže změnit během tvorby rozvrhu) v rozvrhu.
- Výběr algoritmu pro tvorbu rozvrhu.
- Nastavení parametrů jednotlivých algoritmů.
- Zapnutí či vypnutí tvrdých omezení.

- Aplikace poběží v MS Excel.
- Možnost kontroly zadaného rozvrhu vůči aktuálnímu nastavení systému.
- Informace o přípustnosti rozvrhu.
- V případě přípustného rozvrhu, informace o ohodnocení rozvrhu.
- Anglický jazyk.

1.4.2 Obecné požadavky

Do této kategorie spadají všechny ostatní požadavky (např. výkon, kapacita, dostupnost nebo bezpečnost), které nepatří do požadavků funkčních.

- Systém bude implementován pomocí objektově orientovaného programování (OOP):
  - použití programovacího jazyku C#,
  - použití Visual Studio Tools for Office (VSTO) [65] pro propojení MS Excel s C#.
- Aplikace bude dostupná pouze offline.
- Systém nevyžaduje žádné přihlašování – čili bude dostupný komukoliv.
- Aplikaci bude možno lehce rozšířit o nové algoritmy.
- Přehledné a intuitivní ovládání.

1.5 Požadavky na software

Jelikož je hlavním vstupním požadavkem, aby byl jako GUI použit MS Excel, tak logicky je potřeba mít licenci na kancelářský balík Microsoft Office, který také obsahuje právě MS Excel. Pro účely (VSTO propojení) této práce je vyžadován tento balík minimálně ve verzi 2007.

Tento balík (program) lze spustit pouze pod Microsoft Windows [66], takže volba operačního systému (OS) je také zcela jasná. Nicméně jsou vyžadovány následující verze OS Windows [63]:

- Microsoft Windows XP SP2,
- Windows Server 2003 SP1,
- nebo jakákoliv vyšší verze.

1.6 Požadavky na hardware

Požadavky na hardware pro spuštění tohoto systému jsou na dnešní dobu prakticky zanedbatelné. Samotná aplikace potřebuje pro svůj běh jen počítač (nebo notebook, případně netbook). Uživatelé pro svou práci se systémem potřebují:

- počítač,
- monitor,
- klávesnici,
- myš.

Minimální konfigurace pro počítač, na kterém se spouští aplikace, je mixem dvou faktorů – a to operačního systému [64] a softwaru [63], ve kterém je integrována aplikace. Výsledek kombinace je shrnut do tabulky 1.2.

Komponenta	Minimální požadavek
Procesor	500 MHz
Paměť (RAM)	256 MB
Paměť (HDD)	3,5 GB
Displej	1024x768 px*

* Toto rozlišení je silně doporučováno pro pohodlnou práci s aplikací

Tabulka 1.2: Požadavky na hardware

Je nutno podotknout, že čím vyšší bude takt procesoru, tím bude aplikace schopna pracovat rychleji.

Kapitola 2

Rešerše

V dnešní době lze nalézt opravdu mnoho přístupů, které jdou použít pro tvorbu rozvrhu zaměstnanců. Ostatně o tom svědčí i obrovský počet (řádově stovky až tisíce) literatury a odborných článků.

Literatura je ovšem nejednotná v jakékoliv kategorizaci [19, 39, 75, 24]. Co je v jednom zdroji uváděno pod kategorií X, to je uvedeno v jiné publikaci pod kategorii Y. Určitě je to částečně i proto, že mnoho metod pro rozvrhování využívá různých přístupů najednou. V důsledku toho je pak těžké zvolit ten dominantní (je-li to vůbec možné) a díky němu tuto metodu přiřadit do správné kategorie. Jedno základní rozdělení je ale neoddiskutovatelné, a to: úplné metody vs. (meta)heuristiky. Ostatní rozdělení uvedené v této kapitole je nutno brát s určitou rezervou, protože nelze s jistotou říci, jestli je špatné či správné.

Pro vypracování této rešerše, mimo literaturu uvedenou v podkapitolách, byly použity i zdroje s komplexním obsahem jako sborníky Practice and Theory of Automated Timetabling (PATAT) [9, 14, 13, 10, 11] nebo Handbook of Metaheuristics [42].

V níže uvedené podkapitole dojde ještě k objasnění několika termínů týkajících se kategorizace a pak už bude následovat samotný popis přístupů k rozvrhování podle následujícího členění:

- Úplné metody:
  - Celočíselné lineární programování (ILP),
  - Splňování booleovských formulí (SAT),
  - Programování s omezujícími podmínkami (CP).
- (Meta)heuristiky:
  - Lokální prohledávání:
    - Simulated annealing (SA),
    - Hill Climbing (HC),
    - Tabu Search (TS).
  - Evoluční algoritmy:
    - Genetické algoritmy (GA),

    - Memetické algoritmy (MA).
  - Algoritmy rojové inteligence:
    - Particle Swarm Optimization (PSO),
    - Ant Colony Optimization (ACO).

2.1 Vysvětlení pojmů

2.1.1 Úplné metody

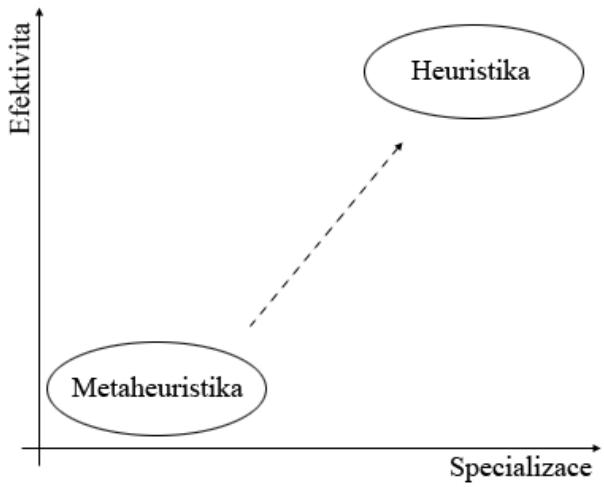
Jak již název napovídá, jedná se o řešení, která procházejí kompletní stavový prostor problému. Takže vždy vrátí nejlepší možný výsledek, pokud vůbec nějaký existuje. Tato výsada je ovšem vykoupena obrovskou časovou náročností (může být klidně desítky let).

2.1.2 Heuristiky

Nevýhodu úplných metod se snaží odstranit heuristické algoritmy, které pracují v rozumných časových intervalech a vrací uspokojivá řešení (nemusí být nutně ta nejlepší, ale cílem je přiblížení se jim). Aby toho dosáhly, tak neprocházejí celý stavový prostor, ale jen jeho část (jakou, to už závisí na přístupu daného algoritmu).

2.1.3 Metaheuristiky

Metaheuristiky jsou v zásadě také heuristiky, ale mají obecné využití na rozdíl od heuristik, které jsou určeny (dají se aplikovat) pro konkrétní záležitosti. Aplikují se na místech, kde není žádný uspokojivý algoritmus či heuristika. Často se z metaheuristiky stává heuristika tím, že se ji snažíme co nejlépe nastavit pro řešený problém, čímž vzniká specializovaný algoritmus (heuristika) [75, 49, 3].



Obrázek 2.1: Vztah mezi heuristikou a metaheuristikou

2.1.4 Hyperheuristiky

Cílem hyperheuristik je automatizace procesu výběru a kombinace několika různých jednoduchých heuristik tak, aby byl co nejefektivněji vyřešen zadaný úkol.

Oproti metaheuristikám, které prohledávají stavový prostor všech možných řešení daného problému, hyperheuristiky prohledávají stavový prostor heuristik. To znamená, že je zde snaha najít správnou sekvenci heuristik namísto přímého řešení problému [18].

2.2 Úplné metody

2.2.1 Celočíselné lineární programování

Vytvoření rozvrhu čistě jen pomocí ILP (Integer Linear Programming) lze efektivně využít pouze za předpokladu, že máme malou instanci vstupních dat (už při desítkách zaměstnanců a jednotkách směn nastávají komplikace, jak je ukázáno v [80]). Použití ILP na problémy s malými instancemi je popsáno v článku L. Trillinga a kolektivu [78], kde bylo ILP aplikováno na instanci 16 zaměstnanců, 4 směn a 7 dnů s minimem základních omezení. Výpočetní časy pro takovéto instance byly kolem 3 minut. Pro instance odpovídající reálným rozvrhům roste výstupní čas exponenciálně s rostoucí velikostí instance, tj. řešení je nalezeno ve výpočetních časech, které jsou pro reálnou aplikaci nepoužitelné.

Z výše zmíněných důvodů je vhodnější (pokud je to možné) použít metodu ILP nikoliv samostatně, ale v kombinaci s nějakým jiným přístupem. Shane G. Henderson a Andrew J. Mason se ve svém článku [44] zabývají iteračním algoritmem využívající ILP. Iterace probíhá v zásadě v tomto cyklu – ILP, simulace (vyzkoušení dosažených výsledků), řez (zmenšení horních a dolních hranic pro řešení problému). Ukončovací podmínkou je rozdíl horní a dolní hranice, kdy se zjišťuje, zda spadá do určité předem definované tolerance. Autoři sice píší, že je v algoritmu hodně míst, kde dochází k malému zlepšení (což ve výsledku znamená relativně znatelné zlepšení), nicméně toto tvrzení není doloženo aplikací jejich heuristiky na známé instance.

2.2.2 Splňování booleovských formulí

SAT (Boolean satisfiability problem) techniky se využívají v ILP, kdy dochází k značnému zrychlení. Využívá se jednak CNF (conjunctive normal form) formulí, tak i PB (pseudo-boolean) omezení. K řešení těchto problémů existuje nespočet volně dostupných SAT-řešitelů (např. PBS [4], SAT4j [6], MiniSat [38], PicoSAT [7], RSat [71]), což nám dává výhodu v tom, že jenom připravíme data v požadovaném formátu a SAT-řešitel se postará o zbytek. Využití SAT v ILP nám umožňuje o trochu zvýšit velikost vstupních dat oproti samotnému ILP (řekněme tedy, že lze využít pro malé firmy).

Fadi Aloul a kolektiv ukazují na příkladu malé firmy [5] tvorbu vstupních dat pro jejich vlastní SAT-řešitel PBS. V experimentech, které provedli, se jen potvrzuje, že vstupní data nemohou být moc velká, jinak bychom se nedočkali řešení v rozumné době. PBS v porovnání s CPLEX (komerční univerzální ILP řešitel od IBM) [45] dosahuje lepších výsledků (nutno

dodat, že je to zřejmě způsobeno univerzálností CPLEX, respektive specializací PBS). Problémem PBS je, že vykonávání algoritmu neskončí (přesněji řečeno v daném časovém okně neskončí), pokud řešíme nesplnitelnou úlohu.

Jako lepší možnost výběru SAT-řešitele oproti PBS se jeví např. SAT4j, který dokáže rozpoznat i nesplnitelné úlohy. Navíc výhodou je jeho integrace s vývojovou platformou JAVA (PBS pracuje přes příkazovou řádku).

2.2.3 Programování s omezujícími podmínkami

Rovněž patří do skupiny algoritmů úplných, tj. takových, které využívají systematického prohledávání a najdou řešení vždy, když existuje. Sophie Demassey a další popisují využití lineárního programování v CP (Constraint Programming) [28], které se používá k přiřazení směnového rozvrhu k zaměstnancům. Další modifikací je sloupcové generování dat, kde jenom nejlevnější rozvrhy jsou iterativně generovány CP.

R. Cipriano a kolektiv zase využívají CP v kombinaci s lokálním prohledáváním [26]. Autoři zvolili dva přístupy. V prvnímm nechali vygenerovat počáteční proveditelný rozvrh pomocí CP a následné zlepšování probíhalo pomocí lokálního prohledávání. Ve druhém případě vytvořili lokální vyhledávací algoritmus, který využívá CP model pro průzkum nejbližšího okolí. Jasně lepší se ukázala první možnost.

2.3 (Meta)heuristiky

Jedná se o skupinu algoritmů, která je založena na náhodném prohledávání stavového prostoru. Může dojít k situaci, že daná vstupní instance má řešení, nicméně náš algoritmus jej nemusí najít. Na druhou stranu, je schopný nalézt nějaké (i neúplné) řešení ve velmi krátkém čase. Tyto algoritmy pracují v zásadě na stejném principu a tím je neustálé zlepšování stávajícího řešení (dochází ke zlepšení ohodnocení všech proměnných zvolením nějakého sousedního ohodnocení).

Tyto algoritmy jsou vhodné pro větší instance vstupních dat. Dokážou se vyrovnat jak se slabým omezením (existuje mnoho řešení, heuristické algoritmy snadno najdou libovolné jedno řešení), tak i s přílišným omezením problému (neexistuje řešení, které by splnilo všechny podmínky, heuristické algoritmy naleznou neúplné (sub-optimální) řešení s minimalizací počtu nesplněných podmínek).

Edmund Burke a kolektiv seznamují s technikami pro volbu sousedů [17] a jejich vzájemného prohazování. Z pozorování vyplývá, že je vhodné nejprve prohazovat malé sousední bloky a teprve pak ty větší. Změnami ve volbě sousedů v průběhu algoritmu můžeme nacházet skrytá nenalezená řešení, které bychom nenašli při použití jednoduché heuristiky.

Na předchozí článek navazuje další [21], který využívá již zjištěných poznatků s tím, že je pár informací doplněno a hlavně je zde nastíněna technická stránka problému v podobě pseudokódu heuristického algoritmu.

2.3.1 Lokální prohledávání

2.3.1.1 Simulated Annealing (Simulované žíhání)

Tato metoda je založena na principu simulace žíhání oceli (materiál je během ochlazování střídavě zahříván a ochlazován). Algoritmus náhodně vybírá sousední rozvrhy. Pokud má sousední rozvrh lepší výsledek objektivní funkce, tak je vybrán. Pokud ho nemá, může být taky s určitou pravděpodobností vybrán. Pravděpodobnost je tím vyšší, čím vyšší je teplota (tu algoritmus postupně snižuje na základě rozvrhu chladnutí). Tato metoda napomáhá úniku z lokálního extrému (díky provádění velkých změn na začátku → čím pomaleji algoritmus konverguje, tím se pomaleji snižuje teplota a tím pádem máme větší šanci dostat se z případného lokálního extrému) [74, 24].

2.3.1.2 Hill Climbing (Horolezecký algoritmus)

Jedná se o gradientní metodu postupně zlepšující (na základě objektivní funkce) ohodnocení všech proměnných. K aktuálnímu ohodnocení prochází pouze jeho sousední množinu. Skončí, jakmile není možné nalézt lepší ohodnocení nebo byl překročen maximální počet iterací. Může lehce uváznout v lokálním extrému, proto je vhodné jej spouštět několikrát z různých míst stavového prostoru. Čímž sice nezaručíme, že opět neuvázne, ale máme alespoň na výběr z více lokálních extrémů a můžeme vybrat ten nejlepší [12, 26].

2.3.1.3 Tabu Search

Metoda používá tzv. Tabu List, který je definován jako seznam dvojic (proměnná, hodnota) o pevné délce n. Daná dvojice vyjadřuje určitou změnu ohodnocení. Pokud chceme změnit hodnotu nějaké proměnné, která už je v seznamu, tak je nám to zakázáno. V praxi se můžeme setkat s použitím aspiračního kritéria, které umožňuje za určitých podmínek tento zákaz ignorovat. Použití TS (Tabu List) umožňuje algoritmu dostat se z lokálního extrému, čímž docílíme efektivnějšího (flexibilnějšího) prohledávání [8, 16, 34].

2.3.2 Evoluční algoritmy

2.3.2.1 Genetické algoritmy

Jsou založeny na Darwinově teorii evoluce. Na začátku máme nějakou počáteční populaci – většinou se jedná o náhodně vygenerovanou sadu jedinců. Jedince ohodnotíme na základě nějaké objektivní funkce. Na základě této hodnoty vybereme nejlepší jedince, na které aplikujeme jednu z následujících metod:

- Křížení – prohodí části jedinců mezi sebou.
- Mutace – náhodně změní část jedince.
- Reprodukce – zkopíruje jedince beze změny do další generace.

Aplikací těchto metod vzniká nová populace (starou můžeme již dále vynechat nebo naopak zachovat) a takto pokračujeme tak dlouho, dokud nedosáhneme určitého stupně generace nebo nenajdeme jedince s vyhovující hodnotou objektivní funkce [2, 68].

D. Abramson a J. Abela se zabývají zrychlením podle nich pomalého genetického algoritmu pomocí paralelizace na více procesorech [1]. Nutno říci, že se jim to povedlo. Při použití dvou procesorů byl výsledný čas dvakrát rychlejší než na procesoru jednom. Nicméně na 5, 10 a 15 procesorech už daná úměra neplatí, ale časy jsou stejně pořad dobré – konkrétně 4x, 8x a 10x rychlejší než na jednom procesoru.

L. Han a další popisují využití TS pro zrychlení genetického algoritmu [43]. Zjistili, že výběr správné kombinace dobrých/špatných jedinců je časově náročné. Proto pro tento úkol zvolili TS, který je znatelně rychlejší. Ten funguje tak, že penalizuje jedince, kteří nemění hodnotu objektivní funkce na n následujících generací. Přičemž z experimentů odvodili, že nejlepších výsledků lze dosáhnout při zvolení n mezi 3 a 6.

2.3.2.2 Memetické algoritmy

„Problémem“ evolučních algoritmů je jejich vyšší časová náročnost, která je způsobena vysokou odolností proti uváznutí v lokálním extrému, protože používají velkou populaci a tím pádem procházejí různé kouty stavového prostoru.

Od genetických algoritmů se liší používáním tzv. memetů (skupina informací) na rozdíl od genů. Pokud gen přejde na potomka, tak nemůže být změněn. Naproti tomu memety si každý potomek může měnit libovolně, třeba pomocí lokálního prohledávání, čímž se může daný potomek přetransformovat na lokálně nejlepší řešení [15]. Porovnáním genetických a memetických algoritmů se zabývá řada článků [31, 73] a prakticky vždy vychází lépe algoritmy memetické.

MA jsou tedy jakýmsi hybridem algoritmů evolučních a lokálního prohledávání. Dochází zde ke kombinaci největších předností obou přístupů: ochrana proti uváznutí v lokálním extrému na jedné straně, rychlost na druhé.

2.3.3 Algoritmy rojové inteligence

Jedná se o techniky inspirované chováním přírodních systémů a evolučními algoritmy. Základem je velké množství agentů působících lokálně, kteří spolu mohou komunikovat přímo i nepřímo a dohromady utvářejí globální řešení problému.

2.3.3.1 Ant Colony Optimization (Mravenčí kolonie)

Tento algoritmus vychází z chování mravenců při hledání potravy (nejkratší cesta z hnízda k jídlu). Je zde značný počet mravenců (agentů), kteří ze začátku náhodně zkoumají terén až do nalezení potravy. Cestou zpět do hnízda za sebou zanechávají feromonovou stopu. Pokud k ní dojdou i jiní mravenci a budou v ní pokračovat, tak jí zesílí svými feromony. Po určitém čase bez obnovy dochází k vyprchání feromonů. Proto když dojde zásoba jídla, feromony se vypaří (mravenci tam už nechodí) a dojde znovu k náhodnému hledání potravy v okolí.

ACO má nízkou režii na komunikaci, je relativně rychlý a především vhodný pro paralelizaci [74, 33, 32].

2.3.3.2 Particle Swarm Optimization (Rojení částic)

PSO je pro změnu inspirován hejnem ptáků, kteří se snaží usednout na nejvyšší kopec v krajině. Ptáci (agenti) zprvu logicky neznají neznámé prostředí a tak jej náhodně prohledávají s libovolnou rychlostí. Každý jednotlivec si pamatuje místo, které si sám našel jako nejvyšší a navíc ví i o místech, které našli okolní ptáci (v některých implementacích se využívá i znalost všech ptáků). V důsledku těchto zjištění pak ptáci létají určitou rychlostí mezi nalezenými nejlepšími místy a při těchto přeletech zároveň hledají nové potencionálně nejvyšší kopce [35, 72].

2.4 Shrnutí

Úplné algoritmy jsou vhodné pro popis a modelování problémů. Jejich řešení je možné dostat pouze pro menší instance. Pro větší, komplexnější problémy je nutné použít (meta)heuristických postupů.

Zde máme na výběr celou řadu možností. Dá se říci, že výběrem jakékoliv neuděláme chybu. Také samozřejmě velmi závisí na vlastní modifikaci základních algoritmů a hlavně na druhu instance. Nicméně se setkáváme s pozorováním, že přeci jen lehce navrch mají algoritmy na bázi lokálního prohledávání. Ty také porovnává s GA E. Burke a kolektiv v [8] a [22], kde v prvním případě je výsledek velmi těsný, nicméně vítězem je lokální prohledávání. V druhém případě už jasně vítězí, a to dokonce i nad komerčním systémem Harmony. Také článek od O. Doria [74] porovnává lokální heuristiky, GA a navíc i ACO. Zde jsou na tom také lépe metody lokálního prohledávání s tím, že ACO dosahuje podobných výsledků. E. Elbeltagi [37] pro změnu srovnává pět metod ze skupiny evolučních algoritmů (v tomto článku jsou do této kategorie zahrnuty i algoritmy rojové inteligence), kde zcela propadly GA následované se značným odstupem MA a naopak zazářilo PSO. ACO a SFL (Shuffled Frog Leaping) skončilo někde mezi MA a PSO.

Následující práce se tedy bude ubírat především směrem k lokálnímu prohledávání, což vyžaduje minimálně implementaci HC, TS, SA + jejich modifikací. Hlavním zdrojem pro další algoritmus Time Pre-defined Variable Depth Search bude článek [21] od E. Burkeho, kde je nastíněn pseudokód, který se pokusím naprogramovat a případně jej vylepšit, pokud je to vůbec ještě možné. Dále bude ještě zhotoven Scatter Search algoritmus [20], který byl vybrán z toho důvodu, že je určitým způsobem podobný evolučním algoritmům. A nakonec se budu snažit vymyslet a naimplementovat vlastní algoritmus na základě nabytých znalostí a zkušeností.

Kapitola 3

Analýza a návrh práce

3.1 Analýza

3.1.1 Funkcionalita aplikace

Funkcionalita systému poměrně jasně vyplývá z katalogu požadavků uvedeného v kapitole 1.4. V následujících odstavcích dojde k jejímu krátkému zanalyzování.

Cílem aplikace je výrazně usnadnit a zlepšit vytváření směnných rozvrhů pro zaměstnance tak, že budou kladeny nízké časové nároky na osobu odpovědnou za generování rozvrhů a jejím výstupem bude co možná nejvíce kvalitní a spravedlivý rozvrh. Výhody a teoretické úspory času zaměstnance při využití rozvrhovacího systému oproti ručnímu tvoření jsou uvedeny v kapitole 1.3.

Práce uživatele s aplikací bude rozdělena na tři fáze, které by měly následovat v uvedeném pořadí za sebou:

- Nastavení všech dimenzí rozvrhu:
  - Zaměstnanci – vyplňovat se bude pouze jméno, příjmení a případná poznámka.
  - Směny – bude se zadávat název, zkratka, čas začátku a konce, doba trvání, barva a poznámka. Délka směny bude muset být nastavena ručně, protože ne vždy je žádoucí operovat pouze s rozdílem mezi jejím začátkem a koncem.
  - Datum – bude potřeba určit interval (začátek a konec), pro který bude tvořen výsledný rozvrh. Navíc bude možno přidat x dnů historie a y budoucích dnů. Tato možnost slouží jak k získání větší jistoty vytvoření proveditelného rozvrhu pro nadcházející období, tak pro lepší návaznost s rozvrhem z období minulého.
- Nastavení dat pro generování rozvrhu:
  - Kvalifikace zaměstnanců – bude nutné přiřadit správné příznaky (ano/ne) k zaměstnancům v závislosti na tom, zda mohou (či nemohou) vykonávat danou směnu.
  - Požadavky na obsazenost směn – zaměstnavatel (uživatel) určí, kolik lidí bude potřeba každý den na všechny uvedené směny.

  - Měkká omezení – zadání preferencí firmy a zaměstnanců (na základě těchto údajů se bude vypočítávat kvalita rozvrhu a cílem bude minimalizovat počet porušení těchto omezení).
  - Specifikace algoritmů – výběr heuristik a jejich nastavení pro vytvoření rozvrhu.

- Práce s rozvrhem (po vyplnění výše zmíněných informací):
  - Kontrola rozvrhu – s rozvrhem bude možné ručně manipulovat a vyvolat kontrolu přípustnosti a ohodnocení rozvrhu.
  - Tvorba rozvrhu – před vytvořením finálního rozvrhu jej bude možné „vylepšit“ zmraženými směnami (tj. takovými, se kterými nemůže algoritmus za žádných okolností hýbat). Po stisknutí patřičného tlačítka dojde ke zpracování a následnému vykreslení rozvrhu.

3.1.2 Technologie pro naprogramování aplikace

Ještě před zahájením implementační práce je vhodné se zamyslet nad volbou programovacího jazyka. Výběr nesprávné technologie může totiž vést ke značnému nárůstu potřebného času pro implementaci oproti předem danému časovému odhadu nebo až k fatálním důsledkům v podobě nedokončení projektu z důvodu nemožnosti provést potřebný úkol. Naopak vybrání vhodné technologie může přispět k velkému zjednodušení práce a úspoře času.

Aby byl proveden správný výběr, je potřeba položit si několik otázek, které jsou nápomocny pro srovnání a zvolení nejlepší technologie. Nejdůležitější otázky jsou (samozřejmě jich existuje více a je možné je také použít, nicméně minimálně následující uvedené by měly zaznít vždy):

- Jaké jsou nároky na hardware, respektive kde všude má aplikace běžet?
  - Viz. kapitola 1.6.
- Má být aplikace platformě nezávislá? Pokud ne, na jakém operačním systému má fungovat?
  - Viz. kapitola 1.5.
- Jaká je vyžadovaná rychlost odezvy systému?
  - Jelikož se jedná o aplikaci, která řeší NP problémy a je hodně závislá na vstupní instanci, nelze přesně definovat čas odezvy. Někdy to mohou být sekundy, jindy zase hodiny. Závisí také na nastavení algoritmů, které má uživatel možnost měnit. Snahou je přednastavit takové hodnoty, aby se čas pro různé instance pohyboval maximálně v hodinách.
- Je potřeba řešit vysoké zabezpečení aplikace?
  - Není.

- Má být systém provozován jako webová aplikace, případně vyžaduje pro svou činnost připojení k internetu?
  - Ne a nevyžaduje.
- Lze využít i placených technologií nebo jen neplacených?
  - Pouze neplacených.
- Je daná technologie dobře zdokumentovaná? Existují k ní příklady použití? Má rozumnou uživatelskou podporu?
  - Všechny uvažované programovací jazyky dávají kladnou odpověď na položené otázky s tím, že žádný není výrazně lepší než ostatní.
- Pokud se budou využívat alespoň dvě technologie, jsou vzájemně kompatibilní (existují možnosti komunikace a předávání potřebných dat mezi nimi)?
  - Viz. kapitoly 3.1.4 a 3.2.4.

Jelikož jedním z hlavních požadavků je, aby byla aplikace integrována do MS Excel, tak se výběr poměrně hodně zúžil, a to jen na dva kandidáty:

- C#,
- Visual Basic .NET.

Problémem ostatních jazyků je nemožnost přímé integrace (práce) s MS Excel. Přesněji řečeno umí „pouze“ operace čtení a zápisu.

3.1.2.1 C#

C# je vysoko úrovňový objektově orientovaný programovací jazyk, který vyvinula firma Microsoft společně s .NET Framework a později jej organizace ECMA a ISO schválily jako standard. Je založen převážně na jazycích C++ a Java. Je možné jej uplatnit při vývoji prakticky čehokoliv od konzolových aplikací přes aplikace mobilní až po systémy na bázi cloud technologií.

Od svého počátku prošel určitým vývojem, kdy současná verze nese značení C# 4.0. Výčet všech verzí a jejich hlavních „přídavků“ oproti verzím minulým je následující:

- C# 1.0 – první verze.
- C# 2.0 – parametrizované typy, částečné třídy, anonymní metody, iterátory, různé typy přístupu k proměnným pro čtení a zápis, delegáti, hodnota *null* jako datový typ.
- C# 3.0 – LINQ, lambda výrazy, inicializátory objektů a kolekcí, anonymní typy, výrazové stromy, rozšiřující metody.
- C# 4.0 – pojmenované a volitelné argumenty funkcí a metod, dynamické vazby.

Výhodou je jednoduchost jazyka, široká komunita a dobrá dokumentace včetně mnoha ukázkových zdrojových kódů. Za mírnou nevýhodu může být považovaná velká závislost na .NET Framework (stejně jako u Visual Basic .NET), protože pokud je potřeba udělat něco, co se v něm nenachází, tak to není zrovna jednoduchá implementační záležitost. Na druhou stranu v dnešní době (kdy už je několikátá verze .NET Framework) lze narazit na tento problém jen minimálně [52, 53].

3.1.2.2 Visual Basic .NET

Visual Basic .NET (VB.NET) je objektově orientovaný programovací jazyk vyvinutý firmou Microsoft běžící na platformě .NET Framework. Jedná se o novou generaci událostmi řízeného jazyka Visual Basic (VB). Výhody VB.NET oproti VB jsou nasnadě, protože vycházejí z použití .NET Framework – nové možnosti jazyka, rychlejší kód (dochází k optimalizaci strojového kódu pro procesor počítače, na kterém se má spustit vybraná aplikace), OOP, webové a mobilní aplikace.

Tak jako C# i VB.NET si prošel svým vývojem (aktuální verze VB 10.0):

- Visual Basic .NET 2003 (VB 7.1) – první použitelná verze (před ní bylo VB 7.0, které bylo ovšem poměrně pomalé).
- Visual Basic 2005 (VB 8.0) – vazby na zdroje dat, částečné třídy, hodnota *null* jako datový typ, bezznaménková celá čísla.
- Visual Basic 2008 (VB 9.0) – XML literály, kladný (true) podmínkový operátor, LINQ, lambda výrazy, anonymní typy, rozšiřující metody.
- Visual Basic 2010 (VB 10.0) – přiblížení k C#, vylepšený kompilátor.

Výhodou je použití mnoha klíčových slov, takže pro programátora (začátečníka či zkušeného, ale neznalého syntaxe jazyka) je poměrně snadné do VB i VB.NET proniknout za předpokladu, že umí anglicky. Nevýhoda je závislost na operačním systému Windows (výjimkou je Linux, který ovšem musí obsahovat framework Mono [48] a zároveň se musí jednat o .NET 2.0 aplikaci) [57, 58].

3.1.2.3 Visual Studio Tools for Office (VSTO)

O propojení MS Excel s implementovanou aplikací se stará Visual Studio Tools for Office (VSTO) [23]. Jiná lepší možnost momentálně neexistuje [60].

Systém je možné implementovat dvěma způsoby:

- úpravy na úrovni dokumentu (Document-level customizations),
- doplňky na úrovni aplikace (Application-level add-ins).

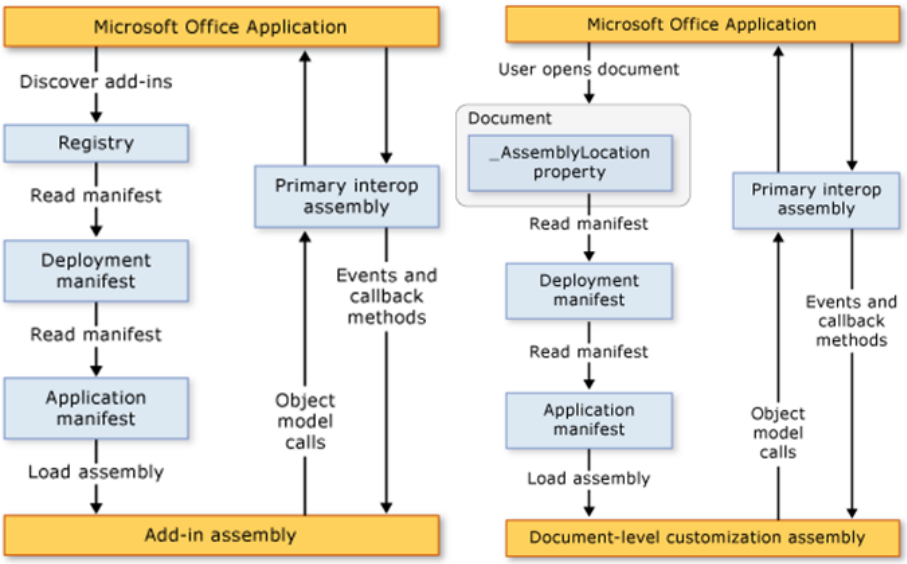
Vyjma níže uvedeného rozdílu se tyto dva přístupy v zásadě ničím velkým neodlišují [54, 55]. Obrázek 3.1 dokumentující jejich architekturu je převzat z [50] respektive [51].

Document-level customizations

Jedná se o seskupení (assembly), které je přiřazeno k právě jednomu dokumentu MS Word nebo sešitu MS Excel. K jeho načtení dojde, jakmile se otevře daný dokument. Všechny jeho možnosti (funkce) je tedy možné využívat pouze v tomto asociovaném dokumentu (neovlivní a neprojeví se v žádném jiném dokumentu). Při zavření patřičného dokumentu se zároveň provede jeho odstranění z paměti.

Application-level add-ins

Jedná se o seskupení, které je přiřazeno k nějaké aplikaci Microsoft Office. K jeho načtení může dojít, jakmile se otevře daná aplikace nebo ho uživatel může ručně spustit sám (stálé ale platí, že aplikace musí být spuštěna). Všechny jeho možnosti (funkce) je tedy možné využívat v asociované aplikaci nezávisle na právě otevřeném dokumentu. Při zavření patřičné aplikace se zároveň provede jeho odstranění z paměti.



Obrázek 3.1: Architektura Application-level add-ins (nalevo) a Document-level customizations (napravo)

3.1.3 Postupy implementace

Metodologií vývoje softwaru se zabývá disciplína softwarové inženýrství. Zde, na rozdíl od kategorizace rozvrhování, je členění jasně dané:

- sekvenční metodiky,
- iterační metodiky:
  - tradiční,
  - agilní.

Ještě před jejich krátkým popisem je potřeba vyjasnit si rozdíl mezi modelem a metodikou. Model popisuje životní cyklus (fáze) softwarového produktu. Metodika na druhou stranu určuje, co se má v daných fázích dělat, jaké postupy se mají aplikovat a co bude výsledkem.

Pro studium této problematiky byla hlavním zdrojem kniha Agilní programování od V. Kadlece [46].

3.1.3.1 Sekvenční metodiky

Tento přístup postupuje dle předem známého sekvenčního schématu takovým způsobem, že je nutné dodržet jeho posloupnost bez jakéhokoliv přeskakování. Schéma je obvykle následující:

1. sběr požadavků,
2. analýza,
3. návrh,
4. implementace,
5. testování,
6. nasazení + případná údržba.

Vodopádový model je bezesporu typickým příkladem a je považován za prvního průkopníka ve svém oboru. Jméno získal díky analogii ke skutečnému vodopádu, kdy dochází k toku vody shora dolů a jinak tomu být nemůže. Postup do další fáze je možný, pouze pokud je současná fáze splněna na 100% a jsou schváleny veškeré potřebné dokumenty. Časem vznikly různé modifikace, které dovolují vrátit se zpět, ale pouze o jeden krok. Nevýhoda je zřejmá, a to prakticky žádná flexibilita při vývoji. Jak se projekt na začátku nastaví, tak musí i skončit, přičemž zákazník vidí hotový produkt až na úplném konci.

3.1.3.2 Iterační metodiky

Jako reakce na nevýhody sekvenčních metodik vznikly metodiky iterační. Jsou navrhnuty tak, aby byly schopné se co nejlépe přizpůsobovat různým změnám v průběhu vývoje. Probíhají tedy v rámci několika iterací, kde v každé z nich zpravidla dochází k průchodu schématem – analýza → návrh → implementace → testování.

Iterativní vývoj je vlastně důsledkem toho, že u většiny projektů není možné na úplném začátku stanovit veškeré požadavky a jsou upřesňovány až během vývoje. Navíc zákazník může vidět postupný vývoj a vyzkoušet si práci s polohotovým produktem. Pokud uvidí, že se vývoj ubírá špatným směrem, má možnost do něj zasáhnout a vrátit jej do správných (požadovaných) mezí. Pro zhotovitele projektu je zase výhodou to, že po každé iteraci dojde k určité analýze pokroku a směru vývoje a je případně možné projekt včas ukončit tak, aby nenastaly fatální důsledky.

Tradiční

Jsou na pomezí mezi sekvenčními a iteračními metodiky. Stejně jako u sekvenčních metodik je fixní položkou funkcionalita a čas s financemi jsou proměnnými položkami. Což

znamená, že je kladen velký důraz na předem dohodnutou funkcionalitu, která musí být v dané míře naplněna, přičemž spotřebovaný čas a peníze mohou růst do nezvratných výšin.

Příkladem budiž spirálový model, který odstraňuje nedostatky vodopádového modelu přidáním iterativního vývoje. Neméně důležitou vlastností je podrobná analýza rizik v každé z iterací, díky níž dochází ke značnému snížení pravděpodobnosti neúspěchu.
Iterace obsahuje v zásadě čtyři kroky:

1. Analýza – stanovení cílů a rozsahu iterace.
2. Zhodnocení – analýza rizik, tvorba prototypů.
3. Vývojový cyklus – vodopádový model, čili od podrobného návrhu až po předání.
4. Plánování – co se bude dělat v další iteraci.

Nevýhodou je předvedení aplikace zákazníkovi až na úplném konci. Během iterací sice dochází k uvolňování prototypů, ale ty se mohou dotýkat jen malých částí systému a nemusí být použitelné pro ostrou (ukázkovou) produkci.

Agilní

Na rozdíl od sekvenčních a tradičních iteračních metodik fixuje položky času s financemi a funkcionalita zde zůstává jako položka proměnná. V praxi to znamená, že zákazník může dostat v dohodnutém čase za smluvené peníze nedokončené dílo, ve kterém ještě nejsou implementovány funkce, kterým byla stanovena společně se zákazníkem nejnižší priorita.

Myšlenkou agilního přístupu je ověřování zhotovované aplikace pomocí průběžné zpětné vazby zákazníka. Cílem tedy je co nejvíce zrychlit vývoj a hlavně pravidelně dodávat systém v krátkých intervalech s částečnou funkcionalitou zákazníkovi k oponování.
Mezi hlavní zástupce patří:

- Extrémní programování (Extreme Programming – XP) – používá se hlavně tam, kde je limitován čas či finance, zákazník nemá moc představu o tom, co vlastně chce, je předpoklad změny systému během určité doby. Naopak se XP nehodí, pokud je tým vývojářů složen z mnoha lidí (>15), akce překlad → linkování → spuštění trvá řádově minuty a více nebo není možná dostatečná konzultace se zákazníkem.
- SCRUM – využívá se tzv. sprintů (jednotlivé iterace), které se pokládají za ukončené v momentě vyhotovení funkční demo aplikace pro zákazníka. Metodika lpí na pravidelných každodenních schůzkách týmu, kde dochází ke kontrole, co se udělalo, co se má udělat a případně jaké nastaly problémy a jak je vyřešit.
- Vývoj řízený testy (Test-Driven Development – TDD) – snaží se prosadit pravidlo psaní testů ještě před samotným kódem. Samotný výkonný kód by měl být implementován přesně v takovém rozsahu, aby prošel testem.
- Vývoj řízený vlastnostmi (Feature-Driven Development – FDD) – je založen na principu vývoje po malých dávkách (elementární funkcionalita). Jednotlivá funkcionalita je získána z dekompozice modelu do seznamu vlastností. Výhodou je snadné zjištění, co je hotovo, co ještě zbývá a dodávka aplikace s novými vlastnostmi zákazníkovi zpravidla ve dvou týdenních intervalech.

3.1.4 Použití knihoven

Při řešení nějakého komplexního problému je vždy vhodné se porozhlédnout po již hotových řešeních (knihovnách), která by mohla být použita pro realizaci některého dílčího podproblému.

Je to nejen z důvodu úspory času, ale je to i otázka kvality řešení. Mnohdy totiž za vývojem nějaké knihovny stojí početný tým vývojářů, uživatelská komunita poskytující zpětnou vazbu a v neposlední řadě je daná knihovna poskytovaná už v několikáté verzi (tj. odladěná s minimem závažných chyb a nedostatků).

Na druhou stranu, pokud je některá aplikace závislá na použití externí knihovny a tým starající se o ni zanikne (absence aktualizací, podpory atd.), může se celý projekt dostat do problému, pokud se například zrovna narazí na nějakou chybu v knihovně, kterou je potřeba odstranit, jinak nemůže být zaručeno budoucí korektní chování systému. Východiskem zůstává využít jiné knihovny řešící daný problém, ale to nemusí být vždy možné a hlavně to zpravidla bývá časově i finančně nákladné.

Knihoven pro různé problémy a programovací jazyky existuje mnoho. Některé jsou placené, jiné zase ve formě open-source (zdarma při dodržení licenčních podmínek). Mají různé velikosti (jednotky kB až stovky MB) a řeší rozmanité problémy od změny jazykových mutací přes tvorbu tiskových sestav až po umělou inteligenci řešící např. rozpoznávání obličejů v obrázcích.

V této práci se nabízí možnost využití knihovny pro generování počátečního rozvrhu pomocí ILP. Bylo by totiž zbytečné a hlavně pracné implementovat vlastního řešitele, když je pro tento problém na internetu mnoho kvalitních a osvědčených knihoven. Z tohoto důvodu je nutné provést průzkum těchto ILP řešitelů.

3.2 Návrh

3.2.1 Technologie pro naprogramování aplikace

Systém bude implementován v programovacím jazyku C# (konkrétně ve verzi 4.0 nad .NET Framework 4.0). Tato technologie byla zvolena spíše ze subjektivního důvodu, kdy má pro mě přijatelnější syntaxi zdrojového kódu než VB.NET.

Aby nezůstalo jen u osobních preferencí, je vhodné tyto dva jazyky porovnat i nějakým objektivním způsobem. Tohoto úkolu se velmi dobře zhostil samotný Microsoft, který publikoval jejich srovnání v rámci vývoje nad Microsoft Office aplikacemi [59], což je přesně odvětví, do kterého spadá náplň této práce.

VB.NET byl lepší volbou než C# pouze do jeho verze 3.0, protože objektové modely Microsoft Office aplikací byly navrhnuty k použití s Microsoft Visual Basic for Applications (VBA), a tudíž byl vývoj pomocí VB.NET mnohem komfortnější, jelikož bylo předpřipraveno spoustu výkonného kódu, který stačilo jen zavolat (v C# byla potřeba psát dodatečné řádky kódu).

Od vydání vývojového prostředí Visual Studio 2010 (tedy i C# 4.0) se rozdíly prakticky smazávají, nicméně se najde pár případů, kde zůstává potřeba psát kód navíc oproti VB.NET. Jelikož se ale jedná o zanedbatelné procento, tak lze konstatovat, že mezi C# 4.0 a VB 10.0

při vývoji nad Microsoft Office aplikacemi nelze najít z globálního hlediska objektivní důvody, proč upřednostnit jednu technologii nad druhou.

Aplikace bude programována pomocí architektury Document-level customizations, protože cílem je pracovat pouze nad jedním dokumentem, který je uzpůsoben kladeným požadavkům na systém.

3.2.2 Konvence

Bývá dobrým zvykem před započetím implementačních prací definovat konvence, podle kterých se bude psát kód, díky čemuž se pak stane mnohem srozumitelnějším a přehlednějším. Zvláště důležitý význam to má v případě, kdy s kódem pracuje více lidí.

Druhou věcí, která se také dotýká konvencí, je jazyk, ve kterém bude aplikace psána. V této práci to bude výhradně angličtina, a to jak u pojmenování souborů, tříd, funkcí a proměnných, tak i u komentářů. Důvodem je fakt, že angličtina je nejpoužívanějším jazykem v technické sféře a prakticky kdokoliv se základní znalostí anglického jazyka by neměl mít problém danému kódu porozumět.

Navrhnuté konvence:

- jména proměnných, funkcí a metod budou používat velbloudí notaci (tj. velká počáteční písmena u všech slov kromě prvního a následné odstranění mezer mezi jednotlivými slovy),
- názvy tříd, souborů a složek budou psány velbloudí notací s prvním písmenem velkým,
- zkratky v názvech tříd, souborů, složek, proměnných, funkcí a metod budou psány vždy velkými písmeny,
- odsazení pomocí tabulátoru o šíři 4 mezer,
- odsazování vnitřních bloků vůči blokům nadřazeným,
- složené závorky budou vždy na samostatném řádku,
- třída typu interface (rozhraní) bude vždy začínat velkým písmenem *i*.

3.2.3 Postup implementace

Už z analýzy je poměrně zřejmé, že sekvenční metodiky nejsou moc vhodné, právě kvůli chybějící flexibilitě. Takže zbývá využití iteračních technik, kde lze rovnou zamítnout tradiční metodiky, protože kladou důraz na funkcionalitu a v této práci je potřeba zafixovat naopak čas (potažmo finance). Téma je totiž tak obsáhlé, že je nejlepší volbou určit mezník, do kdy má být vše hotovo a od toho navrhnout potřebnou funkcionalitu, která se v daném čase stihne naimplementovat a otestovat.

Z tohoto důvodu se použijí agilní metodiky, konkrétně extrémní programování, protože nejlépe vyhovuje situaci kolem potřebné práce na projektu a mám s ním osobně dobré zkušenosti.

3.2.3.1 Naplánování iterací

V rámci každé iterace dojde k návrhu řešení dílčího problému, jeho implementaci a otestování. Jedna iterace bude mít délku dvou až tří týdnů, přičemž v hodinách to bude většinou do 40 hodin. Refactoring (vylepšování vnitřní struktury systému takovým způsobem, aby nedošlo k ovlivnění vnějšího chování kódu, tzn. funkcionality) se bude provádět dle potřeby, nicméně neměl by být tolik používán, pokud budou od samého začátku dodržovány dohodnuté konvence.

Iterace č.	Úkoly
1 a 2	Architektura systému
3	Čtení a zápis z/do MS Excel
4	Modelování tvrdých a měkkých omezení
5	Kontrola ručně zadaného rozvrhu
6	Implementace algoritmu Hill Climbing
7 a 8	Implementace algoritmu Time Pre-defined Variable Depth Search
9	Implementace algoritmu Simulated Annealing
10	Implementace algoritmu Tabu Search
11 a 12	Implementace algoritmu Scatter Search
13 a 14	Implementace vlastního algoritmu
15 až 18	Experimenty a závěrečné testování

Tabulka 3.1: Plán iterací

3.2.4 Použití knihoven

Bylo potřebné zvolit nejlepšího možného ILP řešitele jak z hlediska snadné práce s ním, tak i časového. Poměrně hodně možností bylo v komerční oblasti (např. Extreme Optimization [40], Solver Platfrom SDK [41]), ale cílem bylo najít open-source řešení. Zde byla situace o něco obtížnější, protože přímo pro jazyk C# toho existuje málo.

Nakonec se ale podařilo najít knihovnu lp_solve [36], která poskytuje vše, co je potřeba, je k ní výborný manuál a pracuje se s ní rychle a snadno. V této práci bude využita její verze 5.5.2.0.

ILP řešitel	Výhody	Nevýhody
Extreme Optimization	paralelizace	placené
Solver Platform SDK	rychlost	placené
lp_solve	manuál, ukázkové příklady	těžkopádnější zadávání jednotlivých omezení
PuLP [79]	přehledné zadávací „prostředí“	nutnost použití modulů třetích stran pro umožnění běhu python skriptů
Microsoft.SolverFoundation [56]	přímá součást platformy .NET	složité dynamické zadávání omezení

Tabulka 3.2: Porovnání uvažovaných ILP řešitelů pro C#

3.2.5 Časový odhad

Prvotní práce, která v sobě zahrnuje seznámení se s problematikou, studium relevantní literatury, studium filozofie návrhu aplikací pod MS Excel a modelování problému pomocí UML diagramů by neměla zabrat více jak 300 hodin. Implementace struktury (architektury) aplikace a potřebných algoritmů spolu s jejich laděním by se měla pohybovat kolem 400 hodin. Testování a provádění experimentů by mělo trvat cca 150 hodin.

Při určování potřebného času bylo vycházeno z předchozích zkušeností u rozsahem podobných projektů.

3.2.6 Diagram tříd (Class Diagram)

Má za úkol zobrazit statickou strukturu aplikace skrze třídy a jejich vztahů.

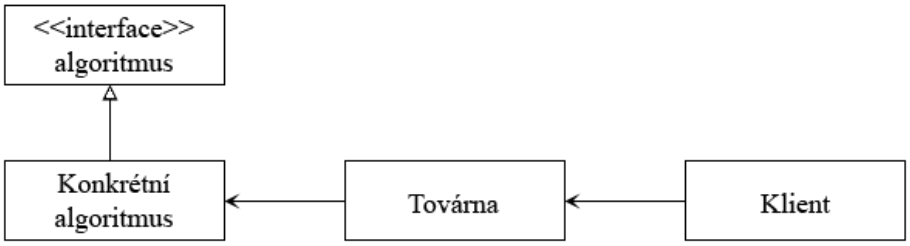
Diagram tříd (viz. G.2) neuvádí seznam atributů tříd, jelikož by docházelo ke zbytečnému překrytí s datovým modelem nebo by byly uvedeny i pomocné proměnné, což by mělo následek v podobě značného zhoršení čitelnosti diagramu. Naopak seznam operací je vyplněn především kvůli částečné představě o tom, co daná třída zhruba dělá (k čemu slouží).

Za zmínku stojí mimo jiné použití návrhové vzoru Factory [69], který má za úkol vrátit správnou instanci konkrétní třídy na základě obdržených parametrů. V této práci se využívá k vrácení konkrétního algoritmu pro rozvrhování na základě preference uživatele.

3.2.7 Datový model (Data Model)

Aplikace sice nevyužívá žádné externí úložiště dat, přesto byl zhotoven datový model (příloha F) pro lepší představu o tom, s jakými daty se v systému pracuje a jaké mají mezi sebou vazby.

Externí úložiště není třeba, protože se nepoužívají data, která by to vyžadovala a na která by nestačilo prosté uložení (a následné načtení v případě potřeby) sešitu MS Excel.



Obrázek 3.2: Ukázka použití návrhového vzoru Factory

3.2.8 Diagram případů užití (Use Case Diagram)

Graficky znázorňuje poskytovanou funkcionalitu aplikace a její interakci s uživateli, kteří budou daný systém využívat.

V diagramu (G.1) je použit pouze jeden aktér, protože aplikace nerozlišuje žádné role a všichni, kdo se ji rozhodnou použít, budou mít naprosto stejné možnosti práce.

3.2.9 Sekvenční diagram (Sequence Diagram)

Zobrazuje časové souvislosti (komunikaci) mezi objekty, které si mohou vzájemně vyměňovat zprávy.

Sekvenční diagram (G.3) je vytvořený pro ukázku toho, kde se co a jak volá při tvorbě rozvrhu.

3.2.10 Grafické uživatelské rozhraní

Jelikož je použit MS Excel, tak není potřeba nijak zvlášť zasahovat do grafiky. Prakticky je to nemožné vyjma kosmetických úprav (barva pozadí, písmo atd.) nebo přidání akčních tlačítek či formulářů.

Rozvržení aplikace bude následující:

- menu – standardní prvky MS Excel,
- obsahová část – obsahuje sadu buněk a dochází zde k zápisu a čtení veškerého obsahu,
- panel akcí – zobrazuje krátkou nápovědu a dostupné akce, které je možno provádět se systémem,
- panel listů – obsahuje seznam všech dostupných listů v rámci sešitu.

Ukázky aplikace lze najít v příloze I.

Kapitola 4

Popis řešení

4.1 Omezení kladená na rozvrh

V následujících podkapitolách jsou uvedeny již konkrétní omezení použité v aplikaci. Jejich obecný popis a význam se nachází v předchozích kapitolách 1.2.3 a 1.2.4.

Každé omezení musí implementovat rozhraní *ICostraint*, které v sobě zahrnuje funkce pro nastavení typu omezení (měkké nebo tvrdé) a kontrolu porušení omezení jak v rámci celého rozvrhu, tak jen pro jednotlivé zaměstnance.

Pro jejich správu byl zhotoven kontejner *ConstraintsContainer*, který umožňuje pohodlné přidávání nových omezení a spouštění kontroly všech obsažených omezení najednou.

4.1.1 Tvrdá omezení

Pokud se při prohledávání rozvrhu najde první porušení daného omezení, tak se okamžitě vrací negativní výsledek a zbývající kontrola se již neprovádí. Aplikace bude umožňovat některá tvrdá omezení zapínat či vypínat (více v příloze D).

Maximálně jedna směna za den

V zadání této práce se omezíme pouze na problémy, kdy v jednom dni je přiřazena maximálně jedna směna. Přesněji, jeden den obsahuje maximálně jeden začátek směny. Tento fakt je zohledněn i v datové struktuře udržující rozvrh.

Požadované obsazení směn

Toto omezení patří k těm nejdůležitějším a určuje počet potřebných zaměstnanců pro každou směnu a den z rozvrhovacího období.

Protože se u všech použitých algoritmů pro rozvrhování v této práci k vylepšování rozvrhu používají pouze vertikální posuny směn, není prakticky nutné toto omezení kontrolovat kromě vygenerování prvotního rozvrhu.

Nicméně aplikace je navrhnuta tak, že je lehké ji rozšířit o nové algoritmy (a tím potencionálně vnést jiné techniky vylepšování rozvrhu), a proto se z důvodu předejití případným

komplikacím a zbytečného zasahování do kódu provádí kontrola vždy. Výrazné časové zpoždění to není (řádově se jedná o setiny až desetiny procent) hlavně díky použité struktuře pro uchování informací o požadované obsazenosti směn namísto procházení celého rozvrhu.

Minimální rozestup mezí směnami

Do rozvrhu nestačí přidat směny jen tak ledabyle podle požadované obsazenosti, ale je potřeba mít na mysli i horizontální stránku věci. Tou je následnost směn, což neznamená nic jiného, než že mezi naplánovanými směnami pro zaměstnance musí být časová prodleva alespoň 12 hodin dle zákoníku práce §90 (1) [84], který doslova říká: „Zaměstnavatel je povinen rozvrhnout pracovní dobu tak, aby zaměstnanec měl mezi koncem jedné směny a začátkem následující směny nepřetržitý odpočinek po dobu alespoň 12 hodin po sobě jdoucích během 24 hodin“.

Zmražené směny

Při vytváření rozvrhu na dané období může být předem známa nějaká skutečnost, kterou je potřeba dát vědět rozvrhovacímu algoritmu. Ten musí s danou skutečností pracovat takovým způsobem, aby se za žádných okolností nezměnila, čili ji musí brát jako fixní.

Skutečností může být např.:

- plánovaná dovolená zaměstnance,
- nutnost zaměstnance absolvovat konkrétní směnu/y v některých dnech,
- potřebná návštěva lékaře zaměstnancem.

Kvalifikace zaměstnanců

Poslední tvrdé omezení opět zesiluje požadavky na umístění směn do rozvrhu. Před přiřazením směny ke konkrétnímu zaměstnanci se totiž kontroluje jeho způsobilost (kvalifikace) vykonávat práci na dané směně.

Ukázkovým příkladem je nemocniční prostředí, kde např. na ranní směnu v pondělí je potřeba x vrchních sester a y zdravotních sester. Směny by tedy byly uvRanní – VS a „Ranní – ZS“ s požadavkem obsazenosti pro pondělky x, respektive y. Zdravotní sestry by měly přiřazenou kvalifikaci pro směnu „Ranní - ZS“, vrchní sestry zase „Ranní - VS“ s možností přidání i „Ranní - ZS“, protože způsobilost k tomu jistě mají.

4.1.2 Měkká omezení

U kontroly libovolného měkkého omezení se vždy projde celý rozvrh, protože nestačí vědět, že došlo k porušení, ale je nutné znát jejich počet. To je důležitý poznatek pro cílovou funkci, na základě které dochází k výběru co možná nejkvalitnějšího rozvrhu.

Pokud by se penalizovala pouhá přítomnost porušení omezení, tak by řešení se stejnou kriteriální funkcí měla odlišnou kvalitu. V případě, že při penalizaci jsou uvažovány i počty porušení, je možné rozvrhy odlišovat dle kvality. Z toho plyne, že rozvrhy s rovnými hodnotami objektivní funkce, které nemusí byt nutně identické, jsou vždy stejně dobré vzhledem k nastaveným omezením.

4.1.2.1 Základní verze

Během počátků vývoje bylo naimplementováno několik hlavních omezení (preferencí), u kterých bylo uživatelem možné měnit jejich váhy.

Celková penalizace (kvalita) rozvrhu je dána součtem dílčích výsledků všech měkkých omezení (4.1).

$$Z = \sum_i P_i \quad , \tag{4.1}$$

kde i označuje zkratku i-té podmínky a je iterováno skrze všechna dostupná omezení.

Vyvážení odpracovaných hodin (VOH)

Aby došlo k vytvoření spravedlivého rozvrhu, je nutné každému zaměstnanci přidělit podobné množství práce měřené v hodinách (4.2).

$$P_{VOH} = \sum_{z=1}^{n_z} |prum_hod - H_z| * V_{VOH} \tag{4.2}$$

Vyvážení typů směn (VTS)

Pod tímto pojmem se skrývá technika, která se snaží docílit toho, aby všichni zaměstnanci měli odpracováno zhruba stejné množství jednotlivých směn (4.3). Důvodem použití tohoto omezení je nejenom vyvážení typů směn z hlediska časového, ale i z hlediska náročnosti jednotlivých typů směn.

$$P_{VTS} = \sum_{s=1}^{n_s} \sum_{z=1}^{n_z} |prum_smen_s - T_{z,s}| * V_{VTS} \tag{4.3}$$

Maximalizace počtu po sobě následujících směn (MNS)

Snahou tohoto omezení je vytvoření rozvrhu, ve kterém budou směny kumulovány do celistvých bloků (4.4). Zaměstnancům to přináší výhodu v podobě delších celistvých bloků volna, které se dají efektivněji zužitkovat než volna jednodenní. V ideálním případě (s nulovou penalizací) by měl každý zaměstnanec naskládané všechny směny za sebou bez přerušení dnem volna. To je ale v praxi nemožné z důvodu několika podmínek, které se buď musí, nebo by měly být splněny. Nejčastěji se lze setkat s blokem o délce maximálně 5 směn.

$$P_{MNS} = \sum_{z=1}^{n_z} pocet_rozd_z * V_{MNS} \tag{4.4}$$

Minimalizace izolovaných pracovních a volných dnů (MSS)

Poslední podmínkou starající se o kvalitu rozvrhu je minimalizace izolovaných pracovních (S) a volných (V) dnů (4.5). Možné vzory jsou následující:

- první dva nebo poslední dva dny rozvrhu: tvar SV nebo VS,

- jinde: tvar SVS nebo VSV.

$$P_{MSS} = \sum_{z=1}^{n_z} pocet_sir_z * V_{MSS} \tag{4.5}$$

4.1.2.2 Vzory

Nevýhodou řešení uvedeného v předešlé podkapitole je neflexibilita. Uživatel využívající aplikaci je odkázán pouze na zhotovená omezení a jediné, co může měnit, jsou váhy penalizací.

Z tohoto důvodu byly zhruba v polovině vývoje naimplementovány tzv. vzory, které právě obstarávají potřebnou volnost a není problémem přes ně namodelovat prakticky každý požadavek. Zároveň jsou používány výzkumnou skupinou ASAP (Automated Scheduling, Optimisation and Planning) [76] v bechmarkových instancích rozvrhovacích problémů [77], čehož je využito v kapitole 5 s experimenty pro porovnání dosažených výsledků s již známými.

Struktura vzoru se skládá z:

- názvu,
- indexu dne, od kterého začíná interval jeho působnosti,
- indexu dne, ve kterém končí interval jeho působnosti,
- typu postupu (minimalizace nebo maximalizace),
- cílového počtu výskytů (CPV) – v kombinaci s předchozí části např. maximálně 5 výskytů,
- váhy,
- výpočtu penalizace – lineárně (4.6) nebo kvadraticky (4.7),

$$P_{vzor} = |pocet_vysk - CPV| * V_{vzor} \tag{4.6}$$

$$P_{vzor} = (pocet_vysk - CPV)^2 * V_{vzor} \tag{4.7}$$

- začátku jeho uplatňování v rámci daného intervalu:
  - kdekoliv,
  - pouze na začátku,
  - pouze v konkrétní den (pondělí až neděle).
- samotného vzoru:
  - omezení délky na 6 slotů,
  - v každém slotu lze vybírat několik položek (den volna a všechny směny).

V tabulce 4.1 je znázorněna transformace omezení z kapitoly 4.1.2.1 do vzorů. Tím je také demonstrován fakt, že pomocí vzorů se dají modelovat různá omezení, a proto je jimi možné tato omezení nahradit.

Omezení	Interval	Postup	CPV	Váha	Začátek	Vzor*	Poznámka
VOH	$1 - n_d$	min.	$\lfloor NS/n_z \rfloor$	V_{VOH}	kdekoliv	S	
	$1 - n_d$	max.	$\lfloor NS/n_z \rfloor$	V_{VOH}	kdekoliv	S	
VTS	$1 - n_d$	min.	$\lfloor N_s/n_z \rfloor$	V_{VTS}	kdekoliv	s	$\forall s \in S$
	$1 - n_d$	max.	$\lceil N_s/n_z \rceil$	V_{VTS}	kdekoliv	s	
MNS	$1 - n_d$	max.	0	V_{MNS}	kdekoliv	SSSSSS	Délka bloku max. 5.
MSS	$1 - 2$	max.	0	V_{MSS}	kdekoliv	VS	
	$1 - 2$	max.	0	V_{MSS}	kdekoliv	SV	
	$(n_d - 1) - n_d$	max.	0	V_{MSS}	kdekoliv	VS	
	$(n_d - 1) - n_d$	max.	0	V_{MSS}	kdekoliv	SV	
	$1 - n_d$	max.	0	V_{MSS}	kdekoliv	SVS	
	$1 - n_d$	max.	0	V_{MSS}	kdekoliv	VSV	

* S = libovolná směna, s = konkrétní směna, V = den volna

Tabulka 4.1: Transformace základních měkkých omezení do vzorů

V případě omezení, které kladou požadavky na rovnost o hodnotě větší než nula, je nutné použít dvou vzorů. Jeden bude minimalizovat ($x < y$) a druhý maximalizovat ($x > y$) stejnou věc, takže ve výsledku se docílí rovnosti ($x = y$). Pro nulu stačí maximalizace, protože počet výskytů je vždy nezáporné číslo. Výhodou tohoto přístupu je možnost nastavit odlišné váhy a výpočty penalizací pro jednotlivé přesahy (např. bude požadavek na dva souvislé dny volna – pokud jich bude méně, tak je to horší situace, než kdyby jich bylo více, a proto se pro první případ nastaví větší váha než pro ten druhý).

4.1.3 Tvrdá omezení modelovaná jako měkká

Při prohledávání stavového prostoru je vždy nově nalezený rozvrh kontrolován, zdali náhodou neporušuje některé z tvrdých omezení. Pokud ano, tak je ihned zahozen a dále se s ním nepracuje.

Někdy je však vhodnější modelovat některá tvrdá omezení jako měkká, aby se rozvrh ihned nezavrhl a operovalo se s ním dál. Nutno podotknout, že je to možné aplikovat pouze u algoritmu Simulated Annealing, který jako jediný v této práci může přijmout i rozvrh s horší penalizací, než je nejlepší doposud známá hodnota.

Modelováním tvrdých omezení jako měkkých se ovšem vystavujeme možnému problému akceptování rozvrhu, který v sobě bude mít právě takto namodelované tvrdé omezení. Obranou proti této nepříjemnosti může být nastavení velkých vah. Otázkou ale je, jak moc velkých?

Podstata řešení je vázána na počáteční rozvrh. Pokud se jej podaří vytvořit, tak máme jistotu, že existuje proveditelný rozvrh s nějakou penalizací. A víme, že se daná penalizace

průchodem přes rozvrhovací algoritmy v žádném případě nemůže zvětšit (právě naopak). Proto bude váha těchto omezení automaticky nastavena přesně na hodnotu penalizace počátečního rozvrhu navýšenou o jedničku.

Tímto je zaručeno, že i kdyby se povedlo počáteční rozvrh upravit takovým způsobem, aby žádné původní měkké omezení nebylo porušeno (rozvrh by byl s penalizací = 0) a zároveň došlo k porušení nějakého (byť jen jednoho) modelovaného omezení, tak tento upravený rozvrh nebude vrácen, protože má penalizaci horší než rozvrh prvotní (čili by byl i výsledkem).

4.2 Počáteční rozvrh

Velmi často se v optimalizaci používá dekompozice problému na získání počátečního řešení a jeho postupného vylepšování. Od počátečního řešení se očekává rychlé vytvoření a přiřazení všech potřebných směn k jednotlivým zaměstnancům. Rychlost se samozřejmě musí odrazit na kvalitě rozvrhu, takže určitě není překvapením vznik velmi špatného rozvrhu. Nicméně bylo zjištěno [21], že kvalita prvotního rozvrhu má pouze velmi malý vliv na rozvrh finální.

V aplikaci jsou použity dva různé přístupy tvorby pomocí:

1. ILP s ohledem na všechna tvrdá omezení,
2. hladového algoritmu respektujícího pouze požadavky obsazenosti směn.

4.2.1 Celočíselné lineární programování

4.2.1.1 Definice problému

Jedná se o optimalizační úlohu [70], která se formuluje jako:

Maximalizace (minimalizace)

$$c^T * x$$

za podmínek

$$A * x \leq b \quad ,$$

kde matice $A \in \mathbb{R}^{m \times n}$, vektor $b \in \mathbb{R}^m$, vektor $c \in \mathbb{R}^n$, vektor $x \in \mathbb{Z}^n$ a cílem tedy je nalézt správný vektor x.

4.2.1.2 Tvorba matematického modelu

K vyřešení problému bude sloužit knihovna lp_solve, která byla vybrána na základě analýzy (kapitoly 3.1.4 a 3.2.4). Všechna tvrdá omezení bude potřeba dostat do nerovnice $A * x \leq b$, přičemž knihovna dává možnost měnit (ne)rovnítko pro každý řádek zvlášť.

Pro konkrétní použití v této práci se neuvažuje žádná kriteriální funkce. Zajímáme se totiž pouze o to, zdali má problém řešení (pokud ano, tak samozřejmě i jeho hodnotu), čili jestli je rozvrh proveditelný. O jeho vylepšení se následně starají jiné fundovanější algoritmy.

Obecné značení a jejich popis je uveden v příloze B.

Uvažují se tři faktory:

- zaměstnanec,
- den,
- směna.

Vektor x bude mít velikost $n_z{*}n_d{*}n_s$. Uvažujme jednoduchý příklad se třemi zaměstnanci, dvěma směnami a dvěma dny. Pak je jeho interpretace následovná (tabulka 4.2).

<table>
<tr><th>index</th><th>zaměstnanec (z)</th><th>směna (s)</th><th>den (d)</th></tr>
<tr><td>1</td><td>1</td><td rowspan="3">1</td><td rowspan="6">1</td></tr>
<tr><td>2</td><td>2</td></tr>
<tr><td>3</td><td>3</td></tr>
<tr><td>4</td><td>1</td><td rowspan="3">2</td></tr>
<tr><td>5</td><td>2</td></tr>
<tr><td>6</td><td>3</td></tr>
<tr><td>7</td><td>1</td><td rowspan="3">1</td><td rowspan="6">2</td></tr>
<tr><td>8</td><td>2</td></tr>
<tr><td>9</td><td>3</td></tr>
<tr><td>10</td><td>1</td><td rowspan="3">2</td></tr>
<tr><td>11</td><td>2</td></tr>
<tr><td>12</td><td>3</td></tr>
</table>

Tabulka 4.2: Mapování jednoduchého příkladu do vektoru

Nyní je nutné zakódovat všechna potřebná omezení (v našem případě se jedná jen o ty tvrdá) do matice A a vektoru b podle následujících matematických vztahů.

Požadované pokrytí směn

$$\sum_{z=1}^{n_z} S_{z,s,d} = PO_{s,d} \quad \forall s, \forall d \quad ,$$

kde

$$S_{z,s,d} = \begin{cases} 1 & \text{zaměstnanec } z \text{ pracuje ve dni } d \text{ na směně } s \\ 0 & \text{jinak} \end{cases} \qquad \text{a}$$

$$PO_{s,d} = \text{počet potřebných zaměstnanců ve dni } d \text{ na směně } s.$$

Zaměstnanec může během dne pracovat maximálně na jedné směně

$$\sum_{z=1}^{n_z} S_{z,s,d} \leq 1 \quad \forall z, \forall d$$

Kvalifikace zaměstnanců

$$\sum_{d=1}^{n_d} S_{z,s,d} \leq n_d * K_{z,s} \quad \forall z, \forall s \quad ,$$

kde

$$K_{z,s} = \begin{cases} 1 & \text{zaměstnanec } z \text{ má kvalifikaci na směnu } s \\ 0 & \text{jinak} \end{cases} \quad .$$

Následnost směn

$$S_{z,s_1,d} + S_{z,s_2,d+1} - N_{s_1,s_2} \leq 1 \quad \forall z, \forall d, \forall s_1, \forall s_2 \text{ a } d < n_d \quad ,$$

kde

$$N_{s_1,s_2} = \begin{cases} 1 & \text{směna } s_2 \text{ může následovat po směně } s_1 \\ 0 & \text{jinak} \end{cases} \quad .$$

Zmražené směny

Uvažujeme slot $L_{z,d}$, jenž je určen zaměstnancem z a dnem d a má hodnotu indexu směny (v případě volna -1).

- Pokud se jedná o volno:

$$\sum_{z=1}^{n_z} S_{z,s,d} = 0 \quad .$$

- Pokud jde o směnu:

$$S_{z,s,d} = 1 \quad .$$

4.2.2 Greedy Algorithm (Hladový algoritmus)

Greedy Algorithm pracuje pouze s jediným omezením, a to požadavky na obsazenost směn. Na rozdíl od ILP nevrací pouze jeden rozvrh, ale hned několik (záleží na požadavku). Tyto rozvrhy jsou navíc od sebe co nejvíce odlišné.

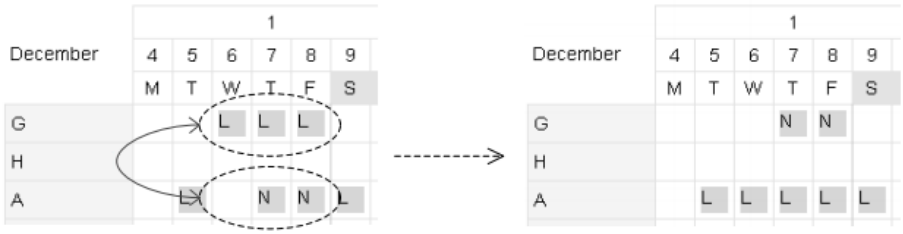
Pokud se vytváří první rozvrh z celé série, tak jsou směny přidělovány prvním možným zaměstnancům. Jestliže již existuje několik rozvrhů (stačí i jeden), tak se skrze všechny spočítá výskyt aktuálně procházené směny v daném dni pro všechny zaměstnance. Přiřadí se pak tomu, kdo ji absolvoval nejméně krát. V případě rovnosti rozhoduje náhoda.

Pseudokódy se nachází v příloze C, konkrétně C.1 a C.2.

4.3 Algoritmy pro rozvrhování

V této kapitole je sepsán seznam použitých algoritmů pro rozvrhování, jejich základní popis, pseudokód a použité lokální úpravy.

U všech se k vylepšování rozvrhu používá prohazování směny/bloků směn vertikálním směrem vždy mezi dvěma zaměstnanci (4.1 převzat z [21]).



Obrázek 4.1: Prohození bloku směn

Pro studium následujících algoritmů byly využity primárně zdroje použité v kapitole 2. Pokud bude charakter některého sdělení vyžadovat dodatečné citování literatury, bude tak učiněno.

Vliv lokálních parametrů a úprav metaheuristik je diskutován v kapitole 5.3.

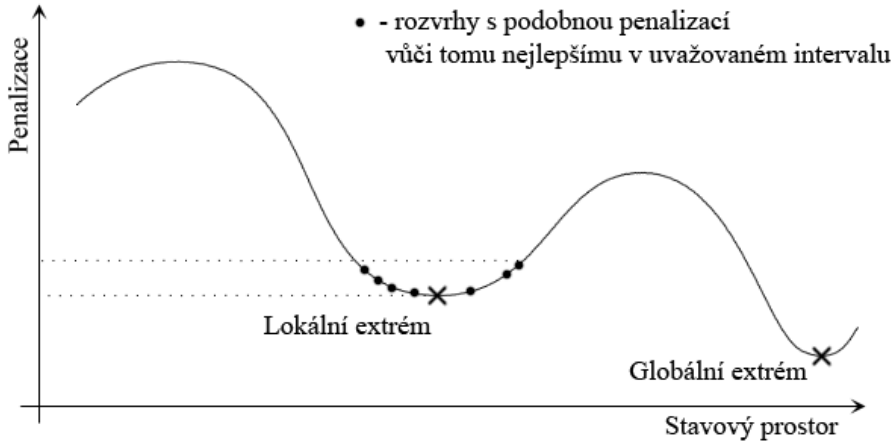
4.3.1 Hill Climbing

Tato metoda se řadí k tomu nejjednoduššímu, co lze použít k prohledávání stavového prostoru. Jedná se o klasickou gradientní techniku, které je potřeba dodat počáteční řešení (v našem případě rozvrh), ze kterého se začne prohledávat. Hledá tak dlouho, dokud nedosáhne určitého počtu kroků nebo už není možné žádné vylepšení dosud nejlepšího nalezeného řešení.

Během průchodu algoritmem postupně dochází ke zkoumání sousedství aktuálního rozvrhu. Sousedstvím se rozumí takový rozvrh, který se liší od toho původního pouhým prohozením směny/bloku směn mezi dvěma zaměstnanci.

Negativem je možné uváznutí v lokálním extrému, které je u této metaheuristiky poměrně časté. Tento jev lze částečně eliminovat opakovaným spuštěním algoritmu, ale vždy z jiného místa stavového prostoru. Výsledkem je pak nejlepší řešení vybrané ze všech nalezených v rámci provedených iterací, které může být řešením optimálním se zvyšujícím se počtem opakování.

K implementaci tohoto vylepšení je v této práci použito restartů. Ty využívají skutečnosti, že pokud se nacházíme v lokálním extrému, tak se množí řešení s velmi podobnou penalizací ve srovnání s nejlepším nalezeným rozvrhem (4.2). Pokud počet takovýchto řešení naroste do určité meze, tak se algoritmu předá signál, že může provést restart a začít práci znovu s novým počátečním řešením.



Obrázek 4.2: Chování algoritmu Hill Climbing v lokálním extrému

Pro vytvoření nedeterministického chování a rozmanitějšího procházení stavového prostoru byla do algoritmu implementována nastavitelná náhoda (procentuální vyjádření, kdy může dojít k nějakému jevu), a to hned na dvou místech:

1. přijmutí nového nejlepšího řešení, i když se jen rovná tomu aktuálnímu,
2. k restartu nemusí dojít hned při překročení meze.

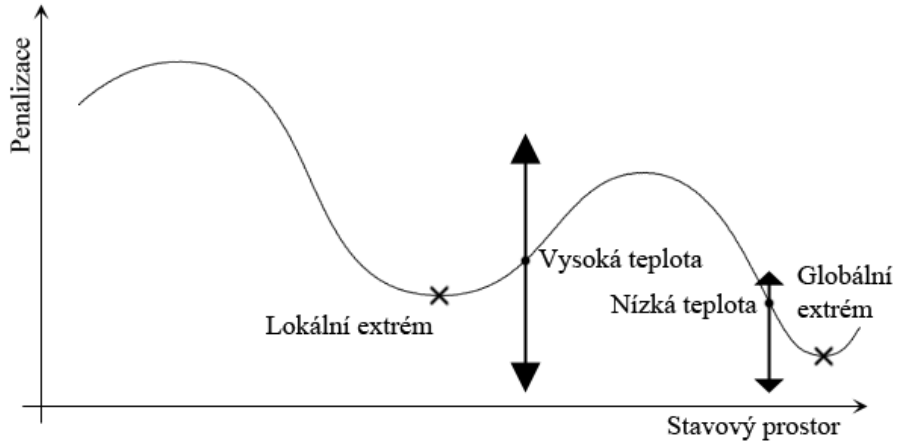
Pseudokód se nachází v příloze C, konkrétně C.3.

4.3.2 Simulated Annealing

Algoritmus Simulated Annealing je inspirován ryze praktickým problémem žíhání tuhého tělesa (např. ocel). Tento postup se provádí kvůli odstranění vnitřních defektů materiálu. Začíná se jeho zahřátím na vysokou teplotu, čímž vlastně dojde k jeho roztopení. To má za následek náhodné uspořádání částic v prostoru. Následně se začne pomalu snižovat teplota a díky tomu se částice pozvolna dostávají do rovnovážné polohy. Na konci procesu, kdy je teplota podstatně nižší než na začátku, jsou už všechny částice v rovnováze a výsledný produkt je bez vnitřních defektů.

Analogie tohoto fyzikálního jevu se používá i v optimalizaci. Na začátku se nastaví nějaká teplota, která nepřímo určuje, jak moc špatný rozvrh oproti nejlepšímu známému je možné přijmout. Postupem času se teplota sníží do takových hodnot, kdy už prakticky nedochází k přijímání horších rozvrhů, ale jen těch kvalitnějších.

Tato modifikace je tedy obranou proti uváznutí v lokálním extrému. Předpokladem je ale pomalá konvergence algoritmu (teploty) tak, aby bylo možné dostatečně dlouhou dobu provádět velké změny v rozvrhu, a tím uniknout z lokálního extrému (4.3).



Obrázek 4.3: Možnost pohybu algoritmu Simulated Annealing v závislosti na teplotě

Nejdůležitější částí metaheuristiky je tzv. Metropolisovo kritérium (4.8), které určuje pravděpodobnost přijmutí (P) nového rozvrhu (n) namísto toho starého (s) na základě aktuální teploty (t) a kriteriální funkce (f).

$$P = \begin{cases} 1 & \text{iff} \quad f(n) \leq f(s) \\ e^{\frac{f(s)-f(n)}{t}} & \text{iff} \quad f(n) > f(s) \end{cases} \tag{4.8}$$

Dalším signifikantním faktorem je bezesporu technika výběru (procházení) sousedních rozvrhů. Zde byly naimplementovány čtyři možnosti:

- náhodný výběr dnů i zaměstnanců,
- iterování dnů s náhodným výběrem zaměstnanců,
- iterování zaměstnanců s náhodným výběrem dnů,
- iterování dnů i zaměstnanců.

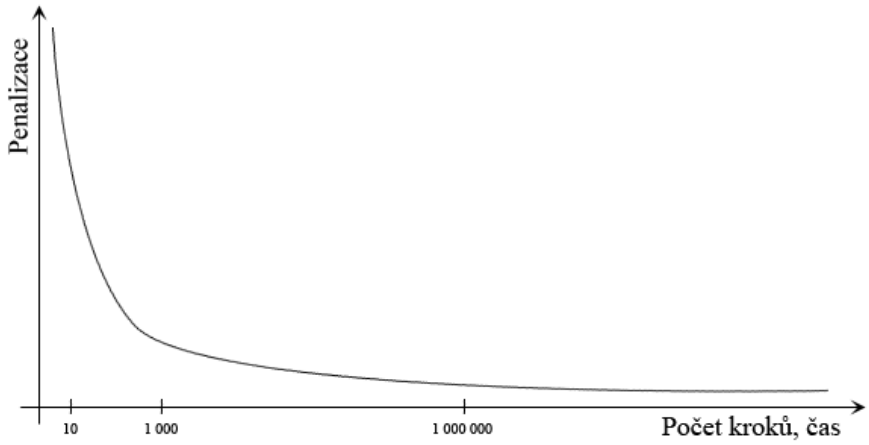
Ukončovací podmínka algoritmu byla uvažována ve dvou modifikacích, a to:

1. dosáhnutí maximálního předem daného počtu kroků,
2. dosáhnutí předem daného počtu kroků bez zlepšení penalizace rozvrhu.

První přístup má výhodu v tom, že se v podstatě zvolí, jak dlouho bude vylepšování rozvrhu probíhat. Dopředu tedy víme, za jak dlouho algoritmus zhruba skončí. Nevýhodou je nutnost správného nastavení. Pokud se zvolí malý počet kroků, tak nemusí dojít k úplnému přiblížení se k optimální hodnotě penalizace. V opačném případě může zase dojít ke zbytečnému plýtvání časem, během jehož konce už nedochází k vůbec žádnému pokroku.

U druhé možnosti tato volba odpadá a je řízena samotným algoritmem. Je zde ale jiné nastavení, a to po kolika krocích, kdy nedojde ke zlepšení, má dojít k ukončení. A i tato

variabilita má podobná úskalí. Jakmile se bude operovat s malou hodnotou, bude docházet k předčasnému konci. Velká hodnota může mít za následek dlouhý běh algoritmu, ale na druhou stranu je výsledné řešení velmi kvalitní a blížící se tomu optimálnímu (samozřejmě taky v závislosti na počáteční teplotě a chladícímu rozvrhu).



Obrázek 4.4: Typický vývoj penalizace v čase u algoritmu Simulated Annealing

V implementaci metaheuristiky byla nakonec zvolena první varianta kvůli možnosti nastavení přibližné doby běhu algoritmu. O nic se tím nepřichází (z hlediska kvality rozvrhu), protože není problémem zvolit takový počet kroků, aby se vyrovnal druhému přístupu. Ten totiž nedělá nic jiného, než úpravu počtu kroků v závislosti na aktuální situaci.

Pseudokód se nachází v příloze C, konkrétně C.4.

4.3.3 Tabu Search

Dalším algoritmem, který se snaží eliminovat nevýhody uváznutí v lokálním extrému u Hill Climbing je Tabu Search. Nástrojem je paměť. Ta je reprezentovaná seznamem, nazývaným Tabu List, který v sobě udržuje informace o provedených přesunech v rámci rozvrhu. Pokud se dostaneme k rozvrhu, který obsahuje změnu vyskytující se v Tabu Listu, tak se s ním dále nepracuje a vybírá se jiné sousední řešení. Tím, že se pamatují řešení, ke kterým se nelze vracet z důvodu jejich navštívení, se právě zamezuje jak zacyklení, tak uváznutí v lokálním extrému.

Otázkou ale zůstává, jak velká paměť se má použít a jak se má chovat. Kdyby se použila „neomezená“ paměť, tak by se s největší pravděpodobností nikdy nedošlo k dobrému řešení, spíše podprůměrnému. S postupem času by byl totiž omezen výběr kandidátů takovým způsobem, že by nebylo z čeho vybírat. Naopak použití malé velikosti by degradovalo TS na úroveň HC bez restartů, čili klasické gradientní metody. V praxi se lze setkat převážně s délkou od 5 – 20, vždy je ale potřeba vzít v potaz typ problému a instance. Paměť je typu FIFO, takže s postupem času dochází k zapomínání nejstarších řešení a jejich nahrazování novějšími.

V některých modifikacích TS se lze setkat s tzv. aspiračním kritériem. To se dostává ke slovu, jakmile se narazí na změnu, která se již nachází v Tabu Listu, a tudíž je zakázaná. A nejen že díky němu může dojít k obejití Tabu Listu, dokonce lze přijmout i horší řešení než je doposud známé (obdoba pravděpodobnosti u SA). V této práci je ale použita pouze základní, nejrozšířenější, verze TS.

Ukončovací podmínka je zde ve formě nemožnosti nalezení lepšího rozvrhu. Pointou je použití struktury pro každého zaměstnance. Ta se skládá pouze z hodnoty nejlepší dosažené penalizace rozvrhu pro daného zaměstnance a počtu kroků (PK), po kterých nedošlo k jeho zlepšení. Jakmile dojde ke zlepšení penalizace, je hodnota PK vynulována. Na začátku každé iterace algoritmu se vybere takový zaměstnanec, který přispívá do penalizace celého rozvrhu nejvíce. Musí ale platit, že hodnota PK není větší jako velikost Tabu Listu. Tato podmínka není nastavena náhodně. Pokud totiž po tu dobu nedojde k žádnému zlepšení, tak už lze prohlásit, že se tak nestane nikdy (v rámci kontextu problému). Během té doby ovšem může docházet k pokrokům v rozvrhu, protože dochází k postupnému uvolňování všech zakázaných změn z Tabu Listu a jejich možnosti použití v následné iteraci. Jakmile jsou všichni zaměstnanci tímto mechanismem zakázaní (není co zlepšovat používaným způsobem), dochází k ukončení metaheuristiky.

Jelikož se zde používají bloky směn, tak bylo potřeba vyřešit, jakým způsobem se budou hledat kolize aktuální změny s těmi v Tabu Listu. Uvažovány byly dvě varianty:

- vnitřní překryvy,
- jakékoliv překryvy.

Příklad

Typ	1. zaměstnanec	2. zaměstnanec	Den	Délka bloku
Záznam v Tabu Listu (TL)	e_1	e_2	5	3
Požadována změna 1 (PZ1)	e_1	e_2	4	2
Požadována změna 2 (PZ2)	e_1	e_2	6	2
Požadována změna 3 (PZ3)	e_1	e_2	3	8
Požadována změna 4 (PZ4)	e_1	e_2	1	3

Tabulka 4.3: Ukázkový příklad pro vyhledávání kolizí v algoritmu Tabu Search

V případě TL jsou ovlivněny tedy kromě pátého dne i den šestý a sedmý. Analogicky je tomu i u ostatních. U vnitřních překryvů dojde ke kolizi pouze v případě PZ2 (celý jeho interval spadá do rozmezí TL), u jakýchkoliv v PZ1, PZ2 a PZ3 (jejich interval se jakýmkoliv způsobem dotýká rozmezí TL).

Pseudokódy se nachází v příloze C, konkrétně C.5 a C.6.

4.3.4 A Time Pre-defined Variable Depth Search

Prvním z algoritmů, který je inspirován literaturou a byl vybrán na základě rešerše, je Time Pre-defined Variable Depth Search (TPVDS). Jedná se o heuristiku, která se skládá ze dvou

hlavních částí (sousedský a zlepšovací algoritmus) a „doprovodných“ fragmentů, které je spojují.

Myšlenka je poměrně prostá. Metoda se snaží postupovat stejným způsobem, jako by se to muselo dělat ručně. Nejprve se vybere jeden zaměstnanec a jeho část rozvrhu se nějakým způsobem vylepší. Tento zásah ale většinou znamená zhoršení individuálních rozvrhů jiných zaměstnanců. Ty se pak musí postupně také začít vylepšovat. Na konci tohoto procesu by měl být rozvrh výrazně lepší kvality než na začátku. Pokud tomu tak není, je potřeba vrátit zpátky provedené směny a začít znovu.

Sousedský algoritmus je určen pouze pro vrácení dosud nenavštíveného sousedního řešení k tomu aktuálně nejlepšímu. Implementace je proto velmi jednoduchá, stačí totiž iterativně projít všechny možné kombinace.

Zlepšovací algoritmus už je složitější a jedná se o stěžejní část. V ní dochází k samotnému vylepšování rozvrhu. Aplikace je navržena tak, aby pro tento problém mohly být použity již hotové metaheuristiky. K tomu, aby tak mohlo být učiněno, je potřeba jen pár modifikací v původním kódu (jde především o vstupní parametry).

Důležitou součástí (modifikací) TPVDS je časová restrikce kladená na dobu běhu. Nejedná se o prosté zadání hodnoty, následné ničím nerušené práce algoritmu a vrácení výsledku po uplynutí času (nebo před, pokud se to stihne). Je zde zabudována určitá dynamičnost:

1. algoritmus skončí dříve než je limit – algoritmus je spuštěn znovu s nějakým novým počátečním rozvrhem (pamatuje si ale nejlepší nalezené řešení),

2. flexibilní hloubka procházení – na základě celkového zbývajícího času (OT), počtu potencionálních zbývajících sousedství (NN) a času procházení aktuálního rozvrhu (AT) určí, zdali je možné jít do další úrovně zanoření (4.9).

$$AT < \frac{OT}{NN} \tag{4.9}$$

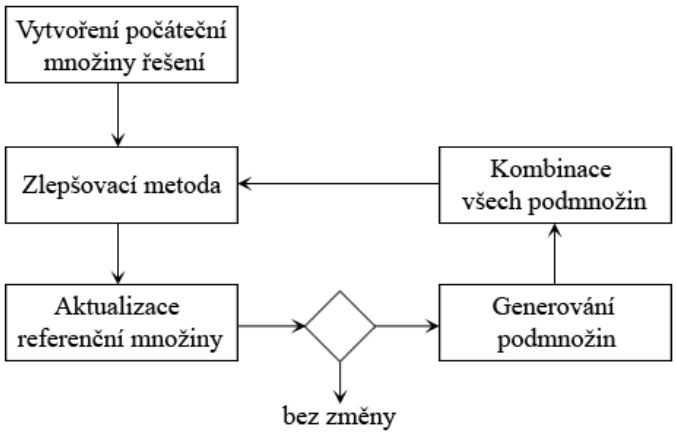
Pseudokód se nachází v příloze C, konkrétně C.7.

4.3.5 Scatter Search

Druhým algoritmem inspirovaným literaturou je Scatter Search (SS). Tento algoritmus podobně jako evoluční algoritmy pracuje s určitou populací řešení. Odlišuje se hlavně přístupem k formování nového řešení. U evolučních algoritmů se využívá různých stochastických elementů k provedení této operace, kdežto SS se snaží tuto náhodnost odstranit a zavádí systematická deterministická pravidla. Dalšími drobnými rozdíly jsou např.:

- všechna řešení jsou použita v kombinační metodě u SS,
- lokální pohledávání jsou nedílnou součástí SS,
- SS není limitováno kombinací dvou rodičovských řešení,
- SS nevytváří počáteční populaci náhodně.

Tato metaheuristika se dá poměrně pěkně dekomponovat na dílčí problémy. Toto rozdělení je znázorněno na obrázku 4.5 a popsáno v následujících odstavcích.



Obrázek 4.5: Dekompozice algoritmu Scatter Search

Nejprve je nutné vytvořit počáteční množinu řešení (P). Její velikost by měla být nastavena zhruba na dvojnásobek velikosti referenční množiny (R). Každé řešení poté projde zlepšovací metodou. Zde, podobně jako u TPVDS, je možné použít již hotové heuristiky. Tentokrát ale bez dodatečných úprav, protože vstupem i výstupem je samotný rozvrh. Následně dojde k vytvoření populace, tedy referenční množiny R. Opět zde hraje roli její velikost. Malá by znamenala nedostatečné pokrytí stavového prostoru. Velká zase razantní zvýšení časové náročnosti. Je potřeba si totiž uvědomit, že každé řešení vždy projde zlepšovací metodou, což je značná zátěž. Referenční množina R se v inicializační fázi vytváří ve dvou krocích:

1. výběr x nejlepších řešení z P a jejich následného vyjmutí z P,
2. výběr y nejodlišnějších řešení z P vůči aktuálnímu stavu v P a jejich následného vyjmutí z P.

Poměr x:y bývá zpravidla 3:2. Jakmile je inicializována R, dojde z ní k vytvoření všech možných podmnožin pro kombinaci. Strategií výběru podmnožin je opět více:

- všechny podmnožiny o dvou elementech,
- výběr x nejlepších řešení,
- podmnožiny o délce 2 doplněné nejlepším řešením nenacházející se v aktuální podmnožině atd.

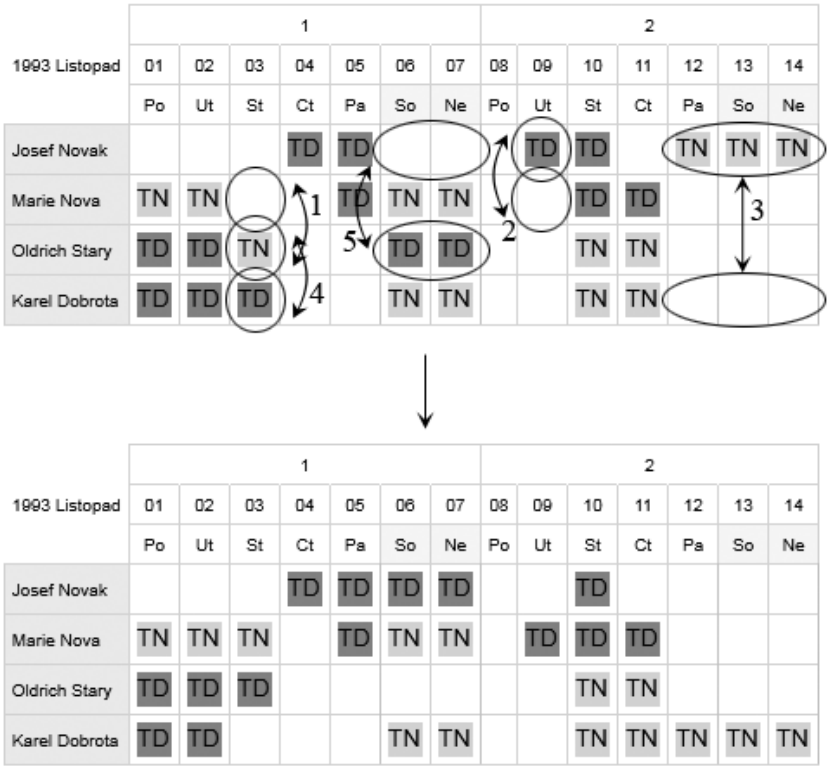
Prvky těchto podmnožin jsou slučovány do jednoho výsledného rozvrhu, který pak ještě musí projít zlepšovací fází. Pokud je kvalita rozvrhu lepší než u nejhoršího z R, tak se provede jejich záměna. Algoritmus skončí v okamžiku, kdy dojde k projití všech vygenerovaných podmnožin. Je potřeba dát pozor na to, že se R v průběhu času může měnit, čili se objevují nové možnosti pro kombinaci podmnožin a je nutné s nimi počítat.

Stěžejní části SS je vytváření nových jedinců (rozvrhů) na základě obdržené podmnožiny rodičovských rozvrhů (PRR). Snahou je dopátrání se výsledku, který toho bude mít co nejvíce společného se svými „předky“. K pochopení pseudokódu pro kombinaci všech podmnožin (C.9) je nezbytné vysvětlit význam dvou termínu užívaných v kontextu této metody. *Kandidáti* jsou množinou všech možných přiřazení směn zaměstnancům v jednotlivých dnech. *Hlasy*, jež dostávají *kandidáti*, jsou shody s rozvrhy z PRR. Jestliže tedy *kandidát* figuruje v nějakém rozvrhu, je mu přidělen *hlas* (pokud ve dvou tak dva, ve třech tři atd.). V množině *kandidátů* jsou pouze prvky s alespoň jedním hlasem.

Pseudokódy se nachází v příloze C, konkrétně C.8 a C.9.

4.3.6 Chain Algorithm

Posledním algoritmem, který zde bude popsán, je Chain Algorithm (CHAIN). Vznikl na základě zkušeností a znalostí nabytých během tvorby této práce. Snaží se aplikovat osvědčené postupy a odstranit jejich nedostatky. Jak už název vypovídá, hlavním mechanismem je vytváření řetězů změn. Nejprve se vybere náhodně nějaký zaměstnanec e_1. K němu se bude hledat nejlepší možné prohození s jiným zaměstnancem (e_2). Nyní už se ale vezme jako pivot zaměstnanec e_2 a k němu se opět nachází nejvýhodnější záměna. Toto se celé opakuje, dokud je potencionální možnost zlepšení.



Obrázek 4.6: Práce s řetězy v Chain Algorithm

Do řetězu nemůže být přijata libovolná změna, ale pouze taková, která se v něm již nenachází. Stejně jako u TS, i zde jsou díky použití bloků směn dvě možnosti řešení konfliktů (vnitřní vs. jakékoliv překryvy, více v kapitole 4.3.3). Navíc do něj lze začlenit i takový element, který nepřináší žádné vylepšení (právě naopak), ale musí platit, že zlepšení generující celý řetězec musí být kladné. S postupem času je tato možnost prakticky zakázaná z důvodu aplikace Metropolisova kritéria popsaného v kapitole 4.3.2.

Pro realizaci ukončovací podmínky bylo potřebné definovat datovou strukturu, která si bude udržovat informaci o možnosti budoucího zlepšení pro každého zaměstnance. Jakmile pro některého z nich nejde vůbec vytvořit řetěz alespoň o délce 1 (jediné prohození), tak jej metaheuristika zakáže. Když už budou všichni zakázaní, CHAIN jenom vrátí uchované nejlepší nalezené řešení. Ke zrušení zákazů dojde při každém zlepšení kvality výsledného rozvrhu.

Pseudokódy se nachází v příloze C, konkrétně C.10 a C.11.

Kapitola 5

Experimenty

V této kapitole jsou uvedeny všechny experimenty, které byly provedeny v průběhu celého cyklu vývoje. Nachází se zde pouze grafy (mnohdy s využitím logaritmického měřítka pro osy) a jejich slovní okomentování. Vertikální osa vždy představuje průměrné hodnoty. Tabulky, které obsahují detailnější informace, lze nalézt v příloze E.

5.1 Průběh testování

Každé měření bylo opakováno stokrát, v grafech jsou pak zaneseny zprůměrované hodnoty penalizace a času. U časově náročných měření (několik hodin) byl počet opakování menší (zpravidla deset).

Experimenty s lokálními parametry algoritmů byly prováděny z důvodu obrovské časové náročnosti na několika strojích (PC1, PC2, PC3, PC4). Na stejném stroji se ale vždy prováděly takové testy, které se měly napříč sebou porovnávat (např. na jednom běžel test vlivu délky bloku na penalizaci pro instanci A, na druhém zase pro instanci B). V případě porovnání algoritmů na benchmarkových instancích už vše probíhalo na stejném stroji (PC1).

Stroj	CPU typ	CPU [GHz]	RAM [GB]	OS verze	MS Office verze
PC1	Intel Xeon E5530	2,4	2	7 32bit	2007
PC2	AMD Opteron 280	2,4	2	7 32bit	2007
PC3	Intel Core 2 Duo E6750	2,66	2	7 32bit	2007
PC4	AMD Athlon 4850e	2,5	4	7 64bit	2007

Tabulka 5.1: HW konfigurace použitých strojů

5.2 Popis testovaných instancí

Dříve, než přejdeme k výsledkům experimentů, je nutné popsat instance, nad kterými se provádělo testování. Tyto instance byly převzaty ze stránek univerzity Nottingham [77], na kterých jsou uveřejněny různé problémy jak z průmyslu, tak i z vědeckých publikací. Některé ovšem musely být mírně modifikovány tak, aby šly použít v této práci. V těchto případech

nebylo možné vyčíst informace o nejlepším řešení přímo ze stránek, ale bylo třeba využít program Roster Booster [27], vyvíjený stejnou univerzitou, kde byl pro řešení instancí použit algoritmus TPVDS (s neznámými aplikovanými modifikacemi). V něm byly takto upravené instance spuštěny a výsledek byl brán jako nejlepší známý.

Instance	Počet zaměstnanců	Počet směn	Počet dnů	Počet měkkých omezení	Počet tvrdých omezení	Nejlepší známé řešení
MILLAR	8	2	14	15	1	0^*
MILLAR-S	8	2	14	10	1	0^*
LLR	27	3	7	8	1	204^+
GPOST	8	2	28	33	1	25^+
VALOUXIS	16	3	28	18	1	20^*
WHPP	30	3	14	13	1	108^+
AZAIES	13	2	28	14	1	4^+
SINTEF	24	5	21	11	1	0^*
ORTEC	16	4	31	49	1	570^+
MUSA	11	1	14	8	1	46^+

* Optimální řešení
$^+$ Řešení generované programem Roster Booster

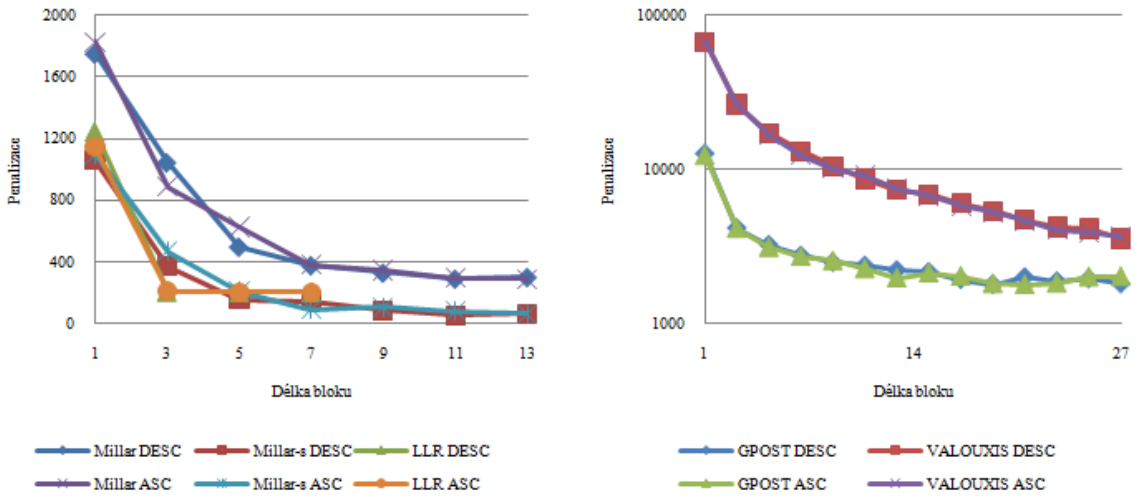
Tabulka 5.2: Popis testovaných instancí

5.3 Experimenty s lokálními parametry algoritmů

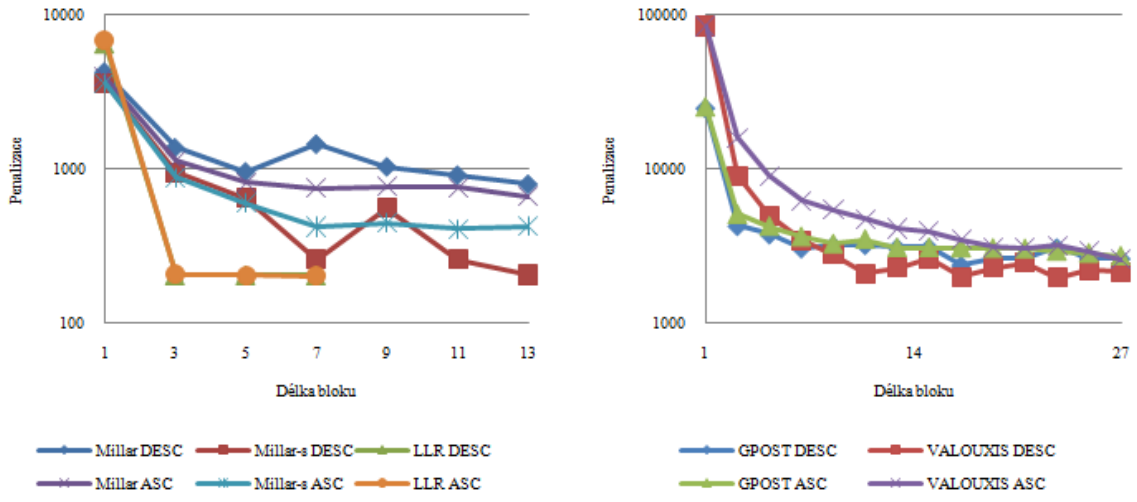
Cílem experimentů v této podkapitole je prozkoumat vliv lokálních parametrů na kvalitu rozvrhu a nalezení jejich nejlepšího možného nastavení. Pro tyto účely bylo vybráno pět různých instancí tak, aby pokryly co možná největší škálu možností vstupních dat.

5.3.1 Délka bloku směn pro prohazování (HC, SA, TS, CHAIN)

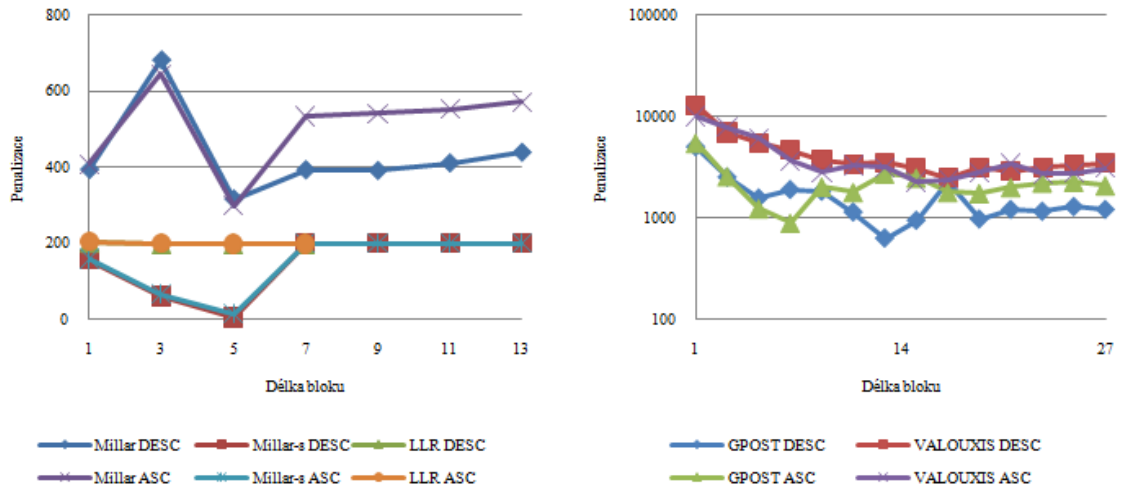
Z výsledků vyplývá, že zavedení bloků směn má určitě smysl. Už při použití bloku délky tři lze pozorovat několikanásobné zlepšení (místy i 10krát). Jako optimální velikost (i vzhledem k času) se jeví hodnota odpovídající zhruba polovině dnů, pro které se tvoří rozvrh. Je to zapříčiněno tím, že větší blok už je kontraproduktivní, protože se snažíme provést tak velkou změnu, která skoro nikdy nevede ke zlepšení. Co se týče procházení bloků, tedy sestupně (DESC) a vzestupně (ASC), zde nelze najít výrazný rozdíl tak, aby bylo možné říct, že jedno řešení je jasně lepší než to druhé. Jakákoliv volba je tedy správná, přičemž při určitém nastavení a typu instance může být některý přístup výrazněji lepší. Časové nároky lineárně rostou se zvyšující se velikostí bloku nezávisle na směru procházení.



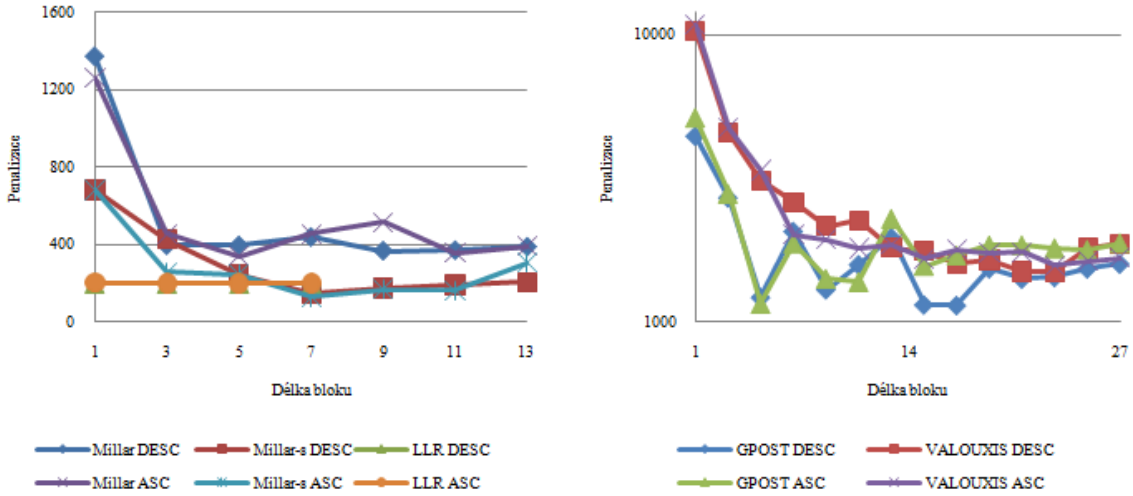
Obrázek 5.1: Vliv délky bloku směn a směru procházení na penalizaci rozvrhu u algoritmu Simulated Annealing



Obrázek 5.2: Vliv délky bloku směn a směru procházení na penalizaci rozvrhu u algoritmu Hill Climbing



Obrázek 5.3: Vliv délky bloku směn a směru procházení na penalizaci rozvrhu u algoritmu Tabu Search

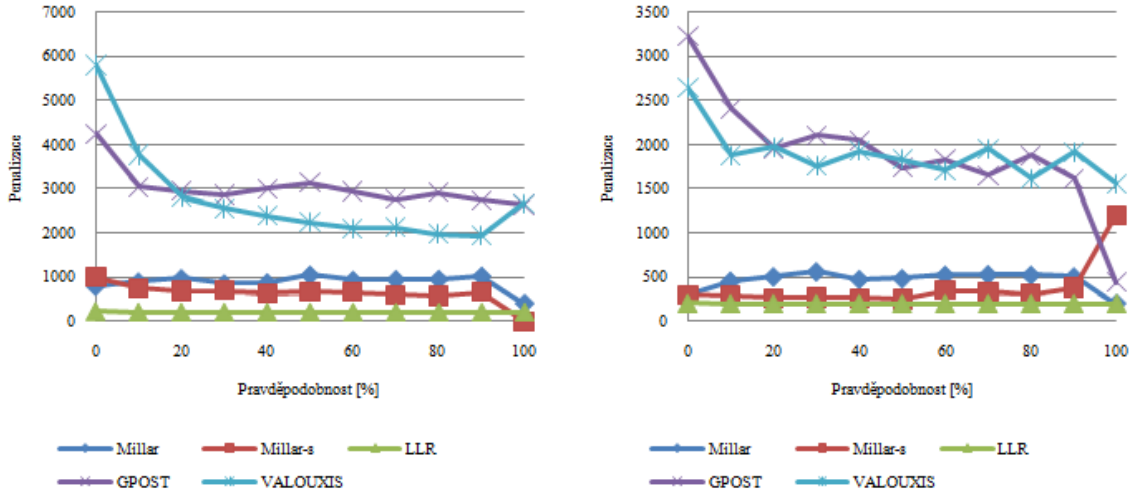


Obrázek 5.4: Vliv délky bloku směn a směru procházení na penalizaci rozvrhu u Chain Algorithm

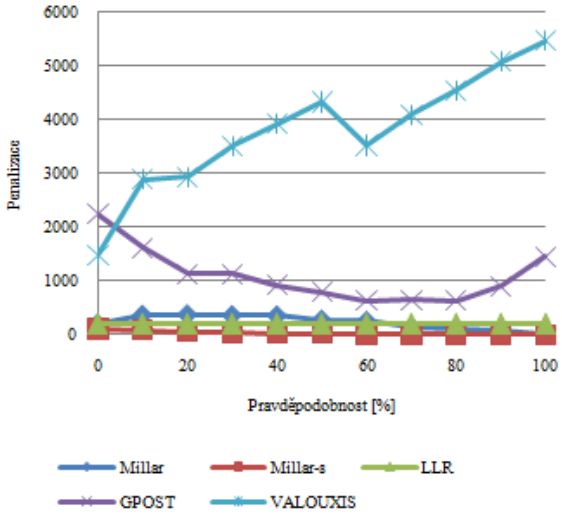
5.3.2 Pravděpodobnost přijmutí stejně ohodnoceného řešení (HC, TS, CHAIN)

Z hlediska časového zde nelze najít rozdíl. V případě deterministického chování (pravděpodobnost se rovná 0% nebo 100%) je kvalita rozvrhu někdy lepší jindy horší než průměrná kvalita u nedeterministického. Nicméně takto se omezíme pouze na jedno možné řešení, což není zrovna žádoucí. Jako ideální hodnota pravděpodobnosti se jeví 40–60%, ostatní ale také nedosahují výrazně horších výsledků, někdy dokonce o něco lepších. V grafu 5.6 si lze všim-

nout rozporu s předchozím tvrzením u instance VALOUXIS. To je pěkný příklad toho, jak pro všechny myslitelné případy nemusí nutně platit obecné dedukce.



Obrázek 5.5: Vliv velikosti pravděpodobnosti přijmutí stejně ohodnoceného řešení na penalizaci rozvrhu u algoritmů Hill Climbing (nalevo) a Chain Algorithm(napravo)



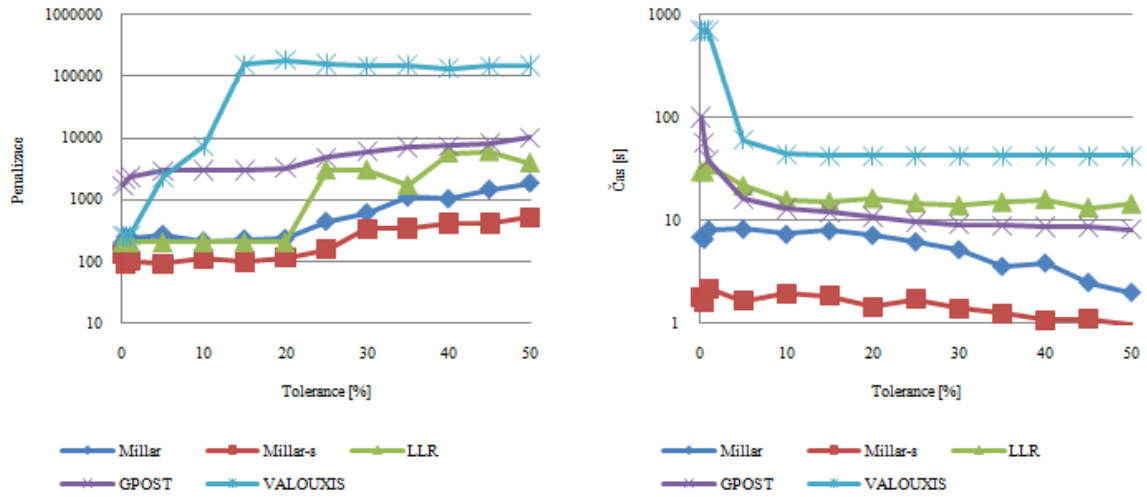
Obrázek 5.6: Vliv velikosti pravděpodobnosti přijmutí stejně ohodnoceného řešení na penalizaci rozvrhu u algoritmu Tabu Search

5.3.3 Restarty (HC)

Tolerance pro určování podobných řešení

Z grafů (5.7) je patrné, hlavně u složitějších instancí, jak moc významné je správné nastavení tolerance. Za pozornost stojí, že není moc důležité, jaké se používají váhy měkkých

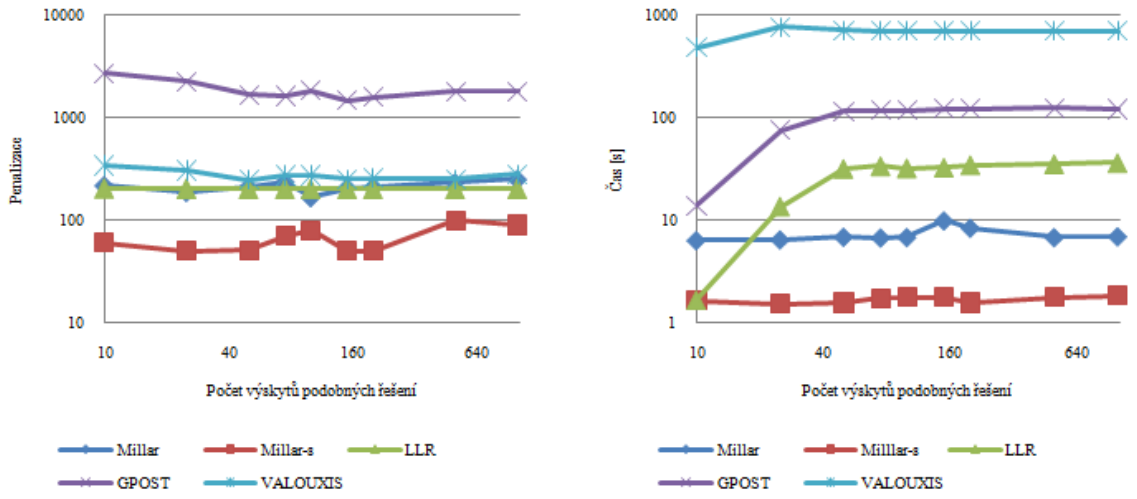
omezení u jednotlivých instancí. Respektive to má vliv pouze na to, od jakého procenta se začne penalizace viditelně zhoršovat. Čím větší váhy (např. u instance Millar jsou to stovky a tisíce), tím lze nastavit toleranci výš. Pokud se totiž přičte 1% nebo 10% ke 100 (výsledky 101 a 110), tak takových hodnot penalizace nejde dosáhnout ani v jednom případě (u instance Millar), a proto bude chování velmi podobné (ne-li stejné). Jako vhodná hodnota se jeví 0–1%. Doba běhu algoritmu prakticky kopíruje křivku penalizace, akorát je horizontálně otočena kolem své osy, tzn. na začátku je čas největší a postupně klesá. Je tomu tak, protože při nízké toleranci dochází k pozdějšímu restartu, a tudíž se musí procházet daleko větší část stavového prostoru. Naopak, pokud je tolerance vysoká, restarty se provádí velmi rychle a výsledný čas je pak menší.



Obrázek 5.7: Vliv velikosti tolerance pro určování podobných řešení na penalizaci a čas rozvrhu u algoritmu Hill Climbing

Počet výskytů podobných řešení

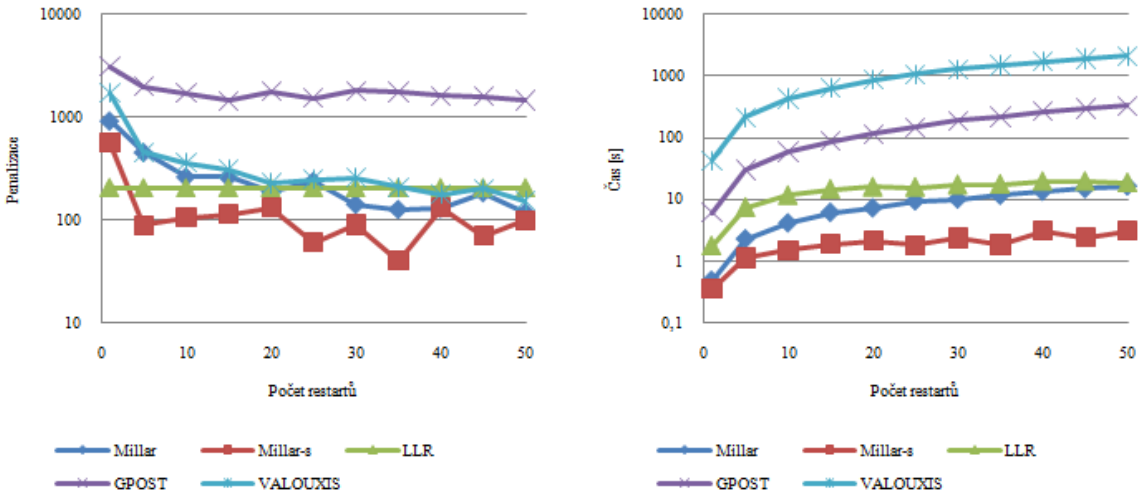
Tento parametr se hodně odvíjí od zvolené tolerance. Pokud je nastavena „dobře“, tak pak už nijak výrazně vzhledem k penalizaci nezáleží na tom, jaký se vybere počet výskytů podobných řešení. Rozdíly zde ale jsou. Nevhodná je nízká hodnota (kolem 10), která sice snižuje výsledný čas, ale poskytuje nejhorší kvalitu rozvrhu. Na druhou stranu, hodnoty zhruba od 50 znamenají lepší penalizaci, ale s vyššími časovými nároky. Vhodným kompromisem mezi kvalitou a časem je tedy hodnota 10–50.



Obrázek 5.8: Vliv počtu výskytů podobných řešení na penalizaci a čas rozvrhu u algoritmu Hill Climbing

Počet restartů

Pokud budeme ignorovat drobné odchylky, tak lze jednoznačně konstatovat, že čím více restartů, tím lepší penalizace a zároveň horší čas. Na tom není nic překvapujícího, protože každým restartem se spouští algoritmus znova. Právě díky restartům může algoritmus Hill Climbing překonávat lokální extrémy, což lze vidět i v grafu 5.9, kdy už při pěti restartech je řešení až 5krát kvalitnější než bez nich.

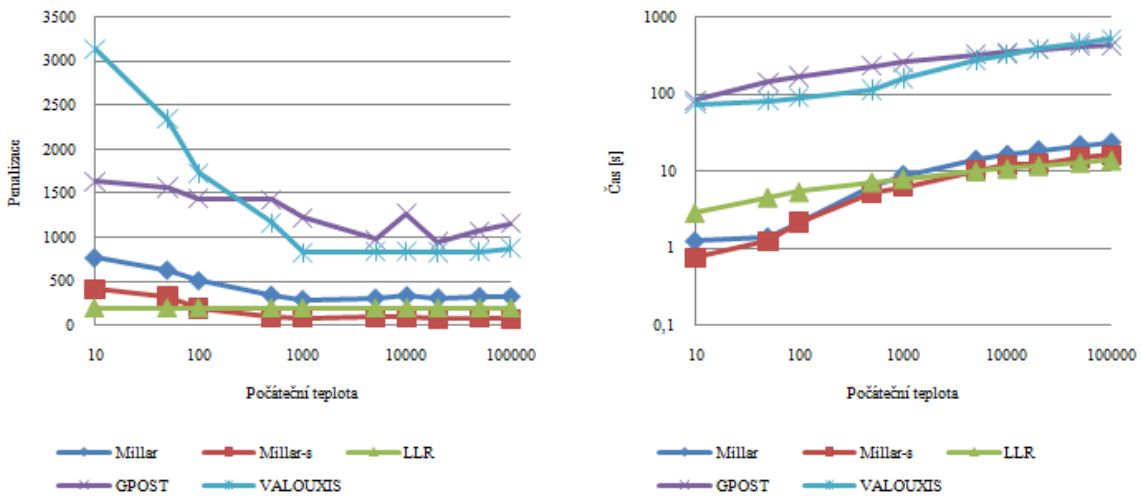


Obrázek 5.9: Vliv počtu restartů na penalizaci a čas rozvrhu u algoritmu Hill Climbing

5.3.4 Chain Algorithm

Počáteční teplota

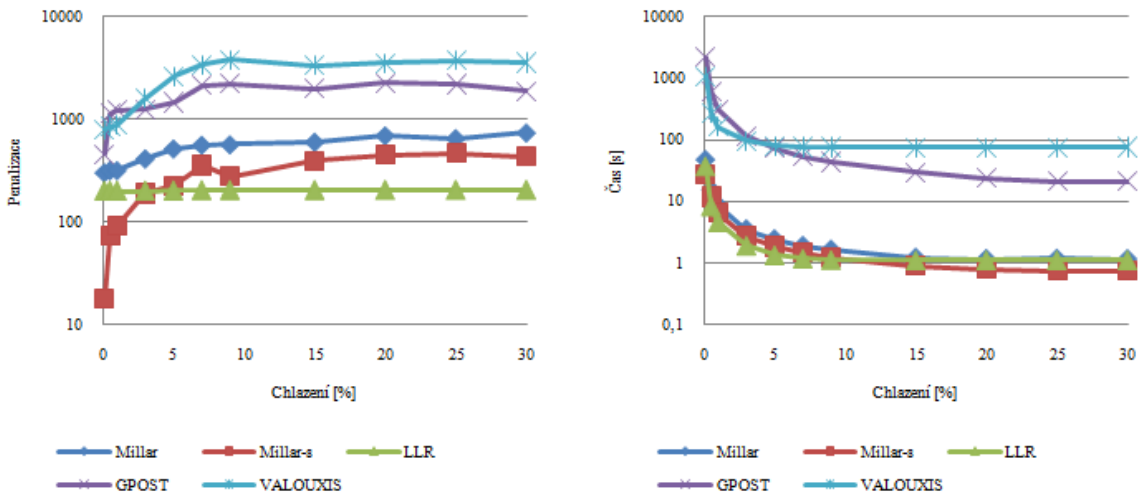
Počáteční teplota v této heuristice udává, jak moc velké zhoršení lze vložit do řetězu. Jako optimální hodnota vzhledem k času a penalizaci se jeví 1000. Je nutné dát pozor na to, že čím větší se použije hodnota teploty, tím vyšší bude výsledný čas. Vysoké hodnoty totiž znamenají přijímaní horších částečných změn po delší dobu, takže proces nasycení (není možné nalézt žádné zlepšení) řetěze trvá déle. Naopak s menší teplotou hrozí uváznutí v lokálním extrému. Velikost teploty také záleží na hodnotě penalizace dané instance, protože pokud se bude pohybovat v miliónech a teplota tisících, tak algoritmus nebude schopen přijmout horší řešení. V opačném případě to problém není, jen by došlo ke zbytečnému plýtvání časem.



Obrázek 5.10: Vliv počáteční teploty na penalizaci a čas rozvrhu u Chain Algorithm

Chlazení

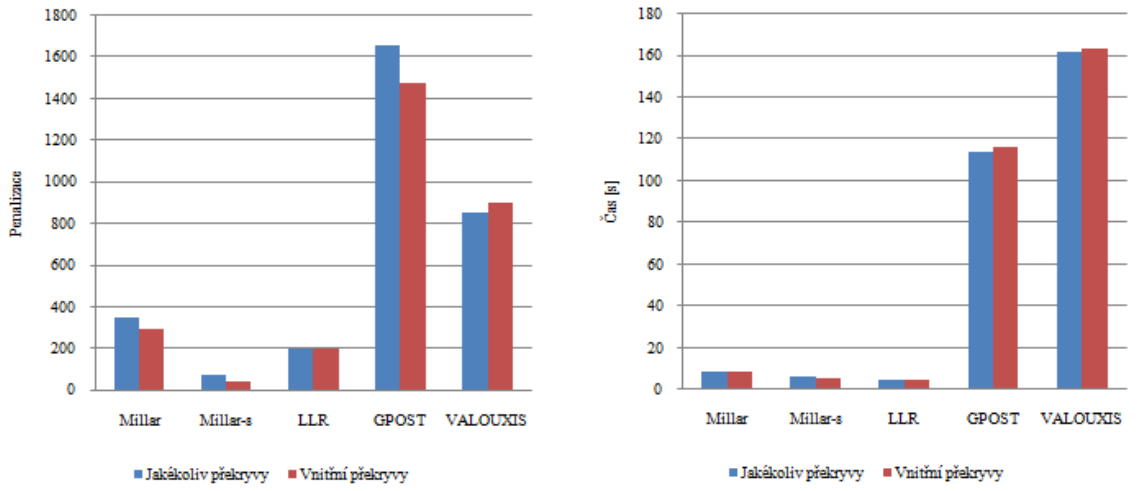
Chlazení úzce souvisí s počáteční teplotou, protože přímo určuje rychlost její konvergence k nule. Čím pomaleji se snižuje, tím déle dochází k přijímání horších řešení a zároveň se snižuje riziko uváznutí v lokálním extrému. Chlazení je vhodné volit v rozmezí 0–1%. Zhruba od dvou procent dochází ke zlomu, kdy se znatelně zhorší penalizace a čas se začne snižovat, protože dochází k rychlé konvergenci.



Obrázek 5.11: Vliv snižování teploty na penalizaci a čas rozvrhu u Chain Algorithm

Vyhledávání kolizí

Zde bylo cílem rozhodnout, který ze dvou přístupu bude používán pro vyhledávání kolizí. Méně restriktivnější pravidlo, tedy vnitřní překryvy, vykazuje nepatrně lepší výsledky, a proto bylo zvoleno. Rozhodně by ale nebylo chybou ani vybrání vnějších překryvů, které kromě jedné instance vykazují velmi podobné penalizace jako překryvy vnitřní. Z hlediska času je rozdíl zanedbatelný.



Obrázek 5.12: Srovnání metod pro vyhledávání kolizí vzhledem k penalizaci a času rozvrhu u Chain Algorithm

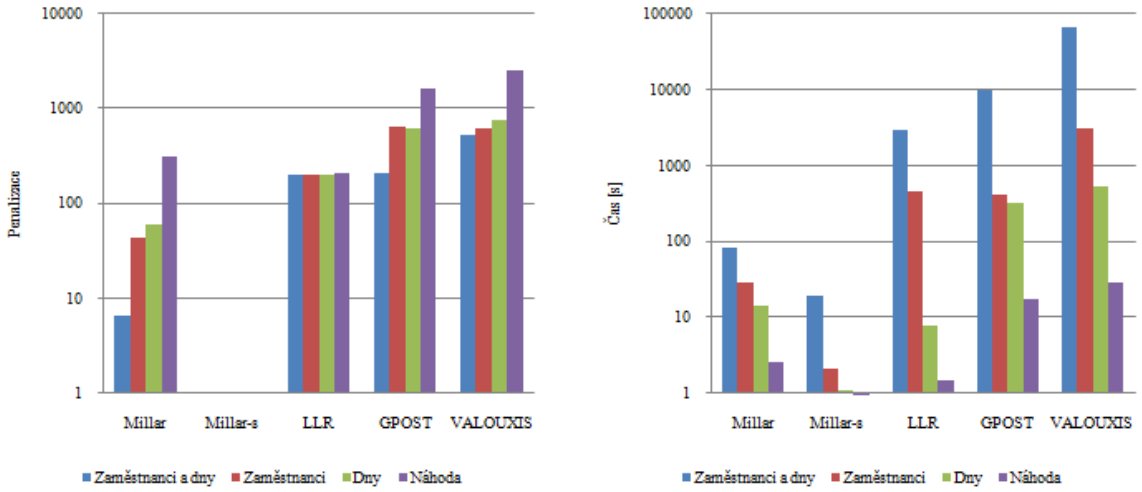
5.3.5 Simulated Annealing

Metody procházení sousedních rozvrhů

Iterace přes:

- zaměstnance a dny – nejkvalitnější rozvrhy, ale velmi časově náročné,
- zaměstnance – dobrá penalizace, horší čas,
- dny – nejlepší kompromis,
- náhodně – velmi rychlé s obstojnou kvalitou rozvrhu.

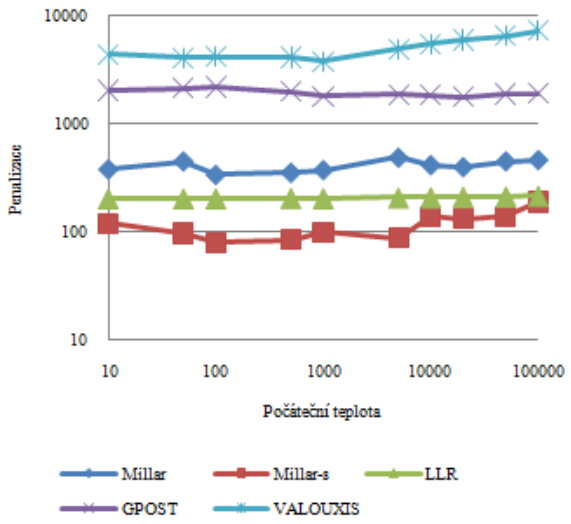
Byly zvoleny náhodné iterace kvůli jejich rychlosti, která se využije nejen u samotného simulovaného žíhání, ale hlavně ve spojení s TPVDS a SS.



Obrázek 5.13: Srovnání metod pro procházení sousedství vzhledem k penalizaci a času rozvrhu u algoritmu Simulated Annealing

Počáteční teplota (náhodné procházení)

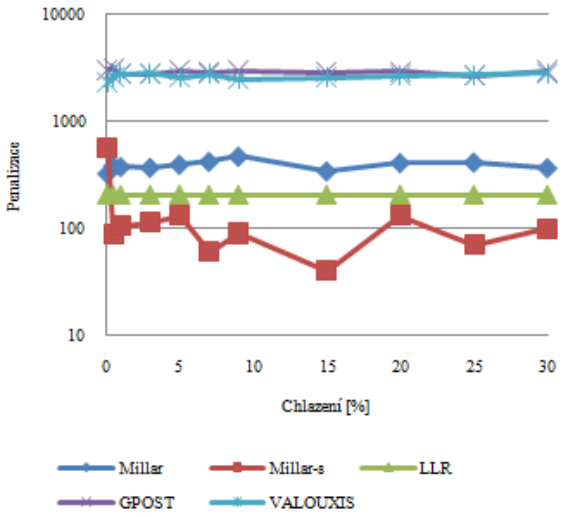
I zde platí závěry, které byly učiněny v podkapitole 5.3.4 s tím, že počáteční teplota udává, jak moc horší rozvrh je možné přijmout vůči nejlepšímu dosud nalezenému a doba běhu algoritmu je neměnná, jelikož je určena počtem předem daných kroků. V tomto grafu 5.14 nelze tak zřetelně pozorovat nekvalitní řešení při nízké teplotě jako u 5.10, což je způsobeno metodou náhodného procházení, která tyto rozdíly částečně smazává.



Obrázek 5.14: Vliv počáteční teploty na penalizaci rozvrhu u algoritmu Simulated Annealing

Chlazení (náhodné procházení)

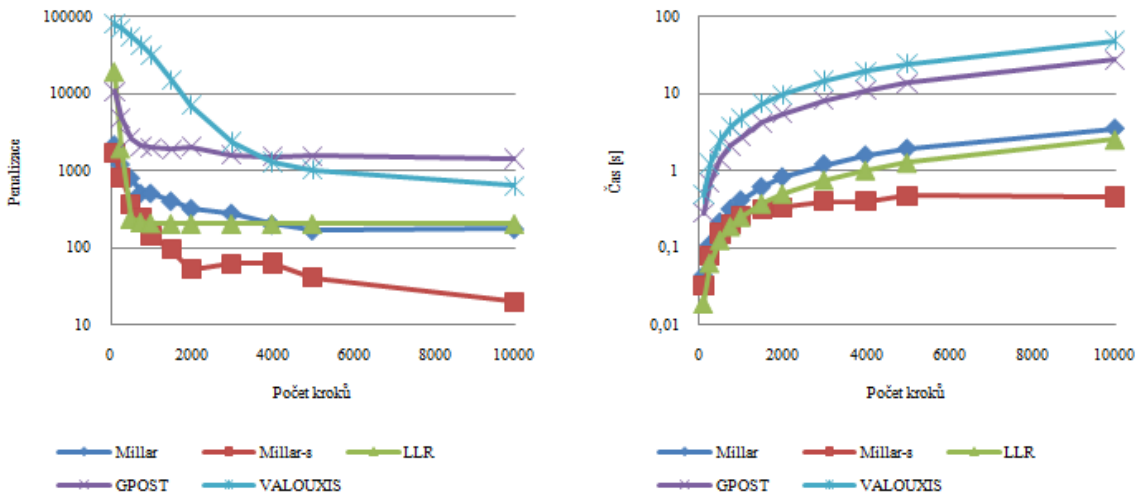
U chlazení v simulovaném žíhání také platí informace uvedené v podkapitole 5.3.4 s tím, že doba běhu algoritmu je neměnná.



Obrázek 5.15: Vliv snižování teploty na penalizaci rozvrhu u algoritmu Simulated Annealing

Počet kroků (náhodné procházení)

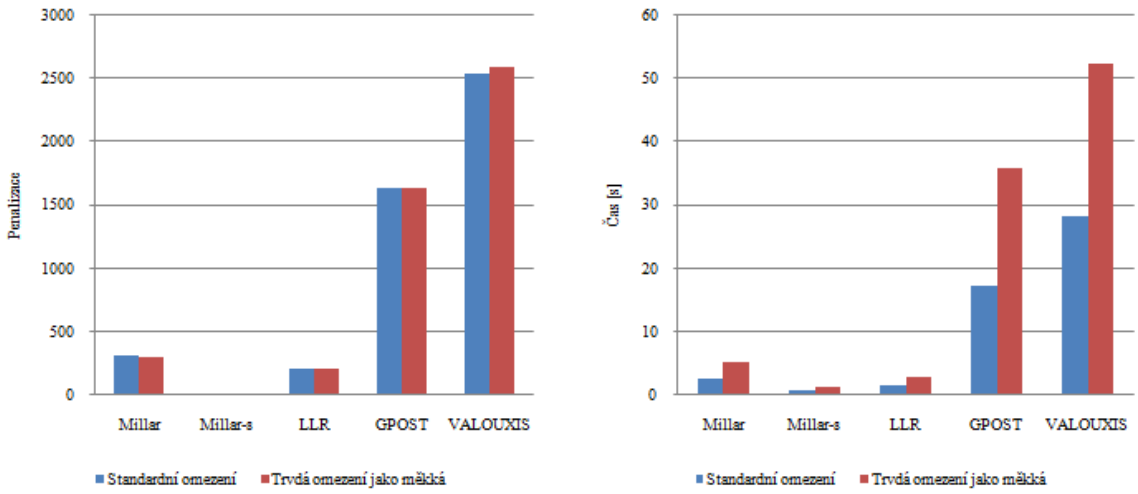
Ze získaných výsledků přirozeně vyplývá, že čím více kroků algoritmus udělá, tím má nižší hodnotu penalizace a běží delší dobu. Jako rozumný kompromis mezi těmito dvěma složkami se jeví hodnota v rozmezí 2000–4000.



Obrázek 5.16: Vliv počtu kroků na penalizaci a čas rozvrhu u algoritmu Simulated Annealing

Tvrdá omezení modelovaná jako měkká (náhodné procházení)

Z hlediska penalizace tato modelace nepřináší žádné zlepšení a z hlediska času je na tom zhruba dvakrát hůře. Rozdíl je způsoben tím, že pokud kontrola tvrdých omezení narazí na nějaké porušení, tak jej nahlásí a dále už nepokračuje v prohledávání. Zatímco u měkkých omezení je ho potřebné dokončit vždy celé. Proto nebyla tato modifikace dále používána.

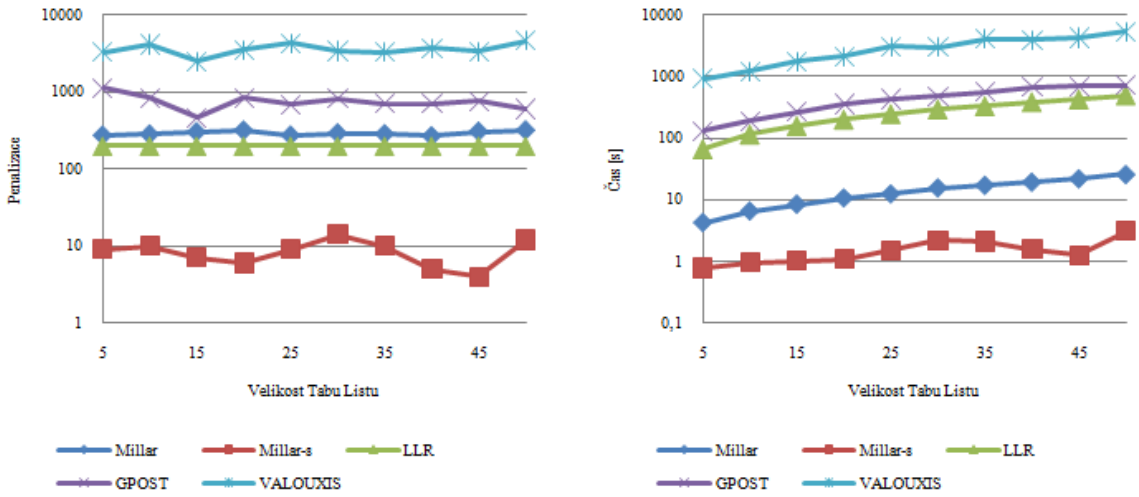


Obrázek 5.17: Srovnání metod pro modelaci tvrdých omezení vzhledem k penalizaci a času rozvrhu u algoritmu Simulated Annealing

5.3.6 Tabu Search

Velikost Tabu Listu

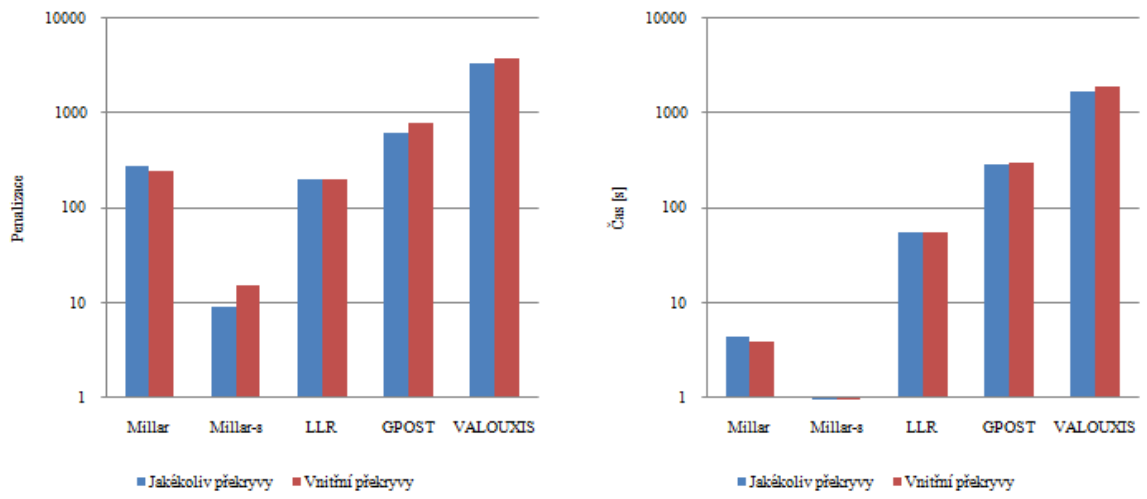
Velikost Tabu Listu je přímo uměrná času potřebného k běhu algoritmu. Malá velikost dosahuje horší kvality rozvrhu z důvodu nedostatečné ochrany před uváznutím v lokálním extrému. Velká je zase kontraproduktivní, protože zbytečně prodlužuje čas a taky může značně omezit vyhledávání sousedních rozvrhů. Z pozorování vyplývá, že nejlepší je rozmezí 10–25, přičemž 15 se projevuje nejlépe.



Obrázek 5.18: Vliv velikosti Tabu Listu na penalizaci a čas rozvrhu u algoritmu Tabu Search

Vyhledávání kolizí

Na rozdíl od vyhledávání kolizí u Chain Algorithm v podkapitole 5.3.4 se zde jeví lépe restriktivní varianta, tedy jakékoliv překryvy. Navíc je to i o něco jasnější, zvláště u penalizace.



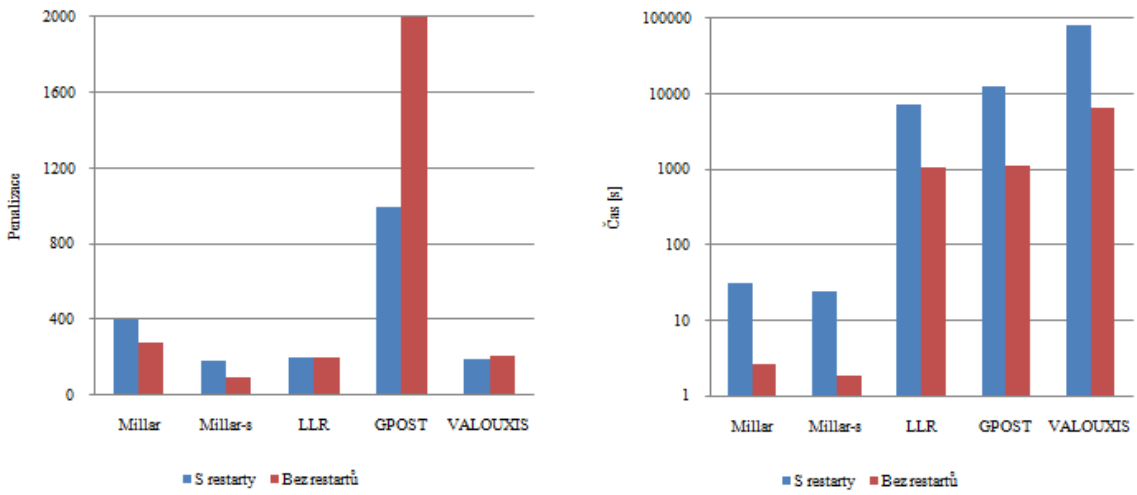
Obrázek 5.19: Srovnání metod pro vyhledávání kolizí vzhledem k penalizaci a času rozvrhu u algoritmu Tabu Search

5.3.7 Time Pre-defined Variable Depth Search

Varianta algoritmu Hill Climbing

Rozhodovalo se mezi použitím algoritmu Hill Climbing s restarty nebo bez. Nutno podotknout, že u TPVDS není absence restartů na škodu, protože má své vlastní mechanismy proti zamezení uváznutí v lokálním extrému. Navíc HC bez restartů je mnohem rychlejší než s nimi (viz. kapitola 5.3.3).

Kdyby nebylo výkyvu penalizace u instance GPOST, dalo by se konstatovat, že jednoznačně lepší variantou je HC bez restartů. Nicméně tam je, ale znamená „pouze“ dvakrát horší řešení. U ostatních je to zhruba na stejno s tím, že mírně lepší výsledky vykazuje HC bez restartů. Z hlediska času je už ale suveréně rychlejší (v průměru 10krát). Proto byla zvolena varianta bez restartů.

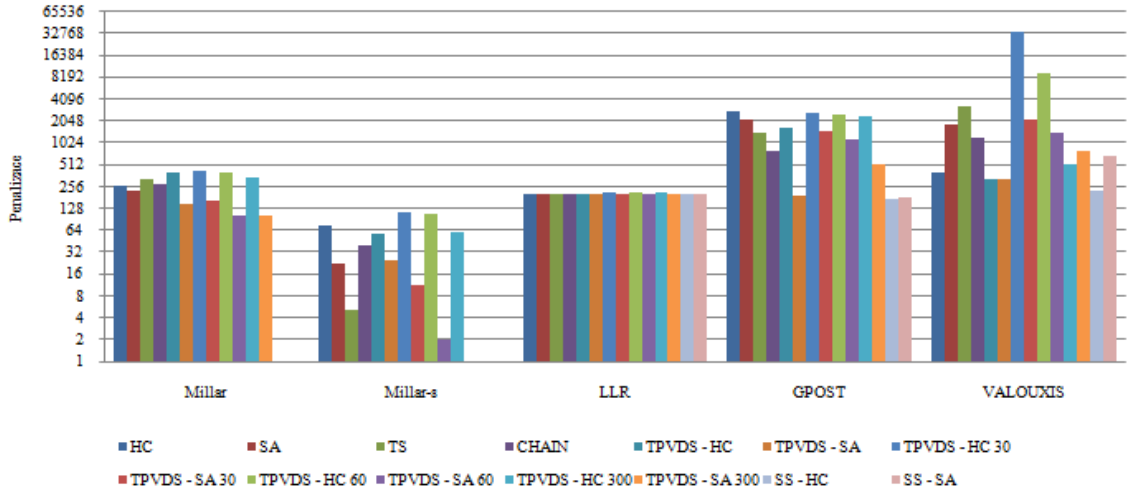


Obrázek 5.20: Srovnání variant algoritmu Hill Climbing vzhledem k penalizaci a času rozvrhu u algoritmu TPVDS

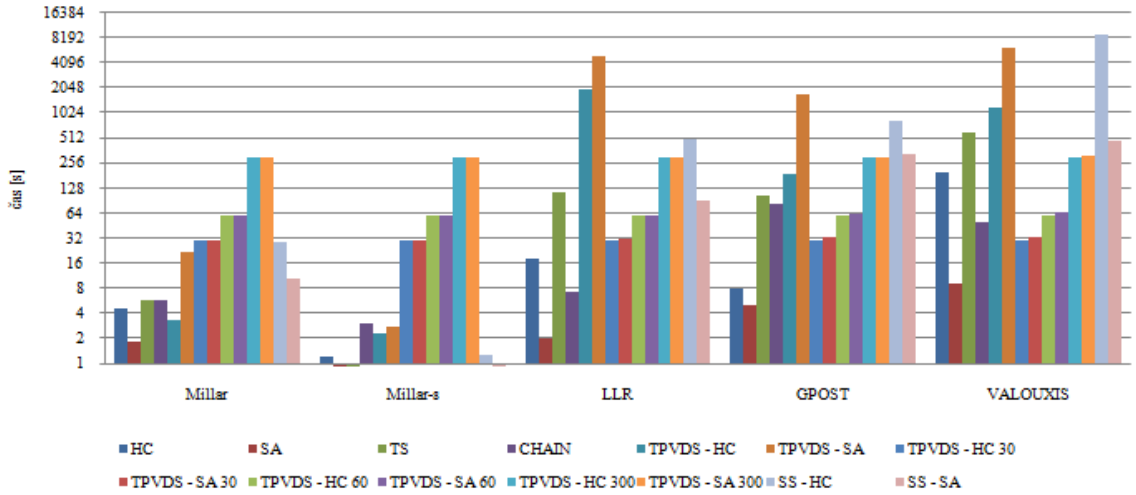
5.4 Porovnání algoritmů na benchmarkových instancích

Nastavení algoritmů pro benchmarky bylo voleno takovým způsobem, aby vycházelo z poznatků předešlé kapitoly 5.2 a bylo použitelné globálně (pro všechny instance stejně):

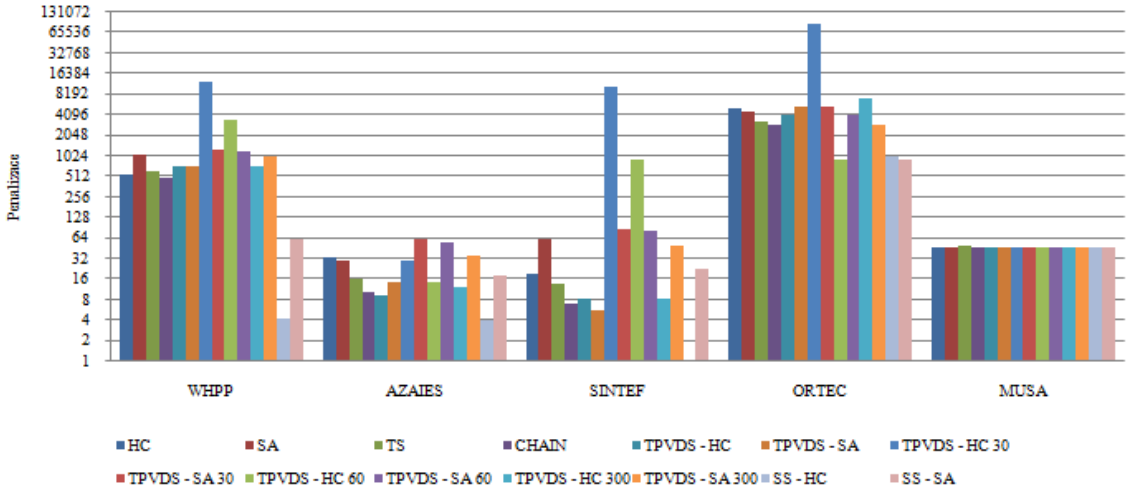
- délka bloku – 6 (HC, SA, TS, CHAIN),
- směr procházení bloků – sestupně (HC, SA, TS, CHAIN),
- pravděpodobnost přijmutí stejně ohodnoceného řešení – 50% (HC, TS, CHAIN),
- počet restartů – 15 (HC),
- tolerance pro podobná řešení – 1% (HC),
- počet výskytů podobných řešení – 35 (HC),
- počáteční teplota – 1000 (SA, CHAIN),
- chlazení – 1% (SA, CHAIN),
- počet kroků – 4000 (SA),
- velikost Tabu Listu – 15 (TS),
- hloubka procházení – 5 (TPVDS),
- časový limit – bez omezení, 30s, 60s a 300s (TPVDS).



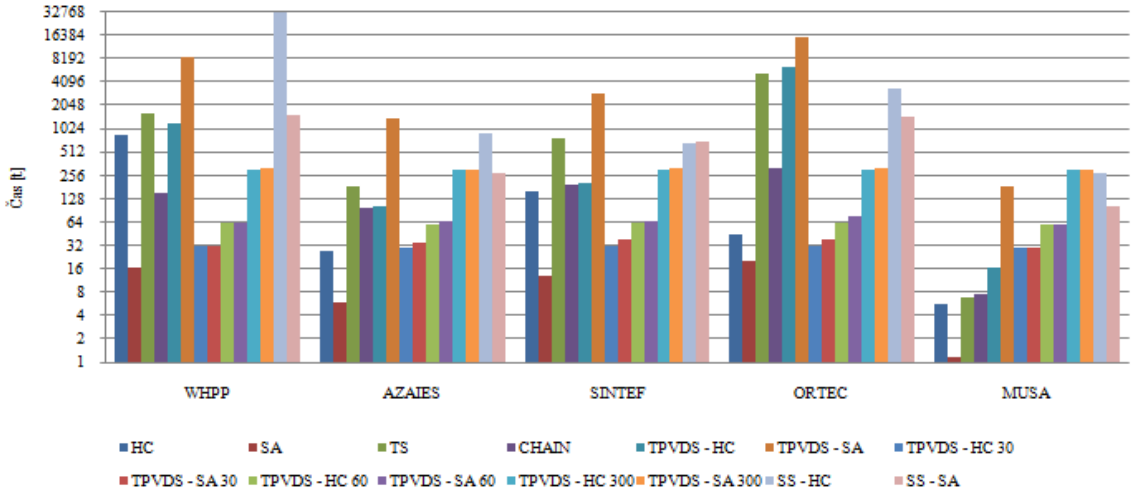
Obrázek 5.21: Srovnání algoritmů na benchmarkových instancích vzhledem k penalizaci I



Obrázek 5.22: Srovnání algoritmů na benchmarkových instancích vzhledem k času I



Obrázek 5.23: Srovnání algoritmů na benchmarkových instancích vzhledem k penalizaci II



Obrázek 5.24: Srovnání algoritmů na benchmarkových instancích vzhledem k času II

5.4.1 Diskuze nad dosáhnutými výsledky

Algoritmy Hill Climbing spolu se Simulated Annealing vykazují nejlepší časy s místy průměrnou až nejhorší dosaženou penalizací.

Výsledky algoritmu Tabu Search nijak nevyčnívají nad ostatními. Dosahuje penalizací, které jsou průměrné, ale čas se řadí k těm horším. Velkou výhodou je malý rozptyl mezi hodnotami nejlepší a nejhorší nalezené penalizace.

Chain Algorithm se pyšní velmi dobrou průměrnou penalizací, kterou prakticky překonávají jenom daleko časově náročnější heuristiky TPVDS a SS. Dobré výsledky dosahuje především u větších instancí. Čas, který potřebuje pro svůj běh, patří mezi ty nejlepší.

Pro TPVDS je lepší upřednostnit jako zlepšovací algoritmus SA před HC, protože ve většině případů vykazuje lepší penalizaci a v těch zbylých moc nezaostává. Pokud není zapnuto časové omezení, tak je jednoznačně rychlejší HC (místy až 10krát). Časový limit je vhodné ponechat na 300 sekundách. Za tuto dobu je schopný TPVDS-SA vracet jednu z nejlepších průměrných penalizací (lepší už vrací pouze SS).

Scatter Search lze bezpochyby považovat za nejlepší testovaný algoritmus. Jeho průměrné hodnoty penalizace jsou prakticky nejlepší. Je schopný nalézt řešení, která odpovídají těm optimálním (pokud existují) nebo se jim alespoň velmi přiblížit. V některých případech překonává i nejlepší známá řešení generovaná programem Roster Booster. Z časového hlediska ale patří k těm nejnáročnějším. Ve variantě s SA je zhruba 2x rychlejší oproti SS-HC, která zase vykazuje lepší průměrné penalizace.

U menších či jednoduchých instancí byly schopny všechny algoritmy nalézt optimální nebo nejlepší známé řešení. U těch větších a složitějších se jim dokázaly minimálně přiblížit. Všechny implementované (meta)heuristiky byly schopné vyřešit problém rozvrhování směn a bylo dosaženo uspokojivých výsledků. Na závěr je nutné podotknout, že nejlepší známé výsledky uvedené v kapitole 5.2 byly často dosahovány se specializovanými heuristiky přímo pro konkrétní instanci a hlavně běžely hodiny až dny. V této práci jsou to maximálně hodiny (převážně minuty až desítky minut).

Kapitola 6

Testování

Tak, jako je analýza a návrh aplikace nedílnou součástí každého většího SW produktu, je i testování velmi důležitým a nezanedbatelným faktorem. Testuje se jak v průběhu vývoje, tak na jeho konci. Cílem je nalezení co největšího počtu chyb nebo problémů a jejich následné opravení. Nikdy se nedocílí toho, aby byl systém absolutně bezchybný. Nicméně kvalitním testováním lze dosáhnout odstranění většiny signifikantních chyb, na které by bylo možné během práce s aplikací narazit.

6.1 Testování funkcionality

Testování funkcionality slouží ke zjištění, zdali daná aplikace splňuje všechny funkční požadavky na ní kladené. Samozřejmě se ověřuje i správná činnost jednolitých komponent, tj. vrácení korektního výstupu s ohledem na vstupní data.

6.1.1 Jednotkové testy

Jednotkové testy (unit testy) se používají k testování malých částí kódu. Jedná se o takové nejmenší fragmenty, které mohou být otestovány. Typicky jsou to funkce. Normálním jevem v praxi je často měnící se kód, kdy ale funkcionalita zůstává zachována. Právě pro tyto případy se vyplatí použití jednotkových testů, které jsou schopny velmi rychle zkontrolovat, zda byla zachována požadovaná funkcionalita a neobjevily se regresní chyby.

Pro zápis a spuštění unit testů byl použit vnitřní mechanismus Visual Studia 2010, přes který lze na pár kliků vytvořit a spustit potřebné testy. Funguje to takovým způsobem, že se zadá patřičný vstup a očekávaný výstup. Při testování se pak zavolá daná metoda se zadaným vstupem a výsledek jejího volání se porovná s očekávaným výstupem. Pokud dojde ke shodě, je test vyhodnocen jako úspěšný, v opačném případě jako neúspěšný.

Bohužel v této práci není možné využít jednotkových testů ve větší míře, protože se aplikace sestává převážně z nedeterministických metod, kde není jak ověřovat výstup, jelikož může nabývat různých hodnot pro stejný vstup. U funkcí, které nemají tyto „problémy“, jsou již unit testy normálně použity a také vedly k odhalení několika drobných regresních chyb, které by jinak bylo poměrně složité hledat.

6.1.2 Benchmarky (srovnávací testy)

Nedá se říci, že by se jednalo o ryzí metodu funkcionálního testování, ale díky ní lze také zkontrolovat správné chování rozvrhovacích algoritmů. Víme totiž, jaké by mělo být optimální (případně nejlepší známé) řešení a pokud jej heuristika dokáže najít, zřejmě bude pracovat dobře. I když jej nenalezne, tak je možné ověřit, že daný rozvrh je proveditelný a má opravdu avizovanou kvalitu.

Navíc při provádění experimentů aplikace běží a počítá různé instance i několik dnů v kuse. Pokud nedojde k pádu systému či jiným problémům, lze konstatovat, že je systém stabilní a nedochází v něm k nějakým skrytým vadám projevujících se časem.

Během vykonávání benchmarků nenastala žádná chyba.

6.2 Testování výkonnosti

Po dokončení implementace každého algoritmu pro rozvrhování byl spuštěn tzv. Profiler (součást Visual Studia 2010 Ultimate). Díky němu lze analyzovat chování aplikace z několika pohledů:

- využití procesoru,
- využití operační paměti,
- volání funkcí a času v nich stráveném.

Využití procesoru nebylo potřeba vůbec optimalizovat, protože v klidovém stavu systému a po dobu běžné práce s ním se pohybuje při nejhorším lehce nad nulou. Při tvorbě rozvrhu se logicky vyšplhá na 100% pro jedno jeho jádro (zbylá jádra nejsou zatížena). Z hlediska využití operační paměti je aplikace poměrně nenáročná, jelikož v průběhu práce se prakticky nemění a pohybuje se kolem 45MB.

Zajímavější však byla analýza času stráveného v jednotlivých funkcích. V Profileru se zobrazují všechny volané funkce spolu s časem vyjadřujícím dobu jejích vykonávání. Ty, které trvají nejdéle, jsou potencionálními kandidáty na optimalizaci kódu (ne vždy je to samozřejmě možné). Po provedení všech úprav se znovu spustí Profiler a jeho výsledek je porovnán s tím předchozím. V porovnání je možno vidět, které funkce se zrychlily a které zpomalily. Poté se znovu provádí iterace operací analýza pomocí nástroje Profiler $\leftrightarrow$ úprava kódu, dokud dochází ke zlepšení výsledných časů.

Profiler se velmi osvědčil při tvorbě prvních algoritmů, kdy bylo díky optimalizaci kódu dosaženo až 30% zrychlení. Při implementaci dalších algoritmů nebylo již po první iteraci zlepšení tak markantní, protože už bylo známo, čeho se vyvarovat a jak správně postupovat při implementaci vzhledem k efektivitě kódu.

Hot Path

The most expensive call path based on sample counts

Function Name	Inclusive Samples %	Exclusive Samples %
TimetableProject.TimetableClasses.TPVDS.improveTimetable()	96,76	1,33
TimetableProject.TimetableClasses.Algorithms.HillClimberAlgorithm.getUnvisitedNei...	48,53	1,94
TimetableProject.TimetableClasses.ConstraintsContainer.checkSoftConstraints(int...	46,08	0,14
TimetableProject.TimetableClasses.Constraints.PatternConstraint.checkIndivid...	45,74	10,62
System.Collections.Generic.List`1.Contains(!0)	**14,79**	**14,79**

Related Views: Call Tree Functions

Obrázek 6.1: Výstup z Profileru – kritická cesta

Functions Doing Most Individual Work

Functions with the most exclusive samples taken

Name	Exclusive Samples %
System.Collections.Generic.List`1.Contains(!0)	29,69
TimetableProject.TimetableClasses.Constraints.PatternConstraint.checkIndividualConstraint(class TimetablePr...	20,95
JIT_GetSharedGCStaticBaseDynamicClass(unsigned long,unsigned long)	19,85
System.DateTime.get_DayOfWeek()	4,48
System.Collections.Generic.List`1.get_Item(int32)	4,28

Obrázek 6.2: Výstup z Profileru – nejvíce pracují funkce

6.3 Testování uživatelského rozhraní

Nestačí testovat pouze funkčnost a výkonnost, ale je nutné se zaměřit také na uživatelské rozhraní (UI), které ve většině případů rozhoduje o úspěchu či neúspěchů aplikace. Zákazník (uživatel pracující se systémem) se nezajímá o to, že program poskytuje „bezchybnou a nejlepší funkcionalitu na světě“, ale o pohodlnou a příjemnou práci s ním. V případě, že aplikace bude z hlediska UI nepoužitelná, tak jí nepomůže ani obrovské množství funkcionality.

Fakt, že aplikace běží v rámci MS Excel, má své výhody a nevýhody. Plusem je využití již zaběhlého a kvalitního UI poskytované MS Excelem. Zároveň je to také mínusem, protože možnost ovlivnit jeho UI je velmi malá.

Níže uvedené testy uživatelského rozhraní jsou poměrně strohé, jelikož:

1. velké změny UI jsou nemožné,
2. aplikace je velmi jednoduchá, a proto nabízí malé množství interakce s uživatelem.

Profil uživatelů

Uživatelé, kteří budou pracovat se systémem, jsou charakterističtí tím, že zvládají bez problému základy práce s PC a hlavně umí dělat s kancelářským balíkem Microsoft Office. To je velmi důležitá informace, protože celá aplikace je právě kvůli tomu postavena na MS Excel. Jiná skupina není uvažovaná.

6.3.1 Kognitivní průchod

Jedná se prakticky o to samé, jako v případě testů použitelnosti prováděných na uživatelích. Rozdíl je právě v těch uživatelích, kdy u kognitivního průchodu figuruje jako tester sám návrhář (nebo nějaký jiný člen vývoje). Ten ale nevystupuje sám za sebe, ale musí se vžít do role klasického uživatele aplikace (viz. profil uživatele uvedený výše). Výhodou je absence testovacích osob, což nepochybně vede k úspoře času a peněz. Navíc se tímto průchodem odstraní na první pohled zřejmé prohřešky, což má za následek, že se uživatelé u testu použitelnosti budou moci soustředit na jiné (pro vývojáře skryté) věci.

V testu je nutné zohlednit tyto otázky:

1. Vím, kde právě jsem a jak budu pokračovat?
2. Mám k dispozici akci, kterou chci provést?
3. Je vybraná akce správná?
4. Chápu zpětnou vazbu po provedení zvolené akce?

Zaměření testu:

- přehledná struktura aplikace,
- dostupnost akcí,
- jednoduchost zadávání dat,
- korektnost zobrazovaných dat a výsledků.

Testované případy užití:

- kontrola rozvrhu,
- tvorba rozvrhu.

Výsledek

Byly provedeny dva průchody. V rámci každého došlo k otestování obou dvou případů užití. Během prvního bylo nalezeno pár drobných nedostatků (tabulka 6.1).

Problém	Řešení
Nedostatečná, respektive žádná, strukturalizace nastavení systému.	Rozdělení do tematických bloků.
Informace na akčních tlačítkách nestačí k popisu práce s aplikací.	Přidána doprovodná nápověda krok po kroku k akčním tlačítkům.
Nelze využít styl pro zmražené směny (viz. kapitola 4.1.1), protože se o něm neví.	Informace přidána do doprovodné nápovědy.

Tabulka 6.1: Nedostatky v UI objevené během kognitivního průchodu

Poté, co byly všechny problémy odstraněny a opraveny, byl proveden druhý průchod, po němž nebyly zjištěny další chyby.

6.3.2 Test použitelnosti

Po provedení kognitivního průchodu a odstranění „závažných“ problémů bylo načase pozvat uživatele k otestování aplikace. Součástí testování je vyplnění dotazníku před a po testu. V mém konkrétním případě nebylo nutné provádět dotazník před testem, protože má za cíl zjistit základní informace o testujících osobách, které už jsem právě znal.

Dotazníky jsou důležitým aspektem k vyhodnocení testu. Pokud se např. o uživateli ví, že je počítačový laik, MS Excel v životě neviděl, tak se nebude zkoumat, proč mu vykonání daného úkolu trvalo několika násobně déle než ostatním. V opačném případě, u zkušeného člověka, by to byla alarmující zpráva a znamenalo by to nejspíše nějaký problém v UI.

Uživatelé

Bylo přizváno celkem pět uživatelů:

- U1 – muž, 22 let, pokročilá znalost PC,
- U2 – žena, 37 let, uživatelská znalost PC – ruční tvorba rozvrhů v MS Excel,
- U3 – žena, 49 let, uživatelská znalost PC – běžné kancelářské práce v MS Excel,
- U4 – muž, 42 let, uživatelská znalost PC,
- U5 – muž, 58 let, základní znalost PC – MS Excel nezná.

Jak je vidět, uživatelé U1 – U4 zapadají do profilu. U5 už ne, ale právě pro to byl zvolen, jelikož může poskytnout cenný náhled na aplikaci, který chybí lidem, jenž se v prostředí MS Excel orientují.

Příprava testu

Testovaným uživatelům byl rozdán vytištěný uživatelský manuál (I), který měli za úkol důkladně nastudovat. Po jeho přečtení měli možnost zeptat se na případné nejasnosti. Nakonec jim bylo vysvětleno, co se od nich očekává:

- otevření souboru MS Excel obsahující integrovanou aplikaci,
- vložení základních dat (zaměstnanci, směny, datum),
- vygenerování tabulek pro vložení dodatečných dat,
- vložení dodatečných dat (kvalifikace zaměstnanců, požadavky na obsazenost směn),
- zadání jednoho měkkého omezení,
- nastavení hlavní heuristiky,
- ruční vytvoření libovolného rozvrhu,
- kontrola předchozího rozvrhu,

- zmražení směn v rozvrhu a odstranění zbytku směn,
- vytvoření rozvrhu pomocí aplikace.

Průběh testu

Každý uživatel byl přiveden k počítači s operačním systémem MS Windows 7. Na stole, u kterého seděli, byl připraven uživatelský manuál a papír s úkoly, které bylo potřeba vyřešit. Já jsem zůstával na místě v roli pozorovatele. Pokud by nastaly nějaké výrazné komplikace, byl jsem připraven pomoci. Po dokončení testu byl rozdán závěrečný dotazník.

Závěrečný dotazník

Otázky:

- O1 – Uvítal/a byste tuto aplikaci u vás v práci?
- O2 – Byla pro vás výhodou práce s MS Excel?
- O3 – Který úkol vám dělal největší starosti a případně proč?
- O4 – Vadilo vám čekání na vygenerování rozvrhu? (poznámka.: pohybovalo se kolem 5 minut)
- O5 – Jste s vygenerovaným rozvrhem spokojeni?
- O6 – Jakou známkou byste ohodnotili přehlednost a jednoduchost aplikace (1 – 10, kde 1 znamená nejlepší a 10 nejhorší)?
- O7 – Co byste zlepšili?

Strukturované odpovědi jsou uvedeny níže v tabulce 6.2.

Výsledek

Podle předpokladů zvládl test nejrychleji U1. U2, U3 a U4 to zvládli za podobný čas, ale oproti U1 třikrát pomaleji. Nejhůře dopadl U5, který byl vůči předchozí skupince ještě dvakrát pomalejší, což se ovšem dalo čekat.

Na základě výsledků testů bylo nutné znovu předělat doplňkovou nápovědu. Dalším problémem byla měkká omezení. Zde si všichni stěžovali a prakticky kromě nejzkušenějšího uživatele U1 to nikdo rozumně nezvládl. Překážkou v tomto případě ale není UI vzorů. Do něj byli ve finále všichni schopni zadat data správně a rychle. Uživatelé nebyli schopni přenést reálná omezení do vzoru. Na základě tohoto největšího nedostatku byl vylepšen manuál, kde bylo důkladně popsáno na několika příkladech, jakým způsobem omezení v aplikaci modelovat.

Otázka / uživatel	U1	U2	U3	U4	U5
O1	Jsem student, ale kdybych takovou práci dělal, tak bych to určitě uvítal, protože cokoliv co nahrazuje zbytečnou práci, je super.	Všemi deseti.	Rozvrhy nepoužíváme, takže ne	Je mi to jedno, já rozvrhy nevytvářím.	Je to zajímavé, ale už jsem v důchodu.
O2	Rozhraní, pokud je dobře udělané, neřeším.	Ano.	Ano.	Nevím.	Neumím se vyjádřit, protože tento program vidím poprvé.
O3	Měkká omezení – musel jsem studovat manuál.	Měkká omezení – neuměla jsem si pořádně představit ten vzor.	Měkká omezení – nepochopila jsem, co mám zadat.	Měkká omezení – vůbec jsem to nepochopil.	Měkká omezení – nerozumím tomu, vypsal jsem tam uvedený příklad.
O4	Já myslím, že to bylo dobré.	Když to srovnám s ruční tvorbou, tak je to nic.	Nevadí mi počkat si, během toho můžu dělat jiné věci.	Ano.	Nevadilo, v manuálu bylo psané, že to může chvíli trvat.
O5	Vypadá to pěkně, ale nemám jak posoudit moc spokojenost.	Ano, opravdu kvalitní rozvrh, nevím, jak dlouho bych ho musela dělat ručně.	Nevím.	To ať mi řeknou ti zaměstnanci.	Je hezký.
O6	3	1	2	3	3
O7	Vzhled, nicméně to asi nepůjde.	Přidala bych barvy.	Manuál ohledně měkkých omezení.	Nic mě nenapadá.	Asi nic.

Tabulka 6.2: Odpovědi na závěrečný dotazník

Závěr

Cílem práce byla rešerše algoritmů vhodných pro rozvrhování směn zaměstnanců a jejich implementace spolu se vzájemným porovnáním. Dále bylo úkolem naprogramovat pro řešení tohoto problému vlastní heuristiku, jejíž výkonnost bude ověřena na benchmarkových instancích v kontextu výsledků ostatních algoritmů. Pro tyto cíle bylo nutné naimplementovat aplikaci, kde by bylo možné snadným způsobem zadat vstup tohoto problému, vybrat si potřebný algoritmus, nastavit si jeho parametry a zobrazit řešení. Tato aplikace byla vyhotovena jako rozšíření programu Microsoft Excel a následně ověřena mimo jiné i uživatelskými testy, které pomohly k finálnímu doladění systému, a tím intuitivnější a přehlednější práce s ním. Architektura aplikace byla navrhnuta takovým způsobem, aby byla co nejflexibilnější a umožnila pohodlné vkládání nových algoritmů a omezení.

Výběr (meta)heuristik byl proveden na základě důkladného zmapování odborných článků věnujících se problému rozvrhování. Z výsledků této rešerše vyplynulo rozhodnutí pro použití algoritmů na bázi lokálního prohledávání. Zároveň posloužila i jako hlavní zdroj všech potřebných informací k tomu, aby mohla být tato práce zdárně zhotovena.

Všechny provedené lokální změny v implementovaných algoritmech vedly k lepším výsledkům než u standardních neupravených verzí, což lze pozorovat v kapitole 5 s experimenty, kde je zkoumám nejen vliv jednotlivých lokálních parametrů, ale i porovnání všech použitých algoritmů na několika různých benchmarkových instancích. Vlastní heuristika, tedy „Chain Algorithm“, dosáhla uspokojivých výsledků jak z hlediska průměrné dosažené penalizace, tak času. Poráží ji prakticky jen Scatter Search a časově neomezený TPVDS s dostatečnou hloubkou procházení, které ale pro svůj běh potřebují mnohem více času. Dá se tedy říci, že CHAIN dosahuje jednoho z nejlepších poměrů kvalita:čas.

Během případného pokračování nad vývojem této aplikace se nabízí do její další verze začlenit několik drobných vylepšení. Jedním z nich je možnost přiřazování vzorů jednotlivcům či skupinám. V současné době je totiž systém schopný uvažovat vzory, které jsou pro všechny zaměstnance stejné. Dále by bylo vhodné doplnit matici vztahů zaměstnanců, která udává, jací lidé nemohou pracovat na společné směně. Důvodem je poměrně častý výskyt tohoto omezení, zvláště ve zdravotnických zařízeních. Pro dosažení komplexního srovnání by bylo také zajímavé naimplementovat a vyhodnotit algoritmy memetické a rojové inteligence. V případě memetických algoritmů je možné využít již hotové struktury SS, a to díky jejich velké vzájemné podobnosti.

Zadání a cíle práce byly úspěšně splněny. Dobrý dosáhnutý výsledek byl zaznamenán v podobě vlastního algoritmu CHAIN, který lze označit za zdařilý.

Literatura

[1] ABRAMSON, D. – ABELA, J. A Parallel Genetic Algorithm for Solving the School Timetabling Problem. *Australian Computer Science Conference*. 1992.

[2] AICKELIN, U. – DOWSLAND, K. A. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*. 2000, 3, s. 139–153.

[3] ALLEN, S. *Heuristic and Metaheuristic Algorithms: přednáška* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www.cs.nott.ac.uk/~sda/teaching/alg.ppt>.

[4] ALOUL, F. A. *PBS4 – 0/1 ILP SAT Solver & Optimizer* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www.aloul.net/Tools/pbs/pbs4.html>.

[5] ALOUL, F. A. et al. Solving Employee Timetabling Problems Using Boolean Satisfiability. *IEEE Innovations in Information Technology Conference*. 2006.

[6] BERRE, D. L. – PARRAIN, A. *SAT4J* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www.sat4j.org/products.php>.

[7] BIERE, A. *PicoSAT* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://fmv.jku.at/picosat/>.

[8] BURKE, E. K. – KENDALL, G. – SOUBEIGA, E. A Tabu-Search Hyperheuristic for Timetabling And Rostering. *Journal of Heuristics*. 2003, 9.

[9] BURKE, E. K. – CAUSMAECKER, P. D. Practice and Theory of Automated Timetabling IV. *Lecture Notes in Computer Science*. 2003, 2740.

[10] BURKE, E. K. – GENDREAU, M. Practice and Theory of Automated Timetabling VII. *Lecture Notes in Computer Science*. 2008.

[11] BURKE, E. K. – MCCOLLUM, B. Practice and Theory of Automated Timetabling VIII. *Lecture Notes in Computer Science*. 2010.

[12] BURKE, E. K. – NEWALL, J. Enhancing Timetable Solutions with Local Search Methods. *Lecture Notes in Computer Science*. 2003, 2740, s. 195–206.

[13] BURKE, E. K. – RUDOVÁ, H. Practice and Theory of Automated Timetabling VI. *Lecture Notes in Computer Science*. 2007, 3867.

[14] BURKE, E. K. – TRICK, M. Practice and Theory of Automated Timetabling V. *Lecture Notes in Computer Science*. 2005, 3616.

[15] BURKE, E. K. – NEWALL, J. P. – WEARE, R. F. A Memetic Algorithm for University Exam Timetabling. 1996.

[16] BURKE, E. K. – CAUSMAECKER, P. D. – BERGHE, G. V. A Hybrid Tabu Search Algorithm for the Nurse Rostering Problem. *Second Asia-Pacific Conference*. 1998.

[17] BURKE, E. K. et al. Variable Neighbourhood Search for Nurse Rostering Problems. 2002.

[18] BURKE, E. K. et al. Hyper-Heuristics: An Emerging Direction in Modern Search Technology. *International Series in Operations Research and Management Science*. 2003, 57.

[19] BURKE, E. K. et al. The state of the art of nurse rsotering. *Journal of Scheduling*. 2004, s. 441–499.

[20] BURKE, E. K. et al. A Scatter Search for the Nurse Rostering Problem. 2007.

[21] BURKE, E. K. et al. A Time Pre-defined Variable Depth Search for Nurse Rostering. *INFORMS Journal on Computing*. 2007.

[22] BURKE, E. K. et al. A Hybrid Heuristic Ordering and Variable Neighbourhood Search for the Nurse Rostering Problem. *European Journal of Operational Research*. 2008, 188, s. 330–341.

[23] CARTER, E. – LIPPERT, E. *Visual Studio Tools for Office 2007: VSTO for Excel, Word, and Outlook*. Addison-Wesley Professional, 1st edition, 2009.

[24] CHEANG, B. et al. Nurse rostering problems — a bibliographic survey. *European Journal of Operational Research 151*. 2003, s. 447–460.

[25] CHIARAMONTE, M. V. Competitive Nurse Rostering and Rerostering. *PhD dissertation*. 2008.

[26] CIPRIANO, R. – GASPERO, L. D. – DOVIER, A. Hybrid Approaches for Rostering: a Case Study in the Integration of Constraint Programming and Local Search. *Lecture Notes in Computer Science*. 2006, 4030, s. 110–123.

[27] CURTOIS, T. *Roster Booster* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www.staffrostersolutions.com/>.

[28] DEMASSEY, S. – PESANT, G. – ROUSSEAU, L. Constraint Programming based Column Generation for Employee Timetabling. *Lecture Notes in Computer Science*. 2005.

[29] DEMEL, J. *Grafy a jejich aplikace*. Academia, 1st edition, 2002.

[30] DEMLOVÁ, M. *Teorie algoritmů: text k přednáškám* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://math.feld.cvut.cz/demlova/teaching/tal_vyuka.html>.

[31] DIGALAKIS, J. – MARGARITIS, K. A Performance Comparison of Parallel Genetic and Memetic Algorithms. *Journal of Complexity International*. 2002, 9.

[32] DORIGO, M. – BLUM, C. Ant Colony Optimization Theory:A Survey. *Theoretical Computer Science*. 2005, 344, s. 243–278.

[33] DORIGO, M. – STÜTZLE, T. *Ant Colony Optimization*. The MIT Press, 1st edition, 2004.

[34] DOWSLAND, K. A. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*. 1998, 3, s. 393–407.

[35] EBERHART, R. C. – SHI, Y. Particle swarm optimization: developments, applications and resources. *Evolutionary Computation*. 2001, 1, s. 81–86.

[36] EIKLAND, K. – NOTEBAERT, P. *lp_solve – MILP řešitel* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://lpsolve.sourceforge.net/5.5/>.

[37] ELBELTAGI, E. – HEGAZY, T. – GRIERSON, D. Comparison among five evolutionary-based optimization algorithms. *Advanced Engineering Informatics*. 2005, 19, s. 43–53.

[38] EÉN, N. – SÖRENSSON, N. *MiniSAT* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://minisat.se/>.

[39] ERNST, A. T. et al. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research 153*. 2004, s. 3–27.

[40] Extreme Optimization. *Výpočetní knihovyn pro .NET* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www.extremeoptimization.com/>.

[41] Frontline Systems Inc. *SDK Platform - optimalizace a simulace pro C#* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www.solver.com/sdkplatform.htm>.

[42] GLOVER, F. – KOCHENBERGER, G. A. Handbook of metaheuristics. *International series in operations research and management science*. 2003.

[43] HAN, L. – KENDALL, G. An Investigation of a Tabu Assisted Hyper-Heuristic Genetic Algorithm. *Congress on Evolutionary Computation*. 2003, 3, s. 2230–2237.

[44] HENDERSON, S. G. – MASON, A. J. Rostering by iterating integer programming and simulation. *Winter Simulation Conference*. 1998, 1, s. 667–683.

[45] IBM. *IBM ILOG CPLEX Optimizer* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/>.

[46] KADLEC, V. *Agilní programování - Metodiky efektivního vývoje softwaru*. Computer Press, 1st edition, 2004.

[47] KITADA, M. – MORIZAWA, K. – NAGASAWA, H. Recursive Search Approach in Nurse Rerostering Following a Sudden Absence of Nurses. *Asia Pacific Industrial Engineering and Management Systems Conference 2009*. 2009.

[48] Komunita přispěvatelů. *Open-source .NET vývojový framework* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www.mono-project.com/>.

[49] LEPŠ, M. *Úvod do stochastických optimalizačních metod (metaheuristik): přednáška* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://klobouk.fsv.cvut.cz/~leps/teaching/mmo/prednasky/prednaska04_intro_SA_TA.pdf>.

[50] Microsoft. *Architektura Application-Level Add-Ins* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://msdn.microsoft.com/en-us/library/bb386298.aspx>.

[51] Microsoft. *Architektura Document-Level Customizations* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://msdn.microsoft.com/en-us/library/zcfbd2sk.aspx>.

[52] Microsoft. *Visual C#* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://msdn.microsoft.com/en-us/library/kx37x362.aspx>.

[53] Microsoft. *Visual C# vývojářské centrum* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://msdn.microsoft.com/en-GB/vcsharp/>.

[54] Microsoft. *Přehled vývoje nad Microsoft Office s VSTO* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://msdn.microsoft.com/en-us/library/hy7c6z9k.aspx>.

[55] Microsoft. *Dostupné vlastnosti Microsoft Office aplikací vzhledem k VSTO* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://msdn.microsoft.com/en-us/library/aa942839.aspx>.

[56] Microsoft. *Návod k použití lineárního programování za pomocí služeb Solver Foundation* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://msdn.microsoft.com/en-us/library/ff628587%28v=vs.93%29.aspx>.

[57] Microsoft. *Visual Basic* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://msdn.microsoft.com/en-us/library/2x7h1hfk.aspx>.

[58] Microsoft. *Visual Basic vývojářské centrum* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://msdn.microsoft.com/en-us/vbasic/>.

[59] Microsoft. *Porovnání vývoje pomocí Visual Basic a Visual C# nad Microsoft Office s VSTO* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://msdn.microsoft.com/en-us/library/522xhsa3.aspx>.

[60] Microsoft. *Visual Studio Tools for Office Runtime* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://msdn.microsoft.com/en-us/library/bb608603.aspx>.

[61] Microsoft. *Microsoft Excel* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://office.microsoft.com/cs-cz/excel/>.

[62] Microsoft. *Microsoft Office* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www.microsoft.com/cze/office/>.

[63] Microsoft. *Technické požadavky potřebné pro Microsoft Office* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www.microsoft.com/officebusiness/products/technical-requirements.aspx>.

[64] Microsoft. *Technické požadavky potřebné pro Microsoft Windows XP* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://support.microsoft.com/kb/314865>.

[65] Microsoft. *Vývoj Microsoft Office aplikací ve Visual Studiu* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://msdn.microsoft.com/en-us/library/d2tx7z6d.aspx>.

[66] Microsoft. *Hlavní strana Microsoft Windows* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www.microsoft.com/windows/>.

[67] Mladá fronta a. s. *Analýza práce vrchních a staničních sester* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www.zdn.cz/clanek/sestra/analyza-prace-vrchnich-a-stanicnich-sester-453230>.

[68] MOZ, M. – PATO, M. V. A genetic algorithm approach to a nurse rerostering problem. *Computers and Operations Research*. 2007, 3, s. 667–691.

[69] PECINOVSKÝ, R. *Návrhové vzory*. Computer Press, 1st edition, 2007.

[70] PELIKÁN, J. *Diskrétní modely v operačním výzkumu*. Professional Publishing, 1st edition, 2001.

[71] PIPATSRISAWAT, K. – DARWICHE, A. *RSat* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://reasoning.cs.ucla.edu/rsat/index.html>.

[72] POLI, R. – KENNEDY, J. – BLACKWELL, T. Particle swarm optimization: An overview. *Swarm Intelligence*. 2007, 1, s. 33–57.

[73] RADCLIFFE, N. J. – SURRY, P. D. Formal Memetic Algorithms. 1994.

[74] ROSSI-DORIA, O. et al. A Comparison of the Performance of Different Metaheuristics on the Timetabling Problem. 2002.

[75] STIDSEN, T. *Evolutionary Algorithms -– simulated evolution: přednáška* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www2.imm.dtu.dk/courses/02719/evo/4evo.pdf>.

[76] The Automated Scheduling, Optimisation and Planning (ASAP) research group. *Hlavní stránka* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www.asap.cs.nott.ac.uk/>.

[77] The University of Nottingham. *Testovací instance pro rozvrhování zaměstnanců* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://www.cs.nott.ac.uk/~tec/NRP/>.

[78] TRILLING, L. – GUINET, A. – MAGNY, D. L. Nurse scheduling using integer linear programming and constraint programming. *Information Control Problems in Manufacturing*. 2006, 12.

[79] Uživatel stuvagas. *PuLP – LP řešitel* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://code.google.com/p/pulp-or/>.

[80] VÁCLAVÍK, R. Rozvrhování zaměstnanců pomocí ILP: semestrální práce. 2010.

[81] WIEGERS, K. E. *Software Requirements 2: Practical techniques for gathering and managing requirements throughout the product development cycle*. Microsoft Press, 2nd edition, 2003.

[82] WREN, A. Scheduling, timetabling and rostering — A special relationship? *Lecture Notes in Computer Science*. 1996, 1153, s. 46–75.

[83] www.motylek.com s.r.o. *Zákoník práce §85* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://zakony-online.cz/?s2&q2=85>.

[84] www.motylek.com s.r.o. *Zákoník práce §90* [online]. [cit. 10. 5. 2011]. Dostupné z: <http://zakony-online.cz/?s2&q2=90>.

Příloha A

Seznam použitých zkratek

GUI Graphical User Interface

MS Microsoft

VS Vrchní sestra

ZS Zdravotní sestra

OOP Object-oriented programming

VSTO Visual Studio Tools for Office

NP Nondeterministic Polynomial

OS Operating System

MHz MegaHertz

MB MegaByte

GB GigaByte

RAM Random-Access Memory

HDD Hard Disk Drive

UML Unified Modeling Language

PATAT Practice and Theory of Automated Timetabling

ILP Integer Linear Programming

SAT satisfiability problem

CP Constraint Programming

SA Simulated Annealing

HC Hill Climbing

TS Tabu Search

CNF Conjunctive Normal Form

PB Pseudo Boolean

PSO Particle Swarm Optimization

ACO Ant Colony Optimization

GA Genetic Algorithm

MA Memetic Algorithm

SFL Shuffled Frog Leaping

ECMA European Computer Manufacturers Association

ISO International Organization for Standardization

LINQ Language Integrated Query

.NET dot NET

VB Visual Basic

XML Extensible Markup Language

XP Extreme Programming

FDD Feature Driven Development

TDD Test Driven Development

VBA Visual Basic for Applications

TPVDS Time Pre-defined Variable Depth Search

SS Scatter Search

NEIGH Neighbour Algorithm

CHAIN Chain Algorithm

UI User Interface

PC Personal Computer

PDF Portable Document Format

GHz GigaHertz

CPU Central Processing Unit

Příloha B

Seznam použitého matematického značení

Proměnná	Význam
s	index pro směny
z	index pro zaměstnance
d	index pro dny
n_z	počet zaměstnanců
n_s	počet směn
n_d	počet dnů
e_z	zaměstnanec z
N_s	počet výskytů směn s v celém rozvrhu
NS	počet výskytů všech směn v celém rozvrhu
H_z	počet odpracovaných hodin zaměstnance z
V_i	váha pro omezení se zkratkou i
$Z_{z,s}$	počet směn s zaměstnance z
P_i	penalizace rozvrhu pro omezení se zkratkou i
Z	výsledná penalizace celého rozvrhu
$prum_hod$	průměrný počet odpracovaných hodin na zaměstnance pro rozvrhované období
$prum_smen_s$	průměrný počet směn s na zaměstnance pro rozvrhované období
$pocet_rozd_z$	počet souvislých bloků směn zaměstnance z snížené o 1
$pocet_sir_z$	počet nalezených izolovaných pracovních a volných dnů u zaměstnance z
$pocet_vysk$	počet výskytů vzoru v daném intervalu

Tabulka B.1: Použité matematické značení

Příloha C

Pseudokódy

Pseudokód C.1 GreedyAlgorithm

Vstup: nSolutions

```
Vytvoř prázdnou množinu rozvrhů (rosters) o velikosti nSolutions
for solution = 1 : nSolutions do
    Vytvoř prázdný rozvrh (roster)
    for day = 1 : nDays do
        for all směny (shift), které je potřeba pokrýt ve dni day do
            call FindSimilarity
            Přiřaď do rozvrhu roster směnu shift ke dni day zaměstnanci employee
        end for
    end for
    Vlož rozvrh roster do množiny rosters
end for
return roster
```

Pseudokód C.2 FindSimilarity

```
if count(rosters) = 0 then
    return první možný zaměstnanec (employee)
else
    Vytvoř pole výskytů směn (similarity) o velikosti nEmployees
    for all rozvrhy (roster) ∈ rosters do
        for emp = 1 : nEmployees do
            if roster[emp, day] = shift then
                similarity[emp] + +
            end if
        end for
    end for
    return zaměstnanec (employee) s nejmenší hodnotou v similarity, pokud jich je
    více, tak vyber zaměstnance náhodně
end if
```

Pseudokód C.3 HillClimbing

Vstup: *roster*

```
1: for restart = 1 : nRestarts do
2:   Nastav proměnnou pro počet výskytů (macthes) na nulu
3:   for blockLength = 1 : maxBlockLength do
4:     for day = 1 : nDays do
5:       for emp1 = 1 : nEmployees do
6:         for emp2 = 1 : nEmployees do
7:           Prohoď směny mezi zaměstnanci emp1 a emp2 v rámci dne day s ohledem
             na délku bloku blockLength
8:           if nastalo zlepšení rozvrhu then
9:             matches = 0
10:          else
11:            if řešení spadá do uvažovaného intervalu then
12:              matches + +
13:            end if
14:            if může dojít k restartu then
15:              Nastav roster na nějaký nový rozvrh
16:              goto 3
17:            else
18:              Vrať rozvrh roster do původního stavu
19:            end if
20:          end if
21:        end for
22:      end for
23:    end for
24:  end for
25: end for
26: return roster
```

Pseudokód C.4 SimulatedAnnealing – varianta náhodného výběru dnů i zaměstnanců

Vstup: *roster*

```
1: for cooldown = 1 : cooldownSteps do
2:   for blockLength = 1 : maxBlockLength do
3:     Náhodně vyber den day, zaměstnance emp1 a emp2
4:     Prohoď směny mezi zaměstnanci emp1 a emp2 v rámci dne day s ohledem na délku
       bloku blockLength
5:     if není splněno Metropolisovo kritérium then
6:       Vrať rozvrh roster do původního stavu
7:     end if
8:   end for
9:   Sniž teplotu
10: end for
11: return roster
```

Pseudokód C.5 TabuSearch

Vstup: *roster*

1: $bestPenalty$ = penalizace rozvrhu $roster$
2: $actualRoster = roster$
3: **while** $bestPenalty > 0$ **do**
4: Vyber zaměstnance (emp_1), který přispívá do rozvrhu největší penalizací a není zakázán ukončovacím mechanismem ($terminationInfo[emp_1].iteration > tabuList.size$)
5: **if** $emp_1 = null$ **then**
6: **break**
7: **end if**
8: **for** $blockLength = 1 : maxBlockLength$ **do**
9: **for** $day = 1 : nDays$ **do**
10: **for** $emp_2 = 1 : nEmployees$ **do**
11: Prohoď směny nad rozvrhem $actualRoster$ mezi zaměstnanci emp_1 a emp_2 v rámci dne day s ohledem na délku bloku $blockLength$
12: $actualPenalty$ = penalizace rozvrhu $actualRoster$
13: Ulož tyto hodnoty do struktury $candidate$
14: **if** existuje změna v $tabuList$ OR se zhoršila penalizace **then**
15: Vrať rozvrh actualRoster do původního stavu
16: **end if**
17: **end for**
18: **end for**
19: **end for**
20: **call** CheckAndUpdate {aktualizace informací o nejlepším řešení a struktury pro ukončení algoritmu}
21: **end while**
22: **return** *roster*

Pseudokód C.6 CheckAndUpdate

1: **if** $actualPenalty < bestPenalty$ **then**
2: $bestPenalty = actualPenalty$
3: $roster = actualRoster$
4: Vynuluj $terminationInfo$
5: Vlož $candidate$ do $tabuList$
6: **else**
7: **if** $actualPenalty < terminationInfo[emp_1].penalty$ **then**
8: $terminationInfo[emp_1].penalty = actualPenalty$
9: $terminationInfo[emp_1].iteration = 0$
10: **else**
11: $terminationInfo[emp_1].iteration++$
12: **end if**
13: Vlož prázdný záznam do $tabuList$
14: **end if**

Pseudokód C.7 TPVDS – varianta bez časové restrikce

Vstup: *roster*

1: $bestRoster = roster$
2: $currentRoster$ = **call** sousedský algoritmus
3: **if** $currentRoster = null$ **then**
4: **return** $bestRoster$
5: **end if**
6: **if** penalty($currentRoster$) < penalty($bestRoster$) **then**
7: **goto** 1
8: **end if**
9: **if** nedošlo ke zlepšení individuálního rozvrhu ani jednoho zainteresovaného zaměstnance OR $maximumDepth <= 1$ **then**
10: **goto** 2
11: **end if**
12: emp_1 = zainteresovaný zaměstnanec, kterému se zvýšila penalizace
13: $currentDepth = 1$
14: $currentRoster, emp_2$ = **call** zlepšovací algoritmus pro zaměstnance emp_1
15: **if** $currentRoster = null$ **then**
16: **goto** 2
17: **else if** penalty($currentRoster$) < penalty($bestRoster$) **then**
18: **goto** 1
19: **else if** $currentDepth < maximumDepth$ **then**
20: $emp_1 = emp_2$
21: $currentDepth++$
22: **goto** 14
23: **else**
24: **goto** 2
25: **end if**

Pseudokód C.8 ScatterSearch

1: **call** GreedyAlgorithm
2: **call** zlepšovací metodu pro všechny řešení
3: R = **call** metodu pro aktualizaci referenční množiny (inicializační část)
4: $RCopy = R$
5: $subsets$ = **call** metodu pro generování podmnožin
6: **for** $\forall subset \in subsets$ **do**
7: $newSolution$ = **call** SubsetCombination
8: $newSolution$ = **call** zlepšovací metodu pro $newSolution$
9: **call** metodu pro aktualizaci referenční množiny (nahrazovací část)
10: **end for**
11: **if** $R \neq RCopy$ **then**
12: $R = RCopy$
13: **goto** 4
14: **end if**
15: **return** nejlepší řešení z R

Pseudokód C.9 SubsetCombination

1: **if** neexistuje žádný *kandidát* **then**
2: **goto** 13
3: **end if**
4: Seřaď kandidáty:
a) sestupně podle hlasů,
b) vzestupně podle úspěšnosti přijímání voličských kandidátů do výsledného rozvrhu (*newSolution*),
c) vzestupně podle možné budoucí penalizace rozvrhu *newSolution*,
d) náhodně.
5: Vyber a následně odstraň prvního *kandidáta*
6: Proveď jeho vložení do rozvrhu *newSolution*, jeli to možné (s ohledem k omezení na obsazenost směn)
7: **if** bylo provedeno vložení **then**
8: **goto** 1
9: **end if**
10: **if** zbyl ještě nějaký *kandidát* **then**
11: **goto** 5
12: **end if**
13: **return** *newSolution*

Pseudokód C.10 ChainAlgorithm

Vstup: *roster*
1: $bestPenalty$ = penalizace rozvrhu *roster*
2: emp_1 = vyber náhodně nějakého zaměstnance
3: **while** $bestPenalty > 0$ **do**
4: **for** $blockLength = 1 : maxBlockLength$ **do**
5: **for** $day = 1 : nDays$ **do**
6: **for** $emp_2 = 1 : nEmployees$ **do**
7: Prohoď směny mezi zaměstnanci emp_1 a emp_2 v rámci dne *day* s ohledem na délku bloku *blockLength*
8: **if** Metropolisovo kritérium splněno AND (zlepšení generované řetězem *chain*+ aktuální zlepšení/zhoršení) > 0 AND nenastává kolize v překryvech **then**
9: Ulož informace o potencionální části řetězu do *candidate*
10: **else**
11: Vrať změny
12: **end if**
13: **end for**
14: **end for**
15: **end for**
16: Sniž teplotu
17: **call** UpdateChain
18: **end while**
19: **return** *roster*

Pseudokód C.11 UpdateChain

```
1: if nastalo zlepšení (candidate ≠ null) then
2:    Přidej změnu candidate do řetězu chain
3:    Zruš všechny zákazy zaměstnanců
4:    Aktualizuj roster a bestPenalty
5:    emp_1 = emp_2
6: else
7:    if chain.size = 0 then
8:       Přidej zákaz zaměstnance emp_1
9:    else
10:      Zruš všechny zákazy zaměstnanců
11:   end if
12:   Vymaž řetěz
13:   if existuje nějaký nezakázaný zaměstnanec then
14:      emp_1 = vyber náhodně nezakázaného zaměstnance
15:   else
16:      break (pro while smyčku v ChainAlgorithm)
17:   end if
18: end if
```

Příloha D

Nastavení aplikace

Následující podkapitoly popisují nastavení aplikace, které je možné měnit uživatelem přímo za chodu systému.

D.1 Formát a vzhled

- Formát data (*dateFormat*):
  - řetězec, výchozí hodnota „d/m/yyyy“,
  - definuje formát data v záhlaví tabulek pro požadovanou obsazenost směn a rozvrh.
- Barva zmražených směn (*frozenBgColor*):
  - barva, výchozí hodnota modro-zelená (Aqua),
  - možnosti – celá paleta barev dostupná v MS Excel,
  - určuje barvu pro označení zmražených směn v rozvrhu, v MS Excel je přidána jako nový styl.

D.2 Omezení

- Prodleva mezi směnami (*minHoursDiffBetweenWorkshifts*):
  - reálné číslo, výchozí hodnota 12,
  - jedná se o nutný odpočinek mezi dvěma směnami a používá se u tvrdého omezení pro následnost směn.
- Váha pro vyvážení odpracovaných hodin (*penaltyBalanceWorkHours*) *:
  - celé číslo, výchozí hodnota 5.
- Váha pro vyvážení typu směn (*penaltyBalanceWorkshiftTypes*) *:
  - celé číslo, výchozí hodnota 1.

- Váha pro maximalizaci počtu po sobě následujících směn (*penaltyMaximumContinuousWorkshifts*) *:
  - celé číslo, výchozí hodnota 10.
- Váha pro minimalizaci izolovaných pracovních a volných dnů (*penaltyMinimizeSingleSlots*) *:
  - celé číslo, výchozí hodnota 15.
- Minimální rozestup mezi směnami (*activeWorkshiftsPrecedences*),
- zmražené směny (*activeFrozenWorkshifts*) a
- kvalifikace zaměstnanců (*activeEmployeesQualification*):
  - logický datový typ, výchozí hodnota NEPRAVDA,
  - možnosti – PRAVDA, NEPRAVDA,
  - určuje, zdali je tvrdé omezení aktivní či nikoliv.

* V aktuální verzi aplikace tyto možnosti nejsou viditelné a používané z důvodu upřednostnění vzorů. Nicméně systém je v sobě zahrnuje, a proto jsou uvedeny i zde.

D.3 Volba algoritmů

- Hlavní (meta)heuristika (*heuristicSkeleton*):
  - výčtový typ, výchozí hodnota SA,
  - možnosti – HC, SA, TS, TPVDS, SS, CHAIN,
  - určuje hlavní algoritmus, kterým bude vylepšován rozvrh.
- Heuristika v TPVDS (*TPVDSHeuristicAlg*):
  - výčtový typ, výchozí hodnota HC,
  - možnosti – HC, SA, TS,
  - určuje algoritmus heuristiky TPVDS, kterým budou vylepšovány jednotlivé rozvrhy.
- Sousedství v TPVDS (*TPVDSNeighbourAlg*):
  - výčtový typ, výchozí hodnota NEIGH,
  - možnosti – NEIGH,
  - určuje algoritmus pro vyhledávání sousedství k danému rozvrhu v TPVDS.
- Zlepšovací algoritmus v SS (*SSImproveAlg*):
  - výčtový typ, výchozí hodnota SA,
  - možnosti – HS, SA, TS, TPVDS, CHAIN,
  - určuje algoritmus, kterým se vylepšují všechny dílčí rozvrhy vzniklé v SS.

D.4 Nastavení algoritmů

Time Pre-defined Variable Depth Search

- Omezení času ($TPVDSTimeRestriction$):
  - logický datový typ, výchozí hodnota NEPRAVDA,
  - možnosti – PRAVDA, NEPRAVDA,
  - určuje, jestli bude běh TPVDS omezen časem či nikoliv.
- Časový limit ($TPVDSTimeLimit$):
  - reálné číslo, výchozí hodnota 30.
- Hloubka procházení ($TPVDSMaximumDepth$):
  - celé číslo, výchozí hodnota 2,
  - určuje, do jaké hloubky může heuristika pracovat.

Scatter Search

- Pravděpodobnost přidání stejně odlišného řešení do referenční množiny ($RSProbabilityInitialRefSet$),
- pravděpodobnost odstranění řešení ze stejně nejhorších z referenční množiny ($RSProbabilityFindWorstSol$),
- pravděpodobnost přidání stejného řešení do referenční množiny ($RSProbabilityAddNewSol$) a
- pravděpodobnost vybrání jednoho ze zaměstnanců se stejným počtem přiřazených směn ($GAProbabilityLeastSimilarity$):
  - celé číslo, výchozí hodnota 50 [%].

Hill Climbing

- Maximální počet restartů ($HCMaxRestarts$):
  - celé číslo, výchozí hodnota 5.
- Tolerance pro výpočet výskytů ($HCRestartsTolerance$):
  - reálné číslo, výchozí hodnota 0,1 (tzn. 10%),
  - definuje interval aktuálně nejlepší penalizace – *<penalizace; penalizace * tolerance>* a pokud do něj spadá aktuálně zpracovávaný rozvrh, dojde k navýšení počtu výskytů podobných řešení.
- Počet výskytů nutných k restartu ($HCRestartsNoMatch$):
  - celé číslo, výchozí hodnota 50.

- Pravděpodobnost provedení restartu (*HCProbabilityRestarts*):
  - celé číslo, výchozí hodnota 50 [%],
  - až se naplní hodnota počtu výskytů nutných k restartu, může dojít s určitou pravděpodobností k restartu.

Simulated Annealing

- Počet kroků chlazení (*SACooldownSteps*):
  - celé číslo, výchozí hodnota 1000.

Tabu Search

- Velikost Tabu Listu (*TSTabuListSize*):
  - celé číslo, výchozí hodnota 10.

NEIGH, HC, SA, TS, CHAIN

- Maximální délka bloků ($\{Neigh\}MaxBlockLength$):
  - celé číslo, výchozí hodnota 5.
- Směr procházení bloků ($\{Neigh\}BlockDirection$):
  - výčtový typ, výchozí hodnota DESC,
  - možnosti – ASC, DESC,
  - určuje směr procházení bloků – tzn. od nejmenšího po největší nebo opačně.

HC, TS, CHAIN

- Pravděpodobnost přijmutí stejného řešení (*{HC}ProbabilityAcceptSameTimetablePenalty*):
  - celé číslo, výchozí hodnota 50 [%].

SA, CHAIN

- Počáteční teplota ($\{SA\}InitTemperature$):
  - reálné číslo, výchozí hodnota 100.
- Chlazení ($\{SA\}Cooldown$):
  - reálné číslo, výchozí hodnota 0,99 (tzn. $100 - 99 = 1\%$),
  - určuje, o kolik procent se v každém kroku sníží teplota.

Příloha E

Tabulkové podklady k experimentům

Vysvětlivky:

- každý řádek zobrazuje jednu testovanou instanci (viz. kapitola 5.2),
- každý řádek s instancí je rozdělen na další čtyři řádky:
  - *min* – nejmenší nalezená hodnota penalizace,
  - *max* – největší nalezená hodnota penalizace,
  - *avg pen* – průměrná hodnota penalizace,
  - *avg time* – průměrný čas (v sekundách) potřebný pro výpočet jednoto výsledku.
- sloupce zobrazují právě jednu uvedenou testovanou hodnotu.

Překryvy		všechny	vnitřní
Millar	min	0	0
	max	1400	800
	avg pen	346	294
	avg time	8,71	8,43
Millar-s	min	0	0
	max	1100	200
	avg pen	75	39
	avg time	6,31	4,96
LLR	min	198	198
	max	201	200
	avg pen	198,81	198,35
	avg time	4,54	4,66
GPOST	min	233	223
	max	5428	3234
	avg pen	1658,28	1474,97
	avg time	113,56	115,76
VALOUXIS	min	500	460
	max	2820	2880
	avg pen	850	897
	avg time	161,33	162,87

Tabulka E.1: Chain Algorithm – srovnání metod pro vyhledávání kolizí

Překryvy		všechny	vnitřní
Millar	min	0	0
	max	800	700
	avg pen	276	245
	avg time	4,35	3,96
Millar-s	min	0	0
	max	200	100
	avg pen	9	15
	avg time	0,72	0,95
LLR	min	198	198
	max	201	201
	avg pen	198,98	199,01
	avg time	55,68	54,63
GPOST	min	236	235
	max	2435	2439
	avg pen	614,64	768,38
	avg time	281,20	301,87
VALOUXIS	min	560	1480
	max	7600	8600
	avg pen	3374	3814
	avg time	1689,73	1890,70

Tabulka E.2: Tabu Search – srovnání metod pro vyhledávání kolizí

Varianta HC		s restarty	bez restartů
Millar	min	0	0
	max	1400	1400
	avg pen	398	280
	avg time	30,69	2,68
Millar-s	min	0	0
	max	1100	1200
	avg pen	184	90
	avg time	23,66	1,85
LLR	min	198	198
	max	198	201
	avg pen	198	198,9
	avg time	7179,95	1063,94
GPOST	min	622	625
	max	2418	3623
	avg pen	990,56	1997,1
	avg time	12134,37	1102,58
VALOUXIS	min	160	120
	max	260	280
	avg pen	193,33	205,2
	avg time	79837,64	6415,16

Tabulka E.3: TPVDS – srovnání variant algoritmu Hill Climbing

Iterace přes		vše	zaměstnance	dny	náhodně
Millar	min	0	0	0	0
	max	100	300	200	1200
	avg pen	6,66	43,33	60	306,66
	avg time	83,17	28,18	13,99	2,58
Millar-s	min	0	0	0	0
	max	0	0	0	0
	avg pen	0	0	0	0
	avg time	19,14	2,09	1,1	0,67
LLR	min	198	198	200	200
	max	198	200	207	207
	avg pen	198	198,53	202,5	203,9
	avg time	2894,88	446,33	7,77	1,44
GPOST	min	25	223	25	228
	max	621	2420	1629	4339
	avg pen	203,73	643,96	610,6	1638,73
	avg time	9532,23	405,53	320,22	17,17
VALOUXIS	min	420	500	600	900
	max	540	740	820	4280
	avg pen	516	623,33	738,66	2532
	avg time	66001,49	3087,88	531,90	28,17

Tabulka E.4: Simulated Annealing – srovnání metod pro prohledávání sousedních řešení

Modelace tvrdých omezení		klasicky	jako měkké
Millar	min	0	0
	max	1200	1100
	avg pen	306,66	292,67
	avg time	2,58	5,12
Millar-s	min	0	0
	max	0	100
	avg pen	0	1,33
	avg time	0,67	1,36
LLR	min	200	199
	max	207	206
	avg pen	203,9	202,8
	avg time	1,44	2,73
GPOST	min	228	228
	max	4339	5229
	avg pen	1638,73	1629,94
	avg time	17,17	35,68
VALOUXIS	min	900	820
	max	4280	5640
	avg pen	2532	2592
	avg time	28,17	52,24

Tabulka E.5: Simulated Annealing – srovnání modelování tvrdých omezení

Sestupně	Blok	1	3	5	7	9	11	13	15	17	19	21	23	25	27
Millar	min	500	100	0	0	0	0	0							
	max	3500	2600	1400	1300	1300	1300	1200							
	avg pen	1751	1046	499	379	331	292	303							
	avg time	0,13	0,41	0,70	0,99	1,30	1,56	1,86							
Millar-s	min	0	0	0	0	0	0	0							
	max	2400	1200	1100	1100	1000	1000	1000							
	avg pen	1059	369	154	143	86	54	65							
	avg time	0,10	0,28	0,40	0,57	0,68	0,68	0,74							
LLR	min	222	201	199	199										
	max	10249	217	212	211										
	avg pen	1243,44	207,74	204,1	203,82										
	avg time	0,14	0,43	0,71	1,00										
GPOST	min	5652	1451	444	856	237	440	238	433	231	40	238	429	228	225
	max	20520	7387	8250	5362	5245	5248	4348	5235	6142	4231	4429	4547	4444	4343
	avg pen	12655,33	4160,24	3239,47	2760,65	2492,84	2376,69	2221,63	2162,65	1932,71	1792,21	1996,1	1879,87	1951,39	1808,68
	avg time	0,38	1,15	1,91	2,68	3,44	4,20	4,98	5,75	6,50	7,37	8,10	8,79	9,55	10,31
VALOUXIS	min	51320	13940	8820	6140	5140	2180	2960	1920	940	860	860	640	820	820
	max	81440	38920	25040	23840	17740	14860	12040	11800	12860	10820	10000	9620	8900	7840
	avg pen	67222,8	26384,6	17161,4	13097	10485,2	8664,2	7311,2	6892,6	6029,8	5380,2	4669	4222,6	4105,8	3552,8
	avg time	0,62	1,87	3,12	4,37	5,61	6,85	8,09	9,34	10,58	11,82	13,06	14,31	15,56	16,79

Tabulka E.6: Simulated Annealing – vliv délky bloku směn pro prohazování v sestupném směru procházení na penalizaci rozvrhu

Vzestupně	Blok	1	3	5	7	9	11	13	15	17	19	21	23	25	27
Millar	min	500	200	0	0	0	0	0							
	max	3300	1600	1500	1400	1400	1200	1300							
	avg pen	1825	887	625	380	345	294	288							
	avg time	0,14	0,43	0,72	1,01	1,29	1,56	1,84							
Millar-s	min	100	0	0	0	0	0	0							
	max	2400	1200	1100	1100	1100	1100	1000							
	avg pen	1095	467	207	93	105	80	64							
	avg time	0,10	0,29	0,41	0,52	0,68	0,74	0,83							
LLR	min	227	200	200	200										
	max	10256	217	210	208										
	avg pen	1143,62	207,22	204,26	203,8										
	avg time	0,14	0,43	0,72	1,00										
GPOST	min	3190	883	870	239	434	435	227	235	239	434	249	230	237	231
	max	24116	7171	7638	5633	5240	5037	4551	5150	5432	4345	4230	4356	4545	4424
	avg pen	12295,37	4139,95	3115,14	2721,94	2543,05	2262,83	1971,43	2123,67	2027,44	1816,51	1784,76	1823,95	2006,13	2015,79
	avg time	0,38	1,16	1,92	2,69	3,45	4,21	4,97	5,74	6,50	7,26	8,02	8,78	9,55	10,37
VALOUXIS	min	52520	15120	8080	7100	5100	3880	2880	1860	680	700	660	860	800	760
	max	83120	37140	28140	20880	16860	15000	12060	11780	11880	7980	8800	8740	9780	8800
	avg pen	66622,2	26139,2	16870,6	12371,2	10103,2	9150,4	7539	6741,8	5830,8	5213,2	4636,6	4043,6	3896	3678,2
	avg time	0,62	1,87	3,12	4,36	5,60	6,85	8,09	9,34	10,58	11,82	13,06	14,31	15,56	16,80

Tabulka E.7: Simulated Annealing – vliv délky bloku směn pro prohazování ve vzestupném směru procházení na penalizaci rozvrhu

Sestupně	Blok	1	3	5	7	9	11	13	15	17	19	21	23	25	27
Millar	min	2900	400	100	800	200	200	200							
	max	6500	2300	1800	2400	2500	2100	1800							
	avg pen	4229	1374	962	1450	1036	916	799							
	avg time	0,13	0,33	0,50	0,73	0,82	0,84	0,89							
Millar-s	min	1900	0	0	0	0	0	0							
	max	5600	2200	2300	600	2200	1200	1200							
	avg pen	3610	947	646	259	557	260	206							
	avg time	0,09	0,24	0,36	0,47	0,54	0,61	0,64							
LLR	min	304	201	201	200										
	max	30354	212	211	210										
	avg pen	6538,37	204,96	205,47	204,5										
	avg time	0,80	2,09	2,90	3,27										
GPOST	min	14749	1468	673	253	830	1435	632	444	430	239	247	640	637	430
	max	35204	8276	6766	5450	6554	5557	5630	6160	4555	7146	6229	5554	6240	5038
	avg pen	24723,57	4285,15	3779,36	3072,51	3207,27	3229,65	3099,28	3105,01	2387,56	2654,29	2605,59	3096,01	2650,67	2634,23
	avg time	0,59	1,71	2,73	3,67	4,54	5,37	6,06	6,69	7,25	7,71	8,09	8,33	8,54	8,62
VALOUXIS	min	69460	2540	1740	820	640	540	540	520	540	480	640	560	580	580
	max	106260	18520	8920	7700	9580	5720	5880	5900	4780	5760	6480	4660	5680	6680
	avg pen	84326	9024	4962,8	3434	2789,2	2087,6	2298	2610,2	1999,6	2294,4	2475,6	1980,8	2204,4	2152
	avg time	5,54	15,97	25,46	34,34	42,40	50,50	57,42	63,04	68,29	72,41	75,77	78,29	80,02	80,85

Tabulka E.8: Hill climbing – vliv délky bloku směn pro prohazování v sestupném směru procházení na penalizaci rozvrhu

Vzestupně	Blok	1	3	5	7	9	11	13	15	17	19	21	23	25	27
Millar	min	2800	400	100	100	200	0	200							
	max	5100	2100	2100	1800	1800	1700	1900							
	avg pen	3926	1132	818	753	766	758	665							
	avg time	0,12	0,34	0,50	0,63	0,74	0,82	0,87							
Millar-s	min	2400	100	0	0	0	0	100							
	max	5200	1600	1500	1400	1400	1300	1300							
	avg pen	3622	884	598	422	448	407	427							
	avg time	0,09	0,24	0,37	0,46	0,54	0,6	0,63							
LLR	min	308	200	199	199										
	max	30362	219	209	206										
	avg pen	6837,15	207,67	203,28	202,78										
	avg time	0,81	2,08	2,87	3,24										
GPOST	min	15616	1983	1443	840	576	637	455	455	449	648	636	633	636	641
	max	33613	8771	7663	6564	6359	6641	5632	5632	7443	6244	6433	6633	6429	6436
	avg pen	25266,76	5082,89	4258,96	3640,71	3270,31	3497,59	3087,73	3087,73	3086,34	3064,22	3042,35	2929,71	2844,26	2727,44
	avg time	0,60	1,74	2,78	3,74	4,60	5,38	6,12	6,12	7,23	7,71	8,05	8,34	8,51	8,59
VALOUXIS	min	66600	8960	4000	680	600	620	1580	600	620	560	620	660	500	560
	max	104420	23840	15620	11660	12800	11460	9540	8820	9760	7760	6760	9480	10640	6660
	avg pen	84174,2	15948,6	9009,8	6215,8	5450	4746	4123	3944,2	3503,2	3124	3070,6	3204,6	2932,6	2585,8
	avg time	5,51	16,13	26,02	35,05	43,08	50,72	57,17	63,03	67,94	72,11	75,67	79,12	76,19	77,70

Tabulka E.9: Hill Climbing – vliv délky bloku směn pro prohazování ve vzestupném směru procházení na penalizaci rozvrhu

Sestupně	Blok	1	3	5	7	9	11	13	15	17	19	21	23	25	27
Millar	min	200	400	0	100	100	100	100							
	max	800	1000	800	1000	900	900	800							
	avg pen	393	682	315	392	390	410	438							
	avg time	2,15	4,81	5,69	8,95	10,79	11,53	12,42							
Millar-s	min	0	0	0	200	200	200	200							
	max	300	300	100	200	200	200	200							
	avg pen	157	61	5	200	200	200	200							
	avg time	1,70	1,62	0,68	5,70	6,62	7,20	7,53							
LLR	min	199	198	198	198										
	max	216	201	201	201										
	avg pen	202,55	199,26	198,75	198,7										
	avg time	18,10	86,24	112,21	131,05										
GPOST	min	2671	1630	642	652	242	438	239	234	1440	228	433	435	444	435
	max	7802	3437	2645	2833	3230	2432	1643	2249	3034	2431	2440	1637	1641	1640
	avg pen	5005,8	2515,65	1567,75	1899	1826,2	1138,85	631,25	945,5	2293,2	973,7	1210,2	1160,95	1290,35	1209,55
	avg time	16,54	40,23	55,48	80,05	88,69	110,54	124,27	156,73	164,63	165,29	145,76	144,57	148,18	149,96
VALOUXIS	min	7840	2660	2540	1540	2600	1560	1440	680	660	1580	580	1520	1420	600
	max	20000	11440	10460	6800	6600	6600	5480	6640	3680	6580	5660	3800	6640	6500
	avg pen	12938	6930	5448	4630	3744	3404	3496	3076	2476	3164	2892	3124	3298	3470
	avg time	114,23	239,96	368,88	517,15	498,69	695,46	712,88	700,93	859,35	839,81	831,51	964,31	859,39	945,09

Tabulka E.10: Tabu Search – vliv délky bloku směn pro prohazování v sestupném směru procházení na penalizaci rozvrhu

Vzestupně	Blok	1	3	5	7	9	11	13	15	17	19	21	23	25	27
Millar	min	200	200	0	200	300	300	200							
	max	700	1000	800	800	800	800	800							
	avg pen	407	645	298	534	541	552	572							
	avg time	2,10	4,89	5,60	9,34	10,56	12,41	13,42							
Millar-s	min	0	0	0	200	200	200	200							
	max	300	300	100	200	200	200	200							
	avg pen	160	67	14	200	200	200	200							
	avg time	1,72	1,93	1,04	5,69	6,61	7,26	7,53							
LLR	min	198	198	198	198										
	max	217	203	203	201										
	avg pen	203,26	199,73	198,65	198,6										
	avg time	18,13	79,11	66,58	73,03										
GPOST	min	2695	1627	638	631	1238	830	1235	1428	826	820	1429	1437	1621	644
	max	10587	4637	2634	1826	3238	2437	3347	4432	3240	2441	2632	2632	3626	2637
	avg pen	5519,9	2576,25	1238,4	885,6	2031,8	1789,8	2728,2	2500,6	1789,5	1750,3	2003,6	2230,8	2270,8	2092
	avg time	15,94	35,89	55,77	65,33	85,37	101,05	122,15	159,21	145,89	177,04	184,87	186,29	210,25	209,02
VALOUXIS	min	6880	4840	2620	580	600	1420	640	720	640	700	1480	600	400	1440
	max	14100	11600	8600	7640	6520	6440	5600	3600	6560	5580	7580	5500	5680	5780
	avg pen	10028	7874	6090	3656	2800	3290	3122	2268	2404	2820	3412	2722	2786	3106
	avg time	110,3	238,39	401,69	457,26	474,60	603,86	734,63	704,54	739,74	782,85	857,02	815,95	845,11	1025,25

Tabulka E.11: Tabu Search – vliv délky bloku směn pro prohazování ve vzestupném směru procházení na penalizaci rozvrhu

Sestupně	Blok	1	3	5	7	9	11	13	15	17	19	21	23	25	27
Millar	min	200	0	0	0	0	0	0							
	max	2500	1600	1500	1500	1600	1500	1500							
	avg pen	1371	393	393	436	365	368	386							
	avg time	0,58	1,71	2,46	3,24	3,74	4,05	4,24							
Millar-s	min	0	0	0	0	0	0	0							
	max	2300	1300	1200	1200	1200	1200	1200							
	avg pen	680	426	243	145	173	190	205							
	avg time	0,44	1,10	1,82	2,38	2,73	3,04	3,02							
LLR	min	198	198	198	198										
	max	205	202	202	203										
	avg pen	200,11	199,12	199,2	199,55										
	avg time	0,74	1,84	2,52	2,83										
GPOST	min	677	1234	238	432	31	238	235	224	227	421	230	230	231	431
	max	11810	4438	2646	4229	3235	3344	4433	4434	3345	3429	3441	3433	4229	3235
	avg pen	4440,37	2709,04	1219,43	2068,52	1298,85	1585,51	1970,34	1149,91	1143,96	1536,18	1414,95	1431,75	1535,01	1595,1
	avg time	5,26	16,07	26,21	35,35	43,79	51,25	57,92	63,81	68,97	73,35	76,61	79,70	80,91	82,59
VALOUXIS	min	4640	1500	540	520	500	600	580	480	600	440	380	480	240	520
	max	16000	9780	6860	7820	6780	3880	3860	5660	3860	3760	3760	3840	4640	6640
	avg pen	10256,8	4554,8	3098	2605,6	2159,6	2251,2	1810,8	1762	1598,4	1644	1494,4	1490	1803,2	1867,2
	avg time	10,32	23,55	37,13	49,16	61,39	71,43	80,16	90,34	97,53	103,38	106,21	112,08	114,76	116,50

Tabulka E.12: Chain algorithm – vliv délky bloku směn pro prohazování v sestupném směru procházení na penalizaci rozvrhu

Vzestupně	Blok	1	3	5	7	9	11	13	15	17	19	21	23	25	27
Millar	min	300	100	0	0	100	0	0							
	max	2900	1600	1500	1500	1500	1300	1500							
	avg pen	1260	454	338	456	513	357	391							
	avg time	0,59	1,69	2,50	3,13	3,79	4,04	4,18							
Millar-s	min	0	0	0	0	0	0	0							
	max	1400	1300	1300	1100	1200	1300	1200							
	avg pen	680	258	245	130	168	161	305							
	avg time	0,43	1,03	1,75	2,31	2,74	2,93	3,03							
LLR	min	198	198	198	198										
	max	208	202	204	203										
	avg pen	200,57	199,15	199,25	199,45										
	avg time	0,76	1,88	2,57	2,89										
GPOST	min	863	650	431	427	230	231	235	223	229	237	426	227	425	225
	max	13693	4437	2635	4227	3228	3428	4142	3428	5242	4229	4232	5228	4233	4228
	avg pen	5126,71	2789,83	1153,9	1873,57	1409,59	1374,75	2280,08	1566,79	1702,74	1843,55	1847,41	1802,9	1780,69	1879,39
	avg time	5,31	16,10	26,09	41,48	51,11	64,61	66,68	66,59	69,60	73,54	77,54	79,10	81,67	88,03
VALOUXIS	min	4120	1840	620	540	480	540	380	500	460	480	540	500	460	440
	max	17120	7680	6660	3940	4660	5780	4620	3780	4600	4620	4660	3620	4660	4800
	avg pen	10824,4	4745,6	3377,6	2011,6	1937,2	1803,6	1859,2	1655,6	1784,8	1737,2	1744,4	1573,6	1629,6	1668
	avg time	10,04	23,19	36,14	49,54	62,12	71,41	81,92	88,95	95,31	101,19	107,01	111,74	116,68	115,39

Tabulka E.13: Chain Algorithm – vliv délky bloku směn pro prohazování ve vzestupném směru procházení na penalizaci rozvrhu

Tolerance [%]		0,1	0,5	1	5	10	15	20	25	30	35	40	45	50
Millar	min	0	0	0	0	0	0	0	0	0	0	0	0	0
	max	1300	1200	1200	1200	1200	1400	2000	2500	2700	4600	4100	4600	4500
	avg pen	200	246	238	269	210	227	231	441	608	1092	1032	1468	1867
	avg time	6,89	6,58	8,15	8,22	7,34	7,97	7,17	6,19	5,18	3,54	3,83	2,47	1,98
Millar-s	min	0	0	0	0	0	0	0	0	0	0	0	0	0
	max	1000	1000	1100	1100	1100	1000	2600	1700	4700	3400	4900	3900	3800
	avg pen	130	90	102	91	111	100	115	157	335	342	414	414	516
	avg time	1,80	1,62	2,15	1,64	1,94	1,85	1,43	1,70	1,39	1,23	1,07	1,1	0,98
LLR	min	201	199	201	201	201	199	200	200	200	199	201	199	201
	max	208	211	207	222	223	216	218	100412	100402	30317	100452	90403	100400
	avg pen	204,37	204,08	203,87	204,35	205,07	204,3	204,39	3010,29	3010,76	1709,39	5718,35	5918,63	3913,49
	avg time	29,69	29,41	34,54	21,59	15,64	15,01	16,26	14,77	13,85	14,93	15,94	13,20	14,40
GPOST	min	25	426	423	624	634	587	435	659	635	842	632	785	835
	max	4229	4634	5485	6525	7065	6613	10223	20171	24696	22391	22449	27259	27109
	avg pen	1698,82	2097,51	2343,96	2933,53	3008,01	2980,58	3260,81	4771,3	6049,69	7091,84	7403,76	8136,17	10059,24
	avg time	100,61	55,79	38,55	16,10	13,05	12,04	10,79	9,70	9,06	8,87	8,66	8,67	8,16
VALOUXIS	min	160	180	120	260	580	540	780	580	660	440	580	600	760
	max	340	300	440	5680	247100	261440	259300	260300	259320	260300	257900	259320	256240
	avg pen	264	252	264,2	2308,2	7359,8	154800,4	182479	156768	146228,4	148935	132931,2	143860,6	147817
	avg time	693,72	692,27	691,68	60,18	43,91	42,56	42,02	42,00	41,96	41,96	42,06	41,96	41,96

Tabulka E.14: Hill Climbing – vliv tolerance pro podobná řešení na penalizaci rozvrhu

Počet výskytů		10	25	50	75	100	150	200	500	1000
Millar	min	0	0	0	0	0	0	0	0	0
	max	1200	1200	1400	1200	1200	1200	1300	1200	1200
	avg pen	218	187	211	237	166	207	208	233	251
	avg time	6,37	6,46	6,88	6,77	6,84	9,97	8,43	6,80	6,94
Millar-s	min	0	0	0	0	0	0	0	0	0
	max	1000	1000	1000	1000	1000	1000	1000	1000	1000
	avg pen	60	50	51	70	80	50	50	100	90
	avg time	1,64	1,52	1,56	1,74	1,78	1,77	1,57	1,78	1,83
LLR	min	201	201	200	201	200	201	200	201	200
	max	214	214	208	208	208	208	209	208	207
	avg pen	205,98	204,97	204,27	204,37	203,95	203,82	204,02	204,1	204,18
	avg time	1,67	13,64	31,77	33,79	32,2	32,72	34,55	35,34	36,16
GPOST	min	622	224	428	34	425	225	224	29	423
	max	5426	5230	3423	3428	3427	3539	4619	4547	3625
	avg pen	2685,44	2256,4	1678	1631,12	1835,14	1461,76	1586,66	1803,68	1792,92
	avg time	13,97	74,86	114,44	117,94	117,49	121,67	122,88	123,81	120,97
VALOUXIS	min	280	260	180	180	180	160	160	140	180
	max	440	360	320	380	420	320	360	340	340
	avg pen	342	306	248	274	274	250	256	250	280
	avg time	479,23	761,44	704,71	700,60	699,83	697,38	697,03	698,41	698,08

Tabulka E.15: Hill Climbing – vliv počtu výskytů podobných řešení na penalizaci rozvrhu

Počet restartů		1	5	10	15	20	25	30	35	40	45	50
Millar	min	0	0	0	0	0	0	0	0	0	0	0
	max	1900	1400	1300	1300	1300	1200	1200	1100	1300	1200	1200
	avg pen	923	453	260	259	188	235	140	125	128	183	117
	avg time	0,48	2,28	4,13	6,14	7,24	9,42	10,08	11,64	13,46	14,89	16,42
Millar-s	min	0	0	0	0	0	0	0	0	0	0	0
	max	1300	1000	1100	1100	1100	1000	1000	1000	1100	1000	1000
	avg pen	558	89	105	113	131	60	90	40	132	70	100
	avg time	0,35	1,13	1,52	1,86	2,13	1,79	2,37	1,85	3,13	2,41	3,05
LLR	min	201	200	200	200	199	200	201	201	200	200	200
	max	212	213	220	223	250	214	226	224	236	227	221
	avg pen	205,57	204,26	204,24	204,33	205,76	204,22	204,33	204,53	204,96	204,49	204,66
	avg time	1,76	7,38	11,51	14,53	15,71	15,63	17,02	17,40	19,31	19,33	18,41
GPOST	min	829	430	425	225	425	225	423	425	227	425	223
	max	5434	4028	4542	3422	5223	2636	5541	4225	3544	4028	3426
	avg pen	3088,32	1973,48	1711,28	1436,18	1774,96	1523,4	1794,38	1753,14	1607,22	1588,1	1455,52
	avg time	6,09	29,83	59,06	88,34	114,87	146,34	187,27	216,84	262,96	294,65	326,74
VALOUXIS	min	560	320	260	180	140	160	180	140	100	160	100
	max	3620	600	480	440	280	380	380	300	280	240	220
	avg pen	1716	458	358	310	228	246	258	208	178	200	152
	avg time	42,63	212,59	426,67	635,75	857,68	1059,27	1274,04	1473,19	1684,49	1893,86	2103,17

Tabulka E.16: Hill Climbing – vliv počtu restartů na penalizaci rozvrhu

Chlazení [%]		0,1	0,5	1	3	5	7	9	15	20	25	30
Millar	min	0	0	0	100	0	100	100	100	100	100	100
	max	900	800	1400	1400	1500	1700	1800	1800	1600	1700	1700
	avg pen	299	314	314	410	515	561	566	600	695	652	741
	avg time	47,18	15,33	8,68	3,51	2,42	1,88	1,64	1,21	1,15	1,16	1,15
Millar-s	min	0	0	0	0	0	0	0	0	0	0	0
	max	300	400	400	1300	1300	1300	1200	1300	1300	1200	1300
	avg pen	18	73	91	189	224	356	278	395	447	467	430
	avg time	27,8	11,82	6,46	2,74	1,85	1,45	1,21	0,89	0,78	0,75	0,74
LLR	min	198	198	198	198	198	199	199	199	199	199	199
	max	204	203	203	204	204	206	207	207	207	209	208
	avg pen	199,44	199,3	198,98	199,19	200,62	202,07	202,4	202,44	202,93	202,68	203,04
	avg time	36,68	8,27	4,57	1,91	1,34	1,17	1,11	1,09	1,10	1,09	1,11
GPOST	min	27	226	425	226	426	438	237	233	439	241	233
	max	2221	2540	3220	3420	4432	3430	4026	4431	4426	4431	3426
	avg pen	457,8	1130,85	1224,65	1247,45	1456,9	2137,3	2207,2	1987,65	2252,35	2189,2	1869,75
	avg time	2182,64	607,82	307,85	113,43	71,71	53,78	43,17	29,27	23,66	21,20	20,82
VALOUXIS	min	560	420	480	460	420	560	480	600	1500	620	660
	max	1800	1920	1900	4600	4840	8940	9480	6520	7600	7740	7420
	avg pen	799,6	846,8	876,4	1590	2594,4	3388,8	3773,2	3330	3503,6	3734	3557,6
	avg time	1063,47	275,24	160,76	94,64	79,76	75,82	75,40	73,60	73,75	74,20	76,13

Tabulka E.17: Chain Algorithm – vliv snižování teploty na penalizaci rozvrhu

Teplota		10	50	100	500	1000	5000	10000	20000	50000	100000
Millar	min	100	0	0	0	0	0	0	0	0	0
	max	1600	1900	1500	1400	1100	700	1200	1200	1200	700
	avg pen	770	629	507	339	291	307	333	310	327	325
	avg time	1,24	1,41	2,13	6,49	8,88	13,99	16,21	18,58	21,58	23,36
Millar-s	min	0	0	0	0	0	0	0	0	0	0
	max	1300	1300	1400	300	400	1100	300	1000	1200	400
	avg pen	412	322	192	89	81	96	90	72	87	74
	avg time	0,76	1,23	2,17	5,20	6,26	10,21	12,05	12,51	14,95	16,37
LLR	min	198	198	198	198	198	198	198	198	198	198
	max	201	201	202	202	203	202	203	205	203	203
	avg pen	198,85	198,84	198,98	199,16	199,35	199,46	199,51	199,35	199,39	199,23
	avg time	2,92	4,64	5,44	7,26	8,02	10,06	10,87	11,83	12,92	13,89
GPOST	min	228	226	225	227	223	34	223	29	223	228
	max	4017	3433	3430	3430	3420	3425	3422	2422	3225	3419
	avg pen	1635,83	1562,93	1436,7	1425,81	1220,55	972,24	1269,83	943,81	1072,29	1158,85
	avg time	82,43	142,64	169,37	230,58	259,73	324,57	347,31	379,13	415,39	428,31
VALOUXIS	min	440	460	460	420	440	420	400	500	400	500
	max	6840	6500	6560	5720	2660	2700	1900	2560	1900	3000
	avg pen	3136,4	2339,6	1728,4	1167	824,2	838,8	839,2	832,2	833,2	872
	avg time	74,17	81,73	90,56	113,82	161,14	277,02	327,56	383,79	459,23	514,74

Tabulka E.18: Chain Algorithm – vliv počáteční teploty na penalizaci rozvrhu

Pravděpodobnost [%]		0	10	20	30	40	50	60	70	80	90	100
Millar	min	800	0	0	0	100	0	200	200	100	100	400
	max	800	1900	2800	2000	1800	2000	1800	1900	2000	2800	400
	avg pen	800	896	975	854	877	1054	934	951	960	1030	400
	avg time	0,28	0,28	0,27	0,27	0,28	0,27	0,28	0,27	0,28	0,28	0,28
Millar-s	min	1000	0	0	0	0	0	0	0	0	0	0
	max	1000	1400	1400	1400	1300	2200	1300	2300	1400	2100	0
	avg pen	1000	745	680	700	630	670	658	601	569	664	0
	avg time	0,21	0,21	0,20	0,20	0,20	0,20	0,20	0,20	0,20	0,21	0,15
LLR	min	222	201	200	200	200	200	201	201	200	200	208
	max	222	216	215	214	213	213	214	214	213	214	208
	avg pen	222	206,58	205,41	205,08	205,16	204,78	205,37	205,95	205,28	205,85	208
	avg time	1,07	1,08	1,08	1,08	1,08	1,07	1,08	1,08	1,08	1,08	1,10
GPOST	min	4240	651	829	633	441	636	428	629	440	635	2636
	max	4240	5634	5445	6367	5831	5560	5554	5438	5561	5426	2636
	avg pen	4240	3034,97	2954,27	2867,93	3007,36	3134,7	2937,9	2757,71	2911,79	2743,81	2636
	avg time	3,42	3,39	3,39	3,39	3,4	3,39	3,39	3,39	3,39	3,39	3,39
VALOUXIS	min	5800	700	600	620	660	600	520	560	500	480	2660
	max	5800	9860	7720	5640	5860	7720	5820	5580	7600	4700	2660
	avg pen	5800	3768,2	2822	2550	2383,8	2233,4	2100,6	2116,2	1976	1939,2	2660
	avg time	25,43	25,45	25,45	25,49	25,46	25,52	25,50	25,50	25,49	25,51	25,45

Tabulka E.19: Hill Climbing – vliv velikosti pravděpodobnosti přijmutí stejně ohodnoceného řešení na penalizaci

Pravděpodobnost [%]		0	10	20	30	40	50	60	70	80	90	100
Millar	min	300	0	0	100	0	0	100	0	0	0	200
	max	300	1500	1700	1600	1400	1600	1500	1700	1500	1900	200
	avg pen	300	454	505	561	475	483	523	530	524	504	200
	avg time	1,48	1,44	1,45	1,45	1,45	1,41	1,44	1,44	1,44	1,42	1,42
Millar-s	min	300	0	0	0	0	0	0	0	0	0	1200
	max	300	1300	1200	1200	1300	1400	1200	1300	1200	1200	1200
	avg pen	300	285	261	265	258	247	349	337	307	379	1200
	avg time	1,15	1,02	1,03	1,01	1,02	1,02	1,04	1,02	1,03	1,06	1,10
LLR	min	201	198	198	198	198	198	198	198	198	198	200
	max	201	202	202	202	202	202	202	202	202	203	200
	avg pen	201	199,39	199,25	199,23	199,12	199,06	199,17	199,25	199,23	199,39	200
	avg time	1,07	1,06	1,06	1,06	1,05	1,06	1,07	1,07	1,11	1,06	1,14
GPOST	min	3223	237	228	427	241	236	234	230	235	421	438
	max	3223	4436	4445	4431	3434	4026	4427	3427	4030	3435	438
	avg pen	3223	2405,71	1953,76	2104,14	2049,87	1735,49	1835,33	1650,91	1878,12	1615,95	438
	avg time	34,92	33,22	33,43	33,51	33,43	33,15	33,33	33,21	33,11	33,48	33,90
VALOUXIS	min	2640	540	440	440	500	540	440	460	480	540	1560
	max	2640	7720	6780	3820	5660	4660	4620	5540	4800	4840	1560
	avg pen	2640	1880,4	1974,8	1760,8	1927,2	1830,6	1711,8	1954	1614	1917,6	1560
	avg time	51,65	50,24	49,91	50,12	49,80	50,17	50,26	50,57	49,83	49,99	50,25

Tabulka E.20: Chain Algorithm – vliv velikosti pravděpodobnosti přijmutí stejně ohodnoceného řešení na penalizaci

Pravděpodobnost [%]		0	10	20	30	40	50	60	70	80	90	100
Millar	min	200	0	0	0	0	0	0	0	0	0	0
	max	200	800	800	800	800	800	800	800	600	600	0
	avg pen	200	362	371	356	350	273	250	144	105	66	0
	avg time	7,92	5,53	4,90	4,30	4,01	3,68	3,42	2,68	1,96	1,37	0,54
Millar-s	min	100	0	0	0	0	0	0	0	0	0	0
	max	100	200	300	300	200	100	100	100	100	0	0
	avg pen	100	83	45	25	14	10	5	3	2	0	0
	avg time	3,59	2,09	1,29	0,94	0,69	0,60	0,46	0,41	0,43	0,32	0,32
LLR	min	198	198	198	198	198	198	198	198	198	198	198
	max	198	201	201	201	202	201	201	200	200	199	198
	avg pen	198	198,57	199,29	199,33	199,38	199,35	199,01	198,7	198,46	198,2	198
	avg time	24,71	51,22	58,08	57,21	61,54	58,87	56,26	52,56	48,61	43,58	39,10
GPOST	min	2234	426	239	239	239	239	239	239	239	239	1448
	max	2234	3554	3442	3442	3227	3440	2435	1630	1448	1454	1448
	avg pen	2234	1610,76	1118,34	1118,34	921,76	791,84	630,62	642,02	622,96	900,22	1448
	avg time	155,81	144,67	134,92	134,92	121,27	123,07	117,34	115,83	117,32	131,30	146,96
VALOUXIS	min	1480	460	460	400	1660	500	480	1560	640	620	5460
	max	1480	6640	8600	8700	10640	12540	6800	9580	7560	7680	5460
	avg pen	1480	2884	2926,4	3504	3921,6	4326,4	3519,2	4096,8	4545,6	5078,4	5460
	avg time	615,76	759,02	752,37	753,34	787,96	694,91	882,35	735,20	835,46	812,16	769,96

Tabulka E.21: Tabu Search – vliv velikosti pravděpodobnosti přijmutí stejně ohodnoceného řešení na penalizaci

Teplota		10	50	100	500	1000	5000	10000	20000	50000	100000
Millar	min	0	0	0	0	0	100	100	0	0	0
	max	1300	1400	1400	1500	1300	1300	1600	1500	1400	1500
	avg pen	380	450	338	353	374	487	416	398	451	463
	avg time	0,64	0,64	0,65	0,64	0,65	0,65	0,65	0,65	0,65	0,65
Millar-s	min	0	0	0	0	0	0	0	0	0	0
	max	1000	1100	1000	1100	1100	1000	1100	1100	1100	1200
	avg pen	119	97	79	84	99	88	138	131	139	189
	avg time	0,34	0,33	0,32	0,36	0,38	0,41	0,41	0,43	0,44	0,46
LLR	min	199	198	200	201	200	201	202	202	202	204
	max	210	213	214	215	215	217	225	225	227	228
	avg pen	203,93	204,32	205,03	206,03	205,64	207,85	209,46	211,36	212,88	215,51
	avg time	0,42	0,42	0,42	0,41	0,41	0,42	0,46	0,41	0,41	0,41
GPOST	min	427	224	232	235	237	234	231	231	434	433
	max	4623	4427	5036	4235	4352	5226	4549	5344	4235	4548
	avg pen	2038,09	2120,95	2187,29	1977,65	1792,52	1876,45	1831,64	1759,67	1889,09	1914,79
	avg time	4,46	4,46	4,46	4,50	4,49	4,48	4,49	4,49	4,5	4,49
VALOUXIS	min	960	760	920	720	680	820	1820	1980	2020	1900
	max	8820	9640	8680	10140	6940	11780	13080	10920	12020	13880
	avg pen	4367,2	4086,2	4183,6	4105,8	3802,6	4876	5490,4	5952,6	6540,8	7240,2
	avg time	7,76	7,76	7,76	7,77	7,77	7,79	7,80	7,79	7,8	7,82

Tabulka E.22: Simulated Annealing – vliv počáteční teploty na penalizaci rozvrhu

Chlazení [%]		0,1	0,5	1	3	5	7	9	15	20	25	30
Millar	min	0	0	0	0	0	0	0	0	0	0	0
	max	1300	1400	1500	1300	1300	1300	1400	1300	1300	1300	1300
	avg pen	329	386	380	373	398	424	474	345	413	418	370
	avg time	0,61	0,66	0,60	0,59	0,61	0,60	0,60	0,60	0,61	0,61	0,59
Millar-s	min	0	0	0	0	0	0	0	0	0	0	0
	max	1300	1000	1100	1100	1100	1000	1000	1000	1100	1000	1000
	avg pen	558	89	105	113	131	60	90	40	132	70	100
	avg time	0,35	1,13	1,52	1,86	2,13	1,79	2,37	1,85	3,13	2,41	3,05
LLR	min	201	200	200	200	199	200	201	201	200	200	200
	max	212	213	220	223	250	214	226	224	236	227	221
	avg pen	205,57	204,26	204,24	204,33	205,76	204,22	204,33	204,53	204,96	204,49	204,66
	avg time	1,76	7,38	11,51	14,53	15,71	15,63	17,02	17,40	19,31	19,33	18,41
GPOST	min	445	435	434	633	431	634	427	635	631	425	453
	max	4834	6559	6807	7556	6681	6189	7103	6212	5442	6196	5780
	avg pen	3042,98	3121,66	2762,12	2772,3	2994,8	2867,98	2947,3	2857,95	2907,25	2681,75	2953,16
	avg time	5,87	8,74	10,71	11,88	13,8	15,08	16,56	17,67	18,84	21,87	21,54
VALOUXIS	min	500	640	660	640	560	600	600	520	600	600	600
	max	7620	6760	7760	7900	6720	8000	5820	7820	6840	7760	8780
	avg pen	2319,6	2612,17	2753,07	2814,87	2585,29	2772,38	2469,1	2549,16	2645,36	2698,59	2787,89
	avg time	41,75	42,31	43,18	52,86	49,95	46,62	48,25	48,42	49,77	48,55	47,54

Tabulka E.23: Simulated Annealing – vliv snižování teploty na penalizaci rozvrhu

Počet kroků		100	250	500	750	1000	1500	2000	3000	4000	5000	10000
Millar	min	700	400	200	100	0	0	0	0	0	0	0
	max	3400	2900	2200	1600	1500	1400	1400	1400	1200	1100	1200
	avg pen	2142	1203	806	512	512	406	326	283	202	170	178
	avg time	0,04	0,10	0,21	0,31	0,41	0,60	0,81	1,18	1,55	1,89	3,45
Millar-s	min	400	0	0	0	0	0	0	0	0	0	0
	max	3400	2400	1100	1200	1200	1100	1000	1000	1000	1000	1000
	avg pen	1659	812	372	240	140	94	52	62	63	41	20
	avg time	0,03	0,07	0,15	0,20	0,26	0,31	0,33	0,4	0,40	0,47	0,45
LLR	min	350	295	218	200	202	200	201	199	199	198	199
	max	40414	20336	265	230	219	215	210	209	207	207	208
	avg pen	18772,37	1929,92	237,72	215,15	209,17	204,98	203,59	203,46	203,58	203,3	203,58
	avg time	0,01	0,06	0,12	0,19	0,25	0,37	0,50	0,76	1,02	1,27	2,54
GPOST	min	4536	1719	504	281	234	431	623	29	29	40	225
	max	15612	9201	4905	5467	4432	5230	4439	4154	4339	4021	4421
	avg pen	10639,71	4860,56	2622,15	2072,89	1967,05	1884,95	2041,64	1585,01	1508,48	1559,21	1420,76
	avg time	0,28	0,7	1,39	2,08	2,77	4,14	5,50	8,23	10,96	13,73	27,41
VALOUXIS	min	63740	56600	43500	29640	22380	8140	2020	780	660	640	440
	max	94640	81800	63360	54380	45260	22440	13320	6200	3140	2100	920
	avg pen	79432	69490,4	54102,8	42520,2	31374,6	15115,6	7053,2	2341,4	1284,8	1017	638,2
	avg time	0,48	1,22	2,44	3,66	4,87	7,33	9,70	14,56	19,36	24,33	48,49

Tabulka E.24: Simulated Annealing – vliv počtu kroků na penalizaci rozvrhu

Velikost Tabu Listu		5	10	15	20	25	30	35	40	45	50
Millar	min	0	0	0	0	0	0	0	0	0	0
	max	900	800	800	800	800	800	800	800	800	800
	avg pen	272	280	300	315	272	290	283	266	305	316
	avg time	4,16	6,43	8,34	10,55	12,33	15,49	17,08	19,17	21,82	25,71
Millar-s	min	0	0	0	0	0	0	0	0	0	0
	max	100	200	100	100	100	200	100	100	200	200
	avg pen	9	10	7	6	9	14	10	5	4	12
	avg time	0,77	0,95	1,00	1,08	1,52	2,16	2,09	1,55	1,26	3,10
LLR	min	198	198	198	198	198	198	198	198	198	198
	max	201	201	201	201	201	201	200	201	201	201
	avg pen	199,26	199,33	199,13	199,23	199,2	199,03	199,16	199,16	199,2	199,3
	avg time	67,10	115,41	158,08	200,31	245,66	291,08	329,54	382,06	420,55	492,49
GPOST	min	239	239	239	239	239	239	239	239	239	239
	max	3228	3230	836	3621	2427	3227	3230	2225	2432	1840
	avg pen	1128,05	830,05	459,2	830,55	678,5	809,8	700,1	708,95	757,7	599,2
	avg time	131,15	186,21	262,50	355,16	429,35	480,23	563,67	673,07	697,40	710,04
VALOUXIS	min	1540	500	600	620	1660	520	500	640	740	1600
	max	5660	8700	4680	7480	6680	7580	6540	7620	6380	8680
	avg pen	3240	4102	2454	3476	4294	3376	3240	3712	3348	4632
	avg time	926,63	1190,90	1753,55	2092,24	3051,27	2987,48	4094,23	3880,70	4218,69	5322,27

Tabulka E.25: Tabu Search – vliv velikosti Tabu Listu na penalizaci rozvrhu

Heuristika		HC	SA	TS	CHAIN	TPVDS-HC	TPVDS-SA	TPVDS-HC 30	TPVDS-SA 30	TPVDS-HC 60	TPVDS-SA 60	TPVDS-HC 300	TPVDS-SA 300	SS-HC	SS-SA
Millar	min	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	max	1200	1100	800	1300	1600	300	1600	1200	1600	400	1600	200	0	0
	avg pen	265	218	315	273	388	143	410	166	390	102	340	100	0	0
	avg time	4,55	1,79	5,69	5,63	3,28	21,29	30,02	30,31	60,02	60,29	300,02	300,13	28,04	10,28
Millar-s	min	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	max	1000	1000	100	200	1100	1000	1100	1000	1000	100	200	0	0	0
	avg pen	73	22	5	40	58	25	114	11	104	2	60	0	0	0
	avg time	1,21	0,51	0,68	3,03	2,31	2,79	30,01	30,01	60,01	60,01	300,01	300,01	1,29	0,50
LLR	min	201	199	198	198	198	192	202	199	201	198	203	198	197	190
	max	209	207	200	200	201	198	211	202	210	201	211	200	203	200
	avg pen	204,48	203,31	198,58	198,17	199,11	197	206,37	200,25	206,06	199,96	205,7	199,1	201,09	196,63
	avg time	17,81	2,03	111,79	7,21	1988,82	4869,00	30,04	30,87	60,03	60,74	300,02	300,72	501,66	88,06
GPOST	min	230	227	441	231	32	32	430	255	429	433	826	48	32	28
	max	6160	4546	2439	2431	3426	433	5632	4047	5222	2639	4232	834	325	428
	avg pen	2716,93	2096,24	1411,52	778,74	1603,8	192,6	2666,27	1500,01	2453,68	1140,24	2323,4	521,7	167,12	184,81
	avg time	7,98	5,07	102,96	80,94	188,81	1725,44	30,12	32,14	60,13	63,51	300,11	300,82	829,54	324,50
VALOUXIS	min	260	620	640	540	220	260	22100	620	3680	660	420	540	160	500
	max	1920	4220	5820	3740	400	420	45080	5080	17880	2940	580	1540	280	780
	avg pen	401,6	1857	3233	1171	322	320	34420,8	2089,2	9046,4	1391,2	506	768	225	658,18
	avg time	196,58	8,97	587,82	49,32	1186,09	6251,77	30,23	32,24	60,55	64,57	300,48	303,55	8781,84	479,43

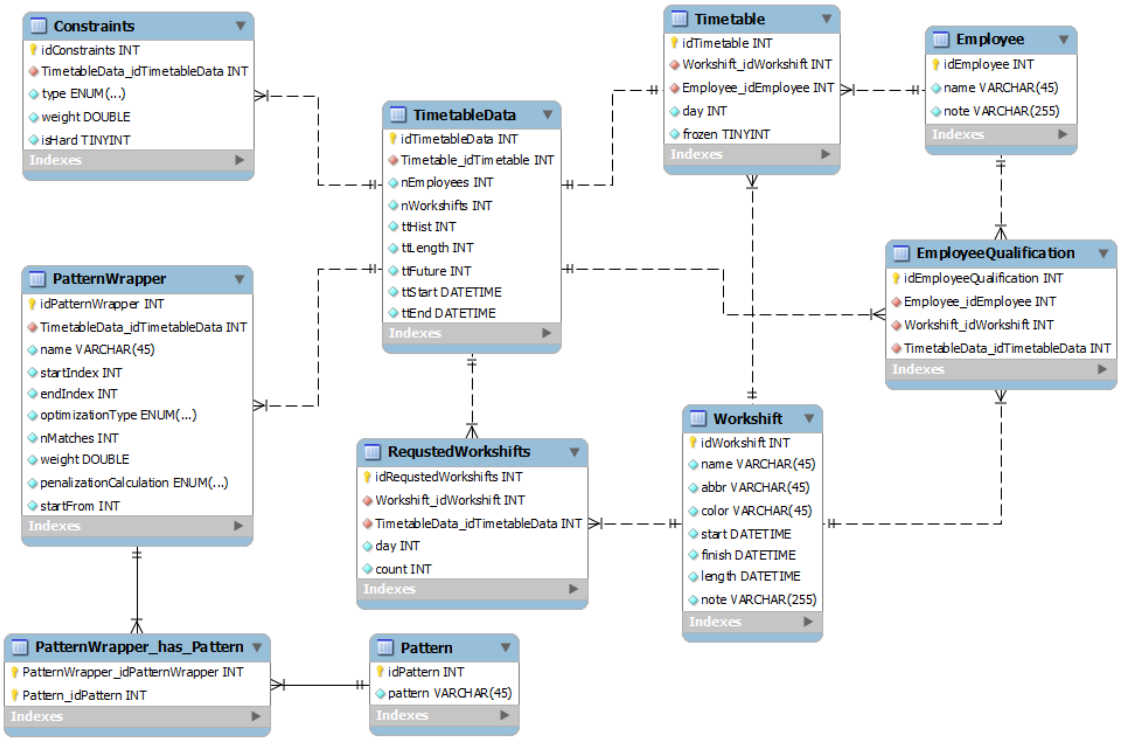
Tabulka E.26: Porovnání algoritmů na benchmarkových instancích I

Heuristika		HC	SA	TS	CHAIN	TPVDS-HC	TPVDS-SA	TPVDS-HC 30	TPVDS-SA 30	TPVDS-HC 60	TPVDS-SA 60	TPVDS-HC 300	TPVDS-SA 300	SS-HC	SS-SA
WHPP	min	420	834	530	416	514	715	4569	1049	1056	1047	522	868	2	17
	max	615	1337	634	717	818	717	19466	1748	6649	1348	832	1061	7	536
	avg pen	541,5	1046,29	596,66	481,54	692	716	12139,06	1241,6	3350,68	1187,76	714,4	1002,6	4,25	61,25
	avg time	845,53	16,83	1628,68	149,67	1179,70	8489,96	30,75	31,56	61,03	63,13	300,92	314,01	32246,81	1523,26
AZAIES	min	3	14	11	4	3	8	10	43	4	33	4	20	1	12
	max	65	47	24	15	18	17	56	76	20	64	18	45	11	21
	avg pen	32,36	28,88	16,12	10,38	9,25	14	29,36	59,64	14,04	54,96	11,9	35	4	17,9
	avg time	27,41	5,87	179,56	95,63	101,78	1356,07	30,21	34,08	60,17	64,01	300,21	301,89	905,28	266,72
SINTEF	min	0	21	10	0	0	1	5940	38	54	54	0	35	0	11
	max	66	101	21	21	20	10	15798	128	2559	117	21	66	0	36
	avg pen	18,72	60,1	13,5	6,81	8,1	5,5	10324,82	85,28	867,72	80,76	8,1	48,6	0	22,25
	avg time	156,22	12,82	770,16	188,64	205,28	2934,64	30,76	38,80	60,92	64,68	300,15	309,05	652,21	681,44
ORTEC	min	2930	1791	2646	1622	3515	2781	79171	2086	54	1985	3846	1028	595	735
	max	7000	8865	3691	3661	4575	9426	85171	9575	2559	7198	11091	4876	1722	1720
	avg pen	4924,78	4479,66	3168,5	2898	4045	5270,66	84451	5300,72	867,72	4017,92	7180,7	2797,5	977,87	879,62
	avg time	44,89	20,28	5222,37	318,31	6348,81	15007,66	31,24	37,38	60,92	74,56	301,06	315,82	3324,08	1420,38
MUSA	min	45	45	46	46	45	45	45	45	45	45	45	45	45	45
	max	47	47	53	48	47	47	48	46	47	46	46	45	45	46
	avg pen	45,93	46,12	47,75	47,13	45,34	45,75	45,5	45,4	45,84	45,32	45,8	45	45	45,03
	avg time	5,57	1,16	6,70	7,48	16,31	185	30,02	30,27	60,02	60,29	300,03	300,53	270,18	102,88

Tabulka E.27: Porovnání algoritmů na benchmarkových instancích II

Příloha F

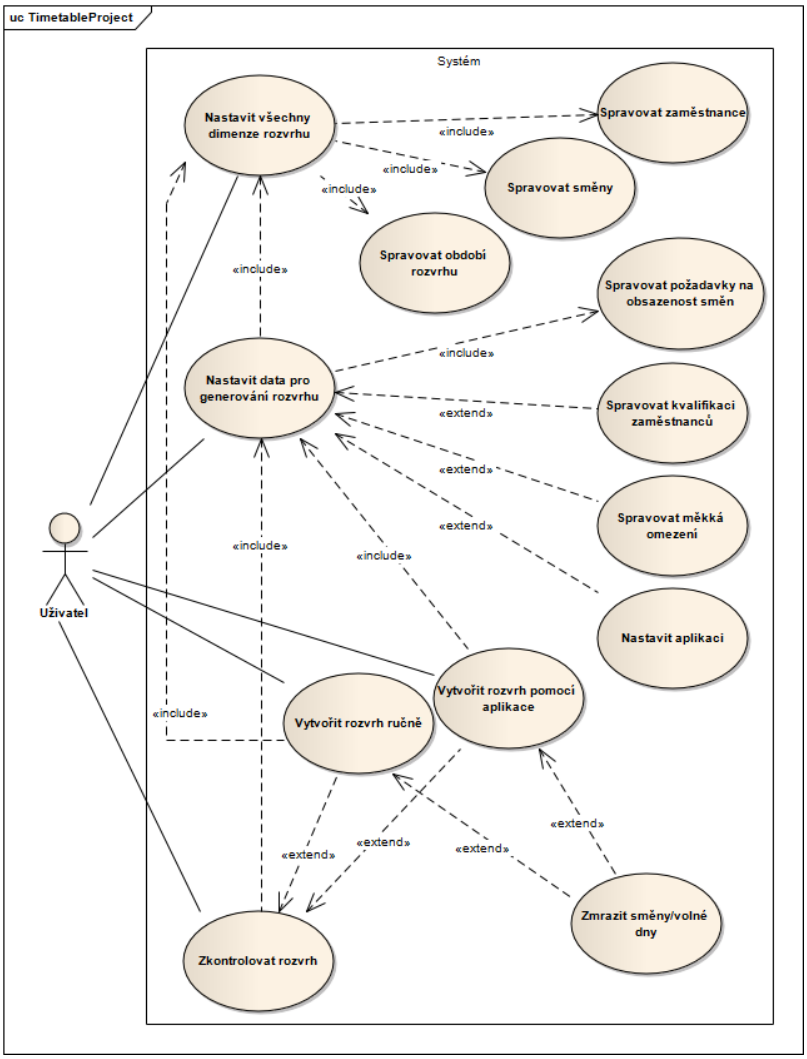
Datový model



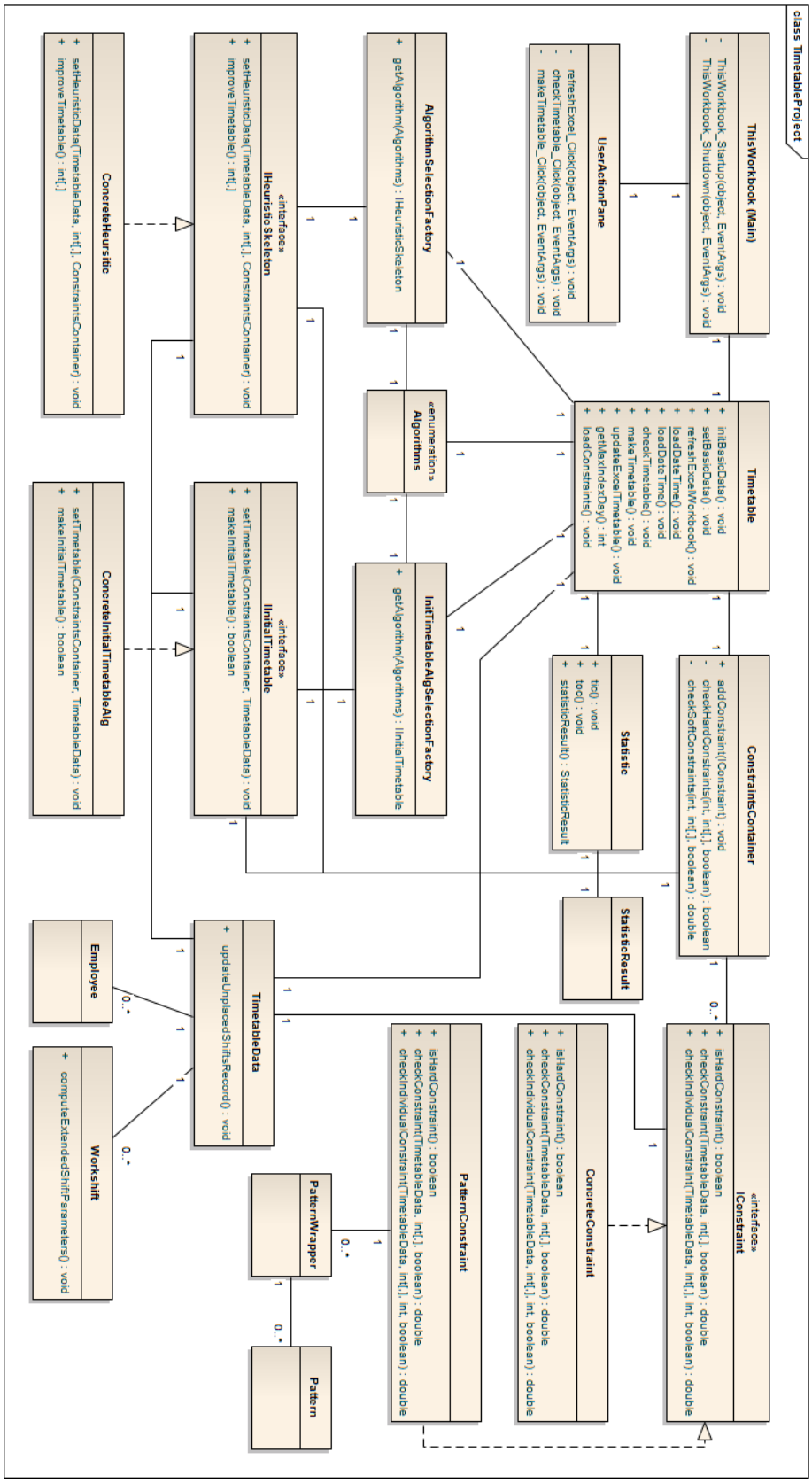
Obrázek F.1: Datový model systému

Příloha G

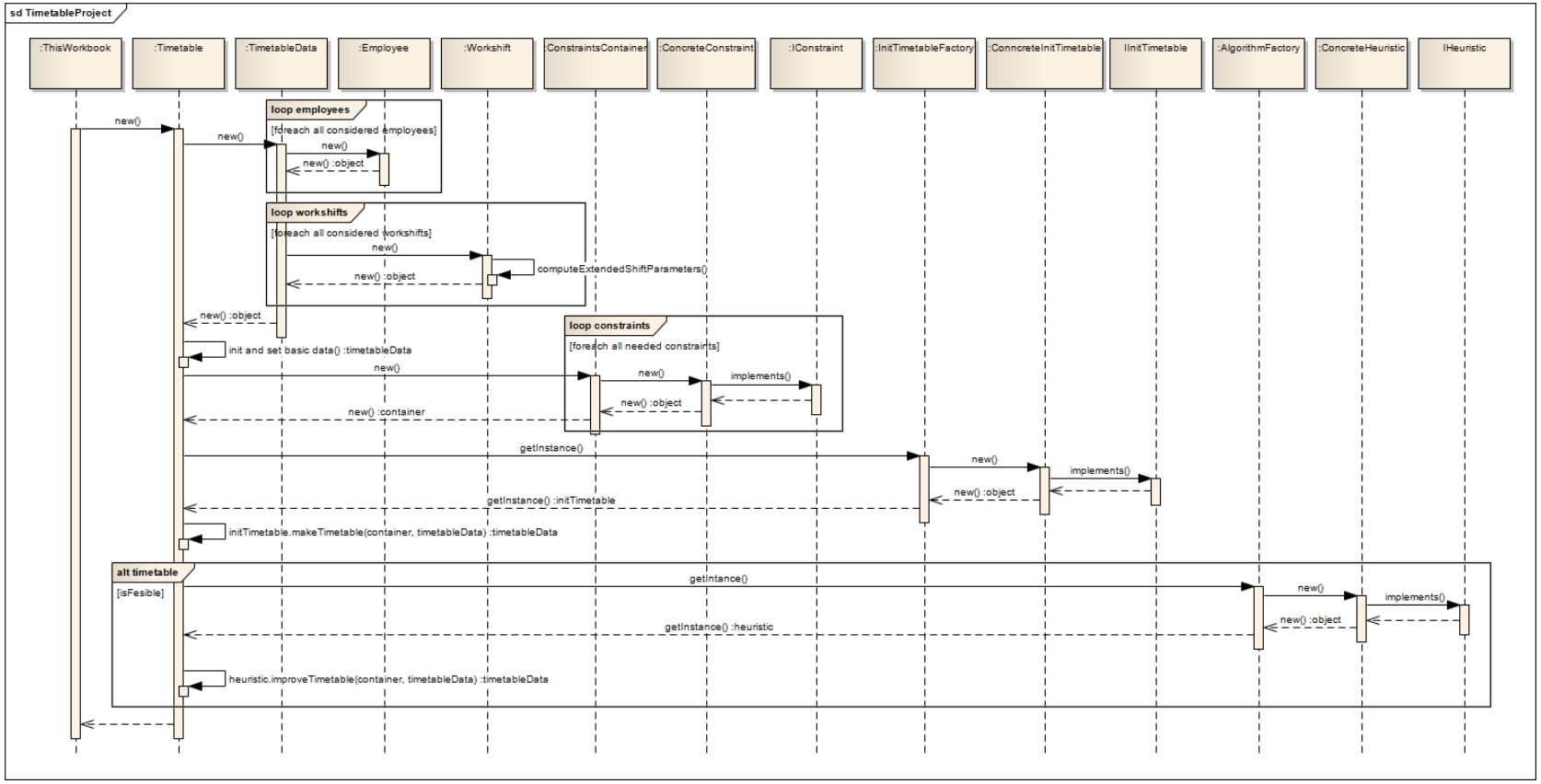
UML



Obrázek G.1: Případy užití – interakce se systémem



Obrázek G.2: Diagram tříd



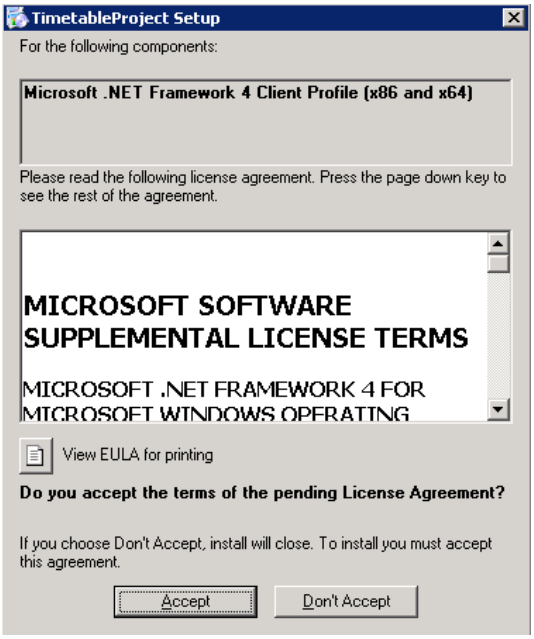
Obrázek G.3: Sekvenční diagram – tvorba rozvrhu

Příloha H

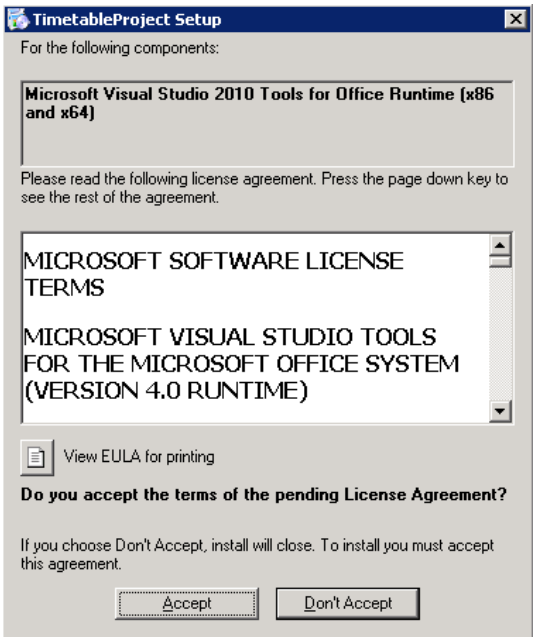
Instalační manuál

Před instalací nejprve zkontrolujete, zda váš počítač splňuje požadavky jak na hardware (1.6), tak i software (1.5). Pokud tomu tak není, zvažte prosím upgrade. Navíc je nutné mít přístup k internetu kvůli stažení podpůrných programů (pouze v případě, že již nejsou nainstalované). Instalační soubory se nachází ve složce *install*.

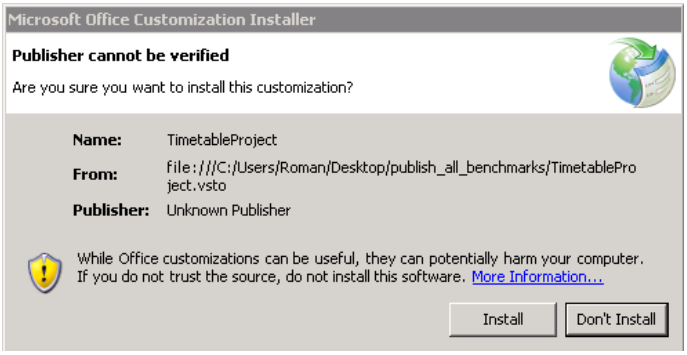
1. Spusťte soubor *setup.exe*.
2. Potvrďte všechny dialogy (H.1, H.2, H.3), které provázejí instalací nezbytných věcí pro chod aplikace (nutný přístup k internetu).
3. Po dokončení instalace podpůrných programů (H.4) spusťte soubor *TimetableProject.xlsx*.
4. Nyní už můžete aplikaci používat (otevírejte souborem *TimetableProject.xlsx*).



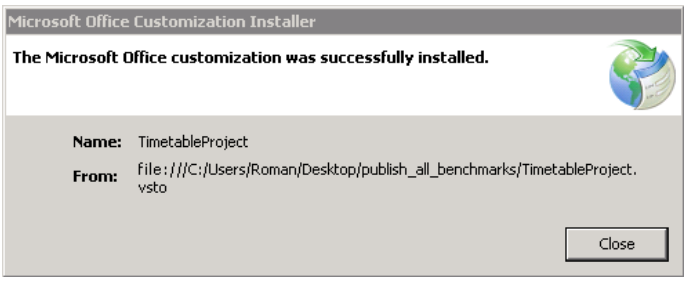
Obrázek H.1: Postup instalace – krok 1



Obrázek H.2: Postup instalace – krok 2



Obrázek H.3: Postup instalace – krok 3

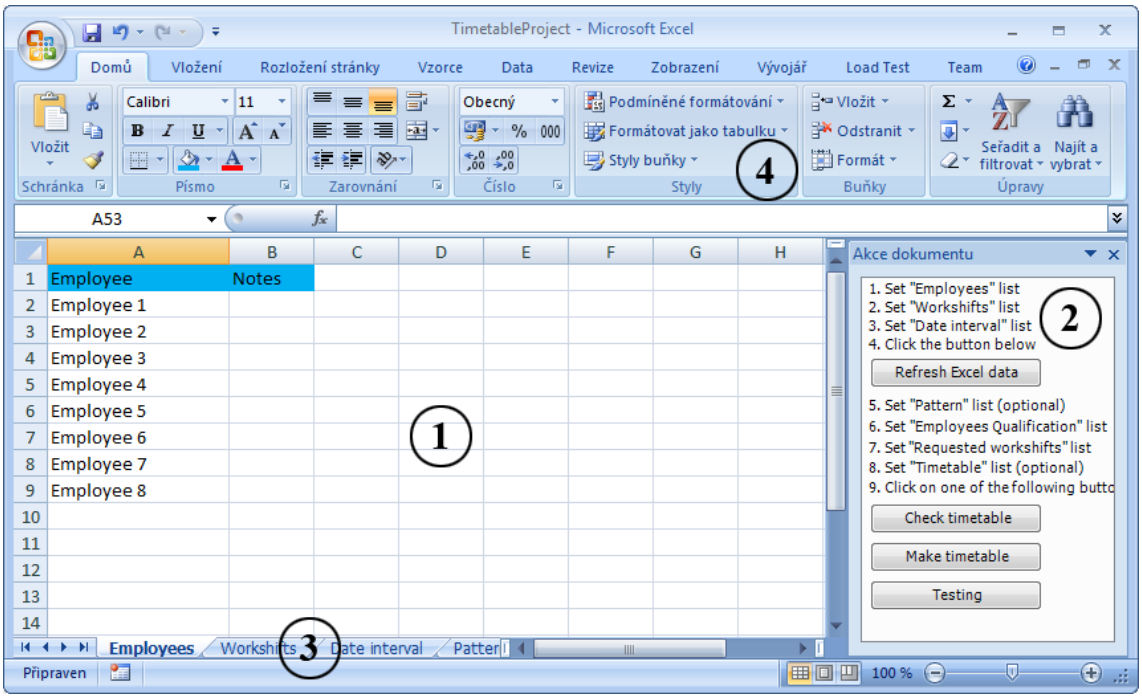


Obrázek H.4: Postup instalace – krok 4

Příloha I

Uživatelský manuál

Vzhled aplikace



Obrázek I.1: Vzhled aplikace

Jednotlivé části:

- obsahová část (I.1, část 1),
- panel akcí (I.1, část 2),
- panelu listů (I.1, část 3),
- menu (I.1, část 4).

Správa zaměstnanců

	A	B
1	Employee	Notes
2	Emp 1	
3	Emp 2	
4	Emp 3	

Obrázek I.2: List se zaměstnanci

V panelu listů vyberte záložku „Employees“. Zde zadejte jména zaměstnanců (nebo jiný identifikátor), kteří mají být v rozvrhu a k nim případné poznámky.

Správa směn

	A	B	C	D	E	F	G
1	Workshift	Abbr	From	To	Length	Color	Notes
2	Day	D	9:00	21:00	12:00		
3	Night	N	21:00	9:00	12:00		

Obrázek I.3: List se směnami

V panelu listů vyberte záložku „Workshifts“. Zde nastavte všechny směny, na kterých mohou zaměstnanci pracovat. Zadává se:

- název a zkratka směny,
- čas, kdy začíná a končí směna (ve formátu $HH : MM$),
- délka směny (ve formátu $HH : MM$),
- barva směny v rozvrhu (změnu provedete vybráním potřebné barvy výplně z menu),
- případné poznámky.

Nastavení období pro rozvrhování

	A	B	C	D
1	No. history days	From	To	No. future days
2	0	2.10.2006	8.10.2006	0

Obrázek I.4: List s obdobím pro rozvrhování

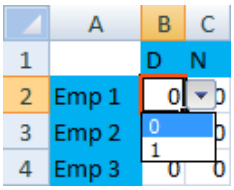
V panelu listů vyberte záložku „Date interval“. Pokud chcete aktuálně tvořený rozvrh navázat na minulý, vyplňte potřebný počet dnů historie. Na ty se nevztahují žádná tvrdá

omezení (1.2.3 a 4.1.1). Jestliže chcete mít jistotu, že na aktuálně tvořený rozvrh půjde navázat nebo jen chcete zjistit, jak bude přibližně vypadat ten následující, vyplňte potřebný počet dnů budoucnosti. Požadavky na obsazenost směn se zkopírují z těch aktuálních. Nutné je vyplnění období pro rozvrhování, tedy datum od a do (ve formátu $DD.MM.RRRR$).

Aktualizace dat

Poté, co projdete všemi výše uvedenými body, klikněte na tlačítko „Refresh Excel data“ v panelu akcí. To způsobí aktualizaci dat vzhledem k těm již vámi vyplněných v následujících záložkách panelu listů.

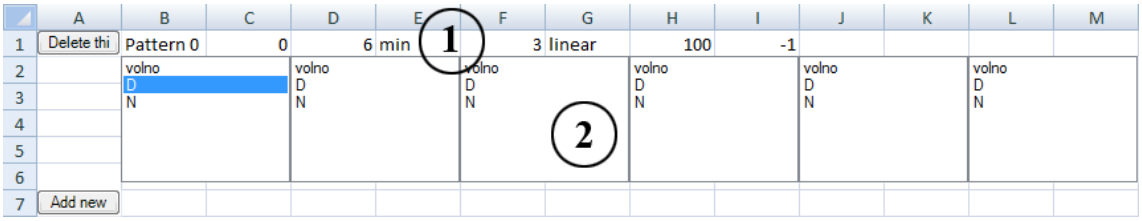
Správa kvalifikace zaměstnanců



Obrázek I.5: List s kvalifikací zaměstnanců

Pokud se rozhodnete používat tvrdá omezení pro kvalifikaci zaměstnanců, tak v panelu listů vyberte záložku „Employees qualification“. Zde přiřaďte každému zaměstnanci jedničku k dané směně, pokud na ní má kvalifikaci nebo nulu v opačném případě.

Správa měkkých omezení



Obrázek I.6: List s měkkými omezeními

Jestliže chcete nastavit různé preference (měkká omezení – 4.1.2.1 a 4.1.2), tak v panelu listů vyberte záložku „Patterns“. Zde pomocí tlačítka „Add new“ v zobrazovací části přidejte nový vzor. Zobrazí se jeho záhlaví (I.6, část 1) a samotná možnost nadefinování vzoru (I.6, část 2).

Popis záhlaví (tak jak jdou sloupce za sebou):

- název celého vzoru,
- index dne, od kterého začíná možná působnost vzoru (možné hodnoty – 0 až délka období rozvrhu zmenšená o jedničku, první den = index 0),

- index dne, ve kterém začíná možná působnost vzoru (možné hodnoty – 0 až délka období rozvrhu zmenšená o jedničku, musí být větší nebo rovno předchozímu indexu),
- jakého minimálního/maximálního počtu výskytů vzoru chcete v ideálním případě dosáhnout (možné hodnoty – „min“ nebo „max“),
- to samé jako v minulém bodě, akorát zde vyplňte právě ten počet výskytů,
- váha tohoto omezení,
- výpočet penalizace (možné hodnoty – „linear“ nebo „quadratic“) – kvadratický volte pouze v případě, pokud chcete zohlednit, jak moc se liší aktuální počet výskytů vzoru v rozvrhu od toho chtěného (čili čím vyšší rozdíl, tím penalizace strměji narůstá),
- začátek uplatňování vzoru v rámci jeho výše definované působnosti:
  - -1: kdekoliv,
  - 0: pouze na začátku,
  - 1–7: pondělí až neděle.

Popis vzoru:

- nachází se zde 6 slotů, kde v každém můžete vybírat z několika položek (všechny směny + den volna),
- slot může zůstat prázdný,
- více položek ve slotu můžete vybrat pomocí přidržení tlačítka „Ctrl“ a následným klikáním levým tlačítkem myší na potřebné položky,
- takto nadefinovaný vzor se pak vyhledává v rozvrhu.

V následující tabulce je uvedeno pár příkladů (další lze nalézt v 4.1.2.2), které by vám měli pomoci se lépe zorientovat. Uvažují se v nich tři směny (denní – D, odpolední – O a noční – N), období dlouhé 20 dnů, váha pro všechny omezení stejná (např. 100) a lineární výpočet penalizace.

Omezení	Působnost	Postup	Počet výskytů	Začátek	Vzor*
Minimálně 3 po sobě jdoucí dny volna	0 – 19	max	0	-1	SVVS
	0 – 19	max	0	0	VVS
	17-19	max	0	-1	SVV
Maximálně 1 celý pracovní víkend	0 - 19	max	1	6	SS
Minimálně 5 odpracovaných denních směn	0-19	min	5	-1	D

* S – libovolná směna, V – den volna

Tabulka I.1: Ukázka zadávání vzorů

Můžete si všimnout, že u prvního omezení se bavíme o minimalizaci, ale vzor nastavujeme na maximalizaci. Tohoto obrácení se využívá poměrně hodně, protože se pak se vzorem dá pracovat mnohem lépe a jednodušeji.

Pomocí tlačítka „Delete this pattern“ můžete vámi vytvořené vzory odstraňovat.

Správa požadavků na obsazenost směn

	A	B	C	D	E	F	G	H
1		2.10.2006	3.10.2006	4.10.2006	5.10.2006	6.10.2006	7.10.2006	8.10.2006
2	D	0	0	0	0	0	0	0
3	N	0	0	0	0	0	0	0

Obrázek I.7: List s požadavky na obsazenost směn

V panelu listů vyberte záložku „Requested Workshifts“. Zde přiřaďte směnám ke všem dnům potřebný počet zaměstnanců.

Zmražení směn/dnů volna

Potřebujete-li nutně přiřadit nějakou směnu nebo den volna některému zaměstnanci, tak v panelu listů vyberte záložku „Timetable“ a zde přiřaďte styl „Frozen Workshift“ buňce, ve které je potřebná hodnota.

Vytvoření rozvrhu ruční cestou

	A	B	C	D	E	F	G	H
1		2.10.2006	3.10.2006	4.10.2006	5.10.2006	6.10.2006	7.10.2006	8.10.2006
2	Emp 1	D		D		N	N	
3	Emp 2	D N				N	N	N
4	Emp 3			N	N		D	D
5								

Obrázek I.8: List s rozvrhem

V panelu listů vyberte záložku „Timetable“ a zde přiřaďte směny pomocí rozevíracího seznamu zaměstnancům v jednotlivých dnech tak, jak potřebujete.

Nastavení aplikace

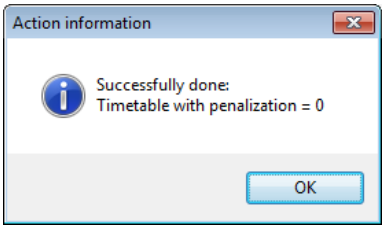
V panelu listů vyberte záložku „Settings“. Zde můžete vybrat algoritmy, které se mají použít pro tvorbu rozvrhu a zároveň je nastavit dle vaší libosti. Popis a vysvětlení jednotlivých možností je uveden v příloze D. Aplikace je ve výchozím režimu nastavena tak, aby co nejlépe vyhovovala různorodým problémům.

Vytvoření rozvrhu pomocí aplikace

Pokud máte vše nutné vyplněno a nastaveno (zaměstnanci, směny, období, požadavky na obsazenost směn), můžete nechat aplikaci vytvořit rozvrh dle vámi zadaných požadavků kliknutím na tlačítko „Make timetable“ v panelu akcí. Ihned poté co systém vygeneruje rozvrh, budete informování dialogovým oknem o proveditelnosti rozvrhu a jeho případné penalizaci. Zároveň také dojde k jeho vykreslení do listu „Timetable“.

Pozor! Vytváření rozvrhu může trvat až několik hodin. Závisí to jak na vstupních datech, tak výběru a nastavení použitých algoritmů.

Kontrola rozvrhu



Obrázek I.9: Informace o rozvrhu

Po vytvoření rozvrhu nebo jeho následné modifikaci můžete nechat rozvrh zkontrolovat kliknutím na tlačítko „Check Timetable“ v panelu akcí. Kontrole také musí předcházet řádné vyplnění nutných dat. Informování budete opět dialogovým oknem.

Příloha J

Obsah přiloženého CD

- Dokumenty: adresář obsahuje diplomovou práci ve formátu PDF.
- Install: adresář obsahuje instalační soubory aplikace.
- TimetableProject: adresář obsahuje zdrojové kódy aplikace.