České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Jakub Levý**

Studijní program: Otevřená informatika (magisterský)
Obor: Počítačové inženýrství

Název tématu: **Algoritmy pro efektivní prohledávání stavového prostoru v rozvrhování zdravotních sester**

Pokyny pro vypracování:

1. Proveďte rešerši problému rozvrhování zdravotních sester zaměřenou na metody pro efektivní prohledávání stavového prostoru.
2. Vyberte a naimplementujte nejvhodnější algoritmus z hlediska obecného použití na různé typy instancí rozvrhování zdravotních sester.
3. Navrhněte a naimplementujte vlastní algoritmus/mechanismus pro efektivní prohledávání stavového prostoru v rozvrhování zdravotních sester.
4. Porovnejte a otestujte naimplementované metody na benchmarkových instancích a diskutujte výsledek.

Seznam odborné literatury:

[1] E. K. Burke, G. Kendall, E. Soubeiga: A Tabu-Search Hyperheuristic for Timetabling and Rostering. Journal of Heuristics, 451 - 470, 2003.
[2] Eric Soubeiga: Development and application of hyperheuristics to personnel scheduling. PhD Thesis, The University of Nottingham, 2003.
[3] Wei Zhang, Thomas G. Dietterich: Solving Combinatorial Optimization Tasks by Reinforcement Learning: A General Methodology Applied to Resource-Constrained Scheduling. Journal of Artificial Intelligence Reseach, 2000.
[4] E. Ikonomovska, D. Gjorgjevik, S. Loskovska: Using Data Mining Technique for Coefficient Tuning of an Adaptive Tabu Search. EUROCON 2007: The international Conference on Computer as a Tool 706 - 713, 2007.
[5] Jingpeng Li, Edmund K. Burke, Rong Qu: Integrating neural networks and logistic regression to underpin hyper-heuristic search. Knowledge-Based Systems, 322 - 330, 2011.

Vedoucí: Ing. Roman Václavík

Platnost zadání: do konce letního semestru 2013/2014

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 5. 2. 2013

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky

Diplomová práce

Algoritmy pro efektivní prohledávání stavového prostoru v rozvrhování zdravotních sester

Jakub Levý

Vedoucí práce: Ing. Roman Václavík

Studijní program: Počítačové inženýrství, Magisterský

Obor: Otevřená informatika

9. května 2013

Poděkování

Rád bych touto cestou poděkoval vedoucímu této práce Ing. Romanu Václavíkovi za jeho časté a přínosné rady.

Prohlášení

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

V Praze, dne 9.5.2013 .....

.....
podpis

Abstract

The aim of this diploma thesis is the proposal of solution for nurse scheduling problem, which will be applicable on the instances with various parameters. Tabu-Search hyperheuristic was used after studying existing possibilities. It uses low-level heuristics which are based on the shift balancing. Considering the hyperheuristic drawbacks the new modified algorithm was designed in this thesis. The adjustments which are based on integrating of the neural network to manage steps of algorithm were done as a substitution of the Tabu-Search method. Finally, these three versions of the algorithm were compared on the set of testing data provided by the science group from Nottingham University.

Abstrakt

Diplomová práce se zabývá problémem rozvrhování směn zdravotních sester a návrhem takového řešení, které by bylo možné aplikovat na instance s různými parametry. Po prostudování různých již existujících možností, byla jako výchozí algoritmus vybrána hyperheuristika řízená pomocí metody tabu vyhledávání, která používá nízkoúrovňové heuristiky založené na bázi tzv. vyvážení směn. S ohledem na jeho nedostatky byl navržen nový, modifikovaný algoritmus. V další fázi na modifikované verzi proběhly úpravy v podobě integrování neuronové sítě pro řízení některých kroků jako náhrada metody tabu vyhledávání. Všechny tři verze algoritmu byly nakonec srovnány na množině testovacích dat uvolněné skupinou z Nottinghamské univerzity.

Obsah

Úvod **1**

1 Popis práce **3**
1.1 Cíle práce . . . 3
1.2 Popis problému . . . 4
1.2.1 Měkká omezení . . . 4
1.2.2 Tvrdá omezení . . . 5

2 Teoretický úvod **7**
2.1 Základní typy metod pro řešení NP úloh . . . 7
2.1.1 Úplné metody . . . 7
2.1.2 Heuristiky . . . 7
2.1.3 Hyperheuristiky . . . 7
2.2 Heuristické algoritmy . . . 8
2.2.1 Horolezecký algoritmus (Hill Climbing) . . . 8
2.2.2 Simulované žíhání (Simulated Annealing) . . . 8
2.2.3 Genetický algoritmus . . . 8
2.2.4 Tabu vyhledávání (Tabu Search) . . . 8
2.2.5 Zpětnovazební učení (Reinforcement Learning) . . . 9
2.3 Hyper-heuristiky . . . 9
2.4 Pokročilé metody . . . 10
2.4.1 Základní charakteristiky . . . 10
2.4.2 Zpětnovazební učení s využitím agentů . . . 10
2.4.3 Zpětnovazební učení s modifikovaným výběrem následujícího kroku . . . 10
2.4.4 Hyper-heuristika řízená tabu prohledáváním . . . 11
2.4.5 Využití vytěžování dat pro ladění koeficientů adaptivního tabu vyhledávání . . . 11
2.4.6 Integrace neuronové sítě a logistické regrese k vylepšení hyper-heuristik 11
2.5 Vybraná řešení . . . 11

3 Analýza a návrh **13**
3.1 Požadavky na použitou technologii . . . 13
3.2 Popis použité technologie . . . 14
3.2.1 Framework NRP . . . 14
3.2.2 Framework NeuronDotNet . . . 15

3.2.3 Roster Booster . . . 15
3.2.4 JNNS . . . 16

4 Řešení **17**
4.1 Výchozí algoritmus . . . 17
4.1.1 Princip algoritmu . . . 17
4.1.2 Popis problému a jeho řešení. . . . 18
4.1.3 Použíté nízko-úrovňové heuristiky . . . 20
4.1.4 Přizpůsobení testovacím datům . . . 20
4.2 Modifikovaný algoritmus . . . 22
4.2.1 Problémy původního algoritmu . . . 22
4.2.2 Návrh modifikovaného algoritmu . . . 23
4.2.2.1 Principy algoritmu . . . 23
4.2.3 Použité (nízkoúrovňové) heuristiky . . . 23
4.2.3.1 Nízkoúrovňová heuristika *mH1* . . . 24
4.2.3.2 Nízkoúrovňová heuristika *mH2* . . . 25
4.2.3.3 Nízkoúrovňová heuristika *mH3* . . . 25
4.2.3.4 Nízkoúrovňová heuristika *mH4* . . . 26
4.2.3.5 Nízkoúrovňová heuristika *mH5* . . . 27
4.2.4 Řízení heuristik . . . 27
4.3 Neuronová síť . . . 27
4.3.1 Princip . . . 27
4.3.2 Využití pro účely této práce . . . 29
4.3.2.1 Klasifikace prostřednictvím neuronové sítě . . . 29
4.3.2.2 Návrh neuronové sítě . . . 30

5 Experimenty **37**
5.1 Popis testovaných instancí . . . 37
5.2 Výchozí (upravený) algoritmus . . . 38
5.3 Modifikovaný algoritmus . . . 39
5.3.1 Přípravné experimenty . . . 39
5.3.2 Hlavní experimenty . . . 41
5.4 Modifikovaný algoritmus s využitím neuronové sítě . . . 44
5.4.1 Přípravné experimenty . . . 44
5.4.2 Hlavní experimenty . . . 45
5.5 Srovnání algoritmů . . . 47
5.5.1 Srovnání všech naimplementovaných algoritmů . . . 47
5.5.2 Srovnání modifikovaného algoritmu bez a s využitím neuronové sítě . . . 49

6 Závěr **55**

A Seznam použitých zkratek **61**

B Seznam použitého matematického značení **63**

C Manuál pro programátora **65**

D Obsah přiloženého CD **69**

Seznam obrázků

3.1 Okno programu JNNS zobrazující chybu v závislosti na počtu učících cyklů (Learning cycles). 16

4.1 Framework hyper-heuristiky řízené tabu vyhledáváním 18
4.2 Příklad vygenerovaného rozvrhu s optimálním rozvržením směn. 21
4.3 Princip heuristiky *mH1*. 25
4.4 Princip heuristiky *mH2*. 25
4.5 Model neuronu (S(x) značí přenosovou funkci). 27
4.6 Struktura neuronové sítě o jedné vstupní, skryté a výstupní vrstvě (generován v programu JNNS). 28
4.7 Chyba neuronové sítě na trénovacích a testovacích datech v závislosti na době učení. 29
4.8 Ukázka minimálního a maximálního bloku směn v rozvrhu. 32
4.9 Diagram zobrazující proces generování vzorků potřebných pro učení neuronové sítě. 33

5.1 Penalizační hodnoty rozvrhu získané aplikací původního algoritmu na testovací instance. 39
5.2 Průběh nejlepších dosažených hodnot penalizace rozvrhu v čase získaných při aplikaci původního algoritmu. 40
5.3 Penalizační hodnoty získané aplikací modifikovaného algoritmu při $N_h = 4$. . . 42
5.4 Penalizační hodnoty získané aplikací modifikovaného algoritmu při $N_h = 6$. . . 43
5.5 Penalizační hodnoty získané aplikací modifikovaného algoritmu při $N_h = 8$. . . 43
5.6 Procentuální úspěšnost heuristiky *mH4* při různě zadaném koeficientu K_{mh4}. 45
5.7 Penalizační hodnoty získané aplikací modifikovaného algoritmu s využitím neuronové sítě při $N_h = 4$ 46
5.8 Penalizační hodnoty získané aplikací modifikovaného algoritmu s využitím neuronové sítě při $N_h = 6$. 46
5.9 Penalizační hodnoty získané aplikací modifikovaného algoritmu s využitím neuronové sítě při $N_h = 8$. 47
5.10 Srovnání hodnot penalizací rozvrhů získaných původním algoritmem, modifikovaným a modifikovaným s využitím neuronové sítě, část 1. 48
5.11 Srovnání hodnot penalizací rozvrhů získaných původním algoritmem, modifikovaným a modifikovaným s využitím neuronové sítě, část 2. 48

5.12 Srovnání hodnot průměrných penalizací rozvrhů vzhledem k počtu spuštěných heuristik v průběhu algoritmu. 51
5.13 Srovnání mediánů penalizací rozvrhů vzhledem k počtu spuštěných heuristik během průběhu algoritmu. 52

C.1 Struktura projektu InitialTSHH. 66
C.2 Struktura projektu ModifiedTSHH. 67

Seznam tabulek

5.1 Seznam použitých výpočetních strojů k experimentům 37
5.2 Seznam testovacích vstupních instancí 38
5.3 Doba běhu heuristiky *mH2* v závislosti na počtu vybraných zaměstnanců na instanci Valouxis. 40
5.4 Doba běhu heuristiky *mH3* v závislosti na počtu vybraných zaměstnanců a dní na instanci Valouxis 41
5.5 Průměrná penalizace rozvrhů a její medián na různých instancích a průměrný počet spuštěných heuristik pro dosažení této penalizace 44
5.6 Průměrné penalizace rozvrhů na různých instancích vygenerovaných modifikovaným algoritmem a modifikovaným algoritmem při použití neuronové sítě 50
5.7 Průměrný počet spuštěných heuristik při průběhu modifikovaného algoritmu a modifikovaného algoritmu s použitím neuronové sítě 50
5.8 Medián penalizace rozvrhů na různých instancích vygenerovaných modifikovaným algoritmem a modifikovaným algoritmem při použití neuronové sítě. . 52
5.9 Rozdíly u generovaných výsledků při různých inicializačních rozvrzích. 53

B.1 Seznam použitého matematického značení - uvedena jsou pouze taková, která se vyskytují i jinde, než jen v daném vzorci, případně textu navazujícím přímo na vzorec. 63

C.1 Popis sady projektů v programu Microsoft Visual 2012 Ultimate. 65
C.2 Popis nejpoužívanějších funkcí frameworku NRP. 65
C.3 Popis nejpoužívanějších funkcí frameworku NeuroDotNet. 66
C.4 Popis proměnných v projektu InitialTSHH. 67
C.5 Popis proměnných v projektu ModifiedTSHH. 67

Seznam pseudokódů

1 Pseudokód heuristiky *mH3* 26
2 Pseudokód modifikovaného algoritmu s využitím neuronové sítě 30

Úvod

Rozvrhování směn zaměstnanců patří mezi typické problémy větších společností, kdy zaměstnanci pracují na směnný provoz. Špatně rozvržené směny mohou mít vliv na efektivitu práce zaměstnanců, případně jejich spolehlivost. V některých situacích mohou být důsledky ještě horší. Takovým případem je mimo jiných rozvrhování směn zdravotních sester, kde by v žádném případě nemělo docházet například k přepracování některé ze sester z důvodu nedodržení odpočinkových dnů.

Existuje velké množství různých řešení tohoto problému, nicméně zatím žádné z nich se nedá nazvat řešením ideálním. Vzhledem k rozdílným požadavkům jednotlivých společností na rozvržení směn zaměstnancům je cílem této práce navrhnout a naimplementovat algoritmus, který bude schopný řešit instance s různou strukturou. Tím se předejde nutnosti zásahu programátora při změně některých z parametrů problému, který má za úkol algoritmus vyřešit.

První část této práce se zabývá popisem problému rozvrhování směn. V další kapitole nazvané *Popis práce* jsou, po obecném rozdělení metod pro řešení NP-obtížných úloh, rozebrány některé ze zajímavých možných postupů při jejich řešení s ohledem na rozvrhování směn zdravotních sester. Dále je zdůvodněn výběr jednoho řešení, ze kterého poté vychází nový algoritmus vytvořený v rámci této práce. Kapitola *Analýza a návrh* se zabývá výběrem vhodného prostředí a podpůrných programů pro vývoj algoritmů. Následuje kapitola *Řešení*, v níž je rozebrán algoritmus vybraný v kapitole *Popis práce* a navržen nový algoritmus, který by měl eliminovat nedostatky původního. Závěrečná část patří experimentům, vyhodnocení a diskuzi k výsledkům naimplementovaných algoritmů.

Kapitola 1

Popis práce

Problém rozvrhování směn patří z hlediska obtížnosti mezi NP-obtížné problémy, proto u většiny vstupních instancí není (v současné době) možné nalézt optimální řešení v rozumném čase [10]. Doba běhu programu se dá považovat za rozumnou v případě, že po jejím uplynutí je řešený problém stále aktuální a doba běhu splňuje požadavky uživatele. Získání řešení, které se alespoň blíží ideálu, v přijatelném čase je ovšem pro některé organizace klíčové. Při nedodržení podmínek zadaných pro přiřazení směn zaměstnancům může dojít například k nechtěnému zvýhodňování některých zaměstnanců na úkor ostatních, nedodržení požadovaného počtu zaměstnanců v daný okamžik či v nejhorším případě nedodržování pracovních podmínek definovaných platným zákonem dané země.

1.1 Cíle práce

Jak již bylo řečeno v úvodu práce, sice existuje velké množství způsobů řešení problému rozvrhování směn zaměstnanců, ale žádný z nich nelze označit za způsob vhodný pro všechny situace. Každý má svá pozitiva i negativa. Při návrhu algoritmu inspirovaného jednou, případně i několika takovými metodami, obvykle směřujeme (mimo jiné) ke dvěma možným základním cílům [6]:

- **Minimalizace penalizace konkrétního typu rozvrhu** - snaha o získání co nejnižší penalizace rozvrhu i za předpokladu, že na rozvrzích s jinými požadavky může být výsledek velmi slabý, či dokonce nebude možné daný algoritmus na jiné případy vůbec použít.
- **Minimalizace penalizace u většího množství typů rozvrhu** - snaha o získání rozvrhu s akceptovatelnou penalizací u různých typů rozvrhů. Není tedy potřeba provádět složité ladění parametrů, případně doprogramovat další moduly pro různé typy rozvrhů. Samozřejmě rozvrh získaný takovými algoritmy má zpravidla vyšší penalizaci než rozvrh získaný za použití algoritmů upravených přímo na míru konkrétnímu rozvrhu.

Tato práce se bude zabývat řešením problému dle druhého bodu, tedy minimalizací penalizace u většího množství typů rozvrhu.

1.2 Popis problému

Problém rozvrhování se zabývá řešením přiřazení úloh ke zdrojům v čase takovým způsobem, aby došlo k co možná největší shodě se zadanými požadavky [28]. Aktivita (úloha) probíhající v časovém úseku, který má definovaný začátek, konec a délku trvání, se typicky označuje jako směna. Při rozvrhování směn zdravotních sester se poté snažíme přiřadit směny zdravotním sestrám a zároveň eleminovat případné konflikty se zadanými omezeními. Cílem je minimalizovat vážený součet porušených měkkých omezení:

$$min \sum_{i=1}^{n} v_i o_i, \tag{1.1}$$

kde n je celkový počet omezení, v_i značí váhu omezení a o_i udává, kolikrát bylo dané omezení v rozvrhu porušeno. Zároveň rozvrh nesmí porušovat žádné z tvrdých omezení.

Omezení se mohou velmi lišit dle konkrétní situace. Rozdělují se na omezení měkká a tvrdá. Suma vah porušených omezení se obvykle nazývá *penalizací* [31].

Pro ohodnocení konkrétního rozvrhu (rozvržení směn) se používá *evaluační (hodnoticí)* funkce, která většinou vychází z hodnoty penalizace.

1.2.1 Měkká omezení

Při porušení měkkých omezení dochází ke zvýšení penalizační hodnoty a tím pádem i snížení kvality rozvrhu, ale stále se považuje za akceptovatelný. Každému měkkému porušení obvykle přiřazujeme váhu, která udává, jak velký vliv má její porušení na kvalitu rozvrhu. Ve většině případů se bohužel nevyhneme tomu, abychom neporušili žádné omezení. V opačném případě můžeme rozvrh směn prohlásit za optimální (za předpokladu současného neporušení žádného z tvrdých omezení).

Typickými příklady měkkých omezení jsou:

- **Preferovaný počet přiřazených směn zdravotní sestře za časový úsek** - obvykle vychází z pracovního úvazku zaměstnance.
- **Maximální a minimální počet pracovních dnů v řadě.**
- **Maximální počet pracovních víkendů za rozvrhovací období.**
- **Plně pracovní víkend** - požadavek, aby daná zdravotní sestra buď pracovala celý víkend, nebo žádný ze dnů víkendu.
- **Volná odpolední páteční směna před volným víkendem** - pravidlo, aby sestra, která nemá o víkendu žádnou směnu, neměla před tímto víkendem ani páteční odpolední (případně noční) směnu [9].

Všechna tato porušení podmínek snižují sice kvalitu rozvrhu, ale i přesto lze nazvat rozvrh akceptovatelným. Samozřejmě může být definováno daleko více omezení anebo naopak některá, z předchozího seznamu, nemusí být uvažována. Jak již bylo řečeno, měkké omezení bývá doplněno o váhu určující velikost vlivu porušení tohoto omezení na kvalitu výsledného rozvrhu. I přesto, že porušením měkkého omezení nedochází automaticky k označení řešení za neakceptovatelné, může být vhodnost rozvrhu omezena maximální hodnotou penalizace.

1.2.2 Tvrdá omezení

Porušením tvrdých omezení se stává rozvrh neakceptovatelným, nesmí k nim tedy v žádném případě dojít. Obvykle jsou mimo jiné spojena s pravidly ze Zákoníku práce. Příkladem tvrdých omezení může být:

- minimální počet zaměstnanců, kteří musí být přiřazeni k dané směně;
- maximální počet pracovních dní, které může zaměstnanec absolvovat;
- zákaz přiřazení zaměstnanci více směn za sebou [1].

Kapitola 2

Teoretický úvod

2.1 Základní typy metod pro řešení NP úloh

2.1.1 Úplné metody

Úplné metody se snaží najít optimální výsledek procházením celého prostoru možných řešení. Příkladem takových metod je celočíselné lineární programování (ILP) nebo programování s omezujícími podmínkami. Stavový prostor řešení je nicméně obvykle příliš velký a v rozumném čase jej lze projít pouze u velice omezených úloh. Z tohoto důvodu se využívají především jako doplněk k dalším metodám, případně k získání suboptimálního řešení [31].

Některé nástroje (např. IBM CPLEX) využívající úplné metody proto umožňují zastavení průběhu algoritmu a vypsání dosavadního nejlepšího výsledku. Aplikací metody programování s omezujícími podmínkami na problém rozvrhování směn se zabývá mimo jiné práce [15].

2.1.2 Heuristiky

Heuristiky jsou algoritmy, které neprocházejí celý stavový prostor řešení, ale pouze jeho část, a zde se pokouší najít akceptovatelný výsledek, který se co nejvíce blíží optimu. Heuristiky mohou být buď deterministické, či stochastické. Deterministické heuristiky mají vždy stejný průchod za předpokladu stejných vstupů. To znamená, že pokaždé dostaneme stejný výsledek. Naopak heuristiky stochastické využívají náhodných jevů a tím se snaží zabránit lokálnímu uváznutí a prohledat tak větší stavový prostor. Obvykle zde není předem jasná doba výpočtu algoritmu [19].

2.1.3 Hyperheuristiky

Hyperheuristiky pracují podobně jako heuristiky, ale neřídí přímo jednotlivé kroky, pouze určují, které heuristiky budou v daném kroku spuštěny. Metoda spouštění jednotlivých kroků může být velice podobná jako u způsobu provádění standardních heuristik. V praxi se však spíše místo heuristik využívají tzv. nízkoúrovňové heuristiky, neboli velmi zjednodušené algoritmy. Typickým příkladem takových nízkoúrovňových heuristik používaných při rozvrhování je záměna (swap) určitých směn mezi dvěma zaměstnanci [5, 29].

2.2 Heuristické algoritmy

2.2.1 Horolezecký algoritmus (Hill Climbing)

Horolezecký algoritmus pracuje na bázi lokálního prohledávání. V každém kroku prohledává své okolí, vybere z něj nejlepší řešení, a pokud je lepší než aktuální, přijme jej. Tento krok opakuje až do chvíle, kdy žádné lepší řešení nenalezne. Horolezecký algoritmus velmi trpí lokálním uváznutím, což se z části řeší restartováním rozvrhu, tzn. několikanásobnou aplikací algoritmu z různých míst stavového prostoru [23]. I přesto výsledky tohoto algoritmu nebývají příliš přesvědčivé [31].

2.2.2 Simulované žíhání (Simulated Annealing)

I algoritmus simulovaného žíhání pracuje na bázi lokálního prohledávání. Princip algoritmu je inspirován procesem žíhání oceli. Algoritmus náhodně vybírá rozvrhy ze svého okolí a v případě, že má vybraný rozvrh nižší penalizační hodnotu, přijme jej. S určitou pravděpodobností může být rozvrh vybrán i v případě, že nižší penalizační hodnotu nemá. Pravděpodobnost přijmutí závisí na aktuální teplotě. Na začátku je teplota nastavena na vyšší hodnotu a postupem času klesá (ekvivalent chladnutí oceli). Tím pádem na začátku algoritmu přichází v úvahu přijmutí i poměrně špatného rozvrhu a postupně se tato pravděpodobnost snižuje, čímž se částečně řeší uváznutí v lokálním extrému [25].

2.2.3 Genetický algoritmus

Algoritmus pracující na principu inspirovaném Darwinovou evoluční teorií. Její ústřední myšlenkou je snaha o zkombinování dobrých jedinců (jednotlivých řešení) z populace (množiny možných řešení) tak, aby vznikl lepší jedinec (řešení). Standardně je využíváno především tří operátorů:

- **Selekce** - výběr jedinců dle jejich kvality, čím lepší jedinci jsou, tím je jejich pravděpodobnost výběru vyšší.
- **Křížení** - kombinace vlastností (v našem případě většinou částí rozvrhů) dvou (či více) jedinců, vzniká nový jedinec.
- **Mutace** - modifikace jednoho jedince, zabraňuje uváznutí v lokálním extrému [26, 27].

2.2.4 Tabu vyhledávání (Tabu Search)

V každém kroku algoritmu tabu vyhledávání se v okolí hledá nejvhodnější řešení, které je poté označeno jako základní řešení pro následující krok. Implementována navíc bývá ještě krátkodobá paměť, případně i dlouhodobá, a můžeme zavést i tzv. aspirační kritérium. V krátkodobé paměti (tabu listu) uchováváme kroky provedené v několika posledních iteracích, které je zakázáno do určité doby zopakovat. Tím zamezujeme zacyklení algoritmu. V dlouhodobé paměti algoritmus ukládá některé informace typu počtu použití určitého kroku, čímž ovlivňuje chod programu. Aspirační kritérium poté umožňuje za určitých podmínek provést krok zakázaný pomocí krátkodobé paměti [31].

2.2.5 Zpětnovazební učení (Reinforcement Learning)

Metoda, při které se využívá znalostí získaných průchodem stavového prostoru řešení. Často se zde setkáváme s následujícími pojmy:

- **Strategie (policy)** - pravidla určující, jaký krok se má provést za daných podmínek.
- **Zpětnovazební funkce (reward function)** - vykazuje informace o tom, zda daný krok vede momentálně k lepšímu řešení (v našem případě zda krok vede k rozvrhu s nižší penalizací).
- **Hodnoticí či evaluační funkce (value function)** [1] - oproti zpětnovazební funkci není výstupem ohodnocení okamžitého zlepšení stavu rozvrhu, nýbrž vlivu na dlouhodobý výsledek.

Primitivní implementace zpětnovazebního učení spočívá v průchodu prostorem řešení, definovaném strategií výběru následujícího kroku (ta obvykle bývá implementována jako naučená neuronová síť). Po každém kroku se ovšem vyhodnocuje, zdali zvolený postup vedl ke zlepšení či nikoliv, a dle výsledku se upravuje strategie. Teoreticky samozřejmě požadujeme, aby kroky vedly k co nejlepšímu dlouhodobému výsledku, nicméně vzhledem k obtížnosti správného určení vlivu aktuální změny na dlouhodobou hodnotu se v praxi obvykle setkáváme s řízením strategie dle okamžité zpětné vazby [33, 3, 30].

2.3 Hyper-heuristiky

Jak již bylo v této práci zmíněno, za hyper-heuristiky označujeme takové algoritmy, u nichž dle nějaké metody řídíme spouštění jiných nízkoúrovňových heuristik. Těmito nízkoúrovňovými heuristikami mohou být základní operace nebo i sofistikovanější metody. Při řízení nízkoúrovňových heuristik je přitom možné využít některé z klasických heuristik, samozřejmě patřičně modifikovaných. Mezi klasické operace při rozvrhování směn, které bývají použity jako nízkoúrovňové heuristiky, patří:

- **Drop (odebrat)** - odebrání směny dle určitých požadavků.
- **Add (přidat)** - přidání směny dle určitých požadavků.
- **Swap (záměna)** - záměna směn dle určitých požadavků.

Samozřejmě není nutností, aby byly tyto operace v hyper-heuristikách implementovány [32].

[1] V tomto případě se rozlišuje okamžitá zpětnovazební funkce a dlouhodobá hodnoticí (evaluační funkce), nicméně dále v práci se bude pojem hodnoticí (evaluační) funkce používat ve smyslu, který popisuje sekce 1.2

2.4 Pokročilé metody

2.4.1 Základní charakteristiky

V praxi se nesetkáváme pouze se základními metodami používanými při řešení rozvrhování směn. Existuje nepřeberné množství řešení, která využívají kombinace těchto metod, případně jsou vylepšována dalšími metodami. Následující část práce se zabývá několika již publikovanými řešeními, která by mohla být inspirací pro vývoj nového algoritmu k efektivnímu prohledávání stavového prostoru v rozvrhování směn zdravotních sester.

2.4.2 Zpětnovazební učení s využitím agentů

Primitivní implementace zpětnovazebního učení trpí častým lokálním uváznutím. Tento jev můžeme eliminovat různými způsoby. V článku [22] řeší autor možné lokální uváznutí použitím několika agentů. Každý agent prochází jinou část prostoru řešení a je řízen vlastní strategií získanou zpětnovazebním učením. Již samotné procházení více míst prostoru řešení samozřejmě směřuje k lepším výsledkům. Klíčovým faktorem pro dosažení lepších výsledků ovšem zůstává kombinace strategií jednotlivých agentů. Záměna strategií agentů znamená změnu jejich chování, což snižuje pravděpodobnost lokálního uváznutí.

2.4.3 Zpětnovazební učení s modifikovaným výběrem následujícího kroku

Další možnost, jak eliminovat rizika lokálního uváznutí a zrychlit průchod algoritmu, a tím rychleji dosáhnout požadovaného výsledku, popisuje článek [33]. Nezabývá se sice přímo rozvrhováním směn, ale podobným problémem - přiřazováním úloh ke zdrojům.

Vybírání sousedů probíhá pomocí dlouhodobého odhadu získaného neuronovou sítí a okamžitou zpětnovazební hodnotou. Konkrétní stav se ohodnocuje pomocí zvláštního mechanismu RDF (Resource Dilation Factor), který je jakýmsi poměrem mezi aktuálním a ideálním řešením, se zohledněním délky průběhu algoritmu. Dále práce elegantně řeší problém neuronové sítě: počet vstupů a výstupů nelze bez opětovného přeučení sítě měnit. Rozvrh je proto popsán několika údaji, jako jsou například:

- počet volných jednotek zdroje;
- časové rozdíly mezi konci a začátky takových úloh, které mají mezi sebou prerekvizitu (podmínku nutnosti ukončení jedné úlohy před začátkem druhé);
- RDF;
- počet jednotek, které jsou přetíženy.

Tím je docílen stabilní počet prvků popisujících daný stav i v případě změny některých parametrů vstupní instance (např. počet zdrojů).

Výstupem neuronové sítě je osm sigmoidálních prvků, které využívají Gaussovy křivky k eliminaci chyby při predikci dlouhodobého RDF.

Tato metoda sice není vhodná pro naše účely, nicméně některé prvky (např. popis rozvrhu pomocí vybraných vlastností) lze využít i pro účely rozvrhování směn.

2.4.4 Hyper-heuristika řízená tabu prohledáváním

Článek [6] popisuje algoritmus, kdy jsou nízkoúrovňové heuristiky řízeny modifikovaným tabu vyhledáváním. Pro rozvrhování směn využívá devíti nízkoúrovňových heuristik, které se pro dosažení vhodného výsledku mimo jiné snaží o vyvážení rozvrhu. Vyváženost v tomto případě znamená, že během dané periody (a každého typu směny - noční i denní) pracuje v průměru dostatek zaměstnanců s potřebnou kvalifikací. V opačném případě se totiž dané řešení nenachází v blízkosti optima.

Autoři článku také poukazují na experimenty, které dokazují vyšší obecnost daného řešení v porovnání s genetickým algoritmem pracujícím na podobné bázi.

2.4.5 Využití vytěžování dat pro ladění koeficientů adaptivního tabu vyhledávání

Práce [14] pojednává o využití vytěžování dat pro účely ladění koeficientů metody tabu vyhledávání, čímž se odstraňuje manuální a často i zdlouhavé přizpůsobování algoritmu konkrétní situaci. Koeficienty mimo jiné v tomto případě řídí evaluační funkci, čímž nepřímo ovlivňují strategii procházení stavového prostoru řešení. Na dané instanci se algoritmus tabu vyhledávání spouští mnohokrát, pokaždé s jinými koeficienty a počátečním stavem, a uloží se výstupní hodnoty. Z těchto dat se poté pomocí Apriori algoritmu vyhodnotí, jak mají být koeficienty nastavené, aby docházelo k co nejlepším výsledkům.

2.4.6 Integrace neuronové sítě a logistické regrese k vylepšení hyper-heuristik

Další způsob možného řízení hyper-heuristik nastiňuje studie [21]. Porovnává přitom využití neuronové sítě a logistické regrese pro určování sekvence použitých nízkoúrovňových heuristik. Algoritmus nejdříve vygeneruje různé sekvence nízkoúrovňových heuristik a poté pomocí již zmiňovaných technologií odhadne, zda daná sekvence povede k získání dobrých výsledků či nikoliv. Bohužel článek uvádí experimenty zabývající se pouze úspěšností zmiňovaných klasifikátorů (neuronové sítě a logistické regrese), ale již nepopisuje reálný dopad na získaná výstupní řešení.

2.5 Vybraná řešení

Tato práce se má zabývat metodami, které umožní řešit problém rozvrhování zdravotních sester na různých instancích a na nichž budou dosahovat akceptovatelných výsledků. Tento požadavek v podstatě splňují tři možné kategorie způsobů řešení:

- využití umělé inteligence pro řízení kroků;
- metody na bázi hyper-heuristiky;
- automatizace ladění koeficientů na míru.

Příkladem třetího bodu je využití vytěžování dat pro ladění koeficientů tabu vyhledávání (viz odstavec 2.4.5). Pro správné zvolení koeficientů nicméně toto řešení vyžaduje velké množství spouštění daného algoritmu. Pokud jsou ovšem již známé výsledky experimentů, ladění koeficientů může probíhat přímo z nich. Vytěžování dat poté funguje pouze jako technologie usnadňující (případně mírně zlepšující) výběr správných hodnot. Lze tedy pochybovat o větším úspěchu při jejím uplatnění v této práci.

Za zajímavější alternativu se dá považovat metoda dle druhého bodu (metody na bázi hyper-heuristiky). Ze zmiňovaných článků vyplývá dostatečná obecnost hyper-heuristik pro jejich nasazení na různé instance daného problému. Poměrně často se pro řízení nízkoúrovňových heuristik používá metoda tabu vyhledávání. Výchozí algoritmus pro rozvrhování směn zdravotních sester proto bude vycházet z techniky, kterou popisuje sekce 2.4.4. Případné nedostatky této technologie budou klíčové pro návrh modifikované verze. Experimenty na závěr prověří schopnosti obou těchto řešení.

Algoritmy využívající umělé inteligence pro řízení kroků začínají taktéž nacházet své uplatnění jak v algoritmech specializovaných na konkrétní problém, tak i u obecných heuristik. Dle experimentů jiných řešitelů NP-úplných problémů pomocí umělé inteligence je možné zrychlit či jinak zdokonalit již fungující algoritmus, případně pomocí ní lze úplně nahradit jiné metody určující směr prohledávání stavového prostoru řešení [33]. Velmi často se jako prvek umělé inteligence aplikuje neuronová síť. V poslední etapě proto tato práce prověří možnost použití neuronové sítě pro řízení nízkoúrovňových heuristik namísto metody tabu vyhledávání. Algoritmus bude vycházet z již zmiňované modifikované verze. Experimenty opět ukážou, zda má smysl inteligentně řídit metodu zaměřenou na řešení obecných instancí problému rozvrhování směn zdravotních sester.

Kapitola 3

Analýza a návrh

3.1 Požadavky na použitou technologii

Vzhledem k tomu, že účelem práce není vytvoření uživatelsky přívětivého softwaru aplikovaného v reálném prostředí, není třeba se zabývat některými aspekty, jako je např. grafické uživatelské rozhraní. Požadavky na technologii pro vývoj programu jsou rozebrány v následujících bodech:

- **Aplikovatelnost na různé typy instancí** - práce má za cíl najít algoritmus, který umožní řešit různé typy úloh. Proto je třeba, aby technologie dovolovala řešit různé instance s různými omezeními.
- **Dostatečná rychlost řešení daných úloh** - největší vliv na rychlost má samozřejmě zvolená metoda řešení, případně nastavení parametru dané metody. Není proto nutné, aby daná technologie využívala např. pouze nízkoúrovňových jazyků a ušetřila tím poměrnou část času na úkor doby implementace, nicméně je důležité, aby nevyužívala metody, které by způsobovaly značné zpomalení při větších vstupních instancích.

Z pohledu druhého bodu se jako ideální řešení nabízí použití vyššího programovacího jazyka (např. C++, C#, Java) a naimplementování programu, který dokáže reagovat na všechny možné vstupy. Takových vstupů ovšem existuje velké množství a naimplementovat všechny možnosti by zabralo velké množství času. Proto byl vybrán již existující framework vytvořený pro ulehčení řešení problému zdravotních sester (v této prácí bude dále nazýván jako framework NRP z anglického *Nurse Rostering Problem*), implementovaný pomocí jazyka C# jako projekt mající kořeny na univerzitě v Nottinghamu. Autoři poskytují pro výzkumné účely framework zdarma [9].

Pro aplikaci neuronové sítě bude využit open-source framework, opět postavený na programovacím jazyce C# - *NeuronDotNet*, který již sice není delší dobu aktualizován (poslední update proběhl 22. 8. 2008), nicméně pro účely této práce disponuje dostatečnou funkcionalitou [11].

Dále bude využit program *Roster Booster* pro analyzování rozvrhů a *JNNS*, který umožňuje vytvoření modelu neuronové sítě a odsimulování jejího chování [9, 18].

3.2 Popis použité technologie

3.2.1 Framework NRP

Framework NRP dovoluje pracovat s rozvrhy, u kterých je možné popisovat pomocí XML souboru jejich vlastnosti. Velké množství těchto údajů se týká rozvrhovacího období (tučně zobrazen název tagu):

- **StartDate** - první den rozvrhovacího období.
- **EndDate** - poslední den rozvrhovacího období.
- **Skills** - popis možných dovedností zaměstnanců.
- **SkillGroups** - možnost rozdělení dovedností do skupin.
- **Contracts** - typy úvazku následně přiřazené jednotlivým zaměstnancům.
- **Employees** - zaměstnanci, kterým mají být rozvrhnuty směny.
- **CoverRequirements** - minimální, maximální a preferovaný počet pracujících zaměstnanců během dané směny.
- **CoverWeights** - definice hodnoty, která se přičte k penalizaci, pokud nebude pokryta určitá směna.
- **DayOffRequests** - seznam požadavků zaměstnanců na volný den.
- **DayOnRequests** - seznam požadavků zaměstnanců na pracovní den.
- **ShiftOffRequests** - směny, kdy zaměstnanci nechtějí pracovat.
- **ShiftOnRequests** - směny, kdy zaměstnanci chtějí pracovat.
- **FixedAssignments** - fixní přiřazení směny nebo naopak volná směna.

Mimo toho je možné definovat vzory sekvencí směn a jejich maximální či minimální počet výskytů v rozvrhu. Tyto vzory framework dovoluje popisovat pomocí tzv. vzorů (ang. patterns).

Na bázi těchto informací poskytuje framework (skrze API) funkci vracející penalizační hodnotu aktuálního rozvrhu, kterou využíváme pro ohodnocení kvality rozvrhu. Snahou je získat rozvrh, kterému tato funkce vrací co nejnižší (penalizační) hodnotu. Na internetu jsou navíc k dispozici rovnou XML soubory odrážející pracovní podmínky různých společností i se záznamy nejlepších dosažených výsledků, což usnadňuje ohodnocení algoritmem dosaženého výsledku [9]. Vzhledem k robustnosti frameworku dochází i k některým nechtěným vlastnostem. Co se týká funkcionality, poskytuje toto řešení dostatečnou škálu možností, nicméně některé funkce frameworku představují poměrně složité algoritmy, které si vybírají svou daň v podobě delší doby zpracování. Jejich modifikace ovšem není vždy jednoduchá, v některých případech je téměř nemožná.

Největší problém představuje funkce *Clone()*, která je součástí třídy *Roster*. Jak již název funkce napovídá, slouží pro duplikaci konkrétního rozvrhu (obsahující jeho vlastnosti, již rozvržené směny atd.). Spolu s překopírováním směn a dalších parametrů volá však navíc časově náročnou funkci *recalculateAllPenalties()*, která počítá penalizaci rozvrhu. Celkově funkce *Clone()* pracuje, vzhledem ke vstupu *počet_ zaměstnanců * počet_ směn_ za_ periodu*, s lineární složitostí, netrvá tedy neúměrně dlouhou dobu, ale kvůli jejímu častému používání patří k největším spotřebičům výpočetního výkonu. Nicméně díky množství pozitiv tohoto frameworku jej lze i přes výše uvedená fakta považovat za dobře použitelný pro řešení problému, kterým se zabývá tato práce [9].

3.2.2 Framework NeuronDotNet

Framework *NeuronDotNet* bohužel již nemá přístupné stránky projektu, proto k němu není dostupná reference (pokud nějaká existovala), avšak ta díky dobře navržené struktuře a komentovanému kódu v podstatě ani není nutná. Framework poskytuje rozhraní (mimo jiné) pro:

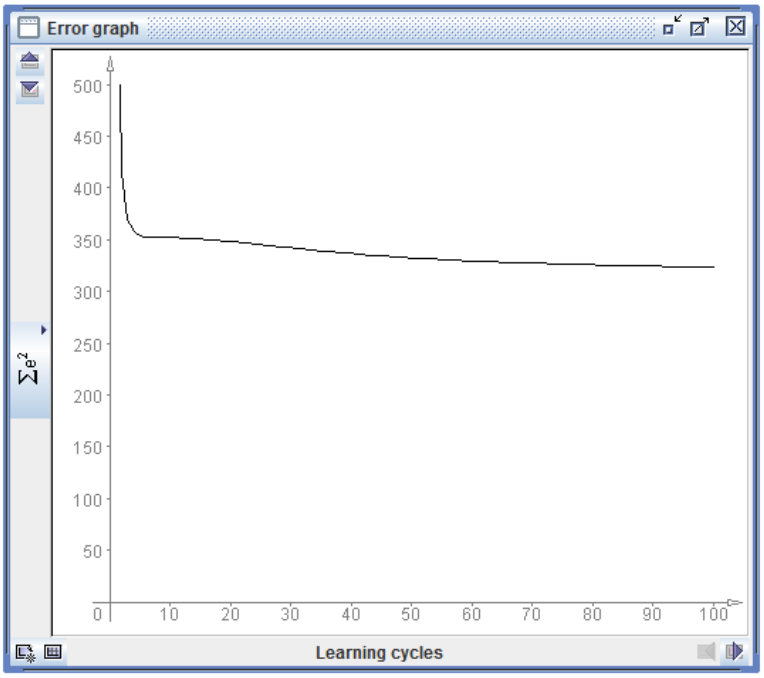
- **Definování struktury neuronové sítě** - počet vrstev, počet neuronů v dané vrstvě, přenosová funkce neuronu apod.
- **Naučení vytvořené neuronové sítě pomocí množiny vzorků** - s možností nastavení rychlosti učení, počtu epoch (tzn. kolikrát projde celou množinu vzorků) apod.
- **Použití naučené neuronové sítě pro odhad výstupních hodnot vzhledem k jejím vstupním hodnotám (tj. klasifikace rozvrhu)**.

Tato funkcionalita by měla být dostatečná pro účely této práce [11].

3.2.3 Roster Booster

Roster Booster je program vyvíjený univerzitou v Nottingamu, umožňující především:

- načtení rozvrhu z XML souboru;
- zobrazení daného rozvrhu;
- informování o struktuře rozvrhu, zadaných omezení, počtu zaměstnanců, atd.;
- výpočet aktuální penalizace a zobrazení seznamu omezení i s výskyty jejich porušení;
- aplikování algoritmu VDS (z anglického Variable Depth Search) pro snížení penalizace rozvrhu [9].



Obrázek 3.1: Okno programu JNNS zobrazující chybu v závislosti na počtu učících cyklů (Learning cycles).

3.2.4 JNNS

Aplikace JNNS (Java Neural Network Simulator), jak již název napovídá, je simulátorem neuronových sítí. Umožňuje navrhnout si vlastní složení neuronové sítě, definovat počet vrstev, přenosové funkce neuronů v jednotlivých vrstvách atd. [18].

Po předložení souboru s testovacími daty (vyžaduje speciální strukturu souboru) lze pomocí tohoto programu například zjišťovat chybovost neuronové sítě v závislosti na délce její učící fáze (viz 3.1).

Kapitola 4

Řešení

4.1 Výchozí algoritmus

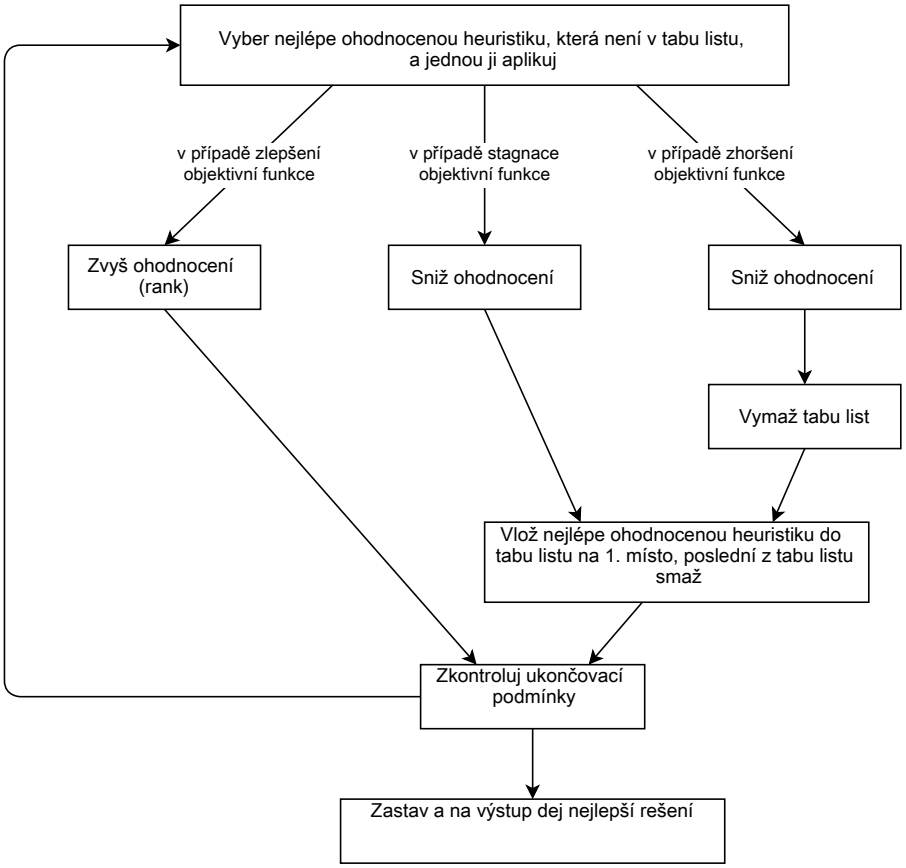
4.1.1 Princip algoritmu

Jak již bylo řečeno, pro inspiraci byl použit algoritmus popsaný v článku [6]. Jedná se v podstatě o algoritmus tabu vyhledávání, který místo řízení konkrétních kroků pouze vybírá, jakou nízkoúrovňovou heuristiku použije v aktuálním stavu. Výběr probíhá na základě předchozí úspěšnosti heuristiky a stavu tabu seznamu.

Na začátku je každá nízkoúrovňová heuristika (pozn.: dále budou nízkoúrovňové heuristiky nazývány pro zkrácení pouze heuristikami) ohodnocena nulou. Po aplikování konkrétní heuristiky se její ohodnocení (proměnná rank) buď sníží, nebo zvýší, vždy se ovšem pohybuje v intervalu $[rn_{min}, rn_{max}]$, kde $rn_{min} = 0$ a $rn_{max} = m$, kde m značí počet heuristik řízených právě algoritmem tabu vyhledávání. Ohodnocení se zvýší o α, pokud po aplikaci dané heuristiky vrátí evaluační funkce u nově vzniklého rozvrhu nižší číslo než u rozvrhu ze stavu před aplikací této heuristiky, v opačném případě dojde k odečtení hodnoty α. Vzhledem k výsledkům ze článku [4] je vhodné použít nastavení α=1.

Tabu seznam na počátku neobsahuje žádné záznamy. Pokud aplikací vybrané heuristiky nedojde ke snížení hodnoty evaluační funkce u daného rozvrhu, vloží se vybraná heuristika do tabu seznamu (fronty) a v případě, že je překročena maximální povolená velikost fronty, nejstarší prvek se z ní odstraní. Vzhledem k počtu použitých heuristik vychází nejlépe nastavení limitu velikosti fronty na hodnotu 4 [4]. Jestliže se evaluační hodnota nejen nesníží, ale naopak zvýší, dojde (před vložením dané heuristiky do tabu seznamu) k vymazání všech záznamů z tabu seznamu. Není totiž důvod omezovat spuštění heuristik v případě, že došlo ke změně rozvrhu s negativním výsledkem [6].

V každé iteraci metoda tabu vyhledávání zvolí ke spuštění heuristiku, která není obsažena v tabu seznamu a má nejvyšší rank. Po vykonání heuristiky se upraví její ohodnocení dle dosaženého úspěchu, případně se změní tabu seznam dle postupu popsaného v předchozích dvou odstavcích. Celý algoritmus končí, pokud dojde ke splnění ukončovací podmínky. Ukončovací podmínkou je zvoleno dosažení 8000 cyklů. Na výstup se načte nejlepší dosažené řešení (řešení s nejnižší penalizací) [6]. Celý proces popisuje diagram 4.1.



Obrázek 4.1: Framework hyper-heuristiky řízené tabu vyhledáváním

4.1.2 Popis problému a jeho řešení.

I přesto, že se v článku [6] mluví o pokroku v oblasti aplikovatelnosti algoritmu na instance s různými parametry, navrhované řešení je poměrně úzce spojeno s konkrétním problémem - rozvrhováním směn sester v jedné z nemocnic ve Velké Británii. V této nemocnici nastává problém s rozvrhováním směn jednotlivých oddělení až s 30 sestrami. Zdravotní sestry mohou mít týden, během kterého pracují buď na nočních, nebo denních směnách (není možné, aby v jednom týdnu měly jak noční, tak denní směny). Denní směny se dále rozdělují na dopolední a odpolední. Zdravotní sestry navíc mohou pracovat na poloviční, nebo plný úvazek. Z těchto specifikací a dalších omezení jsou vygenerovány realizovatelné vzory směn (o délce jednoho týdne) pro denní i noční typ směn v celkovém množství 411 vzorů. Cílem výsledného řešení je minimalizovat penalizační funkci:

$$\sum_{i=1}^{n}\sum_{j=1}^{s} p_{ij}x_{ij} \tag{4.1}$$

při zachování (tvrdých) podmínek:

$$\sum_{j \in F(i)} x_{ij} = 1 \tag{4.2}$$

$$\sum_{j=1}^{s} \sum_{i=1}^{n} b_{ir} a_{jk} x_{ij} \geq S_{kr}, \forall k, r \tag{4.3}$$

V těchto vzorcích parametry značí:

- n - počet zdravotních sester;
- s - počet možných vzorů směn;
- p_{ij} - penalizační hodnota zdravotní sestry i pracující dle vzoru směn j;
- x_{ij} - rozhodovací proměnná, která má hodnotu 1 za předpokladu, že zdravotní sestra i pracuje dle vzoru směn j, a hodnotu 0 v ostatních případech;
- b_{ir} - nabývá hodnoty 1, jestliže sestra i dosahuje úrovně r nebo vyšší, jinak 0;
- a_{jk} - pokud vzor směn j pokrývá směnu k, nabývá hodnoty 1, jinak 0;
- S_{kr} - požadavek na počet sester s úrovní r nebo vyšší přiřazených ke směně k;
- F(i) - množina možných vzorů směn pro zdravotní sestru i.

Vzorec 4.2. popisuje omezení, kdy každá sestra pracuje pouze dle jednoho vzoru směn, vzorec 4.3. ve zkratce znamená, že musí být splněny minimální požadavky na obsazení směny (počet sester dané úrovně či vyšší).

Mimo to je brán ohled na několik desítek měkkých omezení. Úspěšnost té které heuristiky ovšem algoritmus neposuzuje pouze pomocí součtu vah porušených měkkých omezení. Evaluační funkce využívá také hodnoty *InFeas* definované:

$$InFeas = \sum_{r=1}^{g} (\rho \times Bal_r + 1) \sum_{k=1}^{14} max((S_{kr} - \sum_{j=1}^{s} \sum_{i=1}^{n} b_{ir} a_{jk} x_{ij}), 0), \tag{4.4}$$

kde Bal_r je 2, pokud oba typy směn (denní i noční) jsou nevyvážené při požadované úrovni r, 1 - v případě nevyváženosti buď nočního, nebo denního typu směn, jinak 0. Nevyvážeností se přitom myslí stav, kdy celkový počet přiřazených směn sestře dané úrovně (nebo vyšší) za sledovaný úsek (týden) je nižší než požadovaný. Ve vzorci také figuruje parametr ρ, pomocí kterého lze nastavit vliv vybalancování rozvrhu na jeho ohodnocení. Jeho hodnota byla dle článku [8] nastavena na $\rho = 5$. *InFeas* je použitý v evaluační (objektivní) funkci definované:

$$E = PC_e + C_{demand} InFeas, \tag{4.5}$$

přičemž PC_e označuje penalizaci zaměstnanců a koeficient C_{demand} nabývá hodnoty 5, pokud $q = 0$ (viz. vzorec 4.6). Jestliže je ale $q > 0$, poté $C_{demand} = \gamma \times q$.

$$q = \sum_{r=1}^{g} \sum_{k=1}^{14} max((S_{kr} - \sum_{j=1}^{s} \sum_{i=1}^{n} b_{ir} a_{jk} x_{ij}), 0) \tag{4.6}$$

4.1.3 Použíté nízko-úrovňové heuristiky

Použité heuristiky pracují převážně na principu vyváženosti rozvrhu, která již byla v této práci vysvětlena. První heuristika (*H1*) umožňuje uniknutí z lokálního extrému, jelikož funguje na bázi náhodnosti a výběr sousedního řešení je redukován pouze omezeným výběrem vzorů směn. *H2-H5* fungují podobně jako *H1* s ohledem na nezhoršování buď jedné, nebo obou z klíčových hodnot - *InFeas a penalizace zaměstnanců*. *H6-H7* mají za úkol vyřešit problémový stav, kdy dochází k nedostatečnému pokrytí nočních, nebo denních směn (či obou naráz). Heuristiky *H8-H9* pracují na principu přesunutí vzoru směn jiné sestře. Dohromady je pro řešení problému použito tedy 9 heuristik:

- **H1:** Změň vzor směn náhodně vybrané sestry.
- **H2:** Stejná jako *H1*, ale akceptační kriterium je „první zlepšující *InFeas*“.
- **H3:** Stejná jako *H1*, ale akceptační kriterium je „první zlepšující *InFeas* a nezhoršující hodnotu penalizace zaměstnanců“.
- **H4:** Stejná jako *H1*, ale akceptační kriterium je „první zlepšující hodnotu penalizace zaměstnanců“.
- **H5:** Stejná jako *H1*, ale akceptační kriterium je „první zlepšující hodnotu penalizace zaměstnanců a nezhoršující *InFeas*“.
- **H6:** Změna typu vzoru směn náhodně vybrané sestry v případě nevybalancovanosti rozvrhu.
- **H7:** Stejná jako *H6*, ale pouze v případě, že některý z typů směn není vybalancovaný. Pokud jsou oba typy nevybalancované, prohodí se směny mezi sestrami, které mají v daný den rozdílné typy směn.
- **H8:** Nalezení prvního přesunu takového, kdy změníme vzor směn některé sestry a původní vzor přiřadíme sestře jiné.
- **H9:** Stejná jako *H8*, nesmí ovšem dojít ke zhoršení *InFeas*.

4.1.4 Přizpůsobení testovacím datům

Článek [6] sice na experimentech poukazuje na fakt, že i při změně některých podmínek (redukce počtu úrovní sester na 1, náhodném vygenerování penalizací za porušení měkkých podmínek, zavedení záporné penalizace při porušení omezení) vykazuje algoritmus dobré výsledky na testované instanci, nicméně jen s obtížemi si lze představit, že by totožný způsob byl aplikovatelný na množinu testovacích dat uvolněnou vědeckou skupinou z Nottinghamské univerzity [9]. Pro experimentování proto byl použit původní algoritmus přizpůsobený vstupním testovacím datům.

Jak již bylo řečeno, algoritmus hojně využívá k vyřešení úlohy hodnotu velikosti vybalancování směn jak v evaluační funkci, tak i skrze proměnou *InFeas* v heuristikách. Je třeba si uvědomit, z jakého důvodu se algoritmus řídí vybalancováním směn. Evaluační funkce

původního algoritmu je přizpůsobena tomu, že danou směnu bezpodmínečně musí odpracovat požadovaný počet zdravotních sester, což znamená, že se jedná o tvrdou podmínku (v tomto případě bez přímého určení, o kolik se zvýší hodnota penalizace při jejím porušení - porušením se stává rozvrh automaticky neakceptovatelným). Při řešení problému algoritmus potřebuje nějakým způsobem vyhodnotit, zdali došlo ke zlepšení rozvrhu, či nikoliv, a při daném způsobu řešení je třeba v evaluační funkci počítat právě i s porušením tohoto tvrdého omezení. Evaluační funkce se proto skládá z hodnoty součtu penalizací vzniklých porušením měkkých omezení (penalizace zaměstnanců) a z hodnoty vypočtené jako součet velikosti nepokrytí směn potřebnými zaměstnanci (porušení tvrdého omezení) vynásobené určitým koeficientem. Odpovědí na tento požadavek se stala evaluační funkce dle vzorce 4.5.

Spolu s cílovými testovacími vstupními daty (z kolekce Nottinghamské univerzity) ovšem algoritmus získá i množinu informací, mezi nimiž je i velikost penalizace v případě, že se nedodrží požadovaný počet pracujících lidí v daný čas (nemusí být tvrdou podmínkou). Proto není vhodné určovat evaluační hodnotu způsobem uvedeným v článku, ale využít již zmiňovaných informací. Ze stejného důvodu bude v heuristikách podmínka na nezhoršení hodnoty *InFeas* nahrazena výpočtem penalizace za pokrytí rozvrhu.

Původní algoritmus dále počítá s tím, že daná zdravotní sestra pracuje celý týden buď na denní, nebo noční směny. Jak ale ukazuje obrázek 4.2, mezi omezující podmínky cílových testovacích vstupních dat rozhodně tento požadavek nemusí patřit (pozn.: zkratka *TD* signalizuje denní směnu, naopak *TN* noční):

+ -	1							2							
1993 listopad	01	02	03	04	05	06	07	08	09	10	11	12	13	14	
	M	T	W	T	F	S	S	M	T	W	T	F	S	S	
1				TD	TN			TD	TD			TN	TN	TN	0
2			TD	TD					TN	TN		TD	TD	TD	0
3	TN	TN			TD	TD	TD				TD	TD			0
4			TN	TN		TD	TD	TN		TD	TN				0
5	TN	TN	TN			TN	TN				TD	TN			0
6				TN	TN			TN	TN	TN			TD	TD	0
7	TD	TD	TD					TD	TD				TN	TN	0
8	TD	TD			TD	TN	TN			TD	TN				0
														Employees' Penalty	0

Obrázek 4.2: Příklad vygenerovaného rozvrhu s optimálním rozvržením směn.

Z tohoto důvodu jsou heuristiky *H6* a *H7* pro cílová vstupní testovací data bezpředmětné. Pro přehlednost následuje seznam upravených heuristik použitých pro řešení problému rozvrhování směn zdravotních sester. V seznamu se vyskytuje zkratka CP (z anglického *Cover Penalty*), která označuje penalizaci způsobenou nedodržováním preferovaného počtu pracujících zdravotních sester v daném časovém úseku:

- **H1a:** Změň vzor směn náhodně vybrané sestry.
- **H2a:** Stejná jako *H1a*, ale akceptační kriterium je „první zlepšující *CP*“.
- **H3a:** Stejná jako *H1a*, ale akceptační kriterium je „první zlepšující *CP* a nezhoršující hodnotu penalizace zaměstnanců“.
- **H4a:** Stejná jako *H1a*, ale akceptační kriterium je „první zlepšující hodnotu penalizace zaměstnanců“.
- **H5a:** Stejná jako *H1a*, ale akceptační kriterium je „první zlepšující hodnotu penalizace zaměstnanců a nezhoršující *CP*“.
- **H6a:** Nalezení prvního přesunu takového, kdy změníme vzor směn dané sestry a původní vzor přiřadíme sestře jiné.
- **H7a:** Stejná jako *H6a*, nesmí ovšem dojít ke zhoršení *CP*.

I přes tyto úpravy si algoritmus uchovává hlavní ideu původního algoritmu:

- **Použití vzorů směn** - heuristiky nemění jednotlivé směny, ale přiřazují sestrám celé vzory směn určité délky. Délka zůstane na hodnotě sedm dní. V případě, že algoritmus bude rozvrhovat časové rozpětí více než sedm dní, dojde k rozdělení rozvrhu právě dle těchto časových bloků. Kvůli obrovskému počtu potenciálních kombinací není ani možné generovat delší vzory. Vzhledem k tomu, že heuristika musí být co nejobecnější, vygenerují se vždy všechny možné kombinace, nehledě na to, jestli vzor nebude obsahovat žádnou směnu, či naopak každý den bude dnem pracovním.
- **Kombinace heuristik zaměřených na snižování penalizace zaměstnanců a CP** - článek [12], ze kterého z části vychází původní algoritmus, poukazuje na tento jev jako na způsob, který umožňuje dosáhnout dobrých výsledků algoritmu. Změnou heuristik ovšem nezpůsobujeme zánik tohoto případného benefitu.

4.2 Modifikovaný algoritmus

4.2.1 Problémy původního algoritmu

Modifikovaný algoritmus by měl vzniknout jako reakce na nedostatky původního algoritmu, který některými prvky snižuje svoji efektivitu při řešení problému rozvrhování směn zdravotních sester za předpokladu rozdílných vstupních parametrů. Za největší problém lze označit kombinace faktů:

- zdravotní sestra má daný týden přiřazeny buď jen noční, nebo jen denní směny;
- týdenní rozvržení směn sestry se může měnit pouze v závislosti na množině uskutečnitelných vzorů směn.

Tyto problémy byly vyřešeny (jak popisuje část 4.1.4) za prvé zrušením rozdělení na denní a noční směny a za druhé určováním použitelných vzorů směn pouze na bázi délky vzoru směn a počtu typů směn v rozvrhu. Tím byl původní algoritmus zčásti zbaven fixací na jeden konkrétní typ problému. Na druhé straně se razantně zvětšil stavový prostor řešení a tím i doba potřebná pro jeho procházení.

Dalším specifickým parametrem, který algoritmu umožňoval na původních testovacích vstupních datech dosahovat skvělých výsledků, je poměrně omezená délka rozvrhovacího období (1 týden). Množina testovacích dat (z Nottinghamské univerzity) ovšem počítá převážně s rozvrhovacím obdobím alespoň 14 dní. Jako nejjednodušší řešení se nabízí možnost rozdělit delší období do několika menších. S tímto řešením ovšem přichází další problém. Heuristiky z původního algoritmu neumožňují měnit rozvrh směn ve více týdnech najednou (vzhledem k omezené délce rozvrhu, který řešily, ani tuto funkcionalitu nepotřebovaly mít). Existují ovšem situace, kdy pro zlepšení kvality rozvrhu je zapotřebí změnit přiřazené směny různě po celé délce rozvrhovacího období, jelikož při změně pouze jednoho vzoru směn může dojít ke zvýšení penalizace. Tím ovšem dojde k zamítnutí tohoto kroku i přesto, že při dalším spuštění některé z heuristik by mohl znamenat zlepšení kvality rozvrhu. Částečně toto omezení řeší heuristiky, které provádějí změny nehledě na penalizaci za pokrytí nebo penalizaci zaměstnanců (příp. obojí), nicméně vzhledem k velkému množství změn, proveditelných z každého stavu pro získání stavu jiného, je pravděpodobnost provedení pozitivní změny poměrně malá.

4.2.2 Návrh modifikovaného algoritmu

4.2.2.1 Principy algoritmu

Původní algoritmus mimo jeho poměrně rozsáhlých nedostatků pro všeobecné použití poskytuje i některé klíčové benefity, které je vhodné zachovat u modifikovaného algoritmu. Hyper-heuristiky jsou, díky umožnění integrace více různých heuristik, kdy každá má jiné nedostatky a jiná pozitiva, rozhodně vhodné pro aplikaci na problémy s různými vstupními parametry. Ani použití metody tabu vyhledávání pro řízení heuristik se nezdá být překážkou pro dosažení cíle této práce. Jako evaluační funkce bude (stejně jako u upraveného původního algoritmu) využita celková penalizace (pokrytí a zaměstnanců) daného rozvrhu. Hlavním rozdílem je tedy použití jiných heuristik.

4.2.3 Použité (nízkoúrovňové) heuristiky

Pro dosažení obecnosti problému je potřeba, aby i dané heuristiky byly pokud možno co nejobecnější a co možná nejméně se opíraly o metadata popisující konkrétní instanci. To ovšem neznamená, že není účelné zvolit taková omezení, která jsou typická pro většinu vstupních instancí.

Heuristiky původního algoritmu počítají s tím, že může docházet ke zvyšování a snižování počtu přiřazených směn, což se zajisté hodí např. jako prostředek pro uniknutí z lokálního extrému. Změna směn pouze u jednoho zaměstnance (jejímž základem je hned několik heuristik z původního algoritmu) ale znamená obvykle i změnu pokrytí směn, tedy že v případě změny libovolného vzoru směn u rozvrhu s nulovou penalizací za pokrytí se tato penalizace většinou zvýší. Proto v původním algoritmu docházelo ke střídání heuristik, kdy některé se

zabývaly především snižováním penalizace zaměstnanců, a jiné naopak penalizací za pokrytí (penalizace se neustále měnily). Požadovaný počet zaměstnanců přiřazený k dané směně ale patří zpravidla mezi nejsilnější omezení, proto navržený modifikovaný algoritmus nebere v potaz situace, kdy by nebylo toto omezení splněno. Z toho vyplývá, že již od začátku musí mít k dispozici rozvržení směn splňující podmínku pokrytí (bez ohledu na kvalifikaci jednotlivých zaměstnanců). Dále je vhodné, aby již při inicializaci byla brána v potaz v praxi velmi častá podmínka, že buď bude mít zaměstnanec oba dny o víkendu volné, či naopak oba pracovní. Pro účely srovnávání potřebujeme, aby všechny algoritmy vycházely ze stejného inicializačního rozvrhu, proto bude generován pseudonáhodně. Pro definovanou pseudonáhodnost lze s výhodou využít funkci *rand(int seed)* standardně poskytnutou jazykem C# a nastavit jí výchozí bod pomocí hodnoty *seed*.

Ideálně by se tedy suma přiřazených směn neměla měnit, což splňuje operace *záměna směn* (swap), která, jak již název napovídá, směny jednoho zaměstnance přiřadí druhému zaměstnanci a zároveň původní směny druhého zaměstnance přiřadí zaměstnanci prvnímu. Výhodou je velice jednoduchá implementace a poměrná nezávislost na paremetrech rozvrhu, které řeší, na druhé straně není možné pomocí ní projít celý stavový prostor řešení. I přes značné výhody nestačí pouze její primitivní implementace pro efektivní řešení problému rozvrhování směn zaměstnancům, lze ji ovšem označit za základní kámen pro navrhovanou množinu heuristik modifikovaného algoritmu.

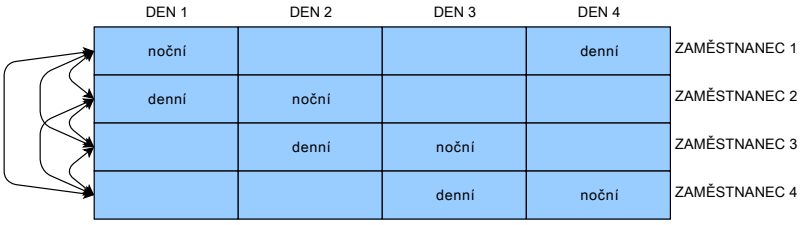
Výhodou a zároveň nevýhodou (upraveného) původního algoritmu je použití vzorů směn. Možností změnit více směn najednou se zvětší množina sousedních řešení, čímž se zvýší pravděpodobnost úniku z lokálního extrému, nicméně jako nežádoucí se jeví statické stanovení délky vzoru, stejně tak jako fixní určení, že první den ze vzoru směn připadá na první den daného týdne. Heuristiky modifikovaného algoritmu tedy budou pracovat s bloky proměnné délky (blokem se rozumí sekvence po sobě jdoucích dní, během nichž může, ale nemusí být přiřazena nějaká směna danému zaměstnanci). Výsledná množina heuristik zahrnuje čtyři, respektive pět následujících heuristik:

- **mH1** - vrátí rozvrh s nejnižší penalizací získaný záměnou bloků směn dvou zaměstnanců.
- **mH2** - vrátí rozvrh s nejnižší penalizací získaný sekvencí (délka sekvence = 2) záměn bloků směn dvou zaměstnanců - prohledávání omezeného počtu dní.
- **mH3** - vrátí rozvrh s nejnižší penalizací získaný sekvencí (délka sekvence = 2) záměn bloků směn dvou zaměstnanců - prohledávání směn omezeného počtu zaměstnanců.
- **mH4** - provede částečný restart rozvrhu.
- **mH5** - vrátí rozvrh s nejnižší penalizací dosaženou během provádění algoritmu.

Poslední z heuristik vzhledem k její triviálnosti nepatří mezi klasické heuristiky.

4.2.3.1 Nízkoúrovňová heuristika *mH1*

Heuristika *mH1* probíhá deterministicky. Otestují se všechny možné záměny bloků směn mezi dvěma zaměstnanci. Záměna, která se ukáže jako nejlepší z hlediska výsledné penalizace

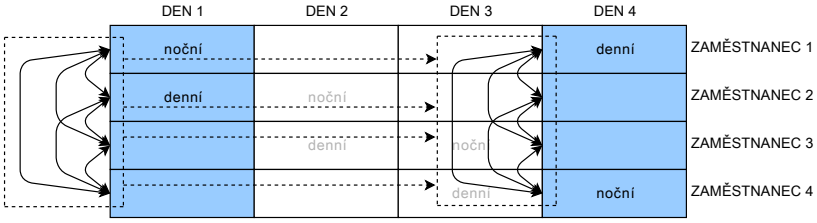


Obrázek 4.3: Princip heuristiky *mH1*.

směny, se provede. V případě, že neexistuje záměna, která by znamenala zlepšení penalizace, žádná změna se v rozvrhu neprovede. Maximální velikost bloku je vzhledem k práci [31] nastavena na hodnotu 5. Princip fungování této metody nastiňuje obr. 4.3.

4.2.3.2 Nízkoúrovňová heuristika *mH2*

Samotná heuristika *mH1* nedokáže řešit případy, kdy je zapotřebí provést více záměn současně, a vzhledem k tomu, že nevrací rozvrh s horší penalizací, než jaký dostane ke zpracování, došlo by velice jednoduše k lokálnímu uváznutí. Částečným řešením je heuristika *mH2*. Ta provede dvě záměny najednou a teprve poté otestuje, zda došlo ke snížení penalizace rozvrhu.



Obrázek 4.4: Princip heuristiky *mH2*.

Na začátku heuristiky se náhodně vyberou dva dny. Poté se provedou všechny možné záměny směn zaměstnanců v bloku dní začínajícím prvním vybraným dnem. Po každé záměně směn v prvním náhodně vybraném bloku se provedou všechny možné záměny ve druhém bloku, začínajícím v druhém náhodně vybraném dnu. Pro názornost je zde uveden obr. 4.4 (který ilustruje záměny směn pouze jednotlivých dní). U všech kombinací se otestuje penalizace takto vzniklého rozvrhu. Změny, které znamenají nejvýraznější snížení penalizace, se provedou. V případě, že není nalezeno lepší řešení, žádné změny provedeny nejsou. V případě aplikace na rozvrh s velkým počtem zaměstnanců by mohlo dojít k neúměrnému zvýšení potřebného času pro dokončení heuristiky *mH2*, proto v některých případech dochází k omezení počtu zaměstnanců, u kterých jsou záměny provedeny.

4.2.3.3 Nízkoúrovňová heuristika *mH3*

Heuristika *mH3* je podobná předešlé heuristice *mH2*, nevybírá ale pouze dva dny, nýbrž dnů více. Tím by se ovšem neúměrně zvýšila její složitost, proto vybere pouze dva zaměstnance,

kteří nejvíce přispívají k penalizaci, a záměny směn se provádějí pouze mezi nimy. Popsat slovně detaily algoritmu by bylo složité, přesný postup algoritmu vysvětluje pseudokód 1. Stejně jako v předešlých případech se provedou z hlediska penalizace nejúspěšnější kroky. V případě neúspěchu hledání lepšího rozvrhu vrátí heuristika rozvrh beze změny.

Pseudokód 1 Pseudokód heuristiky *mH3*

```
1: procedure APLIKUJ_HEURISTIKU_MH3(rozvrh)
2:     nejlepsi_rozvrh ← rozvrh
3:     Najdi zaměstance s nejvyšší penalizací (z1) a druhou nejvyšší penalizací (z2).
4:     Vyber dny, kterých se budou záměny týkat.
5:     Vyber zaměstnance, kterých se budou záměny týkat.
6:     loop
7:         Prověď všechny možné záměny (vzhledem k omezení dnů a zaměstnanců) zaměstnance z1 s ostatními zaměstnanci v rozvrhu rozvrh, po každé záměně vrať rozvrh do původního stavu.
8:         loop
9:             Po každé záměně (z předchozího bodu) otestuj všechny možné záměny (vzhledem k omezení dnů a zaměstnanců) zaměstnance z2 a:
10:            if (penalizace(rozvrh_po_zamenach) < (penalizace(nejlepsi_rozvrh)) then
11:                nejlepsi_rozvrh ← rozvrh_po_zamenach
12:            end if
13:        end loop
14:    end loop
15:    return nejlepsi_rozvrh
16: end procedure
```

K výběru pouze některých dní a zaměstnanců dochází z důvodu snížení výpočetní náročnosti heuristiky.

4.2.3.4 Nízkoúrovňová heuristika *mH4*

V případě, že na rozvrh aplikujeme nějakou z heuristik *mH1-mH3*, může se penalizace rozvrhu pouze snížit, nebo zůstat stejná (jestliže zůstane stejná, nedojde ani ke změně rozvrhu). Proto, pokud z daného stavu není možné libovolnou sekvencí dvou záměn získat lepší řešení, nelze již dále dosáhnout lepšího výsledku. Řešením této situace je zavedení částečného restartu rozvrhu. Heuristika *mH4* tedy odhadne, které směny (myšleno přiřazení směn zaměstnanci) způsobují procentuálně nejvyšší penalizaci. Zjištění problémových směn ovšem není přímočaré. Označení směny za nevhodnou proto probíhá tak, že se daná směna zruší a přepočítá se penalizace zaměstnanců. Dle velikosti snížení této penalizace se vyberou směny, které bude možné odstranit. Po jejich odstranění se provede přesunutí některých směn, načež následuje proces doplnění chybějících směn.

4.2.3.5 Nízkoúrovňová heuristika *mH5*

Po aplikaci každé vybrané heuristiky se zjistí penalizace nově vzniklého rozvrhu a v případě, že se jedná z tohoto hlediska o rozvrh nejlepší, uloží se do paměti. Heuristika *mH5* pouze vrátí rozvrh označený jako nejlepší aktuálně dosažený. Právě proto ji nelze označit za klasickou heuristiku. Důvodem používání této heuristiky je snaha o zlepšení penalizace nejlepšího rozvrhu.

4.2.4 Řízení heuristik

Heuristiky *mH2-mH3* jsou plně řízeny metodou tabu vyhledávání. Heuristika *mH1* je oproti *mH2-mH3* plně deterministická, proto nemá smysl ji aplikovat vícekrát na stejný stav. Z tohoto důvodu se v případě opakovaného vybrání (během stejného stavu) metodou tabu vyhledávání automaticky označí jako neúspěšná, a nedojde tedy k její opětovné aplikaci.

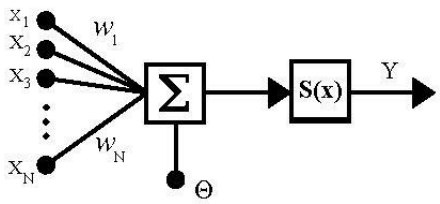
Jak již bylo několikrát zmíněno, heuristiku *mH4* používá algoritmus ve chvílích, kdy je potřeba uniknout z lokálního uváznutí, v ostatních případech ovšem může její spuštění způsobit problémy. Stejným způsobem (jako u *mH1*) dojde k eliminaci spuštění heuristiky *mH4* v případě, že nedošlo ke dlouhodobější stagnaci penalizace. Heuristika *mH4* totiž slouží výhradně jako prostředek pro uniknutí z lokálního extrému, v ostatních případech ji nemá smysl používat. Poslední heuristika *mH5* není řízena metodou tabu vyhledávání, pouze je staticky definován okamžik, za jakých podmínek dojde k jejímu využití.

4.3 Neuronová síť

Metoda tabu vyhledávání použitá v této práci vybírá heuristiky ke spuštění pouze dle úspěchu daných heuristik v poměrně krátkém období. Pro delší období by bylo vhodné využít technologii, která umožňuje řídit kroky algoritmu na bázi dlouhodobějších předchozích zkušeností. Z možných alternativ mající tuto vlastnost byla zvolena neuronová síť (viz sekce 2.5).

4.3.1 Princip

Neuronová síť patří mezi univerzální výpočetní modely inspirované biologickým nervovým systémem. Často se používá pro řešení problémů, kdy nepotřebujeme nutně znát přesný výsledek, protože její typická úspěšnost (při dobrém naučení neuronové sítě) je okolo 95% [16].



Obrázek 4.5: Model neuronu (S(x) značí přenosovou funkci).

Neuronová síť se skládá alespoň z jednoho neuronu (viz obr. 4.5) [20]. V případě této minimální struktury se nazývá *perceptronem*. Každý neuron má n vstupů $(x_1 \dots x_n)$, které jsou ohodnoceny vahami $(w_1 \dots w_n)$, a práh θ .Vnitřní potenciál neuronu se určuje:

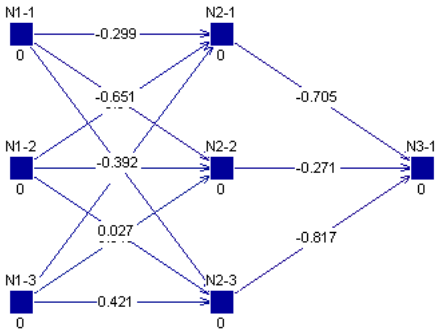
$$\xi = \sum_{i=1}^{n} w_i x_i + \theta \tag{4.7}$$

Výstupní hodnota neuronu se vypočítá na základě hodnot θ, ξ a použité přenosové funkce. V našem případě se použije pro neuronovou síť často aplikovaná funkce sigmoidální.

Tím pádem lze výstupní funkci neuronu (v případě sigmoidy) zapsat jako

$$y = \frac{1}{1 + e^{-\xi}} \tag{4.8}$$

Vzhledem ke zvolené přenosové funkci a předpokládanému rozsahu vstupu neuronové sítě (viz sekce 4.3.2.1) je zachována prahová hodnota $\theta = 0$, jelikož v daném rozsahu vstupních hodnot poskytuje sigmoida rozumnou a především stabilní strmost výstupní funkce, a tak není nutné tento rozsah měnit. Neuronová síť samozřejmě obvykle neobsahuje pouze jeden neuron. Jedno z jejich možných zapojení zobrazuje obrázek 4.6.



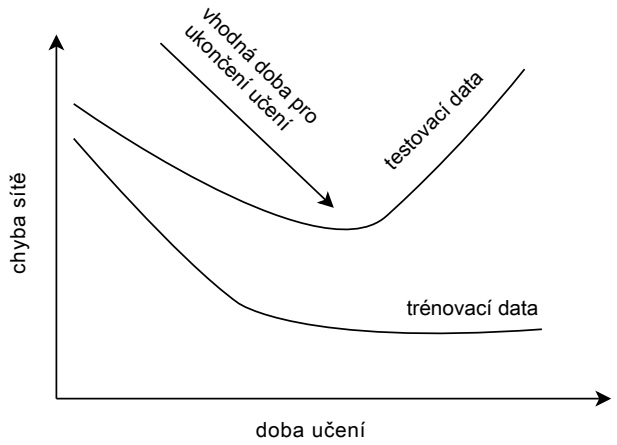
Obrázek 4.6: Struktura neuronové sítě o jedné vstupní, skryté a výstupní vrstvě (generován v programu JNNS).

Zobrazená síť obsahuje pouze jednu skrytou vrstvu, nicméně skrytých vrstev může být neomezeně. S větším počtem skrytých vrstev lze zpřesnit dělení prostoru a tím i výsledky poskytnuté neuronovou sítí na základě určitého vstupu, na druhé straně se taková síť déle učí i klasifikuje. Pro účely této práce postačí neuronová síť s jednou skrytou vrstvou. Uzly (z obrázku 4.6) označené jako N1-1, N1-2 a N1-3 značí neurony vstupní vrstvy, N2-1, N2-2 a N2-3 neurony vrstvy skryté a poslední neuron (N3-1) patří do vrstvy výstupní. Hodnoty na spojnicích mezi uzly jsou vstupní váhy neuronu.

Architekturou patří neuronová síť z obrázku 4.6 mezi vrstevnaté sítě. Vstupní vrstva splňuje mimo jiné funkci přeposílání vstupního signálu na neurony další vrstvy. Skrytá a poté i výstupní vrstva patří mezi „inteligentní části“ sítě, kde se výstup jednotlivých neuronů vypočítává dle již uvedeného vzorce 4.8. Výstupní vrstva oproti skryté již nemá propojený výstup s dalšími neurony. [7, 17]

Pro naučení neuronové sítě se v tomto případě používá algoritmus zpětné propagace (anglicky Backpropagation) s dynamicky se vyvíjecím koeficientem *rychlosti učení* (s průměrnou hodnotou 1,75). *Rychlost učení* udává, jak rychle se během učení mění vstupní váhy neuronů. Při vysoké hodnotě *rychlosti učení* dochází k rychlejšímu, ale potenciálně nekvalitnímu naučení sítě. Proto na začátku učení, kdy je chybovost sítě větší, dochází k rychlejšímu učení, ale postupem času klesá.

Na začátku dojde k inicializaci vstupních vah, což obvykle znamená nastavení těchto parametrů náhodně v rozmezí <-1, 1>. Dále se neuronové síti předloží učicí data skládající se z množiny vzorků. Každý vzorek obsahuje vstupní a adekvátní výstupní hodnoty, neboli dvojice, které určují, jaký výstup má neuronová síť vykazovat při daném vstupu. V případě, že výstup generovaný neuronovou sítí neodpovídá výstupní hodnotě uvedené v aktuálním vzorku, provede se změna vah dle již zmiňovaného algoritmu zpětné propagace, založeného na tzv. gradientní metodě. Algoritmus je podrobněji popsán v článku [16].



Obrázek 4.7: Chyba neuronové sítě na trénovacích a testovacích datech v závislosti na době učení.

Při neúměrně dlouhé době učení může dojít k přeučení dané neuronové sítě, proto je možné ověřit naučení testovacími daty. Vhodnou dobu pro ukončení trénování sítě naznačuje obr. 4.7. Délkou učení se myslí, kolikrát předložíme množinu učících dat neuronové síti. Počet opakování se někdy označuje jako počet epoch [24, 13].

4.3.2 Využití pro účely této práce

4.3.2.1 Klasifikace prostřednictvím neuronové sítě

Velice zjednodušeně by se dalo říct, že metoda tabu vyhledávání řídí spouštění heuristik na základě jejich úspěšnosti v posledním krátkém období (délku období je možné prodlužovat pomocí zvyšování konstanty rn_{max} popsané v části 4.1.1). Ve fázích, kdy je penalizační hodnota (vzhledem k možným reálně dosažitelným hodnotám) vysoká, v podstatě takové chování vyhovuje, a není příliš důvod jej měnit. Nicméně ve chvílích, kdy se pravděpodobnost zlepšení daného rozvrhu použitím definovaných heuristik začne snižovat, bylo by vhodné použít co nejsofistikovanější metodu, např. právě rozhodování pomocí neuronové sítě.

Pseudokód 2 Pseudokód modifikovaného algoritmu s využitím neuronové sítě

```
1: procedure MOD_TABU_VYHLEDAVANI_S_NEURO_SITI(rozvrh)
2:     inicializujTabuList()
3:     inicializujHodnotyRank() //ohodnocení heuristik pro účely metody tabu vyhledávání
4:     inicializujPozadovanouNeuronovouSit()
5:     naucNeuronovouSit()
6:     stejnaPenalizace ← 0
7:     while !splnenyUkoncovaciPodminky() do
8:         if stejnaPenalizace < 2 then
9:             aplikujHeuristikuVybranouTabuVyhledavanim(rozvrh)
10:        else
11:            aplikujHeuristikuVybranouNeuronovouSiti(rozvrh)
12:        end if
13:        hodnota1 = penalizace(rozvrh_pred_apl_heuristiky)
14:        hodnota2 = penalizace(rozvrh_po_apl_heuristiky)
15:        if hodnota1 > hodnota2 then
16:            stejnaPenalizace ← 0
17:            zvysRankVybraneHeuristiky()
18:        else
19:            stejnaPenalizace ← stejnaPenalizace + 1
20:            snizRankVybraneHeuristiky()
21:            vlozHeuristikuDoTabuListu()
22:        end if
23:    end while
24:    return rozvrh
25: end procedure
```

Z důvodu lepšího srovnání algoritmů není neuronová síť používána ve všech případech, ale pouze tehdy, když pomocí metody tabu vyhledávání nedochází ke snižování penalizace rozvrhů (viz kapitola 5.4.1). Průběh algoritmu popisuje pseudokód 2.

4.3.2.2 Návrh neuronové sítě

U neuronové sítě je nejdříve potřeba rozhodnout, jaké vstupy a výstupy, jejichž počet by měl být neměnný, bude mít. Při změně počtu vstupů nebo výstupů se totiž již naučená neuronová síť musí opětovně naučit na nových datech taktéž s modifikovaným počtem vstupních i výstupních prvků. Není proto vhodné na vstup přivádět pole hodnot udávající přiřazení směn jednotlivým zaměstnancům, protože např. při změně počtu zaměstnanců by bylo nutné vygenerovat novou trénovací množinu dat a s nimi neuronovou síť opětovně naučit. Pro účely neuronové sítě je zapotřebí najít jiný vhodnější způsob, kterým se docílí stabilního počtu prvků. Alternativu nabízí článek [33], který několika parametry popisuje pouze strukturu vstupní instance, ale bohužel se zabývá odlišným problémem (plánováním zdrojů), a tak je nutná modifikace. Modifikovaný vstup neuronové sítě se skládá z osmi vstupních parametrů:

- nejmenší blok směn v rozvrhu;

- největší blok směn v rozvrhu;
- největší penalizace zaměstnance;
- penalizace rozvrhu (2x);
- počet iterací bez zlepšení penalizace rozvrhu (2x);
- příznak stagnace velikosti penalizace rozvrhu.

První čtyři body přímo popisují strukturu, popřípadě vlastnosti rozvrhu. Nejmenší blok směn znamená nejmenší počet pracovních dní přiřazených jednomu zaměstnanci za sebou. Naopak největší blok udává nejdelší sekvenci takových dní (viz 4.8). Neuronová síť (při použití sigmoidální přenosové funkce) pracuje se vstupními hodnotami v omezeném rozsahu. Při vyšších (kladných i záporných) vstupních hodnotách (zhruba od hodnoty 3) se výstupní hodnota již téměř nemění. Proto je v této práci snahou dodržovat maximální vstupní hodnotu zhruba do hranice 1 a vzhledem k ní dochází ke konverzi hodnot [2]. Pokud nejmenší nalezený blok je 2 a největší je 5, poté vstupem neuronové sítě tedy budou hodnoty $vstup_1 = 0,2$ a $vstup_2 = 0,5$, čili desetkrát menší.

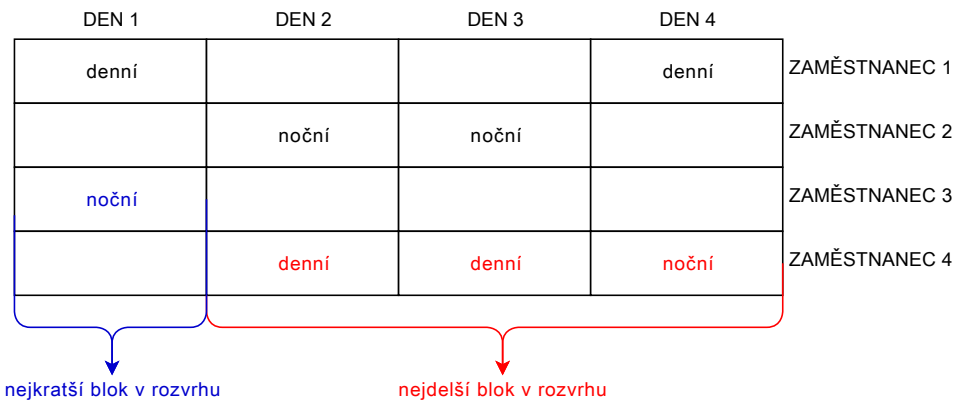
Největší penalizaci zaměstnance není třeba nějak zdlouhavě vysvětlovat, protože použitý framework přímo poskytuje tuto hodnotu (suma porušených měkkých omezení) u každého zaměstnance. Samozřejmě, opět kvůli omezenému vstupnímu rozsahu, dojde k převodu, konkrétně $vstup_3 = \log_{10} maximalni_penalizace_zamestnance/5$, což umožňuje bez problému reagovat na penalizaci od 0 do více než 100 000, která by měla pro účely této práce dostačovat. Překročení penalizace rozvrhu nad hodnotu 100 000 neznamená (za předpokladu stanovení rozumně velké váhy omezení, tzn. maximálně do řádu tisíců) pro řízení heuristik výrazný problém, obojí totiž signalizuje nekvalitně rozvržený rozvrh. V případě, že bychom pracovali s instancemi, kde rozdíly vstupních hodnot jsou daleko větší, a nechtěli bychom u nich z nějakého důvodu použít logaritmický převod, je možné použít místo statického přepočtu lineární mapovací funkci:

$$\begin{aligned} nova_hodnota = (&\frac{(stara_hodnota - stare_minimum)}{(stare_maximum - stare_minimum)} \times \\ &\times (nove_maximum - nove_minimum)) + nove_minimum \end{aligned} \tag{4.9}$$

Hodnota (celkové) penalizace rozvrhu, neboli součet penalizace zaměstnanců a penalizace za nedostatečné pokrytí rozvrhu, musí být opět uzpůsobena vstupům neuronové sítě, stejným způsobem jako u *největší penalizace zaměstnance*. Vzhledem k vyšší důležitosti penalizace se hodnota přikládá na dva vstupy neuronové sítě, čímž se zvýší vliv na její výstupní hodnoty.

Informace o době stagnace penalizace pomáhá neuronové síti rozhodnout, zdali má smysl dále zkoušet zlepšit konkrétní stav, anebo provést částečný restart (popřípadě načíst doposud zjištěný nejlepší stav). Opět nelze použít přímo hodnotu určující počet iterací, během nichž nedošlo ke změně penalizace rozvrhu, proto:

$$vstup_6 = pocet_iteraci_bez_zmeny_penalizace/20.$$



Obrázek 4.8: Ukázka minimálního a maximálního bloku směn v rozvrhu.

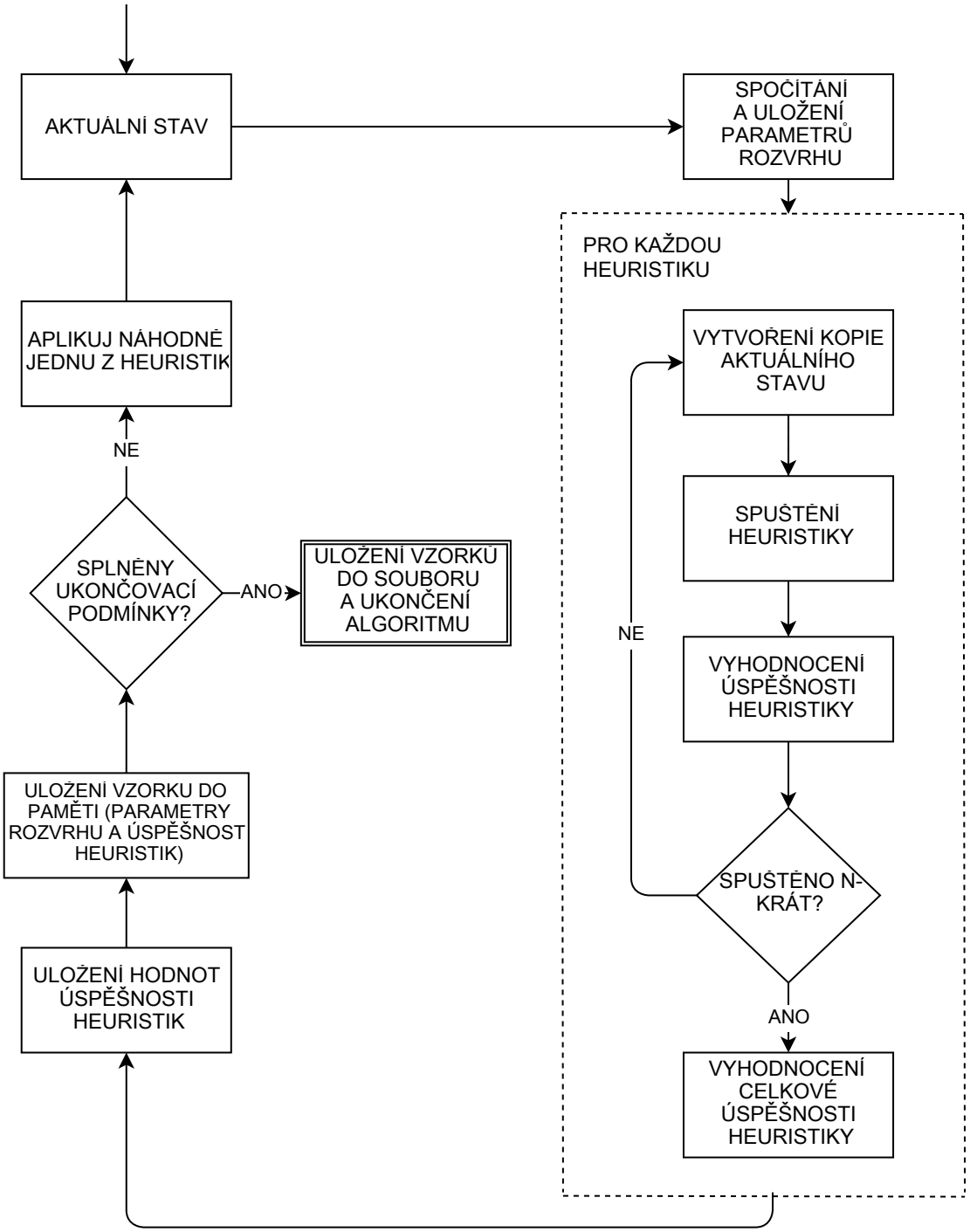
Opět jde o poměrně důležitou hodnotu, takže $vstup_6 = vstup_7$. Posledním bodem v seznamu a zároveň posledním vstupem se rozumí speciální příznaková hodnota, která pouze indikuje, že došlo ke stagnaci penalizace (1 znamená, že se penalizace použitím poslední heuristiky nezměnila, 0 značí případ opačný). Tuto hodnotu lze bez problému určit z počtu iterací bez zlepšení penalizace. Důvod jejího nasazení bude vysvětlen dále v této kapitole.

Výstup neuronové sítě nám musí nějakým způsobem poskytnout informaci o tom, které heuristiky mají v případě spuštění v danou chvílí největší pravděpodobnost na úspěch. Nejjednodušší by zajisté bylo, aby výstupní hodnota udávala číslo heuristiky, která má být spuštěna. Takový způsob by ovšem mohl znamenat časté lokální uváznutí, jelikož pro stejný stav by se vždy aplikovala stejná heuristika. Bylo by proto vhodné, aby neuronová síť podávala informace o pravděpodobnosti úspěšnosti každé z heuristik a řídící algoritmus se mohl na jejich základě rozhodnout, jak se zachová.

Pro kvalitní naučení neuronové sítě potřebujeme poměrně velké množství vzorků (vstupní parametry a požadované výstupy) dat z různých míst stavového prostoru řešení. Pro procházení prostoru je využita trojice heuristik (*mH1, mH2 a mH3* z modifikované verze algoritmu tabu vyhledávání popsaného v této práci), které se náhodně spouštějí. V případě uváznutí v lokálním extrému dojde k částečnému restartu (*mH4*) a algoritmus dále pokračuje, dokud nedojde ke stanovenému počtu iterací. Po každém spuštění jakékoliv heuristiky se v daném stavu spustí všechny používané heuristiky (*mH1, mH2 a mH3* vždy, *mH4* pouze pokud dojde k zastavení zlepšování penalizace rozvrhu) a vyhodnotí se jejich úspěšnost (jestli došlo ke zlepšení penalizace rozvrhu či nikoliv). V každé iteraci jsou spuštěny heuristiky několikrát. Po každém spuštění se zaznamená, zdali došlo ke zlepšení penalizace rozvrhu, či nikoliv. Z těchto záznamů se poté vypočítá úspěšnost jejich použití jednoduchým vzorcem:

$$P_{hd} = \frac{1}{n} \sum_{i=1}^{n} u_{id}, \tag{4.10}$$

kde n značí počet spuštění heuristiky (z původního stavu), u_{id} nabývá hodnoty 1, pokud došlo ke zlepšení penalizace rozvrhu po i-tém aplikování heuristiky d, a 0, pokud se stav nezměnil. P_{hd} označuje úspěšnost heuristiky d.



Obrázek 4.9: Diagram zobrazující proces generování vzorků potřebných pro učení neuronové sítě.

Obr. 4.9 zobrazuje průběh algoritmu pro generování vzorků (celý průběh se opakuje několikrát z důvodu vygenerování většího množství dat). Pouze omezení spouštění heuristiky *mH4* (pro částečný restart) v něm není vyznačeno - heuristika *mH4* se testuje pouze ve fázích, kdy delší dobu nedochází aplikací heuristik ke zlepšení rozvrhu. Nemá totiž smysl pokoušet se částečně restartovat rozvrh ve chvílích, kdy není dosaženo lokálního extrému. Navíc by testování *mH4* v těchto fázích znamenalo neúměrné zpomalení algoritmu, protože tato heuristika má odlišné vlastnosti oproti *mH1-mH3*, a proto vyhodnocení její úspěšnosti probíhá daleko déle.

U heuristik *mH1-mH3* lze okamžitě po jejich aplikaci jednoduše zjistit úspěšnost pomocí rozdílu penalizace rozvrhu ze stavu před a po proběhnutí heuristiky. Pokud bychom stejnou metodu použili i pro *mH4*, prakticky by neměla šanci na úspěch, jelikož částečné zrestartování rozvrhu zpravidla znamená získání rozvrhu s daleko vyšší penalizací. Ideálním řešení z hlediska přesnosti by bylo zkusit aplikovat *mH4* a následně spouštět sekvenci heuristik *mH1-mH3* do té doby, než se tímto způsobem dosáhne nějakého lokálního extrému. Tímto řešením by ale docházelo k neúměrnému zvyšování času potřebného pro generování vzorků.

Zvolený kompromis funguje tím způsobem, že se porovnají penalizace původního rozvrhu a rozvrhu vzniklého následovně:

1. Aplikace heuristiky *mH4*.
2. Opakovaná aplikace heuristiky *mH1* do doby, dokud dochází ke změně rozvrhu.
3. Spuštění jedné z heuristik *mH1-mH3*. Výběr probíhá pseudonáhodně. Tento bod se opakuje čtyřikrát.

Pro počáteční sekvenci je zvolena heuristika *mH1* ze dvou důvodů. Za prvé se jedná o deterministickou metodu, to znamená, že pokud v daném stavu jednou neuspěje, není již důvod ji zkoušet znovu. Za druhé, vzhledem k jejímu charakteru se dá předpokládat, že z heuristik *mH1-mH3* bude trvat nejkratší dobu, čímž se sníží i doba potřebná pro ohodnocení úspěchu aplikace heuristiky *mH4* za daného stavu. Tato sekvence ovšem (především při řešení složitějších rozvrhů) se z hlediska počtu použitých heuristik nemusí blížit sekvenci předcházející rozvrhu, na který aplikujeme heuristiku *mH4*. Proto, aby nedocházelo ke znevýhodnění, nesrovnávají se přímo penalizace ze stavu před a po aplikaci *mH4*, ale úspěšnost heuristiky definuje vzorec (4.11):

$$P_{h4} = \begin{cases} 1, & ((1+K_{mh4})penalizace_pred_apl_mH4 \geq penalizace_po_apl_mH4) \\ 0, & ((1+K_{mh4})penalizace_pred_apl_mH4 < penalizace_po_apl_mH4) \end{cases} \tag{4.11}$$

Koeficient K_{mh4} ve vzorci 4.11 poté nabývá „vhodné“ nezáporné hodnoty a P_{h4} označuje úspěšnost heuristiky *mH4*. Nastavením K_{mh4} se bude tato práce zabývat v kapitole 5.

Z vygenerovaných vzorků jsou pro řešení nejvíce důležité ty, které se týkají situací, kdy dochází ke stagnaci ve vývoji penalizace rozvrhu, a proto je vhodné se zaměřit právě na ně. Teoreticky by bylo možné nepoužívat pro učení vzorky, které se těchto situací netýkají. Tedy vyfiltrovat takové, které byly vygenerovány ze stavů, kdy bylo jednoduché najít lepší řešení a zároveň je pravděpodobné, že heuristiky *mH1-mH3* budou mít vysokou úspěšnost nalezení

rozvrhu s nižší penalizací. Nicméně i tyto vzorky pomáhají utvářet logiku pro vyhodnocení vhodnosti použití té které heuristiky.

Vzhledem k nedeterministickému chování heuristik *mH2-mH3* se může např. stát, že stav, který během generování bude označen za bezproblémový (došlo k okamžitému nalezení rozvrhu s nižší penalizací), se při aplikaci neuronové sítě stane stavem problémovým (dojde ke stagnaci penalizace rozvrhu). Místo filtrace „nevhodných“ vzorků se proto použije metoda opačná - zvýšení důležitosti vzorků, které jsou vygenerovány během problémových stavů. Toho se docílí tím, že během učení se neuronové síti předloží vzorek dvakrát. Jednou standardně, tak jako vzorky ostatní, a podruhé s již zmiňovaným příznakem.

Kapitola 5

Experimenty

Časová náročnost většiny experimentů je obrovská, proto muselo být použito více výpočetních strojů, jejichž seznam zobrazuje tabulka 5.1.

Stroj	**CPU**	**Počet jader CPU**	**Frekvence CPU**	**Verze OS**
P1	C2D T6670	2	2,2 GHz	W7 64bit
P2	C2D T6670	2	2,2 GHz	W7 32bit
P3	Phenom II X4 955	2	3,2 GHz	W7 64bit
P4	Core i5 2410M	2	2,3-2,9 GHz	W7 64 bit
P5	Core i5 3317U	2	1,7-2,6 GHz	W7 64 bit
P6	2x Xeon X5460	2x 4	3,16 GHz	W2008 64bit
P7	Xeon E5530	4	2,4-2,66 GHz	W2008 64bit
P8	Xeon E5-2630	6	2,3-2,8 GHz	W2008 64bit

Tabulka 5.1: Seznam použitých výpočetních strojů k experimentům

Velikost operační paměti u výpočetních strojů není důležitá, jelikož naimplementovaný program zabírá pouze řádově desítky MB volné paměti. Bohužel u většiny z těchto strojů není zajištěn stabilní výpočetní výkon (z důvodu sdílení systémových prostředků s dalšími procesy), proto nejsou průběhy experimentů obvykle poměřovány z hlediska doby běhu algoritmu. Pouze u některých experimentů, kdy je zapotřebí srovnávat reálný čas zpracování daných úloh, jsou použity stroje *P1* a *P2*, u kterých může být z velké části zajištěna exkluzivita výpočetních prostředků.

5.1 Popis testovaných instancí

Pro otestování efektivity naimplementovaných algoritmů byly použity instance publikované univerzitou v Nottinghamu [9]. Z časových důvodů experimenty neprobíhaly na všech dostupných instancích, ale pouze na výběru z nich. Snahou bylo zvolit takovou podmnožinu těchto vstupních dat, aby zahrnovala problémy s různými parametry. Instance použité pro experimenty jsou popsány v tabulce 5.2.

Instance	Počet dní	Počet směn	Počet zaměstnanců	Nejlepší známé řešení - penalizace
Millar-1	14	2	8	0
GPost	28	2	8	5
GPost-B	28	2	8	3
Valouxis	28	3	16	20
LLR	7	3	27	301
SINTEF	21	5	24	0
MUSA	14	1	14	175
Millar-1.1	14	2	8	0

Tabulka 5.2: Seznam testovacích vstupních instancí

5.2 Výchozí (upravený) algoritmus

Původní algoritmus, z něhož vychází tato práce, byl upraven tak, aby fungoval pro všechny testovací vstupní instance (viz kapitola 4.1.4). Heuristky používané v tomto upraveném algoritmu vrací (až na *H1a* a *H6a*) rozvrh s nižší penalizací za pokrytí směn nebo nižší penalizací zaměstnanců. V případě, že by ale takový rozvrh neexistoval, mohlo by dojít k zacyklení algoritmu. Z tohoto důvodu musí být nastaven maximální počet pokusů na získání požadovaného rozvrhu. Při nastavení vyššího počtu se zvyšuje i pravděpodobnost úspěchu dané heuristiky, ovšem spolu s tím zároveň vzrůstá výpočetní náročnost. Vzhledem k těmto okolnostem byl nastaven limit na 1000 pokusů.

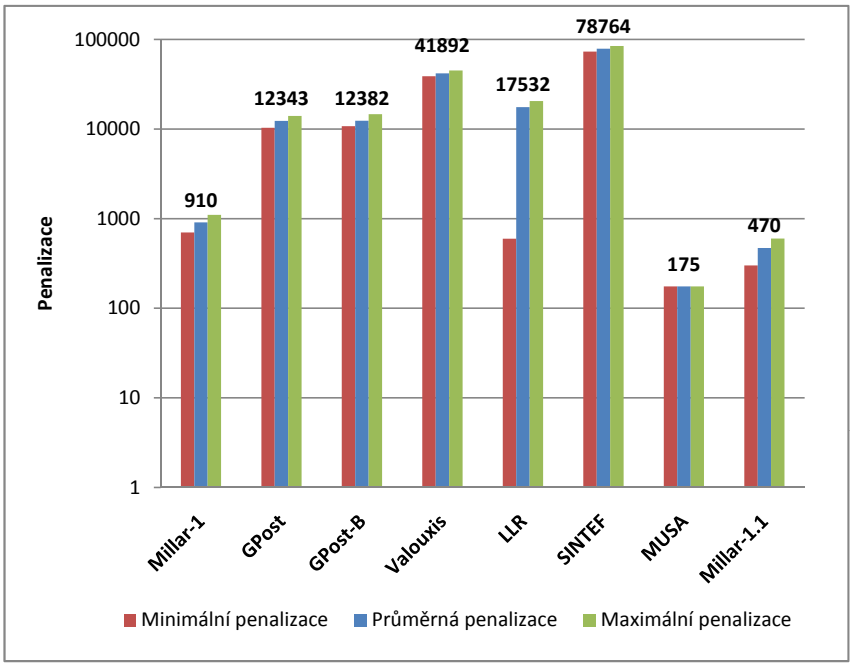
Jedná se o poměrně vysoké číslo, díky kterému trvá stanovený počet spuštěných heuristik (8000) v průběhu algoritmu tabu vyhledávání řádově desítky minut (a více než hodinu u nejnáročnější používané instance - Valouxis). Nicméně snahou bylo získat představu o schopnosti tohoto algoritmu a počet iterací může být kdykoliv snížen, proto pro experimentování byla hodnota počtu spuštěných heuristik zachována. Kvůli vysoké časové náročnosti byl tento algoritmus spuštěn vždy pouze desetkrát. Dosažené výsledky zobrazuje graf z obr. 5.1, na kterém jsou číselně vyznačeny průměrné hodnoty penalizace výsledného rozvrhu.

Z výsledků je patrné, že algoritmus při několikanásobném spuštění generuje rozvrhy s poměrně stabilní penalizací (s výjimkou LLR). Bylo tedy možné předpokládat, že v případě vícenásobného spuštění algoritmu by se výsledky příliš neměnily. V porovnání s nejlepšími dosaženými hodnotami z tabulky 5.2 nedopadly zrovna nejlépe, až na instanci MUSA, kde naopak algoritmus exceloval a vždy dosáhl optimálního řešení.

Instance Millar (verze 1 i 1.1) a MUSA jsou si poměrně podobné, proč se tedy v prvním případě penalizace neblíží k penalizaci nejlepšího zjištěného řešení, a v druhém případě ano? Druhý případ má totiž některé vlastnosti, které s výhodou původní algoritmus využíval.

Za prvé nízký počet typů směn, čímž snižuje i počet možných vzorů směn. Při délce vzoru 7 dní a pouze možnosti přiřazení jednoho typu směny (anebo nepřiřazení směny žádné) existuje pouze 2^7 neboli 128 vzorů směn. Instance Millar ovšem pracuje se dvěma typy směn, tzn. 3^7 (2187) různých kombinací (dvě možné směny anebo volný den).

Za druhé, původní algoritmus počítal s rozvrhovacími obdobími o délce jednoho týdne, čili délka vzoru se rovnala i délce celého rozvrhovacího období. U obou instancí byl sice po-



Obrázek 5.1: Penalizační hodnoty rozvrhu získané aplikací původního algoritmu na testovací instance.

čet rozvrhovaných dní dvojnásobný (14), ale MUSA směřuje měkká omezení na definování požadavků za každý týden zvlášť, čímž se více blíží problému, který řešil původní algoritmus. Experimenty tedy odpovídaly předpokladu o nedostatku obecnosti (viz 4.1.2). Dalším případem podobným k instanci MUSA je LLR, jenž dokáže nalézt i řešení s poměrně nízkou penalizací (naměřená minimální hodnota - 597), ale obvykle bývá o dost vyšší. LLR má sice pouze sedmidenní rozvrhovací období, na druhé straně ovšem pracuje se třemi různými typy směn, kvůli kterým se počet různých vzorů zvýšil na 4^7, čili 16384.

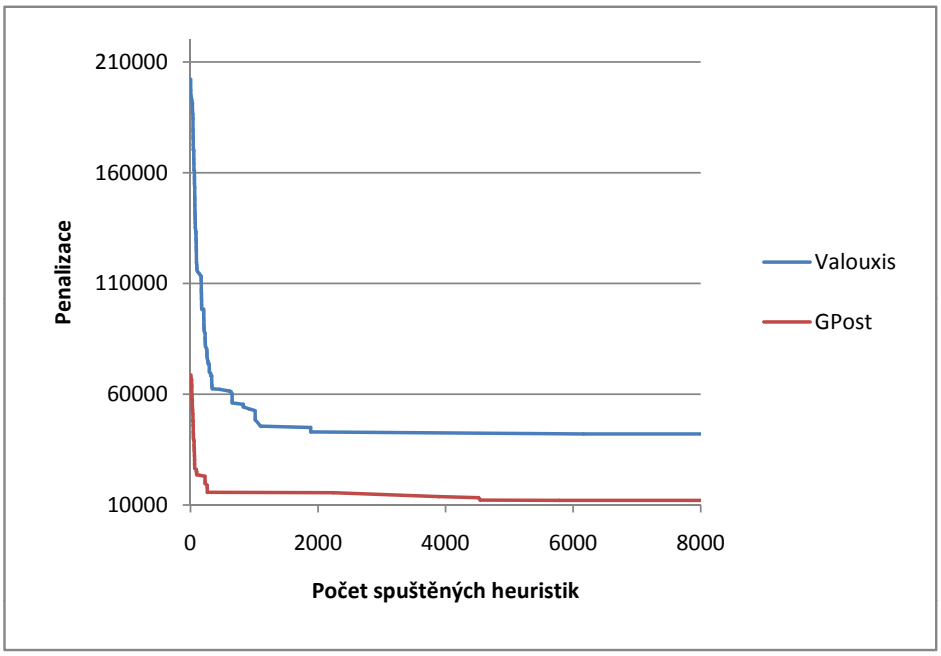
V případě špatně navrženého algoritmu může dojít k trvalému lokálnímu uváznutí velmi brzy a kvůli tomu poté nedochází ke znatelnému vylepšování penalizace ani při jeho spuštění na velmi dlouhou dobu. Příklad průběhu tohoto algoritmu si lze prohlédnout na obr. 5.2.

Z průběhu lze vypozorovat, že sice k největšímu zlepšení dochází pochopitelně na začátku, kdy je penalizace vysoká, nicméně i v pozdější fázi postupně penalizace mírně klesá. Způsob, jakým původní algoritmus řeší problém uváznutí v lokálním extrému, lze tedy prohlásit za funkční.

5.3 Modifikovaný algoritmus

5.3.1 Přípravné experimenty

Změna použitých heuristik byla klíčovým rozdílem modifikovaného algoritmu od původního. Jak již bylo řečeno, heuristiky *mH2* a *mH3* mohou být v některých případech časově velmi náročné. Z tohoto důvodu musí být u heuristiky *mH2* implementováno omezení na maximální počet zaměstnanců, mezi nimiž budou probíhat záměny směn. Z tabulky 5.3 je možné



Obrázek 5.2: Průběh nejlepších dosažených hodnot penalizace rozvrhu v čase získaných při aplikaci původního algoritmu.

vyčíst doby běhu této heuristiky při uvedeném omezení. Test proběhl na instanci Valouxis, která patří mezi časově nejnáročnější, za použití výpočetního stroje *P1*. Výsledky jsou pouze orientační, protože tyto heuristiky nemají (kvůli optimalizaci) stabilní dobu výpočtu.

Počet vybraných zaměstnanců	**Průměrná doba**
2	0,02 s
4	0,22 s
6	1,24 s
8	3,86 s
10	10,22 s

Tabulka 5.3: Doba běhu heuristiky *mH2* v závislosti na počtu vybraných zaměstnanců na instanci Valouxis.

Každé spuštění heuristiky by mělo ideálně trvat maximálně 2-3 vteřiny, protože lze předpokládat, že pro dosažení rozvrhu akceptovatelné kvality je zapotřebí aplikovat stovky heuristik a celý průběh algoritmu by měl trvat maximálně několik desítek minut. Z tohoto hlediska se jeví jako ideální zvolit pouze šest zaměstnanců, nicméně tento počet je velice nízký (např. Valouxis se zabývá rozvrhováním 16 zaměstnanců), a proto byla tato hranice posunuta alespoň na hodnotu osm.

U heuristiky *mH3* dochází k ještě obtížnější situaci, jelikož složitost rychle roste nejen s počtem zaměstnanců, ale i s počtem dnů. Proto zde dochází k omezení obou těchto parametrů. Časové experimenty heuristiky *mH3* zobrazuje tabulka 5.4.

Počet vybraných zaměstnanců	**Průměrná doba**		
	8 dní	**10 dní**	**12 dní**
2	0,07 s	0,09 s	0,1 s
4	0,31 s	0,43 s	0,56 s
6	0,7 s	1,07 s	1,52 s
8	1,41 s	2,25 s	3,2 s
10	3,15 s	4,59 s	5,72 s

Tabulka 5.4: Doba běhu heuristiky *mH3* v závislosti na počtu vybraných zaměstnanců a dní na instanci Valouxis

.

Časové požadavky na heuristiku *mH3* jsou zhruba stejné jako u *mH2*, proto byla, s ohledem na tabulku 5.4, zvolena kombinace výběru deseti dní a osmi zaměstnanců. Při tomto nastavení parametrů by výpočet heuristiky *mH3* na stroji *P1* měl v průměru trvat zhruba něco málo přes 2 vteřiny.

Heuristika *mH4* se svým chováním velmi odlišuje od *mH1-mH3*. Jejím účelem je únik z lokálního extrému pomocí systematického odstranění problematických směn, zpřeházení rozvrhu a následovného doplnění chybějících směn. Proto není vhodné nechávat její řízení pouze na metodě tabu vyhledávání. Spuštění této heuristiky je tedy povoleno pouze v případě, že ani po N_h spuštění libovolných heuristik nedojde ke změně rozvrhu (tím pádem zůstane i penalizace stejná). Nelze jednoduše určit, jaké hodnoty by měla konstanta N_h nabývat, a tak byly experimenty spouštěny při $N_h = 4$, $N_h = 6$ a $N_h = 8$. Při zvolení $N_h < 4$ lze totiž předpokládat, že dojde ke zbytečně častému spouštění této heuristiky, na druhé straně, pokud $N_h > 8$, může docházet ke zbytečně dlouhému setrvání v lokálním extrému.

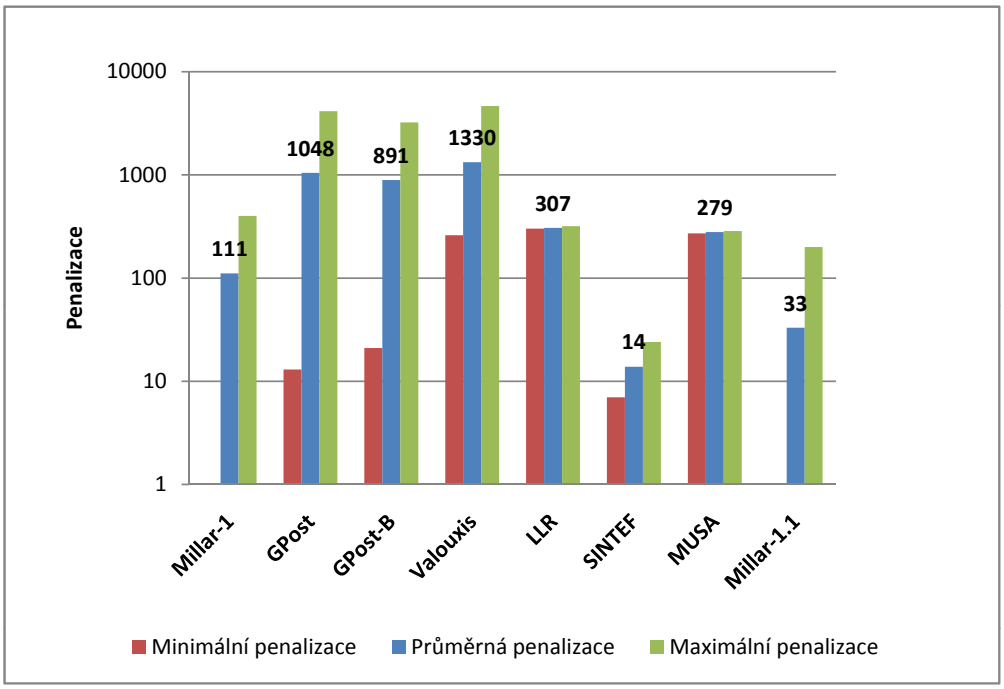
Jak již bylo v této práci řečeno, *mH5* nepatří mezi klasické heuristiky a není vhodné ji řídit pomocí algoritmu tabu vyhledávání, který řídí spouštění heuristik dle jejich předchozí úspěšnosti. Heuristika *mH5* má fungovat jako alternativa k *mH4*, což znamená, že je žádoucí ji používat pouze ve chvílích, kdy dochází k uváznutí v lokálním extrému. Její spuštění bylo omezeno podobně jako v předchozím případě (u *mH4*). Dokud dochází ke zlepšování rozvrhu a nedojde k aplikaci N_h neúspěšných heuristik, nebude nikdy spuštěna. Po této době se spustí s pravděpodobností 50%, tím pádem po delší době uváznutí v lokálním extrému je šance na její spuštění jak u heuristiky *mH4*, tak i *mH5*.

5.3.2 Hlavní experimenty

Ukončovací podmínkou pro modifikovaný algoritmus není stanovený počet spuštěných heuristik, nýbrž počet částečných restartů (realizovaných heuristikou *mH4*), po kterých zůstane rozvrh s nejnižší dosaženou penalizací nezměněn. Pevně stanovený počet iterací totiž u této metody může způsobovat ukončení algoritmu v jeho různých fázích - např. ve fázi těsně po restartu, kdy je penalizace vysoká, nebo naopak ve fázi, kdy se blíží k nalezení rozvrhu s nízkou penalizací. To by mohlo znamenat velké výkyvy v penalizaci výsledného rozvrhu.

Otázkou zůstává, jak vhodně nastavit ukončovací podmínku, neboli po kolika „neúspěšných restartech rozvrhu“ ukončit algoritmus. Po provedení několika testů byla hodnota na-

stavena na 5. Pokud bychom chtěli, aby algoritmus v průměru trval kratší dobu (a tím pravděpodobně dosahoval horších průměrných výsledků), stačí tuto hodnotu snížit. Její zvýšení by poté mělo mít přesně opačný efekt. Výsledky experimentů s různou konstantou N_h zobrazují obrázky 5.3., 5.4 a 5.5. Číselně označené jsou opět průměrné hodnoty. Každý test byl spuštěn 200 krát. Ideální by bylo spustit test řádově vícekrát, z časových důvodů to ovšem není možné.



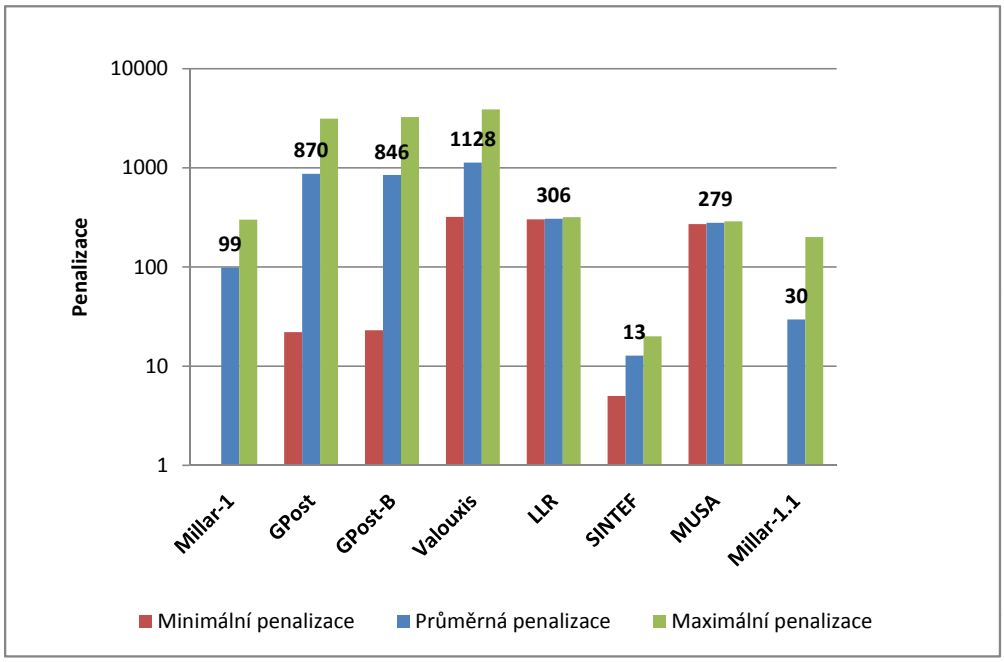
Obrázek 5.3: Penalizační hodnoty získané aplikací modifikovaného algoritmu při $N_h = 4$.

Modifikovaný algoritmus vykazuje dobré výsledky. I u nejnáročnější instance (Valouxis) dostáváme rozvrh s poměrně slušnou průměrnou penalizací (1000 - 1330) a některé nejnižší hodnoty se blíží nejlepším známým dosaženým hodnotám (např. GPost s penalizací 13, viz obr. 5.3).

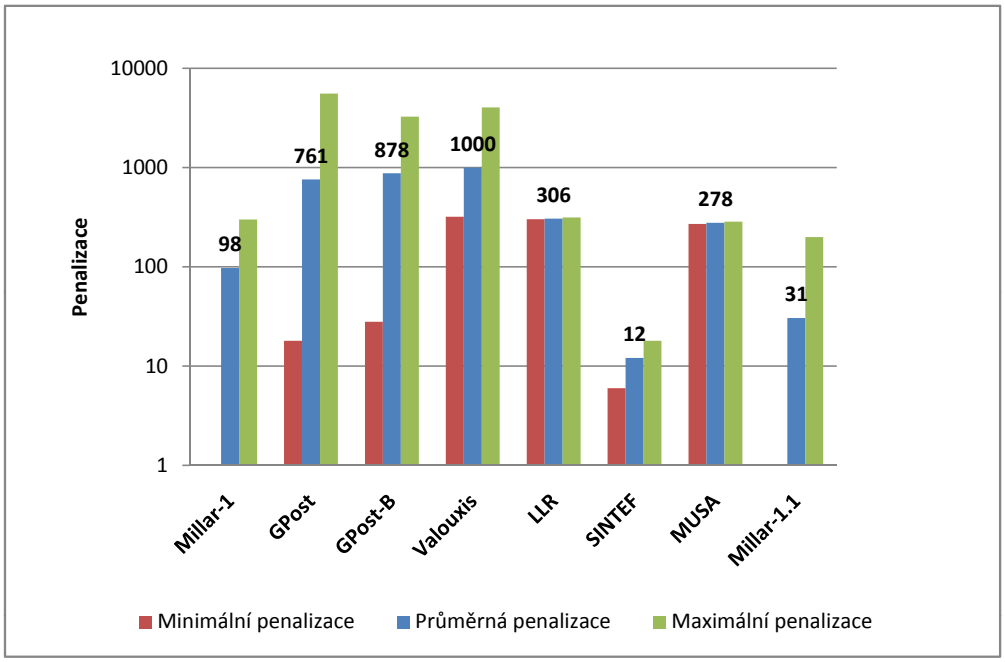
Ze získaných dat lze vypozorovat, že se zvyšováním konstanty N_h dochází ke snižování průměrné penalizace. Tím ovšem dochází i ke zvyšování počtu spuštěných heuristik, tím pádem se dá předpokládat i zvýšení času potřebného pro jeho spuštění. Údaje o průměrném počtu spuštěných heuristik za daného nastavení konstanty N_h nabízí k prohlédnutí tabulka 5.5.

Z údajů, jež jsou uvedeny v tabulce 5.5, vyplývá, že rozdíl mezi průměrnou penalizací rozvrhů při nastavení $N_h = 4$ a $N_h = 6$ je poměrně velký, zatímco mezi $N_h = 6$ a $N_h = 8$ je podstatně menší, počet spuštěných heuristik ale stoupá zhruba stejným tempem. Z tohoto důvodu se jeví jako nejvhodnější varianta $N_h = 6$.

Modifikovaný algoritmus generuje rozvrhy s akceptovatelnou průměrnou penalizací, nicméně na některých vstupních instancích s poměrně velkým rozdílem mezi minimální a maximální



Obrázek 5.4: Penalizační hodnoty získané aplikací modifikovaného algoritmu při $N_h = 6$.



Obrázek 5.5: Penalizační hodnoty získané aplikací modifikovaného algoritmu při $N_h = 8$.

instance	Průměrná penalizace			Medián penalizace			Průměrný počet heuristik		
	$N_h = 4$	$N_h = 6$	$N_h = 8$	$N_h = 4$	$N_h = 6$	$N_h = 8$	$N_h = 4$	$N_h = 6$	$N_h = 8$
Millar-1	111	99	98	100	100	100	154	176	198
GPost	1048	870	761	1073	661	548	459	569	678
GPost-B	891	846	878	625	581	574	494	600	660
Valouxis	1330	1128	1000	1390	760	690	564	639	686
LLR	307	306	306	306	306	306	389	423	492
SINTEF	14	13	12	14	12	12	606	676	755
MUSA	279	279	278	280	279	278	232	270	317
Millar-1.1	33	30	31	0	0	0	104	111	126

Tabulka 5.5: Průměrná penalizace rozvrhů a její medián na různých instancích a průměrný počet spuštěných heuristik pro dosažení této penalizace

.

hodnotou. Vzhledem k tomu stojí za zmínku medián penalizace vygenerovaných rozvrhů, který je taktéž uveden v tabulce 5.5.

Hodnoty mediánu jen potvrzují předchozí doporučené nastavení N_h na hodnotu 6. Dále se nezdá být vhodná volba $N_h = 4$, jelikož například u instance Valouxis či GPost znamená tato volba znatelně vyšší hodnotu mediánu než při zvolení vyššího N_h.

5.4 Modifikovaný algoritmus s využitím neuronové sítě

5.4.1 Přípravné experimenty

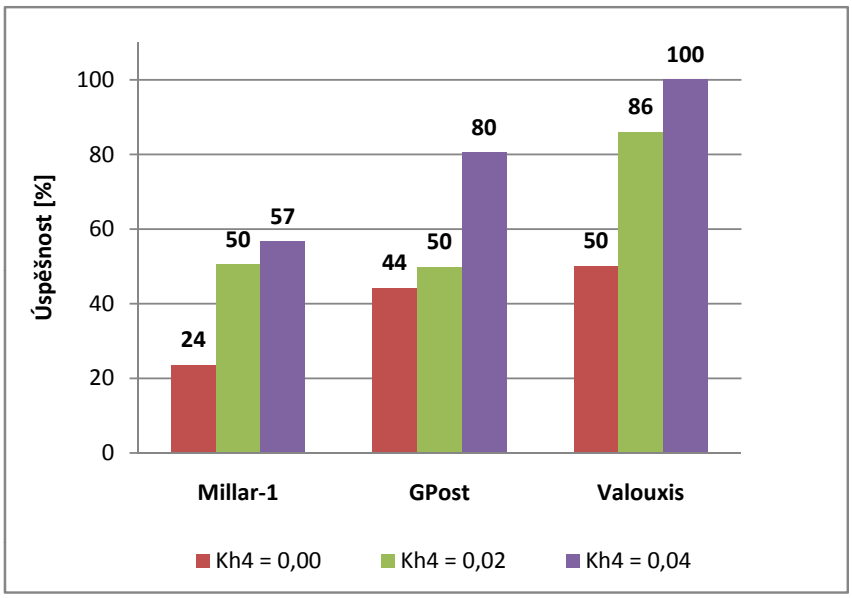
Metoda tabu vyhledávání i neuronová síť používané v této práci ohodnocují úspěšnost heuristik *mH1-mH2* pouze pomocí rozdílu penalizace mezi dvěma stavy (před a po aplikaci heuristiky). V době, kdy nedochází k problému najít pomocí heuristik rozvrh s nižší penalizací (typicky např. po částečném restartu pomocí heuristiky *mH4*), proto nedokáže metoda tabu vyhledávání ani neuronová síť rozhodovat, aplikací které heuristiky se dosáhne kvalitnějšího rozvrhu z dlouhodobého hlediska. I přesto mohou mít různě zvolené sekvence heuristik v této fázi algoritmu velký vliv na konečnou penalizaci vzniklého rozvrhu. Nevědomě totiž mohou heuristiky směřovat do oblasti stavového prostoru, kde se nenachází uspokojivé řešení, a uniknout z této oblasti může být problémové. Aby bylo možné co nejlépe výsledky při řízení metodou tabu vyhledávání srovnávat s výsledky při použití neuronové sítí, byla neuronová síť aplikována pouze v případě, kdy nedošlo ke zlepšení rozvrhu ani po aplikaci dvou heuristik. Jedná se tedy o kombinaci řízení pomocí tabu vyhledávání a neuronové sítě.

Při generování testovacích dat byly nastaveny následující podmínky, které by měly zaručit kompromis mezi délkou trvání generování těchto dat a jejich kvalitou:

- z každého stavu rozvrhu jsou heuristiky spuštěny třikrát, z jejich výsledků se vypočítá průměrná úspěšnost (dosáhnutí nižší penalizace);
- testování úspěšnosti heuristky *mH4* je povoleno po době, kdy během čtyř po sobě aplikovaných heuristikách nedošlo ke zlepšení penalizace rozvrhu.

Detailní popis těchto dvou bodů se nachází v kapitole 4.3.2.2. V této kapitole dále byla zmíněna nutnost nastavení parametru K_{mh4} ze vzorce 4.11. Jedná se o parametr, který

určuje, o kolik může být penalizace rozvrhu po aplikaci heuristiky *mH4* a následné sekvenci dalších heuristik vyšší (opět viz 4.3.2.2) než před touto aplikací, aby byla aplikace *mH4* stále označena za úspěšnou. Pro otestování vlivu nastavení tohoto koeficientu byly zvoleny testovací instance Millar, GPost a Valouxis. Výsledky jsou patrné z grafu na obr. 5.6.



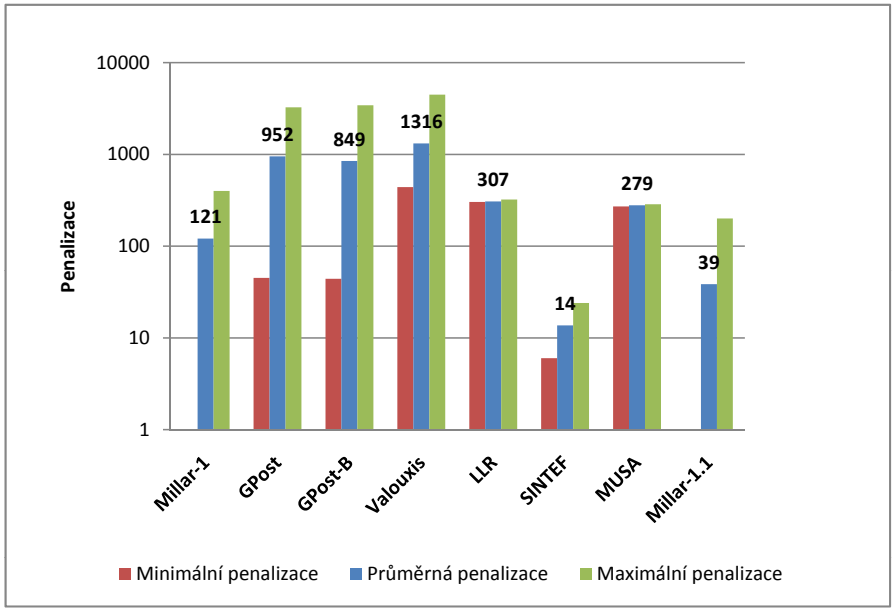
Obrázek 5.6: Procentuální úspěšnost heuristiky *mH4* při různě zadaném koeficientu K_{mh4}.

Hodnoty získané z tohoto experimentu nám však nedávají dostatečnou informaci o tom, jak často by měla být heuristika *mH4* označena za úspěšnou. Tato informace byla získána z pozorování chování běhu algoritmu. Při generování je vhodné označit heuristiku *mH4* za úspěšnou takovým způsobem, aby následovným řízením spouštění heuristik nedocházelo k příliš časté aplikaci *mH4* a naopak aby nedocházelo ke zbytečnému setrvávání v lokálním extrému. Při pozorování běhu algoritmu za použití neuronové sítě s různými učicími daty (s daty generovanými při různých hodnotách koeficientu K_{mh4}) došlo k získání názoru, že aplikace heuristiky *mH4* by u těchto testovacích dat měla být označena za úspěšnou zhruba v 50% případů. Tím pádem se jeví, s ohledem na data získaná experimenty, jako nejvhodnější zvolit hodnotu parametru $K_{mh4} = 0,02$.

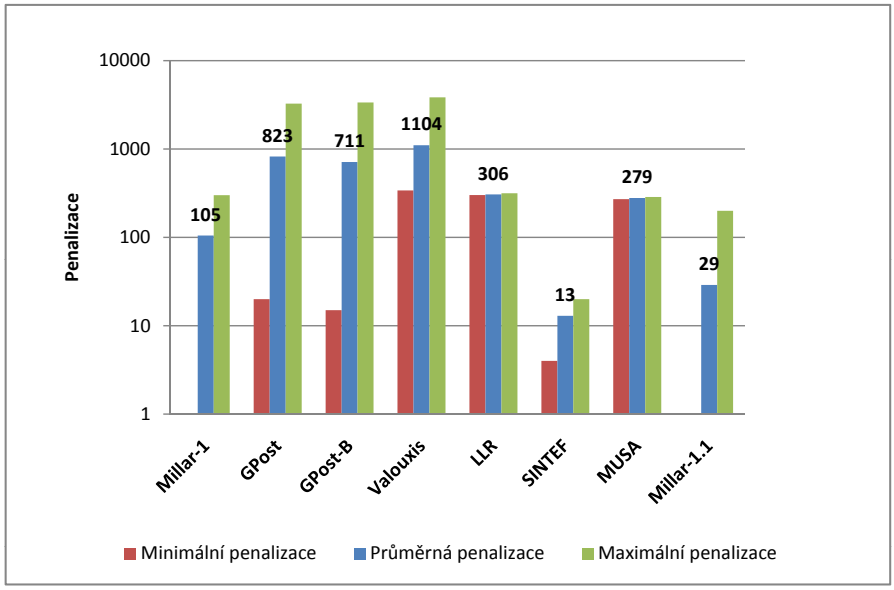
Zatím v této práci nebylo zodpovězeno, jak dlouho probíhalo učení neuronové sítě, neboli kolik epoch (tzn. kolikrát neuronové síti předložit množinu učicích vzorků) k učení zvolit. Pro stanovení počtu epoch byl použit program *JNNS*, který umožňuje při poskytnutí učicích a testovacích dat určit jejich vhodnou hodnotu. Po otestování několika instancí se ukázalo jako vhodné nastavit počet epoch na hodnotu 30 při zhruba 1500 vzorků v množině dat. Proto se neuronová síť učí po dobu 45000/*pocet_testovacich_vzorku* epoch.

5.4.2 Hlavní experimenty

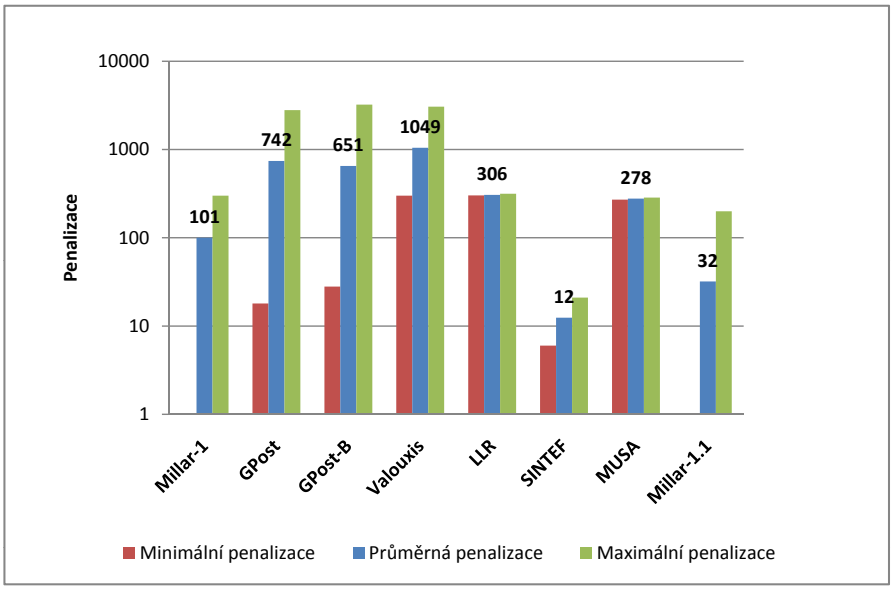
Experimenty u modifikovaného algoritmu s řízením heuristik s pomocí neuronové sítě probíhaly stejně jako v předchozí verzi algoritmu (bez neuronové sítě). Výsledky experimentů zobrazují obr. 5.7,5.8 a 5.9.



Obrázek 5.7: Penalizační hodnoty získané aplikací modifikovaného algoritmu s využitím neuronové sítě při $N_h = 4$



Obrázek 5.8: Penalizační hodnoty získané aplikací modifikovaného algoritmu s využitím neuronové sítě při $N_h = 6$.



Obrázek 5.9: Penalizační hodnoty získané aplikací modifikovaného algoritmu s využitím neuronové sítě při $N_h = 8$.

Výsledné průměrné hodnoty penalizace jsou velice podobné hodnotám získaným z experimentů bez použití neuronové sítě. Proto nebudou výsledky diskutovány přímo, ale až ve srovnání s ostatními algoritmy v kapitole 5.5.

5.5 Srovnání algoritmů

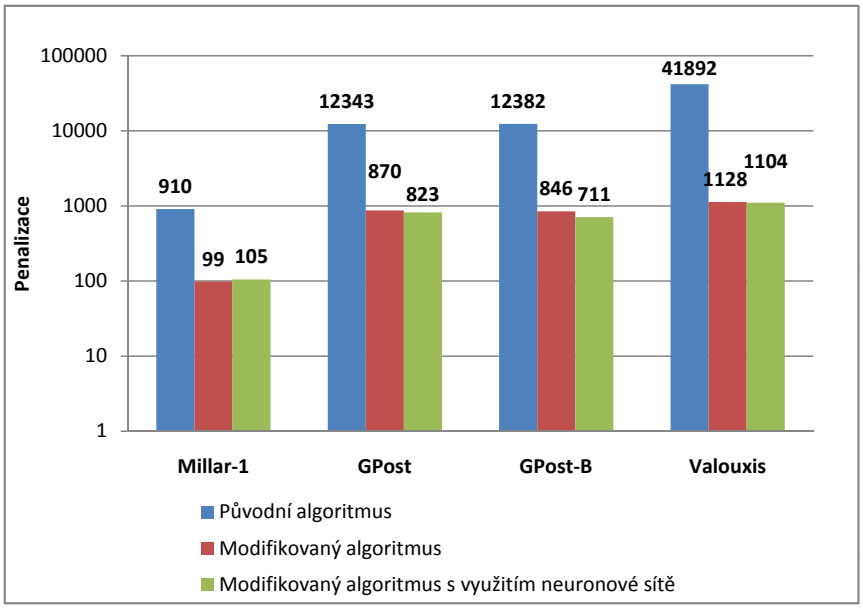
Práce se nejprve zabývá srovnáváním všech tří algoritmů (původní algoritmus s úpravami pro obecné použití, modifikovaný algoritmus a modifikovaný algoritmus s použitím neuronové sítě). Další část práce se poté zaměří na detailní srovnání modifikovaného algoritmu bez a s využitím neuronové sítě.

5.5.1 Srovnání všech naimplementovaných algoritmů

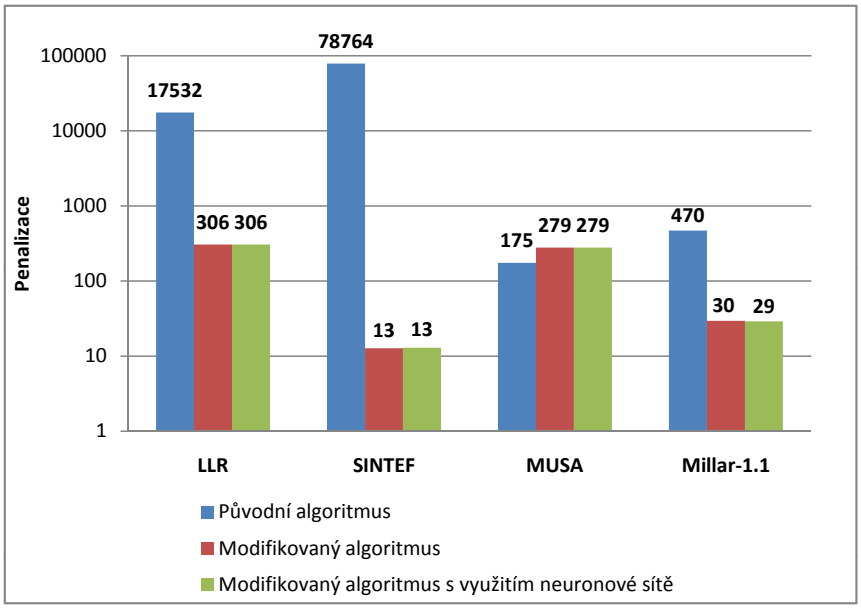
Pro srovnání výsledků všech tří algoritmů byly vybrány následující hodnoty:

- průměrné hodnoty penalizace původního algoritmu;
- průměrné hodnoty penalizace modifikovaného algoritmu při nastavení $N_h = 6$;
- průměrné hodnoty penalizace modifikovaného algoritmu s využitím neuronové sítě při nastavení $N_h = 6$.

Srovnání všech tří algoritmu zobrazují obr. 5.10 a 5.11.



Obrázek 5.10: Srovnání hodnot penalizací rozvrhů získaných původním algoritmem, modifikovaným a modifikovaným s využitím neuronové sítě, část 1.



Obrázek 5.11: Srovnání hodnot penalizací rozvrhů získaných původním algoritmem, modifikovaným a modifikovaným s využitím neuronové sítě, část 2.

Z výsledných průměrných penalizací je zřejmé, že modifikovaný algoritmus, ať již ve verzi bez anebo s použitím neuronové sítě, vykazuje na všech instancích mimo MUSA rozvrhy s penalizací dokonce řádově nižší než u původního algoritmu. A to navíc za kratší dobu - např. u instance Valouxis, která je časově nejnáročnější testovanou instancí, trvala v průměru doba běhu původního algoritmu přes hodinu, zatímco u obou modifikovaných verzí zhruba 20 minut. I v případě jednoduchých instancí dosahují modifikované verze lepších časových výsledků. Zatímco původní algoritmus u instance Millar1.1 potřebuje zpravidla více než deset minut, modifikované verze nikdy nepřekročily v průměru 5 minut. I pokud vezmeme v potaz poměrně značnou nepřesnost měření z důvodu spouštění na různých strojích, je zřejmé, že modifikovaný algoritmus časově i kvalitou rozvrhování předčí původní algoritmus na všech instancích s výjimkou MUSA.

Úspěšnost původního algoritmu na instanci MUSA byla již diskutována v předchozí kapitole 5.3.2. Oba modifikované algoritmy se jen stěží mohou blížit k ideálnímu. Optimálního stavu (rozvrhu s penalizací 175) ani nemohou dosáhnout, poněvadž nejsou schopny přidávat či odebírat směny. Bohužel u instance MUSA je nutné počítat s touto možností. Optimální rozvrh získává totiž penalizaci výhradně za nedodržení preferovaného pokrytí směn. Pro vyřešení těchto případů by bylo zapotřebí přidat další heuristiku, schopnou odstraňovat a přidávat další směny. Při této modifikaci je ovšem nutné brát v potaz, že může dojít k velkému nárůstu výpočetní náročnosti algoritmu. V reálném prostředí ovšem pravděpodobně nebudeme mít informaci o tom, zdali existuje optimální řešení [2] při požadovaném pokrytí všech směn, tím pádem ani nebude jisté, zda je heuristika zapotřebí a zda její zavedení nebude mít spíše negativní vliv na konečný výsledek.

5.5.2 Srovnání modifikovaného algoritmu bez a s využitím neuronové sítě

Rozdíly v penalizaci rozvrhů generovaných pomocí modifikovaného algoritmu bez a s použitím neuronové sítě jsou poměrně malé, proto jsou hodnoty z důvodu lepší přehlednosti zobrazeny v tabulce 5.6.

Tabulka 5.6 zobrazuje průměrné penalizace rozvrhů generované sledovanými algoritmy. Průměrné penalizace jsou u obou algoritmů poměrně vyvážené. V některých případech poskytuje penalizačně lepší rozvrhy algoritmus používající neuronovou síť, v některých je situace opačná, jindy jsou srovnatelné.

Ukončovací podmínkou algoritmu ovšem není, tak jako u původního algoritmu, určitý počet spuštěných heuristik, nýbrž počet neúspěšných restartů pomocí heuristiky *mH4*. Proto je taktéž důležité srovnávat počet heuristik potřebných k získání daného rozvrhu. Srovnání na základě počtu spuštěných heuristik *mH1-mH4* je uvedeno v tabulce 5.7.

Jak je vidět, algoritmus používající neuronovou síť ve většině případů spouští méně heuristik než algoritmus bez ní. Z hodnot z tabulky 5.7 však není přímo možné poznat, jestli rozdíl v počtu spuštěných heuristik je adekvátní k rozdílu penalizace. To lze zjistit pomocí následujícího vzorce:

$$PP_{inst-alg} = \frac{PH_{inst-alg}/PH_{inst-ref}}{HC_{inst-ref}/HC_{inst-alg}} \tag{5.1}$$

[2] Optimální řešení nemusí nutně znamenat nulovou penalizaci rozvrhu.

Instance	**Modifikovaný algoritmus**			**Modifikovaný algoritmus s neuronovou sítí**		
	$N_h = \mathbf{4}$	$N_h = \mathbf{6}$	$N_h = \mathbf{8}$	$N_h = \mathbf{4}$	$N_h = \mathbf{6}$	$N_h = \mathbf{8}$
Millar-1	111	99	98	121	105	101
GPost	1048	870	761	952	823	742
GPost-B	891	846	878	849	711	651
Valouxis	1330	1128	1000	1316	1104	1049
LLR	307	306	306	307	306	306
SINTEF	14	13	12	14	13	12
MUSA	279	279	278	279	279	278
Millar-1.1	33	30	31	39	29	32

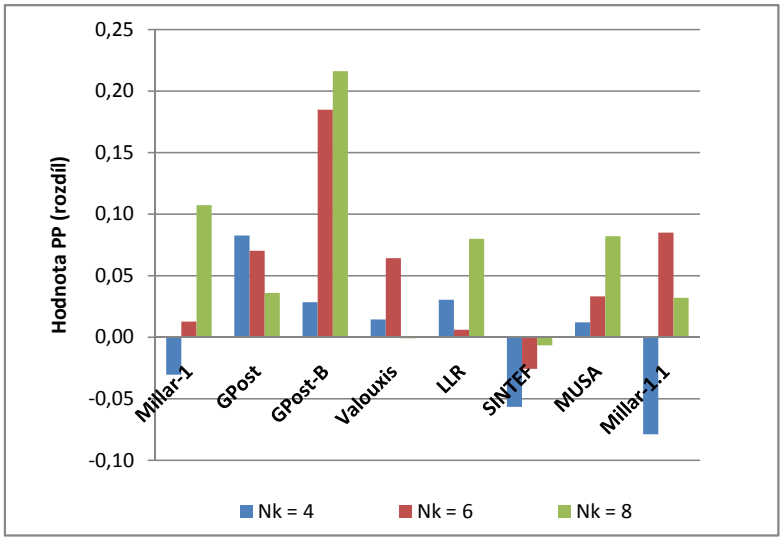
Tabulka 5.6: Průměrné penalizace rozvrhů na různých instancích vygenerovaných modifikovaným algoritmem a modifikovaným algoritmem při použití neuronové sítě

.

Instance	**Modifikovaný algoritmus**			**Modifikovaný algoritmus s neuronovou sítí**		
	$N_h = \mathbf{4}$	$N_h = \mathbf{6}$	$N_h = \mathbf{8}$	$N_h = \mathbf{4}$	$N_h = \mathbf{6}$	$N_h = \mathbf{8}$
Millar-1	154	176	198	146	162	171
GPost	459	569	678	461	557	670
GPost-B	494	600	660	499	562	698
Valouxis	564	639	686	563	613	655
LLR	389	423	492	374	421	452
SINTEF	606	676	755	651	683	741
MUSA	232	270	317	228	259	291
Millar-1.1	104	111	126	97	101	116

Tabulka 5.7: Průměrný počet spuštěných heuristik při průběhu modifikovaného algoritmu a modifikovaného algoritmu s použitím neuronové sítě

Ve vzorci 5.1 značí PH penalizační hodnotu, HC počet spuštěných heuristik, $inst$ zvolenou instanci, alg zvolený algoritmus a ref značí referenční algoritmus. Jako referenční je zvolen modifikovaný algoritmus, který nepoužívá neuronovou síť s $N_h = 8$. Hodnota $PP_{inst-alg}$ (poměrná penalizace) tedy značí poměrnou úspěšnost algoritmu vzhledem k počtu spuštěných heuristik vztaženou k referenčnímu algoritmu. Čím nižší hodnotu daný algoritmus má, tím je tedy efektivnější. Srovnání obou algoritmů pomocí vypočteného $PP_{inst-alg}$ lze nalézt na grafu z obr. 5.12.



Obrázek 5.12: Srovnání hodnot průměrných penalizací rozvrhů vzhledem k počtu spuštěných heuristik v průběhu algoritmu.

Záporná hodnota na grafu znamená, že modifikovaný algoritmus má lepší poměr počtu spuštěných heuristik k průměrné penalizaci generovaných rozvrhů než modifikovaný algoritmus používající neuronovou síť, kladná hodnota značí opak. Dle těchto experimentů se ukazuje jako lepší varianta algoritmus používající neuronovou síť ve většině případů, s výjimkou instance SINTEF, kde je situace opačná. Penalizace se ovšem příliš neliší, průměrné hodnoty penalizace rozvrhů (u instance SINTEF) jsou v rozsahu 12 až 14.

Průměr penalizace má zajisté vysokou vypovídací hodnotu, ovšem v případě spouštění jednotlivých testů by mohlo být více zajímavé dozvědět se, jaké hodnoty získáme s pravděpodobností alespoň 50%. K tomu nám slouží medián, který byl v této práci již použit. Medián získaných výsledků udává tabulka 5.8.

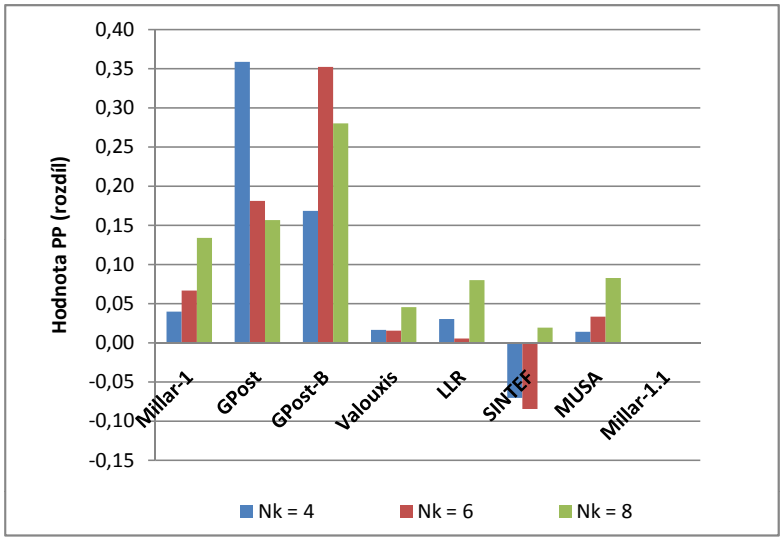
Získané hodnoty naznačují, že došlo k prohloubení rozdílů mezi oběma srovnávanými algoritmy. Pro přehlednost je opět uveden graf s rozdílem hodnot $PP_{inst-alg}$, ale se záměnou průměrných hodnot za medián (viz obr. 5.13).

Graf z obr. 5.13 potvrzuje předchozí slova o prohloubení rozdílů. Navíc v některých případech došlo k přehodnocení situace - např. u instance Valouxis při $N_k = 8$ došlo k přechodu z neutrální nulové hodnoty na kladnou hodnotu (z hlediska neuronové sítě).

I přesto existují instance, kde je stále lepší modifikovaný algoritmus nepoužívající neuronovou síť. V práci bylo již zmíněno, že neuronová síť určuje následující krok pouze tehdy,

Instance	Modifikovaný algoritmus			Modifikovaný algoritmus s neuronovou sítí		
	$N_h = 4$	$N_h = 6$	$N_h = 8$	$N_h = 4$	$N_h = 6$	$N_h = 8$
Millar-1	100	100	100	100	100	100
GPost	1073	661	548	780	554	467
GPost-B	625	581	574	491	382	391
Valouxis	1390	760	690	1380	780	690
LLR	306	306	306	306	306	306
SINTEF	14	12	12	14	13	12
MUSA	280	279	278	279	279	278
Millar-1.1	0	0	0	0	0	0

Tabulka 5.8: Medián penalizace rozvrhů na různých instancích vygenerovaných modifikovaným algoritmem a modifikovaným algoritmem při použití neuronové sítě.



Obrázek 5.13: Srovnání mediánů penalizací rozvrhů vzhledem k počtu spuštěných heuristik během průběhu algoritmu.

pokud došlo k zastavení snižování penalizace rozvrhu. V těchto fázích je totiž nejdůležitější rozhodnout, která heuristika má být spuštěna. I přesto, že algoritmy byly upraveny tak, aby bylo umožněno co nejlépe srovnat jejich schopnost nalézt lepší řešení v dobách, kdy se rozvrh nachází právě v těchto fázích, nebylo možné zaručit totožné podmínky.

Chování modifikovaného algoritmu při využití neuronové sítě ve fázi, kdy řízení obsluhuje metoda tabu vyhledávání, je podobné, ale nikoliv stejné. Řízení metodou tabu vyhledávání je totiž samozřejmě ovlivňováno aktuálním stavem tabu listu a příslušným ohodnocením heuristik (jejich přiřazenými hodnotami *rank*). Při přepnutí řízení na neuronovou síť ovšem dochází i k odlišnému chování při evaluaci heuristik pro pozdější fáze, kdy dojde k opětovnému návratu k řízení metodou tabu vyhledávání. Metoda tabu vyhledávání řídí typicky rozvrh v době např. po aplikování heuristiky *mH4*, kdy zvolené heuristiky ovlivňují, do jaké části stavového prostoru řešení se algoritmus dostane. Tím může být výsledná penalizace ovlivněna jak negativně, tak i pozitivně, protože k řešení klíčových situací (kdy není jednoduché najít rozvrh s lepší penalizací) je přistupováno z různého stavového prostoru řešení.

Pro názornost, jak může stav, ze kterého algoritmus vychází, ovlivňovat výslednou penalizaci rozvrhu, byl proveden test na instanci GPost se dvěma různými počátečními řešeními. Počáteční řešení je generováno pseudonáhodně, vždy se volí nějaký tzv. *seed*, který určuje posloupnost generovaných čísel. Pokud nastavíme *seed* stejně, získáme i stejně zinicializovaný rozvrh. Z dosavadních testů byla vybraná jedna taková *seed* hodnota, která nepřinášela příliš pozitivní výsledky, a druhá s opačným efektem. V tabulce 5.9 jsou uvedeny hodnoty získané průchodem modifikovaného algoritmu bez použití neuronové sítě při dvou rozdílných hodnotách *seed*. Při každém nastavení bylo spuštěno 40 testů.

	seed = 150	**seed = 115**
Průměrná penalizace	547	1104
Medián penalizace	336	1159

Tabulka 5.9: Rozdíly u generovaných výsledků při různých inicializačních rozvrzích.

Není pochyb o tom, že stav, ze kterého algoritmus vychází, má vliv na jeho výsledky. Pokud vezmeme dále v úvahu, že například u instance Valouxis v případě modifikovaného algoritmu s využitím neuronové sítě řídí neuronová síť pouze zhruba necelých 20% všech kroků, dají se pochopit případy, kdy má větší vliv právě zmiňovaný původní stav než změna v řízení způsobená aplikací neuronové sítě.

Pro zlepšení této situace se nabízí možnost změny chování neuronové sítě tak, aby reagovala nejen v rovině toho, jestli dojde ke zlepšení penalizace či nikoliv, ale i v rovině velikosti zlepšení penalizace. Zde ovšem musíme brát v úvahu, že v takovém případě může snáze dojít k uváznutí v lokálním extrému.

Další možností je nebrat v úvahu pouze následující krok, ale uvažovat i nad dlouhodobějším vývojem, například pomocí technologie vytěžování dat (data mining). Při vygenerování velkého množství průběhů algoritmu by mělo být možné určit kroky, které jsou z dlouhodobého hlediska pravděpodobně prospěšné, a které nikoliv.

Hodnoty uvedené v tabulce 5.9 navíc naznačují, že částečný restart pomocí heuristiky *mH4* nemusí pracovat ideálně. Velký rozdíl v naměřených hodnotách může způsobovat i nevhodné částečné restartování touto heuristikou. Hlavní princip heuristiky *mH4* (odstraňování

problémových směn) nemusí vždy znamenat úspěšné vyřešení lokálního uváznutí, protože jde o částečně deterministickou heuristiku, čímž hrozí opakované prohledávání stejného stavového prostoru řešení. Řešením by mohlo být zavedení další heuristiky s funkcí částečného restartu, která by pracovala více na bázi náhodnosti, a v případě neúspěchu aplikace heuristiky *mH4* aplikovat tuto.

Kapitola 6

Závěr

Cílem této práce bylo vybrat jedno z již existujících řešení problému rozvrhování směn zdravotních sester vhodné pro aplikaci na různé typy instancí a navrhnout vlastní algoritmus, který bude porovnán s vybraným řešením.

Z velké škály existujících algoritmů byla vybrána jedna konkrétní implementace hyperheuristiky řízené pomocí metody tabu vyhledávání. Ačkoliv mezi hlavní výhody hyperheuristik patří možnost využití na širokém spektru instancí, po naimplementování a provedení experimentů se ukázalo, že zvolené nízkoúrovňové heuristiky byly příliš šité na míru konkrétnímu problému. Penalizace rozvrhů vzniklých aplikací testovacích instancí na tento algoritmus se převážně ani neblížila hodnotám, které by se daly považovat za uspokojivé. S výjimkou jediné instance (MUSA), jež se svou strukturou blíží té, s níž původní algoritmus pracoval. Na této instanci algoritmus naopak dosahoval excelentních výsledků.

Vzhledem k nedostatkům původního algoritmu došlo k jeho modifikaci. Žádna z původních nízkoúrovňových heuristik nebyla zachována, místo nich byly navrženy nové, pracující na bázi záměny směn mezi dvěma zaměstnanci a co možná nejméně se opírající o metadata vstupní instance. V další fázi poté bylo řízení nízkoúrovňových heuristik částečně nahrazeno klasifikací pomocí neuronové sítě pro otestování vlivu využití umělé inteligence na kvalitu výsledných rozvrhů.

Experimenty prokázaly, že modifikovaný algoritmus, ať již s použitím neuronové sítě, nebo bez ní, dokáže generovat rozvrhy s lepší penalizací než algoritmus původní, s výjimkou již zmiňované instance MUSA. Při částečném použití neuronové sítě při řízení nízkoúrovňových heuristik dochází celkově ke zlepšení z hlediska snížení potřebného počtu spuštěných heuristik pro vygenerování rozvrhu, čímž se potvrzuje, že neuronová síť dokáže lépe předpovídat, jaká heuristika s největší pravděpodobností dokáže zlepšit aktuální rozvrh. V porovnání se standardním řízením pomocí tabu vyhledávání ovšem rozdíly nejsou příliš velké.

Modifikovaný algoritmus nabízí široké možnosti k jeho rozšíření. Jednou z nich je naimplementování další heuristiky, která by umožňovala odebrání nebo přidání směny. Tím by mohlo dojít ke zlepšení výsledků u instancí podobných instanci MUSA. Z pohledu neuronové sítě by bylo vhodné klasifikovat kroky nejen z krátkodobého, ale i dlouhodobého hlediska. Za úvahu by zajisté stálo využití techniky vytěžování dat pro stanovení, které kroky provedené neuronovou sítí byly vhodné, a dle těchto informací neuronovou síť přeučit. Reálné dopady takových změn ovšem vyžadují řádné otestování jejich účinku, proto tyto kroky pouze doporučuji pro budoucí výzkum.

Literatura

[1] BEDDOE, G. R. – PETROVIC, S. A novel approach to finding feasible solutions to personnel rostering problems. In *Proceedings of the 14th Annual Conference of the Production and Operations Management Society (POM)*, Savannah, Georgia, United States, April 2003.

[2] BISKUP, R. *Možnosti neuronových sítí.* PhD thesis, ČZU, 2009. s. 23 - 62.

[3] BROEKENS, J. – HINDRIKS, K. – WIGGERS, P. *Reinforcement Learning as Heuristic for Action-Rule Preferences.* Springer Berlin Heidelberg, 2012. s. 25-40.

[4] BURKE, E. – SOUBEIGA, E. Scheduling Nurses Using a Tabu-Search Hyperheuristic. In *Proc. of the 1st MISTA*, s. 197–218, 2003.

[5] BURKE, E. K. et al. A Classification of Hyper-heuristic Approaches. In *International Series in Operations Research & Management Science Volume 146*, s. 449–468, 2010.

[6] BURKE, E. – KENDALL, G. – SOUBEIGA, E. *A Tabu-Search Hyperheuristic for Timetabling and Rostering.* Journal of Heuristics, 2003. s. 451-470.

[7] CHALUPNÍK, V. Biologické algoritmy (5) - Neuronové sítě. [online], 2012. Dostupné z: <http://www.root.cz/clanky/biologicke-algoritmy-5-neuronove-site/>.

[8] COWLING, P. – KENDALL, G. – SOUBEIGA, E. *Hyperheuristics: A robust optimisation method applied to nurse scheduling.* Parallel Problem Solving from Nature VII, PPSN 2002, 2002.

[9] CURTOIS, T. Automated employee scheduling benchmark instances. Dostupné z: <http://www.cs.nott.ac.uk/~tec/NRP/index.html>.

[10] DEMLOVÁ, M. Teorie algoritmu: text k prednáškám. [online], 2013. Dostupné z: <http://math.feld.cvut.cz/demlova/teaching/tal_vyuka.html>.

[11] DINESHA, V. NeuronDotNet - Neural Networks in C#. Dostupné z: <http://sourceforge.net/projects/neurondotnet/>.

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

[13] DRCHAL, J. Artificial Neural Networks - MLP, Backpropagation. [online], 2013. Dostupné z: <https://cw.felk.cvut.cz/lib/exe/fetch.php/courses/a4m33bia/a4m33bia-03backprop-2013.pdf>.

[14] IKONOMOVSKA et al. Using Data Mining Technique for Coefficient Tuning of an Adaptive Tabu Search. In *EUROCON, 2007. The International Conference on #34;Computer as a Tool #34;*, s. 706–713, 2007.

[15] JANUŠ, P. Rozvrhování zdravotních sester pomocí omezujících podmínek. Master's thesis, Masarykova univerzita, 2011.

[16] KAČENKA, P. Neuronové sítě. [online], 1998. Dostupné z: <http://mks.mff.cuni.cz/library/NeuronoveSitePK/NeuronoveSitePK.pdf>.

[17] KLÁN, P. – ŠTĚDRÝ, A. Neuronové sítě. [online], rok neznámý. Dostupné z: <http://www.cs.cas.cz/pklan/CHI7.pdf>.

[18] TUBINGENU. JavaNNS. Dostupné z: <http://www.ra.cs.uni-tuebingen.de/software/JavaNNS/>.

[19] KOMÍNEK, J. Heuristické algoritmy pro optimalizaci. Master's thesis, VUT v Brně, 2012. s. 12-23.

[20] KOMUNITA. Model neuronu. [online]. Dostupné z: <http://cs.wikipedia.org/wiki/Soubor:NeuronModel.jpg>.

[21] LI, J. – BURKE, E. K. – QU, R. *Integrating neural networks and logistic regression to underpin hyper-heuristic search.* Knowledge-Based Systems, 2011. s. 322 - 330.

[22] MARTIN, S. et al. *A generic agent-based framework for cooperative search using pattern matching and reinforcement learning.* European Journal of Operational Research, 2011.

[23] MATOUŠEK, R. Horolezecký algoritmus a úloha celočíselného programování v prostředí Matlab. [online], 2004. Dostupné z: <http://dsp.vscht.cz/konference_matlab/matlab04/matousek.pdf>.

[24] PAVLÍK, D. Využití neuronové sítě při identifikaci znaku v obraze. Master's thesis, VUT v Brně, 2007.

[25] POKORNÝ, M. Simulované žíhání jako nástroj k hledání optimálního řešení. [online], 2011. Dostupné z: <http://fyztyd.fjfi.cvut.cz/2011/cd/prispevky/sbpdf/zihani.pdf>.

[26] POLOMSKÝ, T. Algoritmy pro rozvrhování směn na grafických kartách. Master's thesis, ČVUT v Praze, 2012.

[27] POSPÍCHAL, P. Akcelerace genetického algoritmu s využitím GPU. Master's thesis, VUT v Brně, 2009.

[28] RUDOVÁ, H. Rozvrhování zaměstnanců. [online], 29. 4. 2012. Dostupné z: <http://www.fi.muni.cz/~hanka/rozvrhovani/prusvitky/jedenacta_bw.pdf>.

[29] SOUBEIGA, E. *Development and application of hyperheuristics to personnel scheduling.* PhD thesis, The University of Nottingham, 2003. s. 84-134.

[30] SUTTON, R. S. – BARTO, A. G. *Reinforcement Learning: An Introduction.* The MIT Press, 1998.

[31] VÁCLAVÍK, R. Algoritmy pro rozvrhování směn. Master's thesis, ČVUT v Praze, 2011.

[32] WEBERSCHINKE, J. Evolutionary hyper-heuristics for heuristic selection. Master's thesis, ČVUT v Praze, 2012.

[33] ZHANG, W. – DIETTERICH:, T. G. *Solving Combinatorial Optimization Tasks by Reinforcement Learning: A General Methodology Applied to Resource-Constrained Scheduling.* Journal of Artificial Intelligence Reseach, 1991.

Příloha A

Seznam použitých zkratek

NP Non deterministic

ILP Integer linear programming

RDF Resource Dilation Factor

NRP Nurse rostering problem

XML Extensible Markup Language

HTML HyperText Markup Language

API Application Programming Interface

VDS Variable depth search

JNNS Java Neural Network Simulator

CPU Central Processing Unit

GHz GigaHertz

CP Cover Penalty

Příloha B

Seznam použitého matematického značení

Proměnná	**Význam**
$rank$	ohodnocení heuristik - tabu vyhledávání
rn_{min}	minimální hodnota ohodnocení (rank) heuristik - tabu vyhledávání
rn_{max}	maximální hodnota ohodnocení (rank) heuristik - tabu vyhledávání
$InFeas$	ohodnocuje nevyváženost rozvrhu
C_{demand}	koeficient určující vliv hodnoty *InFeas* na evaluační funkci
ξ	vnitřní potenciál neuronu
w	váha vstupu neuronu
θ	práh neuronu
y	výstupní funkce neuronu
P_{hd}	úspěšnost heuristiky d (mHd)
K_{mh4}	maximální poměrná tolerance, která určuje, kdy je použití heuristiky $mH4$ považováno za úspěšné
$PP_{ins-alg}$	poměrná penalizace na instanci *ins* při zvolení algoritmu *alg*
N_h	počet restartů pomocí $mH4$ než dojde k ukončení testu

Tabulka B.1: Seznam použitého matematického značení - uvedena jsou pouze taková, která se vyskytují i jinde, než jen v daném vzorci, případně textu navazujícím přímo na vzorec.

Příloha C

Manuál pro programátora

Naimplementování algoritmu proběhlo v programu Microsoft Visual Studio 2012 Ultimate. Řešení (*Solution*) se skládá z několika projektů (viz tabulka C.1).

Název projektu	**Popis projektu**
Core	jádro frameworku NRP
InitialTSHH	původní algoritmus
ModifiedTSHH	modifikovaný algoritmus (s možností využití neur. sítě)
NeuroCore	framework neuronové sítě (neuroDotNet)
Solver	pomocný projekt frameworku NRP
SolverTests	pomocný projekt k frameworku NRP, načítá požadovaný rozvrh, zavolá danou metodu řešení a poté vygeneruje výstupní soubory

Tabulka C.1: Popis sady projektů v programu Microsoft Visual 2012 Ultimate.

Z velké množiny funkcí integrovaných ve frameworku NRP se používají především funkce uvedené v tabulce C.2.

Název funkce	**Umístění**	**Popis funkce**
Clone()	Core\Roster	vytvoří kopii rozvrhu
RecalculateAllPenalties()	Core\Roster	přepočítá penalizaci rozvrhu
AssignShift(Employee, Shift)	Core\Roster	přiřadí směnu zaměstnanci
UnAssignShift(Employee, Shift)	Core\Roster	odebere směnu zaměstnanci

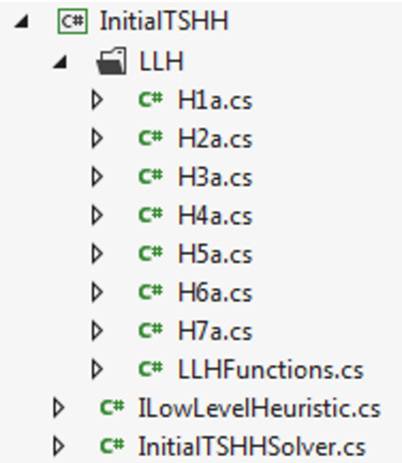
Tabulka C.2: Popis nejpoužívanějších funkcí frameworku NRP.

Hlavní funkce frameworku *NeuroDotNet* používané pro účely této práce popisuje tabulka C.3.

Struktura implementace původního algoritmu je zobrazena na obr. C.1. Třídy *H1a-H7a* reprezentují stejně pojmenované heuristiky popsané v této práci. Třída *LLHFunctions* obsahuje podpůrné metody používané především třídami *H1a-H7a*. Třída *ILowLevelHeuristic* definuje rozhraní heuristik. Hlavní třídou tohoto projektu je *InitialTSHHSolver*, která mimo jiné implementuje logiku tabu vyhledávání.

Název funkce	**Umístění**	**Popis funkce**
Learn(TrainingSet, epochs)	NeuroCore\Network	proběhne učení dané neuronové sítě
Run(input)	NeuroCore\Network	vrátí výstupní hodnoty neuronové sítě při daném vstupu

Tabulka C.3: Popis nejpoužívanějších funkcí frameworku NeuroDotNet.



Obrázek C.1: Struktura projektu InitialTSHH.

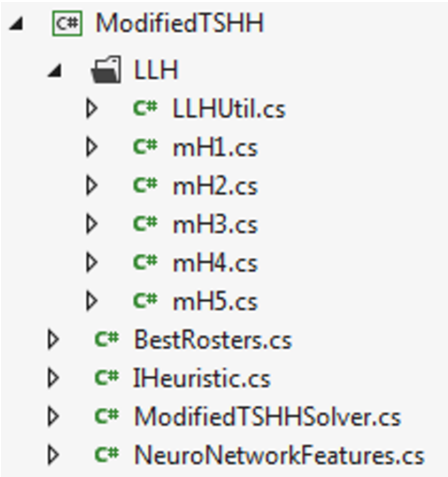
Strukturu implementace modifikovaného algoritmu poté zobrazuje obr. C.2. Třídy *mH1-mH5* implementují logiku stejnomenných heuristik. Třídy *LLHUtil, BestRosters a NeuroNetworkFeatures* jsou podpůrnými třídami obsahujícími funkce k danému problému (problém naznačuje jméno třídy). *IHeuristic* definuje rozhraní modifikovaných heuristik a třída *ModifiedTSHHSolver* je hlavní třídou obsahující řízení heuristik pomocí metody tabu vyhledávání či neuronové sítě.

Pro spuštění požadovaného testu je nejprve nutné ve třídě *Test* (projekt *SolverTests*, který je zároveň startovacím projektem) nastavit požadovanou vstupní instanci pomocí řetězové proměnné *schedulingPeriodID*. Dále nastavit proměnou *search* v metodě *ExecuteSearch()*:

- search = new InitialTSHHSolver() - pro spuštění původního algoritmu.
- search = new ModifiedTSHHSolver() - pro spuštění modifikovaného algoritmu

Každá z těchto voleb umožňuje nastavit některé další parametry týkající se průběhu algoritmu. Parametry lze volit přiřazením hodnoty proměnným, které jsou na začátku hlavních tříd daných algoritmů. Seznam proměnných souvisejících s původním algoritmem obsahuje tabulka C.4 a s modifikovaným algoritmem (s možností povolit řízení pomocí neuronové sítě) tabulka C.5. Po zvolení těchto parametrů samozřejmě následuje spuštění projektu.

Při zvolení spuštění více testů (proměnná *cycles* u původního algoritmu a *tabuOrNeuroCycles* u modifikovaného algoritmu) se po jejich ukončení vypíšou nejnižší penalizace rozvrhů v rámci každého testu a rozvrh s celkově nejnižší dosaženou penalizací se uloží do souboru



Obrázek C.2: Struktura projektu ModifiedTSHH.

Název proměnné	**Funkce proměnné**
cycles	nastavení počtu opakování testu
maxIterations	nastavení počtu spuštěných heuristik v jednom testu

Tabulka C.4: Popis proměnných v projektu InitialTSHH.

Název proměnné	**Funkce proměnné**
generateSampleForNN	v případě hodnoty *true* dojde nejprve k vygenerování dat pro neuronovou síť a následnému spuštění algoritmu řízeného metodou tabu vyhledávání a neuronovou sítí, v opačném případě se volba algoritmu řídí proměnnou *useNeuralNetwork*
useNeuralNetwork	v případě hodnoty *true* se použije k řízení heuristik metoda tabu vyhledávání i neuronová síť, v opačném případě se neuronová síť nepoužije
N_h	udává počet neúspěšných restartů pomocí heuristiky *mH4*, než se test ukončí
generateSampleMaxSteps	maximální počet iterací v jednom cyklu při generování vzorků pro učení neuronové sítě
generateSampleCycles	počet spuštěných cyklů při generování vzorků pro učení neuronové sítě
fileToCreate	název souboru, kam se uloží data vygenerovaných vzorků pro učení neuronové sítě
tabuOrNeuroCycles	počet spuštěných testů při spuštění modifikovaného algoritmu nebo modifikovaného algoritmu s použitím neuronové sítě
dir	cesta k adresáři, kde jsou uloženy soubory s vygenerovanými daty pro učení neuronové sítě
filename	jméno souboru, ve kterém jsou uložena data vygenerovaná pro učení neuronové sítě

Tabulka C.5: Popis proměnných v projektu ModifiedTSHH.

ve formátu XML do složky *html\data\solutions\xml* spustitelný v programu *RosterBooster* a standardního HTML souboru do složky *html\data\solutions\html*. Soubory obsahující vzorky pro neuronovou síť se nacházejí ve složce *neuroData*.

Příloha D

Obsah přiloženého CD

```
.
|-- Text     - adresář obsahující diplomovu práci v PDF
\-- Hyper-heuristics     - adresář obsahující projekt MS Studia 2012 + související data
     |-- Core     - adresář obsahující zdrojové kódy frameworku NRP
     |-- html     - adresář obsahující doplňkové soubory k frameworku NRP
     |-- InitialTSHH     - adresář obsahující zdrojové kódy původního algoritmu
     |-- ModifiedTSHH     - adresář obsahující zdrojové kódy modifikovaných algoritmů
     |-- neuroData     - adresář obsahující doplňkové soubory pro modifikovaný algoritmus
     |-- NeuronDotNet Source     - adresář obsahující zdrojové soubory
     |                                       frameworku NeuronDotNet
     |-- Solvers     - adresář obsahující zdrojové kódy frameworku NRP
     \-- SolverTests     - adresář obsahující zdrojové kódy frameworku NRP
```