ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kratochvíl** Jméno: **Ondřej** Osobní číslo: **420054**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Rozvrhování domácí péče

Název diplomové práce anglicky:

Home healthcare scheduling

Pokyny pro vypracování:

1) Navrhněte algoritmus pro problém rozdělování domácí péče zdravotním sestrám
2) Implementujte algoritmus řešící tento problém s využitím několika heuristik přístupů
3) Otestujte algoritmus na reálných vstupních datech, srovnejte jednotlivé heuristické přístupy a vyhodnoťte vzhledem k výsledkům publikovaným v literatuře

Seznam doporučené literatury:

[1] Guericke, D.; Suhl, L., The home health care problem with working regulations, OR SPECTRUM, Volume: 39 Issue: 4 Pages: 977-1010, DOI: 10.1007/s00291-017-0481-3 , Published: OCT 2017
[2] Christian Fikar, Patrick Hirsch, Home health care routing and scheduling, Computers and Operations Research archive, Volume 77 Issue C, January 2017, Pages 86-95.
[3] The state of the art of nurse rostering. E. Burke, P. de Causmaecker, Greet Vanden Berghe and Hendrik Van Landeghem UGent (2004) JOURNAL OF SCHEDULING. 7(6). p.441-499.

Jméno a pracoviště vedoucí(ho) diplomové práce:

prof. Dr. Ing. Zdeněk Hanzálek, oddělení průmyslové informatiky CIIRC

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **17.01.2018** Termín odevzdání diplomové práce: 25.5.2018

Platnost zadání diplomové práce: **30.09.2019**

prof. Dr. Ing. Zdeněk Hanzálek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

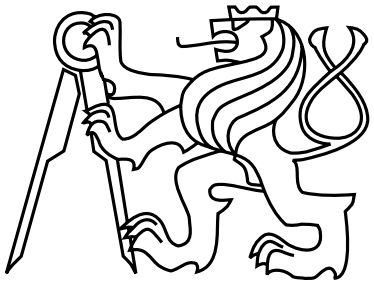
III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

24.4.2018 Datum převzetí zadání	Kratochvíl Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Diplomová práce

Rozvrhování domácí péče

Bc. Ondřej Kratochvíl

Vedoucí práce: prof. Dr. Ing. Zdeněk Hanzálek

Studijní program: Otevřená informatika, Magisterský

Obor: Softwarové inženýrství

23. května 2018

Poděkování

Rád bych poděkoval prof. Dr. Ing. Zdeňkovi Hanzálkovi za vedení diplomové práce a čas strávený konzultacemi. Dále bych také chtěl poděkovat své přítelkyni a rodině za zázemí, trpělivost a podporu nejen při psaní této práce, ale během celého studia.

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 24. 5. 2018 .....

Abstract

Home health care scheduling problem is a task from the $\mathcal{NP}$ hard complexity class that combines aspects of both travelling salesman problem and nurse rostering problem. The goal is to create individual schedules for medical assistants that perform demanded services in the client's home environment while also considering any potential constraints.

The aim of this master's thesis is to analyze the problem and to design and implement an algorithm that finds a suboptimal schedule for given input data. The services may also be specified in a flexible way by a time window, or can have certain constraints (soft or hard). The solution is found by a two-phase algorithm that initially creates a partial schedule for all assistants and subsequently iteratively improves it by applying two types of heuristics (insertion and removal). During the designing of the algorithm, an emphasis was put on its extensibility, easy configurability and independence of all individual components, which facilitates any potential changes, such as altering the objective function or using different heuristics for schedule improvement. Benchmarking of the efficiency of the algorithm was measured on a set of both real and generated instances and the usability of the individual heuristics was evaluated based on analysis of the results.

Keywords: Home care, scheduling, NP hard, heuristics

Abstrakt

Problém rozvrhování domácí péče je úlohou ze třídy $\mathcal{NP}$ hard problémů, která kombinuje aspekty problému obchodního cestujícího a plánování směn zdravotních sester, známý jako nurse rostering problem. Cílem je vytvořit rozvrhy pro zdravotní asistenty, kteří vykonávají objednanou péči v domácím prostředí klientů, se zohledněním případných omezení.

Cílem této diplomové práce je provést analýzu problému a navrhnout a implementovat algoritmus, který najde suboptimální rozvrh pro daná vstupní data. Objednané asistence mohou být navíc zadány flexibilně pomocí časového okna, nebo mohou mít určitá omezení (*soft constraints* či *hard constraints*). Řešení je hledáno dvoufázovým algoritmem, který nejprve vytvoří částečný rozvrh pro všechny sestry a následně jej iterativně vylepšuje pomocí dvou typů heuristik (*insertion* a *removal*). Při návrhu algoritmu byl kladen velký důraz na rozšiřitelnost, snadnou konfiguraci a nezávislost jednotlivých komponent, díky čemuž jej lze velice jednoduše upravovat, například změnit kriteriální funkci nebo použít jiné heuristiky pro hledání výsledného rozvrhu. Měření efektivity algoritmu proběhlo na řadě reálných i generovaných dat a na základě analýzy výsledků byla vyhodnocena využitelnost použitých heuristik.

Klíčová slova: Domácí péče, rozvrhování, NP hard, heuristiky

Obsah

Seznam obrázků

Seznam tabulek

Seznam algoritmů

Kapitola 1

Úvod

Stárnutí populace, spočívající ve zvyšování podílu starých lidí v populaci, je demografický jev, se kterým se v současné chvíli potýkají vyspělé státy v Evropě i ve zbytku světa. Tento fenomén souvisí s demografickými změnami, mezi které patří zejména pokles úmrtnosti, porodnosti a zvyšování střední délky života [43], a má mnoho ekonomických a sociálních implikací. Podle Českého statistického úřadu byl v ČR v roce 2016 mediánový věk 41,5 let, což představuje od roku 2000 nárůst o 4,2 roky a podíl obyvatel starších 65 let byl 18,3% [44], přičemž v budoucnosti je očekáván ještě další nárůst těchto hodnot.

Jedním z důsledků tohoto jevu je zvyšování nároků na zdravotní péči. Její vykonávání však již není limitováno pouze na nemocnice a zdravotnická zařízení. V posledních letech se poměrně výrazně rozrůstá sektor domácí péče [27], kdy pacienti zůstávají v domácím prostředí a zdravotní asistenti za nimi dojíždějí a vykonávají potřebnou péči na místě. Častým případem využití je například paliativní péče, tedy péče o (většinou) starší pacienty s nevyléčitelnou nemocí v terminálním stádiu, nicméně senioři nejsou jedinými klienty; často tuto službu využívají také osoby se zdravotním postižením. Jednou z hlavních výhod tohoto způsobu péče o nemocné pacienty je psychologický aspekt, který je důležitý zejména právě pro seniory, kteří často preferují pobyt v domácím prostředí s rodinou, přičemž kvalifikované úkony provádějí dojíždějící sestry.

Typy péče se liší na základě potřeb pacienta. Pro některá speciální ošetření je potřeba přítomnost sestry s určitými kvalifikacemi či certifikací; v jiných případech je pacientem vyžadována konkrétní sestra či jedna z vybraných sester. Tak je tomu často u starších osob, kteří péči objednávají pravidelně a jsou zvyklí na konkrétní asistenty, kteří k nim dojíždějí. Asistence mohou být objednány na konkrétní čas, nebo mohou být zadány volněji pomocí časového okna – například dvouhodinová asistence, která může být provedena kdykoliv mezi 8:00 a 15:00. Z pohledu zdravotních sester je třeba splnit smluvený pracovní úvazek a reflektovat vybrané dny volna a dovolené. A jelikož péče o pacienty není vykonávána na jednom konkrétním místě, jak je tomu v případě hospitalizace, je do pracovní doby zdravotního asistenta třeba započítat také přesun mezi jednotlivými klienty, ke kterým asistent v daný den jede.

Problematika rozvrhování domácí péče se zabývá tvorbou rozvrhu pro zdravotní asistenty na základě jejich dostupnosti a asistencí objednaných jednotlivými klienty. Tento rozvrh popisuje přiřazení objednaných asistencí zdravotním sestrám v konkrétní čas a popisuje pořadí klientů, za kterými bude sestra v daný den cestovat. Případné speciální potřeby klientů jsou

vyjádřeny formou omezení, která mohou být porušitelná (*soft constraints*), či neporušitelná (*hard constraints*). Kvalita výsledného rozvrhu je posuzována na základě několika různých kritérií, mezi které patří počet přiřazených asistencí, počet porušených soft constraints či efektivita využití pracovní doby sestry (tj. doba, kdy daná sestra vykonává péči u klienta nebo je na cestě ke klientovi).

Samotný proces tvorby výsledného rozvrhu je však náročný, jelikož problém rozvrhování domácí péče zahrnuje dva podproblémy ze třídy $\mathcal{NP}$ hard – *nurse rostering problem* (NRP) a *vehicle routing problem with time windows* (VRPTW), který je rozšířením problému obchodního cestujícího (TSP). Z toho důvodu není možné řešit tento problém pomocí exaktních algoritmů, které hledají optimální rozvrh; místo toho je vhodné zvolit heuristický přístup, kterým bude vytvořen suboptimální rozvrh.

Cílem této práce je provést analýzu problému a stávajících řešení a implementovat algoritmus pro řešení problému rozvrhování domácí péče. Navržený algoritmus je rozdělen do dvou fází – v první fázi je vytvořen počáteční, částečný rozvrh a seznam nepřiřazených asistencí a ve druhé fázi je tento rozvrh iterativně vylepšován za cílem dosažení lepší hodnoty kriteriální funkce. Druhá část algoritmu je inspirována metaheuristikou ALNS (*adaptive large neighborhood search* [22]), která využívá dvou typů heuristických metod – odebírání nevhodně umístěných asistencí z rozvrhu a přidávání nových asistencí ze seznamu nepřiřazených do rozvrhu. Pro posouzení efektivity algoritmu je provedena analýza výsledků různých typů implementovaných heuristik na různých množinách vstupních dat a dosažené výsledky jsou porovnány s literaturou.

Struktura této práce je následující: v kapitole 2 je představen a formalizován řešený problém spolu s analýzou jeho složitosti a obecných přístupů k řešení problémů ze třídy $\mathcal{NP}$ hard. Kapitola 3 je zaměřena na rešerši konkrétních přístupů pro řešení problému rozvrhování domácí péče v literatuře a vyhodnocení jejich efektivity. Návrhem frameworku pro řešení představeného problému, který je následně použit pro implementaci vlastního algoritmu, se zabývá kapitola 4; samotná implementace je popsána v kapitole 5. Popis testovací strategie a typů implementovaných testů pro ověření správnosti navrženého konceptu je obsažen v kapitole 6. Kapitola 7 se věnuje analýze výsledků experimentů pro měření efektivity algoritmu na množinách generovaných a reálných dat. Nakonec, v kapitole 8 je provedena diskuse dosažených výsledků a použitelnosti algoritmu spolu s různými možnostmi budoucí práce.

Kapitola 2

Analýza

2.1 Definice problému

Cílem této práce je navrhnout algoritmus pro zdravotnickou firmu poskytující domácí péči (*HHC – home health care*), jenž by měl sloužit k efektivnímu rozvrhování směn pro zdravotnické asistenty (dále také sestry).

Směny jsou plánovány na následující měsíc, tvorba rozvrhu tedy probíhá typicky na konci měsíce a v současné chvíli je prováděna ručně. Před začátkem období obdrží firma požadavky od klientů specifikující datum, požadovaný typ péče a čas, kdy má být vykonávána. Kromě klientských požadavků je také známa časová dostupnost zdravotních sester – ta je určena typem úvazku, ale nemusí být totožná pro každé plánovací období, jelikož reflektuje například čerpání dovolené či jiný typ volna.

Klientské požadavky obsahují několik typů informací. Nejdůležitější z nich je doba asistence - ta je buď fixní (např. 8:00 – 12:00), nebo klouzavá, tedy zadána pomocí časového okna (např. dvouhodinová péče kdykoliv mezi 10:00 a 17:00); někteří klienti ale také využívají asistenty pouze na doprovod k lékaři. Počet požadavků na den není nijak omezen a v praxi se často stává, že si klient objedná nesouvislou péči ve stejný den na dopoledne i odpoledne.

V některých případech může péče o klienta vyžadovat specifické dovednosti; takovou asistenci pak mohou provádět jen některé zdravotní sestry, které mají požadovanou kvalifikaci. Často je také preferováno (nikoliv však vyžadováno), aby dostával klient přiřazeného stejného asistenta jako v minulosti z důvodu osobních preferencí.

Důležitým aspektem je doba přejezdu mezi jednotlivými klienty či bydlištěm asistenta, která nemůže být v reálném prostředí zanedbána. Dvě asistence po sobě nemohou být přiřazeny stejnému asistentovi, pokud by byla doba přesunu od jednoho klienta k druhému delší, než doba mezi koncem první směny a začátkem druhé. V současné chvíli je snahou, aby byly přesuny při tvorbě rozvrhů co nejmenší a aby sestra jezdila za klienty z okolí jejího bydliště.

Vzhledem k povaze problému nelze považovat za jediné přípustné řešení takový rozvrh, který přiřadí zdravotním sestrám všechny objednané asistence, jelikož vstupní data nezaručují, že bude v každou chvíli k dispozici dostatečný počet sester a že mohou být splněna všechna omezení, jako např. požadovaná kvalifikace. Z toho důvodu musí být algoritmus

schopen vytvořit i částečný rozvrh, kde budou přiřazeny pouze některé asistence, které neporušují žádnou z nutných podmínek. Ostatní asistence, které nelze přiřadit bez porušení takových omezení budou zaznamenány jako nepřiřaditelné. Tato data mohou být následně dále analyzována a na jejich základě může dojít k úpravě údajů – například ke zkrácení některých asistencí po komunikaci s klientem, k prodloužení směny sestry, k odebrání některých omezení apod. Preferovaným (optimálním) výsledkem je však pochopitelně rozvrh, který maximalizuje počet přiřazených směn a minimalizuje počet porušených omezení.

Při návrhu algoritmu je důležitý důraz na jeho modifikovatelnost a schopnost adaptace při změně podmínek – například je možné, že v budoucnu vyvstanou nová omezení či preference. Algoritmus by měl být připraven na takové situace a integrace těchto změn by měla být snadná bez nutnosti většího zásahu. Modulární návrh je tedy z tohoto hlediska velice důležitý.

2.2 Formalizace problému

Pro exaktní popis problému bude v této práci využito formulace omezení a invariant pomocí soustavy lineárních rovnic a nerovnic, což je technika, která je používána například pro programování s omezujícími podmínkami (*CP – constraint programming*) nebo celočíselné lineární programování *(ILP – integer linear programming)*. Oba tyto přístupy jsou příkladem deklarativního programování, což je paradigma, které se soustředí na popis řešeného problému a ne na způsob jeho řešení, jak je tomu u imperativního způsobu programování. Obecně, lineární programování (*LP*) umožňuje formalizovat problém jako soustavu lineárních rovnic a nerovnic, přičemž hodnoty proměnných mohou mít libovolné hodnoty z oboru reálných čísel. Pokud je jejich definiční obor omezen, je možné LP dále rozdělit na celočíselné lineární programování, kdy jsou všechny použité proměnné celá čísla, nebo smíšené lineární programování (*MILP – mixed integer linear programming*), kdy mohou být definičním oborem proměnných jak celá, tak reálná čísla. Problém rozvrhování domácí péče může být alternativně formulován vytvořením soustavy lineárních rovnic a nerovnic, popisující jednotlivá pravidla a omezení, a definicí *účelové funkce*, vyjadřující kritérium, které má být optimalizováno (např. minimalizace nepřiřazených asistencí, minimalizace doby, kdy sestra není u klienta ani na cestě apod.) [13].

Omezení pro úlohu rozvrhování domácí péče (*HHCS*) lze rozdělit do dvou skupin – *hard constraints* a *soft constraints*. *Hard constraints* jsou taková omezení, která musí být splněna pro každý výsledný model rozvrhu – pokud by bylo některé z nich porušeno, je model považován za nesplnitelný (*infeasible*) [3]. Do této skupiny omezení patří například nutná specializace sestry pro provedení asistence, minimální doba přejezdu mezi dvěma místy nebo nepřekročení začátku a konce dostupnosti dané sestry.

Soft constraints vyjadřují spíše ideální hodnoty, kterých by mělo být dosaženo, než omezení. Výsledný rozvrh může být považován za validní (*feasible*), přestože došlo k porušení některých soft constraints, nicméně pokud se dva validní rozvrhy budou lišit pouze počtem těchto porušených omezení, je považován za lepší ten rozvrh, který jich porušuje méně. Takového chování lze dosáhnout zavedením penalizačních konstant pro každé z těchto omezení. Pro výsledný model je pak vypočítáno, která omezení jsou porušena, a suma penalizačních konstant pro porušená omezení je součástí účelové funkce. Konfigurací těchto konstant lze také rozlišit závažnost porušení jednotlivých soft constraints – pokud by bylo porušení

Tabulka 2.1: Seznam použitých parametrů

Parametr	Definiční obor	Popis
$\mathcal{N}$	$\mathcal{N} \in \mathbb{N}$	Množina asistentů (sester)
$\mathcal{C}$	$\mathcal{C} \in \mathbb{N}$	Množina klientů
$\mathcal{J}$	$\mathcal{J} \in \mathbb{N}$	Množina objednaných asistencí
j^n_{start}	$j^n_{start} \in \mathcal{J}; n \in \mathcal{N}$	Pomocná asistence, první vrchol cesty asistenta $n \in \mathcal{N}$
j^n_{end}	$j^n_{start} \in \mathcal{J}; n \in \mathcal{N}$	Pomocná asistence, poslední vrchol cesty asistenta $n \in \mathcal{N}$
s_j	$s_j \in \mathbb{N}; j \in \mathcal{J}$	Začátek časového okna asistence $j \in \mathcal{J}$[1]
e_j	$e_j \in \mathbb{N}; j \in \mathcal{J}$	Konec časového okna asistence $j \in \mathcal{J}$[1]
l_j	$l_j \in \mathbb{N}; j \in \mathcal{J}$	Délka časového okna asistence $j \in \mathcal{J}$
$\mathcal{S}$	$\mathcal{S} \in \mathbb{N}$	Množina soft constraints
p_s	$p_s \in \mathbb{Z}; s \in \mathcal{S}$	Penalizace za porušení omezení $s \in \mathcal{S}$
p_u	$p_u \in \mathbb{Z}$	Penalizace za nepřiřazenou asistenci
$d_{i,j}$	$d_{i,j} \in \mathbb{N}; i, j \in \mathcal{J}$	Doba cesty mezi asistencemi $i, j \in \mathcal{J}$
w^n_s	$w^n_s \in \mathbb{N}; n \in \mathcal{N}$	Začátek dostupnosti asistenta $n \in \mathcal{N}$
w^n_e	$w^n_e \in \mathbb{N}; n \in \mathcal{N}$	Konec dostupnosti asistenta $n \in \mathcal{N}$

Tabulka 2.2: Seznam použitých proměnných

Proměnná	Definiční obor	Popis
v^j_s	$v^j_s \in \{0,1\}$ $s \in \mathcal{S}, j \in \mathcal{J}$	$\begin{cases} 1, & \text{pokud je omezení } s \in \mathcal{S} \text{ porušeno pro asistence } j \in \mathcal{J} \\ 0, & \text{jinak} \end{cases}$
$x^n_{i,j}$	$x^n_{i,j} \in \{0,1\}$ $n \in \mathcal{N}, i, j \in \mathcal{J}$	$\begin{cases} 1, & \text{pokud je asistence } j \in \mathcal{J} \text{ přiřazena právě před } i \in \mathcal{J} \\ 0, & \text{jinak} \end{cases}$
n_j	$n_j \in \{0,1\}$ $j \in \mathcal{J}$	$\begin{cases} 1, & \text{pokud asistence } j \in \mathcal{J} \text{ není přiřazena} \\ 0, & \text{jinak} \end{cases}$
y^n_j	$y^n_j \in \mathbb{N}$ $n \in \mathcal{N}, j \in \mathcal{J}$	Čas, kdy sestra $n \in \mathcal{N}$ začne vykonávat asistenci $j \in \mathcal{J}$

omezení c_1 závažnější než porušení omezení c_2, musí pak platit $p_1 > p_2$, kde p_1 a p_2 jsou penalizační konstanty pro c_1, respektive c_2.

V tabulce 2.1 je popsán seznam parametrů, které popisují vstupní data, definice proměnných pak v tabulce 2.2. Vytvořený model popisuje přiřazení objednaných asistencí sestrám v jeden konkrétní den – je tedy pouze částí celkového modelu. Jelikož jsou však rozvrhy pro jednotlivé dny disjunktní množiny, je možné model pro celé plánovací období dekomponovat na dny, pro které stačí pouze měnit příslušné parametry (objednané asistence, dostupnost sester apod.). Důvodem pro takovou dekompozici je zejména zvýšení přehlednosti rovnic a nerovnic – mnoho z nich by pro celkový model obsahovala navíc sumu přes všechny dny, jelikož vybrané parametry by měly o index více (vyjadřující den). Mezi takové parametry patří např. objednané asistence, jejich délka, začátek a konec dostupnosti sester apod.

[1] Začátek a konec časového okna může být reprezentován například jako počet minut od půlnoci daného dne. V takové podobě lze pak s těmito parametry zacházet jako s čísly a aplikovat na ně standardní aritmetické operace.

Model dále také popisuje množinu nepřiřazených asistencí, zajišťuje dodržování hard constraints a minimalizuje penalizaci za porušené soft constraints. Formulace problému vypadá následovně:

$$min(\sum_{j\in\mathcal{J}}\sum_{s\in\mathcal{S}} v_s^j \cdot p_i) + \sum_{j\in\mathcal{J}} u_j \cdot p_u \tag{2.1}$$

Rovnice 2.1 popisuje účelovou funkci. První část udává celkovou penalizaci modelu za porušené soft constraints, pro každou asistenci je přičtena hodnota penalizační konstanty pro každé porušené omezení. Druhá část rovnice vyjadřuje penalizaci za nepřiřazené asistence.

$$(\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{J}} x_{i,j}^n) + u_j = 1 \qquad ,\forall j \in \mathcal{J} \tag{2.2}$$

$$y^n_{j^n_{start}} = w_s^n \qquad ,\forall n \in \mathcal{N} \tag{2.3}$$

$$\sum_{j\in\mathcal{J}} x^n_{j^n_{start},j} = 1 \qquad ,\forall n \in \mathcal{N} \tag{2.4}$$

$$\sum_{i\in\mathcal{J}} x^n_{i,j^n_{end}} = 1 \qquad ,\forall n \in \mathcal{N} \tag{2.5}$$

$$d_{i,j^n_{end}} = 0 \qquad ,\forall i \in \mathcal{J}, \forall n \in \mathcal{N} \tag{2.6}$$

$$\sum_{j\in\mathcal{J}} x^n_{j,i} = \sum_{j\in\mathcal{J}} x^n_{i,j} \qquad ,\forall i \in \mathcal{J} \setminus \{j^n_{start}, j^n_{end}\}, \forall n \in \mathcal{N} \tag{2.7}$$

$$(\sum_{i\in\mathcal{J}} x^n_{i,j} \cdot y^n_j) \geq w^n_s \qquad ,\forall n \in \mathcal{N}, \forall j \in \mathcal{J} \tag{2.8}$$

$$(\sum_{j\in\mathcal{J}} x^n_{i,j} \cdot y^n_i) + l_i \leq w^n_e \qquad ,\forall n \in \mathcal{N}, \forall i \in \mathcal{J} \tag{2.9}$$

$$(\sum_{i\in\mathcal{J}} x^n_{i,j} \cdot y^n_j) + l_j \leq (\sum_{k\in\mathcal{J}} x^n_{j,k} \cdot (y_k + d_{j,k})) \qquad ,\forall n \in \mathcal{N}, \forall j \in \mathcal{J} \tag{2.10}$$

Fakt, že je každá asistence přiřazena právě jednou některé sestře, nebo je v množině nepřiřazených asistencí, zajišťuje rovnice 2.2. Proměnná $x^n_{i,j}$ slouží ke dvěma účelům – v první řadě udává pořadí asistencí pro danou sestru, lze ji však také využít jako indikátor, zda je některá asistence přiřazena, nebo ne. Hodnota výrazu $\sum_{n\in\mathcal{N}}\sum_{i\in\mathcal{J}} x^n_{i,j}$ vyjadřuje, zda *existuje sestra n, která má přiřazenou asistenci j po nějaké jiné asistenci i.* Každou asistenci $j \in \mathcal{J}$ lze totiž také vnímat jako vrchol orientovaného grafu, přičemž množina $\mathcal{J}$ obsahuje také pomocné asistence j^n_{start} a j^n_{end}, které reprezentují počáteční a konečný vrchol cesty grafem pro sestru n. Tato cesta popisuje posloupnost asistencí, které sestra v daný den vykoná. Jelikož se sestra na začátku pracovní doby nachází na adrese svého bydliště, musí platit rovnice 2.3, 2.4 (pomocná asistence j^n_{start} je přiřazena právě před jednou jinou asistencí), 2.5 (pomocná asistence j^n_{end} je přiřazena právě po jedné jiné asistenci). Na

konci pracovní doby se sestra nemusí nacházet na adrese svého bydliště, tedy konec poslední přiřazené asistence může být roven konci dostupnosti sestry (w_e^n). Toto pravidlo je pokryto rovnicí 2.6.

Rovnice 2.7 zajišťuje, že do každého vrcholu cesty povede právě jedna vstupní a právě jedna výstupní hrana. Jelikož asistent nemusí končit v místě svého bydliště, neplatí toto omezení pro pomocné asistence j_{start}^n a j_{end}^n.

Další dvě nerovnice zaručují, že žádná asistence nebude přiřazena sestře $n \in \mathcal{N}$ dříve, než začíná její dostupnost (2.8), nebo později, než končí její dostupnost (2.9). Nakonec, nerovnice 2.10 ošetřuje korektní reflexi dojezdových vzdáleností – pokud jsou jedné sestře přiřazeny dvě asistence j_1 a j_2 za sebou, musí být součet konce j_1 a dojezdové vzdálenosti $d_{1,2}$ menší nebo roven začátku asistence j_2. Jelikož v proměnných modelu není explicitně ukládán konec přiřazené asistence, je dopočítán jako začátek asistence plus její délka.

2.3 Modelový příklad

Pro lepší ilustraci problému zde bude formulován minimalistický modelový příklad, který bude demonstrovat všechny požadavky problému rozvrhování domácí péče zmíněné v kapitole 2.1. Předpokládejme dostupnost sester viz tabulka 2.3 – k dispozici jsou dvě sestry, přičemž Radmila Lukešová je dostupná pouze 1. 12. od 8 do 17 hodin (veškerá jména byla automaticky vygenerována a slouží pouze pro snadnou identifikaci). Eliška Nešporová má pracovní den 1. 12. rozdělený do dvou časových intervalů: od 10 do 13 hodin a po dvouhodinové pauze od 15:00 do 18:00. 2. 12. je dostupná v souvislém intervalu 8:00 – 12:00. Navíc má také kvalifikaci pro provádění peritoneální dialýzy, tedy může být přiřazena ke klientům, kteří požadují takový typ ošetření. Tato kvalifikace je zde uvedena pro demonstraci různých dovedností, které tvoří hard constraints vybraných asistencí.

Radmila Lukešová

Datum	Dostupnost
1.12.2017	8 : 00 – 17 : 00

Eliška Nešporová

Datum	Dostupnost
1.12.2017	10 : 00 – 13 : 00
	15 : 00 – 18 : 00
2.12.2017	8 : 00 – 12 : 00

Specializace: per. dialýza

Tabulka 2.3: Dostupnost sester

Požadované asistence klientů jsou zobrazeny v tabulce 2.4. Formát je stejný jako popis dostupnosti sester; některé časy asistencí jsou navíc specifikovány časovým oknem, jak je tomu např. 2. 12. u Tomáše Suka – hodinová péče kdykoliv mezi 8. a 12. hodinou. Informace v závorkách popisují soft, či hard constraints, konkrétně preferovanou/požadovanou sestru (Vítězslav Kudela či Mojmír Vladyka, popsáno v závorce iniciály sestry), nebo požadovanou dovednost (Tomáš Suk, popsáno opět v závorce zkratkou dovednosti – PD = peritoneální dialýza, OS = ošetřování stomií). Pro úplnost by měly tabulky obsahovat ještě bydliště klienta či sestry; předpokládejme však pro jednoduchost, že jakýkoliv přesun mezi lokacemi trvá 30 minut.

Mojmír Vladyka

Datum	Asistence
1.12.2017	$9:00-12:00$ $14:00-15:00$ $15:30-16:30$
2.12.2017	$8:30-9:30$ *(pož. EN)*

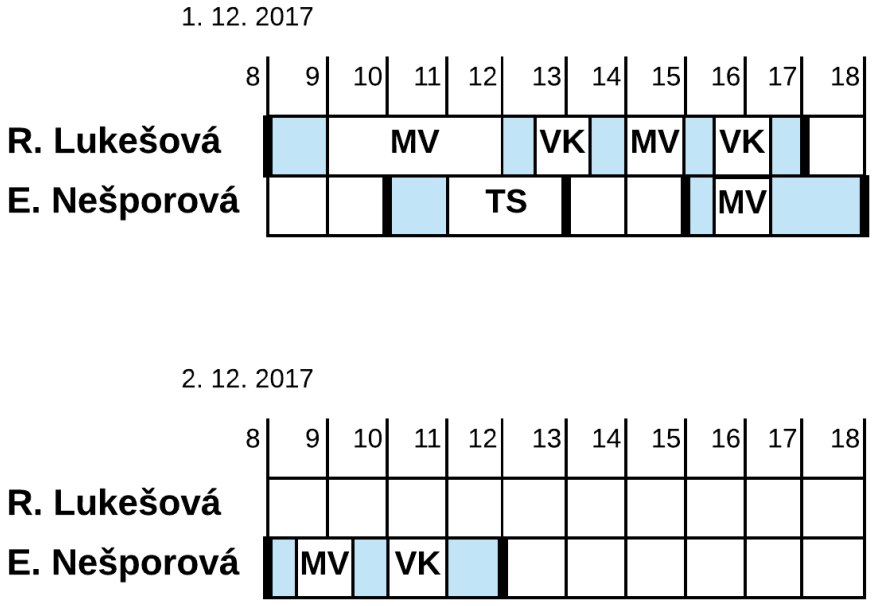
Vítězslav Kudela

Datum	Asistence
1.12.2017	$12:30-13:30$ $(pref.RL)$ $15:30-16:30$ $(pref.RL)$
2.12.2017	$9:00-15:00$ 1h $(pref.EN)$ $11:00-12:30$

Tomáš Suk

Datum	Asistence
1.12.2017	$11:00-13:00$ $(dov.PD)$ $14:00-15:00$ $(dov.OS)$
2.12.2017	$8:00-12:00$ 1h

Tabulka 2.4: Požadované asistence klientů



Obrázek 2.1: Příklad výsledného rozvrhu

Příklad výsledného rozvrhu je znázorněn na obrázku 2.1. Každý řádek vyjadřuje rozvrh dané sestry, přičemž modrá část indikuje její původní dostupnost, jejíž začátek a konec je zvýrazněn tučnou vertikální čarou. Jednotlivé asistence jsou znázorněny bílými bloky s iniciály pacienta. Z celkových 11 objednaných asistencí jich lze přiřadit 8 – první nepřiřaditelná asistence je 2. 12. objednaná Vítězslavem Kudelou, 11:00 – 12:30 – zde je očividné, že nemůže být s aktuální dostupností přiřazena, jelikož žádná sestra není dostupná do půl jedné. Další nepřiřaditelná asistence je 1. 12. 14:00 – 15:00, Tomáš Suk – není dostupná žádná sestra, která by měla kvalifikaci na ošetřování stomií. Poslední je asistence 2. 12. objednaná opět

Tomášem Sukem, hodinová péče v rozmezí od 8:00 do 12:00. Ta na první pohled přiřadit lze, nicméně doba péče se překrývá s dalšími objednanými asistencemi. Přestože je udána časovým oknem, nelze ji umístit tak, aby bylo možné stihnout všechny asistence objednané na 2. 12. a jelikož mají všechny stejnou délku (1h), nezáleží z hlediska kriteriální funkce na tom, která bude vybrána. V celém výsledném rozvrhu není porušena žádná soft constraint a navíc jsou všechny sestry v průběhu celého dne efektivně využity – tráví čas buď u klienta, nebo na cestě. Výjimkou je čas před první a po poslední přiřazené asistenci, nicméně o tento čas jim může být zkrácena pracovní doba.

2.4 Související problémy

Problematika hledání optimální cesty (*routing problem*) je disciplína z oblasti teorie grafů a kombinatorické optimalizace, která získala na popularitě zejména díky velkému množství aplikací v reálném světě. V průběhu posledního století, zejména ve druhé polovině, bylo tomuto problému věnováno mnoho pozornosti odborníků nejen z oblasti matematiky, ale také informatiky či fyziky a na toto téma vzniklo mnoho studií.

Jedním z prvních a pravděpodobně nejznámějších problémů z této oblasti je *problém obchodního cestujícího (TSP – travelling salesman problem)*. Jeho počátky nejsou přesně známy, ale za první zmínku je považován článek německého obchodního cestujícího z roku 1832 [5]. Ten sice nemá odbornou podobu, avšak pomohl přilákat vědeckou pozornost k tomuto problému. První odborná práce byla vytvořena Karlem Mengerem až ve 30. letech ve Vídni; ve stejné dekádě se obdobnému problému věnoval Hassler Whitney z Harvardovy univerzity a také Merrill Flood z Kolumbijské univerzity, který hledal optimální cestu pro školní autobusy [11].

Definice problému obchodního cestujícího je velice jednoduchá a tvoří základ pro problematiku rozvrhování domácí péče, její podoba zní následovně:

Definice 2.4.1 *Nechť $G = (V, E)$ je graf, kde V je množina n vrcholů a E je množina hran, a nechť $C = c_{i,j}$ je matice vzdáleností. Najděte hamiltonovskou kružnici H v grafu G, jejíž cena $\sum_{e \in E(H)} c(e)$ je minimální. [25]*

Pro úplnost je třeba dodefinovat dva pojmy z definice 2.4.1. Za prvé, pro každý prvek $c_{i,j} \in C$ platí, že $c_{i,j}$ je cena hrany z vrcholu i do vrcholu j; $i, j \in V$. Většina aplikací problému obchodního cestujícího splňuje podmínky *metrického TSP*, které vychází z tří předpokladů [11]:

- Graf G je úplný
- Graf G je neorientovaný, tedy platí $c_{i,j} = c_{j,i}$ (matice C je symetrická)
- Vzdálenosti v grafu G splňují trojúhelníkovou nerovnost: $c_{i,j} + c_{j,k} \geq c_{i,k}, \forall i, j, k \in V$

Za druhé, je třeba definovat hamiltonovskou kružnici:

Definice 2.4.2 *Hamiltonovská kružnice v grafu G je kružnice, která obsahuje každý vrchol grafu G. [4]*

Pokud jsou reálné aplikace optimalizačních problémů formulovány jako problém obchodního cestujícího, velice často splňují podmínky metrického TSP v případě, že jsou popsány v *euklidovském prostoru* – např. pokud matice vzdáleností C popisuje vzdálenost vzdušnou čarou mezi dvěma městy; nicméně i neeuklidovské prostory mohou splňovat podmínky metrického TSP za předpokladu, že je splněna trojúhelníhová nerovnost. Tak je tomu i pro formulaci rozvrhování domácí péče. V mnoha ohledech se však problematika HHCS liší od výše zmíněné definice TSP. V první řadě jsou plánovány cesty pro vícero sester, nikoliv pouze pro jednu. Tato modifikace je v literatuře známa jako *m-TSP – multiple travelling salesman problem*, které je zobecněním klasického problému obchodního cestujícího. Ani ta však nepokrývá požadavky rozvrhování domácí péče, jelikož pro m-TSP se předpokládá, že všichni obchodní cestující začínají a končí ve stejném bodě [2]. Mimo to je ve všech formulacích TSP velkým omezením hledání hamiltonovské kružnice, které by analogicky v problému HHCS znamenalo, že žádný z klientů nesmí být navštíven vícekrát, než jednou [27], což je v praxi poměrně častým případem.

Další modifikací TSP, která se přibližuje problému rozvrhování zdravotní péče, je *problém obchodního cestujícího s časovými okny – TSPTW*. Ten přidává temporální omezení pro každé město, které musí být navštíveno pouze v předem specifikovaném časovém okně [26]. Tato omezení sice částečně zahrnují požadavky pro rozvrhování domácí péče zmíněné v kapitole 2.1, avšak například již neuvažují dobu strávenou samotným vykonáváním péče; navíc má *TSPTW* stejné nedostatky jako všechny ostatní modifikace TSP, jak je popsáno výše.

Mezi varianty, respektive generalizace problému obchodního cestujícího, patří také *vehicle routing problem (VRP)*, do češtiny občas překládáno jako úloha okružních jízd. VRP vychází z problému m-TSP; cílem je naplánovat cesty váženým grafem, které budou hamiltonovské kružnice, tak, že jejich cena bude minimální. Místo obchodníků zde figurují vozidla, která rozváží zboží nebo jiný náklad zákazníkům. Tento problém byl poprvé představen v roce 1959 [38] americkými matematiky G. B. Dantzigem a J. H. Ramserem, kteří řešili problematiku rozvozu benzínu do čerpacích stanic [14]. Oproti m-TSP zavádí VRP omezení vztahující se konkrétně k dopravní problematice, v nejjednodušší formulaci je to kapacita vozidla a samozřejmě výše poptávky pro každý vrchol grafu [24]. Navíc je také v každém vrcholu předem definována doba trvání vykládání zboží, čímž se VRP přibližuje problematice rozvrhování domácí péče. V ostatních ohledech je však totožný s problémem obchodního cestujícího, tudíž se ze stejných důvodů liší od HHCS. Stejně jako u výše zmiňovaných problémů existují i u úlohy vehicle routing problem různé modifikace a rozšíření – například heterogennost vozidel (jednotlivá vozidla se liší kapacitou), zavedení více druhů nákladu nebo časových oken (VRPTW).

Jiným typem problému, který nevychází z problému obchodního cestujícího, ale velice úzce souvisí s HHCS, je *nurse rostering problem (NRP)*, problém rozvrhování směn zdravotních sester (v literatuře známý také jako *nurse scheduling problem*). Ten patří do kategorie *scheduling* problémů, které jsou významnou disciplínou kombinatorické optimalizace s širokou aplikací v reálném světě. Navzdory velice podobnému názvu se však zásadně liší od rozvrhování domácí péče, jelikož popisuje problematiku rozvrhování směn v nemocnici, tedy vůbec neobsahuje problém obchodního cestujícího – zdravotnický personál za pacienty nemusí cestovat. Na druhou stranu však zahrnuje mnoho důležitých aspektů problematiky HHCS, které naopak chyběly v TSP. Jedním z nich je specifikace dostupnosti sester – ta je

předem známa, stejně jako specifikace směn (čas, délka apod.). Mimo to také NRP pracuje s pojmy *soft* a *hard constraints* [35], viz kapitola 2.2. Konkrétní podoba obou typů omezení je odlišná pro každý konkrétní problém, tudíž je složité NRP generalizovat, avšak většinou mezi hard constraints patří nutnost splnění všech požadavků směny a nutnost přiřazení všech směn sestrám [35]. První zmiňované omezení je analogické k hard constraints pro asistenci v HHCS (např. požadované dovednosti nebo požadovaná konkrétní sestra); druhé omezení je v definici v kapitole 2.1 relaxováno do podoby soft constraint. Oproti problému rozvrhování domácí péče se NRP snaží přiřadit zdravotní sestry ke konkrétním směnám, které jsou předem známy a sestra má většinou přiřazenu nejvýše jednu směnu za den, přičemž u HHCS jsou přiřazovány sestry k jednotlivým asistencím, jejichž posloupnost ve výsledném rozvrhu teprve definuje směnu.

Jak je patrné z předchozí analýzy, problém rozvrhování domácí péče obsahuje prvky různých jiných problému z oblasti kombinatorické optimalizace. Z hlediska prostředí je nejvíce příbuzným *nurse rostering problem*, který pokrývá požadavky na soft a hard constraints, pomocí nichž lze modelovat složité situace reálného prostředí a vyjádřit určité preference, jejichž nesplnění neznamená neplatnost výsledného rozvrhu, ale pouze zhoršení jeho kvality. Co však NRP pokrýt nedokáže, je prostorová lokalita klientů. Na tu se zaměřuje problém obchodního cestujícího, konkrétně jeho modifikace *PVRPTW – periodic vehicle routing problem with time windows* [36]. Samotný VRP zahrnuje standardní požadavky m-TSP, tedy vícero sester, které cestují ke klientům, a zachování doby přejezdu mezi jednotlivými lokalitami a navíc také dobu trvání jednotlivých asistencí. Modifikace *with time windows* pokrývá požadavky časových oken, kdy si klient může objednat asistenci například v podobě "dvě hodiny dopoledne mezi 7. a 12. hodinou". Nakonec, modifikace *periodic* poukazuje na horizont plánování směn pro sestry na následující měsíc.

Z výše zmíněných důvodů lze považovat problém HHCS jako kombinaci NRP (nurse rostering problem) a PVRPTW (periodic vehicle routing problem with time windows) [36, 22]. Ne všechny předpoklady jsou sice splněny, například VRP předpokládá, že vozidla vyjíždějí z a vracejí se na stejné místo, nicméně problémy takového typu lze často vyřešit úpravou vstupního grafu přidáním nových vrcholů či hran s vhodně zvolenou cenou.

2.5 Třída složitosti

Jedním z hlavních důvodů, proč je problémům z oblasti rozvrhování či hledání optimální cesty v grafu věnováno tolik pozornosti v posledních několika dekádách, je jejich složitost. Než však určíme, do které třídy složitosti spadají, je třeba nejprve definovat několik pojmů a představit širší kontext problematiky.

V teorii složitosti se pracuje zejména s *rozhodovacími úlohami*, tedy s takovými problémy, jejichž řešení je odpověď ano, nebo ne [16]. Výše popsané problémy však patří do kategorie *optimalizačních úloh* (hledání přípustného řešení, které je podle určitých kritérií optimální), nicméně tyto úlohy lze transformovat na ekvivalentní rozhodovací úlohy [12]. Například rozhodovací verze problému obchodního cestujícího může znít následovně: *"Je dán graf* $G = (V, E)$ *s váhami hran* $c_{i,j} \in \mathbb{N}$ *a číslo* K*. Existuje minimální kostra, jejíž délka je nejvýše* K*?"* [16].

Dále, třídy složitosti udávají buď časovou, nebo paměťovou složitost. Pro jejich popis je využívána *Landauova notace* (neboli *omikron notace*, angl. *big O notation*) $\mathcal{O}$, která popisuje

asymptotické chování funkcí a omezuje jejich funkční hodnoty shora (obdobně existují také Ω notace pro omezení zdola a Θ notace pro omezení shora i zdola zároveň). Pokud je zkoumána časová složitost algoritmu, je popisován čas, který algoritmus potřebuje pro vyřešení instance o velikosti n. Pochopitelně, velikost instance není jediný indikátor, který ovlivňuje dobu běhu algoritmu – uvažme například úlohu řazení čísel algoritmem *bubble sort*. Doba běhu algoritmu se bude výrazně lišit, pokud bude vstupní pole již seřazeno, nebo pokud bude seřazeno sestupně, přestože tato pole budou mít stejnou velikost. Proto je nejčastěji využívána Landauova notace, která omezuje funkci shora, neboli analyzuje nejhorší případ pro danou instanci. Časová jednotka může být milisekunda, ale například také instrukční cyklus procesoru, díky čemuž zůstane funkce nezávislá na výkonnosti stroje. Nejlepší vstupní instancí pro *bubble sort* je právě již seřazené pole, v tom případě stačí pro vyřešení úlohy o velikosti pole n pouze n porovnání; naopak, pokud bude pole seřazeno sestupně, je třeba provést n^2 porovnání. Počítání paměťové složitosti probíhá analogicky. Formální definice Landauovy notace zní následovně:

Definice 2.5.1 *Je dána nezáporná funkce $g(n)$. Nezáporná funkce $f(n)$ patří do třídy $\mathcal{O}(g(n))$, pokud existuje kladná konstanta c a $n_0 \in \mathbb{N}$ tak, že $f(n) \geq c \cdot g(n) \quad \forall n \geq n_0$. [16]*

Definice 2.5.1 popisuje vztah funkcí. V následujícím textu bude pro zjednodušení používán obdobný vztah i pro složitost problémů, např. *Problém řazení patří do třídy $\mathcal{O}(n \cdot log(n))$*, což vyjadřuje fakt, že *Existuje algoritmus, který řeší problém řazení, jehož funkce popisující dobu běhu patří do třídy $\mathcal{O}(n \cdot log(n))$*.

Pomocí výše definovaných prostředků je možné formulovat třídy složitosti, které popisují množinu problémů majících stejnou složitost. První a pravděpodobně nejrozšířenější je třída $\mathcal{P}$, sdružující rozhodovací úlohy, které lze *vyřešit v polynomiálním čase* [12], tedy takové problémy, které patří do třídy $\mathcal{O}(n^k)$ pro nějaké nezáporné k. Formálně lze třídu $\mathcal{P}$ definovat následovně:

Definice 2.5.2 *Rozhodovací úloha $\mathcal{U}$ leží ve třídě $\mathcal{P}$, pokud existuje deterministický Turingův stroj, který rozhodne jazyk $\mathcal{L}_\mathcal{U}$ v polynomiálním čase. [16]*

Do této třídy patří mnoho problémů, mj. např. již výše zmiňované řazení pole, hledání minimální kostry v grafu, nejkratší cesta v acyklickém grafu a mnoho dalších. Nutno zmínit, že do třídy $\mathcal{P}$ patří také problémy složitosti $\mathcal{O}(n^k)$ pro *libovolně* velké k, jejichž řešení je prakticky nerealizovatelné pro velké vstupní instance.

Další, velice významnou třídou složitosti, je třída $\mathcal{NP}$, do které patří takové problémy, jejichž *řešení lze ověřit v polynomiálním čase* [12]. Formálně lze tuto třídu definovat obdobně pomocí Turingova stroje:

Definice 2.5.3 *Rozhodovací úloha $\mathcal{U}$ leží ve třídě $\mathcal{NP}$, pokud existuje nedeterministický Turingův stroj, který rozhodne jazyk $\mathcal{L}_\mathcal{U}$ v polynomiálním čase. [16]*

Jak je patrné z definice, $\mathcal{P} \subseteq \mathcal{NP}$, jelikož každý deterministický Turingův stroj je triviálně také nedeterministický. Zajímavá otázka však je, zda $\mathcal{P} = \mathcal{NP}$, tedy důkaz opačné inkluze:

$\mathcal{NP} \subseteq \mathcal{P}$. Odpověď na ni je však již několik desítek let nezodpovězená a otázka je jedním z problémů tisíciletí, které vyhlásil v roce 2000 Clayův matematický ústav [2].

Poslední dvě třídy složitosti, které zde budou uvedeny, jsou třídy $\mathcal{NPC}$ a $\mathcal{NP}$ hard. Před jejich definicí je však třeba zavést pojem *polynomiální redukce*:

Definice 2.5.4 *Rozhodovací úloha $\mathcal{U}$ se redukuje na rozhodovací úlohu $\mathcal{V}$ tehdy, jestliže existuje algoritmus, který pro každou instanci $\mathcal{I}$ úlohy $\mathcal{U}$ sestrojí instanci $\mathcal{I}'$ pro úlohu $\mathcal{V}$ tak, že $\mathcal{I}$ je ANO-instance $\mathcal{U}$ právě tehdy, když $\mathcal{I}'$ je ANO-instance $\mathcal{V}$. Pokud redukční algoritmus běží v polynomiálním čase, je redukce nazývána polynomiální. [16]*

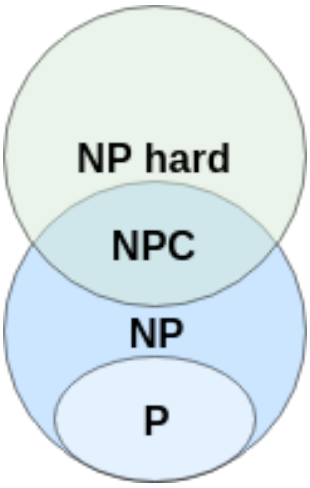
$\mathcal{NPC}$, neboli třída $\mathcal{NP}$ úplných úloh popisuje, neformálně řečeno, ty nejtěžší problémy z třídy $\mathcal{NP}$. Formálně je popsána následovně:

Definice 2.5.5 *Rozhodovací úloha $\mathcal{U}$ leží ve třídě $\mathcal{NPC}$, pokud $\mathcal{U}$ patří do třídy $\mathcal{NP}$ a každý problém ze třídy $\mathcal{NP}$ lze polynomiálně redukovat na $\mathcal{U}$. [40]*

Obdobným způsobem je formulována definice třídy $\mathcal{NP}$ hard, neboli $\mathcal{NP}$ obtížné úlohy:

Definice 2.5.6 *Rozhodovací úloha $\mathcal{U}$ leží ve třídě $\mathcal{NP}$ hard, pokud existuje nějaká úloha ze třídy $\mathcal{NPC}$, která se polynomiálně redukuje na úlohu $\mathcal{U}$. [40]*

Vztahy tříd složitosti $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NPC}$ a $\mathcal{NP}$ hard za předpokladu, že $\mathcal{P} \neq \mathcal{NP}$ jsou zobrazeny na obrázku 2.2.



Obrázek 2.2: Vztahy tříd složitosti, pokud $\mathcal{P} \neq \mathcal{NP}$

Porozumění vztahům mezi těmito třídami složitosti je nutné k pochopení složitosti problému rozvrhování domácí péče. Jak je zmíněno v kapitole 2.4, HHCS je kombinace problémů NRP (*nurse rostering problem*) a PVRPTW (*periodic vehicle routing problem with time windows*). Druhý zmiňovaný je generalizací problému VRP, který vychází z m-TSP, což je opět

[2] http://www.claymath.org/millennium-problems

rozšíření klasického problému obchodního cestujícího. Již však samotné hledání existence hamiltonovské kružnice, které je základem problému obchodního cestujícího, je problémem ze třídy $\mathcal{NPC}$ [16]. Z toho důvodu je zřejmé, že i rozhodovací verze TSP musí být $\mathcal{NP}$ úplná, protože se ptá, zda existuje hamiltonovská kružnice určité ceny v daném grafu [5]. Jelikož standardní, optimalizační verze problému obchodního cestujícího je alespoň stejně náročná jako rozhodovací verze, která patří do třídy $\mathcal{NPC}$, patří optimalizační verze TSP do třídy $\mathcal{NP}$ hard (viz obr. 2.2). Důkaz faktu, že je stejně, nebo více náročná než rozhodovací verze, je triviální, jelikož v rozhodovací verzi stačí najít pouze jednu hamiltonovskou kružnici, která bude mít stejnou nebo nižší cenu, než je dáno, kdežto v optimalizační verzi je nutno nalézt *všechny* hamiltonovské kružnice a následně vybrat tu nejlevnější. Navíc, optimalizační verze TSP patří mezi ty $\mathcal{NP}$ hard problémy, které *nepatří* do třídy $\mathcal{NPC}$, tedy nejsou ani ve třídě $\mathcal{NP}$, protože ta obsahuje pouze rozhodovací problémy.V této formulaci navíc není možné ověřit řešení algoritmu v polynomiálním čase. Pro důkaz správnosti řešení TSP je nutno dokázat, že nalezená hamiltonovská kružnice má nejnižší možnou cenu. To je však možné zjistit jedině tak, že nalezneme všechny možné hamiltonovské kružnice a nenajdeme žádnou, která by byla levnější, čímž však problém obchodního cestujícího vyřešíme a jak je zmíněno výše, řešení nelze nalézt v polynomiálním čase.

Druhou součástí problému rozvrhování domácí péče je NRP z kategorie scheduling problémů, které (v jejich obecné formě) patří do třídy $\mathcal{NPC}$ [15]. Oproti TSP se NRP liší tím, že zahrnuje celou množinu problémů, které se výrazně liší skladbou soft a hard constraints, tudíž nelze jednoznačně určit, do kterých tříd složitosti patří a do kterých ne. V literatuře je však tento problém většinou uváděn jako $\mathcal{NP}$ obtížný [15, 37], občas ale také jako $\mathcal{NP}$ úplný [35, 17] – záleží na skladbě omezení. Obě kategorizace však nejsou v kontradikci, viz obr. 2.2 a definice 2.5.6.

Problém HHCS, jakožto kombinace problému obchodního cestujícího a úlohy *nurse rostering problem*, patří do třídy $\mathcal{NP}$ hard [18, 23], jelikož oba podproblémy do této třídy patří také. Intuitivně lze říci, že je jedním ze složitějších problémů z této třídy, jelikož vyžaduje navíc několik modifikací oproti původním verzím těchto $\mathcal{NP}$ hard podproblémů. Ty jsou popsány v kapitole 2.4, pro připomenutí je to například vícero obchodních cestujících (zdravotních sester), časová okna, nebo výběr omezení. Zajímavé je tvrzení týkající se omezení, které říká, že pokud je problém VRP s nějakou hard constraint $\mathcal{NP}$ obtížný, zůstane ve stejné třídě složitosti, i pokud by bylo ono omezení nahrazeno jeho soft verzí [17].

2.6 Aproximační algoritmy a heuristiky

Mnoho optimalizačních problémů z reálného světa patří do třídy $\mathcal{NP}$ hard. Vzhledem k jejich složitosti je často prakticky nereálné nalézt optimální řešení pro velké instance, a to i přes stálý růst výpočetního výkonu, který předpovídá Mooreův zákon. Některé problémy mají navíc omezený čas pro výpočet, po jehož překročení pak nemají výsledky žádnou hodnotu. Tak je tomu i u problému rozvrhování domácí péče – plánování probíhá zpravidla na následující měsíc a požadavky klientů a sester jsou dostupné pouze s omezeným předstihem. Pokud by algoritmus nalezl řešení až po začátku daného měsíce, nebylo by možné rozvrh aplikovat, byť by byl jakkoliv kvalitní.

Z tohoto důvodu se pro řešení reálných problémů většinou využívá aproximačních či heuristických algoritmů. Pokud existuje aproximační algoritmus k řešení nějakého optima-

lizačního problému, zaručuje, že nalezne suboptimální řešení, přičemž kvalitu řešení udává *faktor* aproximačního algoritmu. Formálně jej lze definovat následovně:

Definice 2.6.1 *Aproximační algoritmus $\mathcal{A}$ optimalizačního problému $\mathcal{P}$ se nazývá k-aproximační, pokud existuje konstanta $k \geq 1$ taková, že pro všechny vstupní instance x problému $\mathcal{A}$ platí $A(x) \geq \frac{1}{k} \cdot P(x)$ (pokud je kriteriální funkce problému $\mathcal{P}$ maximalizována). [1]*

Jinými slovy, definice 2.6.1 říká, že k-aproximační algoritmus najde řešení, které bude nejhůře k-krát horší než optimální hodnota. Například $\frac{3}{2}$-aproximační algoritmus problému obchodního cestujícího zaručuje, že nalezená hamiltonovská kružnice bude mít nejhůře o polovinu vyšší cenu než optimální[3].

Oproti tomu heuristiky nezaručují kvalitu řešení, ani rychlost výpočtu. Heuristické algoritmy jsou často založeny na domněnce získané analýzou problému či vstupních dat a nalezení určitých vzorů. Není ani výjimkou, že heuristický algoritmus je použitelný pouze pro určité kategorie vstupních dat či po relaxování některých omezení, tedy neřeší problém univerzálně, avšak pro reálné použití může být mnohdy dostačující. Jejich častým nedostatkem a hlavním rizikem je uvíznutí v lokálním optimu, což omezuje kvalitu výsledku. Příkladem heuristik je např. hladový (*greedy*) algoritmus, lokální prohledávání nebo dynamické programování [19]. Pojmy heuristika a aproximační algoritmus jsou však v literatuře často zaměňovány a používány obecně pro pojmenování algoritmu, který hledá suboptimální řešení.

Velice silným nástrojem pro řešení složitých problémů z oblasti kombinatorické optimalizace jsou *metaheuristiky*. Na rozdíl od heuristik nejsou závislé na řešeném problému – obsahují množinu heuristik, které jsou vnímány jako black box, a určují způsob, kterým bude prohledáván prostor řešení, například mohou v průběhu adaptivně měnit parametry podřazených heuristik. Jinými slovy, metaheuristika je iterativní proces, který řídí chování heuristik, ze kterých se skládá, za cílem nalezení kvalitnějšího řešení [19]. Mezi typické příklady patří evoluční algoritmy, simulated annealing nebo iterativní lokální prohledávání.

Ještě vyšší úroveň abstrakce nabízejí *hyperheuristiky*. Ty, stejně jako metaheuristiky, operují nad množinou (meta)heuristik, avšak neprohledávají prostor řešení, ale prostor heuristik, a umožňují jejich výběr, kombinaci apod. Díky tomu se konkrétní hyperheuristika nemusí vztahovat k jednomu konkrétnímu problému, ale k celé třídě problémů [6]. Navíc také umožňují vhodnou kombinaci různých přístupů v různých fázích algoritmu – například zpočátku lze použít genetické programování a následně přejít ke zcela odlišné (meta)heuristice, např. simulated annealing. Tento výběr může být řízen dynamicky na základě aktuálního stavu algoritmu.

Jak je již napsáno na začátku této kapitoly, heuristické algoritmy jsou často voleným přístupem k řešení problémů z oblasti kombinatorické optimalizace a NRP a VRP/TSP nejsou výjimkou. Mezi používané techniky pro řešení NRP patří již zmiňovaný simulated annealing, genetické algoritmy nebo tabu search [7]; pro řešení vehicle routing problemu je pak využíván například algoritmus nejbližších sousedů (*nearest neighbour*), nebo Clarke-Wright algoritmus [31]. Podrobné analýze použitých heuristik pro řešení problému rozvrhování domácí péče se věnuje kapitola 3.

[3] Christofidův algoritmus pro metrické TSP [9]

2.7 Shrnutí

Rozvrhování domácí péče je náročnou úlohou, která v sobě zahrnuje dva $\mathcal{NP}$ obtížné problémy – nurse rostering problem, který používá soft a hard constraints, jež jsou důležitou součástí tohoto problému, a PVRPTW – generalizace jedné z nejpopulárnějších a nejstarších $\mathcal{NP}$ hard úloh, problému obchodního cestujícího. Standardním přístupem k řešení těchto složitých úloh jsou aproximační, nebo heuristické algoritmy, jelikož pro velká vstupní data je z důvodu jejich asymptotické složitosti prakticky nemožné nalézt optimální výsledek. Často používanou technikou jsou metaheuristiky, které umožňují kombinaci více různých heuristických algoritmů.

V konkrétní formulaci problému HHCS pro tuto práci je cílem vytvořit rozvrh přiřazených asistencí na následující měsíc. Objednané asistence lze specifikovat časovým oknem, nebo mohou být omezeny prostřednictvím soft nebo hard constraints, kterými jsou v současné době preferovaná sestra, požadovaná sestra a nutná kvalifikace/dovednost. Při návrhu je však nutno mít na paměti, že tato omezení se mohou v budoucnu změnit, či může dojít k přidání nových.

Kapitola 3

Rešerše

V kapitole 2 byla analyzována problematika úlohy rozvrhování domácí péče, její dekompozice a obecné přístupy k řešení podobných problémů. Cílem této kapitoly je provést důkladnou rešerši konkrétních stávajících přístupů k řešení HHCS, porovnat typy požadavků a omezení, které pokrývají, a popsat jejich efektivitu. Jednotlivé přístupy budou sdruženy do podkapitol podle typu použitého algoritmu.

3.1 Celočíselné lineární programování

Celočíselné lineární programování (ILP) je poměrně populárním přístupem k řešení optimalizačních problémů a rozvrhování domácí péče není výjimkou – v literatuře je tento přístup často volen pro porovnání s jinými, heuristickými algoritmy. Důvodem pro volbu celočíselného programování je zejména deklarativní způsob řešení problému, na rozdíl od ostatních algoritmů, které jsou většinou imperativní. Pro vyřešení problému rozvrhování domácí péče pomocí ILP stačí pouze formulovat problém a veškerá omezení pomocí soustavy rovnic a nerovnic a následně využít některého z existujících solverů pro výpočet. Ten slouží jako univerzální nástroj pro řešení lineárního programování – vstupem je ILP formulace libovolného problému a výstupem jsou optimální hodnoty všech proměnných. Mezi populární solvery patří například *Gurobi*[1], nebo *GLPK*[2].

Pro velké instance s velkým počtem proměnných a omezení je však výpočet pomocí ILP náročný, jelikož dokonce i pouhá rozhodovací verze celočíselného programování patří do třídy $\mathcal{NP}$ úplných problémů [16], na rozdíl od standardního lineárního programování, které je ze třídy $\mathcal{P}$. Optimalizační verze je $\mathcal{NP}$ hard úlohou, jelikož třída $\mathcal{NP}$ popisuje pouze rozhodovací úlohy. Vzhledem k tomu, že ILP formulace problému rozvrhování domácí péče pracuje s celočíselnými proměnnými a kriteriální funkce popisuje optimalizační kritérium, vztahují se k HHCS právě výše zmíněná rizika.

V článku od D. Guericke a kol. [22] je problém formulován pomocí smíšeného lineárního programování, přičemž je umožněno specifikovat čas asistencí pomocí časových oken, je zohledněna doba přesunu mezi jednotlivými klienty apod. Mimo základní atributy problému je dále také počítáno s pracovními regulacemi, mezi které patří například typ směny (ranní,

[1] https://www.gurobi.com
[2] GNU Linear Programming Kit, https://www.gnu.org/software/glpk

noční), rotace nepopulárních směn (práce o víkendu nebo o svátcích) tak, aby byly rozděleny spravedlivě mezi všechny sestry, pauzy během pracovní doby nebo omezení, která se vztahují k typu úvazku (např. pokud by sestra na částečný úvazek mohla sloužit pouze denní směny apod.). Tyto regulace lze chápat jako množinu soft a hard constraints, pokud by byl problém popsán podle definice v kapitole 2.1.

Kriteriální funkce použitá v této formulaci jednoduše minimalizuje délku všech cest, jelikož tato verze počítá s tím, že každá asistence musí být přiřazena právě jedné sestře. Ostatní vlastnosti problému a výše zmíněné regulace jsou popsány pomocí omezení ve formě rovnic a nerovnic. Problém je však jejich počet – těch je v této verzi uvedeno 36, což výrazně zvyšuje komplexitu a způsobuje praktickou nepoužitelnost algoritmu pro větší vstupní data. Z analýzy v článku vyplývá, že pro mnoho instancí nebyl nalezen výsledek, přestože algoritmus běžel 12 hodin; pro vybraná vstupní data algoritmus nenašel řešení v téměř polovině případů. Pouze pro instance s nejvýše patnácti klienty byl nalezen optimální rozvrh. Proto by v reálném prostředí, kdy mohou být sester a klientů desítky až stovky, tento přístup nemohl být využit. Navíc v případě, že by byla přidána další omezení, by se výkonnost algoritmu ještě více zhoršila.

Podobné výsledky měření popisuje ve svém článku i D. S. Mankowska et al. [27]. Formulace problému, kterému se věnuje, obsahuje stejně jako výše zmiňovaný článek časová okna, liší se ale zejména řadou omezení, mezi která patří například požadované kvalifikace pro výkon asistence, nutnost účasti dvou a více sester (např. u postižených, nebo obézních pacientů) a další. Efektivita algoritmů byla měřena na generovaných instancích, které se lišily velikostí (10 – 300 pacientů a 3 – 40 sester) a doba běhu pro MILP solver byla omezena na 10 hodin. Optimální výsledek pro instanci, která obsahovala 10 pacientů a 3 sestry, zvládl solver najít během 20 sekund, nicméně již pro druhou nejmenší (25 klientů a 5 sester) nebylo optimální řešení nalezeno v limitu deseti hodin. Pro tuto instanci však ještě byl nalezen alespoň nějaký přípustný výsledek – na žádné z větších instancí (více, než 50 klientů a 10 sester) se to již nepodařilo. Závěrem měření bylo doporučení použití heuristických algoritmů již pro instance s více než 25 klienty a 5 sestrami. Nutno zmínit, že oproti výše analyzovanému článku bylo pro ILP formulaci použito pouze 13 omezení.

Pozornost v tomto článku si také ještě zaslouží parametrizovatelná kriteriální funkce. D. S. Mankowska sice počítá s časovými okny, avšak jejich porušení je povoleno za cílem přiřazení více asistencí i za cenu, že budou ve výsledném rozvrhu mírně zkráceny. Tyto pozdní příchody jsou zahrnuty v kriteriální funkci. Článek popisuje tři různá kritéria – délka cesty sestry, suma pozdních příchodů a maximální zpoždění, jaké ve výsledném rozvrhu měla některá ze sester. Všechny tyto skutečnosti jsou zohledněny v kriteriální funkci; navíc jsou ještě vynásobeny konstantou, která může být před spuštěním algoritmu nastavena uživatelem. Díky tomu lze snadno udat priority jednotlivých kritérií.

Odlišnou strategii pro formulaci ILP použil ve svém článku Zhi Yuan a kol. [42]. Ten se věnoval tvorbě rozvrhu na následující týden, přičemž jednotlivá data asistencí nemusí být fixní, avšak klient může mít určité preference (analogie s penalizací porušených soft constraints). Mezi omezení patří například minimální počet dnů mezi ošetřeními u jednoho klienta pro určitý typ asistencí, maximální délka směny sestry nebo preferovaný pracovní den (časové možnosti sester jsou tedy udány volněji než v této práci). Formulace celočíselného lineárního programování není založena na problému obchodního cestujícího, jak je tomu v případech popsaných výše, ale je popsána pomocí modelu multikomoditních toků a obsahuje

12 omezení. Transformaci TSP na tento model popisuje C. E. Miller ve svém článku [28]. Kriteriální funkce je formulována tak, aby prioritou bylo využití co nejmenšího počtu sester pro vykonání všech požadovaných asistencí a v druhé řadě aby byly směny sester co nejkratší.

Závěry z výsledků měření jsou velice podobné jako závěry v obou článcích popisovaných výše. Jako vstupní instance byla tentokrát použita reálná data obsahující 99 pacientů, 9 sester a 460 objednaných asistencí. Jelikož experimenty ukázaly, že takto velká data nejsou spočitatelná pro ILP solver, byla z této instance extrahována data a vytvořeny menší instance. Jediný případ, kdy byla nalezena optimální hodnota, byla instance s pěti pacienty, dvěma sestrami a 22 asistencemi; pro větší data sice bylo nalezeno řešení, avšak již ne optimální. Instance, které obsahovaly více než 75 asistencí, nebyly vyřešeny v časovém limitu šesti hodin, navrhovaným řešením je opět heuristický algoritmus. Zajímavé je porovnání velikosti vygenerovaných ILP modelů pro nejmenší instanci (22 asistencí) a největší instanci (460 asistencí): nejmenší model obsahoval 5300 proměnných, 8400 omezení a 4300 nenulových koeficientů v matici omezení, zatímco největší byl složen z 8.5 milionu proměnných, 8.9 milionu omezení a 69.8 milionu nenulových koeficientů. Toto srovnání dobře ilustruje praktickou nepoužitelnost lineárního programování k řešení problému rozvrhování domácí péče, jelikož reálné instance jsou často ještě mnohem větší, než s jakými bylo počítáno v tomto článku.

3.2 Heuristiky

V kapitole 2.6 je uvedeno, že pro řešení problému rozvrhování domácí péče, stejně jako pro většinu dalších úloh ze třídy $\mathcal{NP}$, se v praxi využívá zejména heuristických algoritmů. Tento fakt dokazují i všechny články zmiňované v kapitole 3.1, ve kterých autoři analyzovali efektivitu solverů řešící HHCS pomocí celočíselného programování a ve všech případech představili heuristiky, pomocí kterých bylo nalezeno dostatečně kvalitní řešení pro výrazně větší instance, než tomu bylo pro ILP. Následující kapitoly se budou věnovat jednotlivým kategoriím heuristických algoritmů, které jsou v literatuře používány k řešení problému rozvrhování domácí péče.

3.2.1 Evoluční algoritmy

Prvním typem používaných metaheuristik jsou evoluční algoritmy (EA), které jsou inspirovány reálnou biologickou evolucí a využívají stejných principů, mezi které patří výběr nejsilnějších jedinců, jejich křížení, mutace apod. Evoluční algoritmus je, jakožto metaheuristika, iterativní proces, který prohledává prostor řešení tak, aby maximalizoval hodnotu kriteriální funkce, nazývané *fitness*. Obecné schéma evolučního algoritmu je popsáno pseudokódem 3.1.

EA jsou založeny na iterativním vylepšování populace, která je složena z jednotlivých jedinců, přičemž každý jedinec popisuje jedno možné řešení. Na začátku je vytvořena počáteční generace; ta může sestávat z náhodných jedinců, nebo může být vygenerována sofistikovanějším algoritmem, např. lokálním prohledáváním. V obou případech však musí jedinci reprezentovat přípustné řešení; velice důležitá je také diverzita populace, jelikož v opačném případě hrozí riziko konvergence do lokálního optima. Následně je spuštěn iterativní proces, který je ukončen podmínkou – tou může být například minimální hodnota fitness funkce

Algoritmus 3.1 Schéma evolučního algoritmu

```
P ← initializePopulation();
evaluateFitness(P);
BSF ← updateBSF(P);                                   ▷ best so far
while not terminationCondition() do
    P_s ← selection(P);
    P_c ← crossover(P_s);
    P_m ← mutation(P_c);
    evaluateFitness(P_m);
    BSF ← updateBSF(P_m);
    P ← replacement(P, P_m);
end while
return BSF;
```

(dostatečná kvalita řešení) za předpokladu, že je známa, nebo počet iterací (časové omezení algoritmu). V každé iteraci probíhají tři základní procesy [32]:

- **Selekce** – Výběr nejlepších jedinců z populace. Strategií může být například turnajová selekce, nebo ruletová selekce.
- **Křížení** – Křížení vybraných jedinců a vytváření nových. Typickými příklady jsou *single point crossover* nebo *uniform crossover*; zvolená strategie udává, které geny budou vybrány od rodičů k tvorbě potomka. Počet rodičů nemusí být omezen pouze na dva.
- **Mutace** – Mírná randomizovaná změna genů nových jedinců. Jejím cílem je zachovat diverzitu populace, případně nabízí šanci k tvorbě nových genů, které v populaci nejsou zastoupeny.

Pomocí kombinace těchto procesů je vytvořena nová generace. Její kvalita je po každé iteraci vyhodnocena výpočtem hodnoty fitness funkce a následně je aktualizován nejlepší nalezený jedinec (*BSF – best so far*). Nakonec je aplikován rekombinační operátor na původní a novou generaci – dvěma základními typy jsou *generační* strategie, kdy je původní populace kompletně nahrazena, a *steady-state* strategie, kdy je nahrazena pouze část populace novými jedinci.

Výhodou použití evolučních algoritmů k optimalizaci rozvrhování domácí péče je, stejně jako u ILP, deklarativní přístup k řešení problému. Při psaní programu stačí pouze popsat, jakým způsobem bude počítána fitness funkce (ta může být identická s kriteriální funkcí v ILP) a jaké evoluční operátory budou použity. Na druhou stranu, při neefektivní implementaci těchto funkcí nebo při nevhodném zvolení parametrů je u evolučních algoritmů velké riziko uvíznutí v lokálním optimu.

Y. Shi a kol. ve svém článku [34] používá k řešení HHCS genetický algoritmus (*GA*, podmnožina evolučních algoritmů). Verze problému, kterou popisuje, se více podobá klasickému VRP s časovými okny, jelikož sestry vyrážejí ze stejného místa a všechny končí v laboratoři, navíc vozidlem rozváží také léky a ostatní potřeby k jednotlivým pacientům. Kapacita

vozidla je omezena a pokud sestra zjistí, že nemá dostatek prostředků k vykonání požadované péče, musí se vrátit do nemocnice a chybějící léky vyzvednout. Problém se však liší od standardního VRP tím, že výše poptávky není známa dopředu a sestra ji vždy zjistí až u pacienta. V ostatních ohledech je problém stejný, jako standardní HHCS.

Genotyp jedince, tedy vnitřní reprezentace řešení, kterou každý jedinec vyjadřuje, je seznam pacientů (vrcholů grafu) v pořadí, ve kterém je daná sestra navštíví. Jelikož musí sestra za den zpravidla vícekrát navštívit nemocnici, aby doplnila léky, obsahuje genotyp seznam cyklů, které jsou odděleny těmito zastávkami v nemocnici. V počáteční populaci je část jedinců vygenerována náhodně (avšak tak, aby byl rozvrh validní) a část pomocí metody konstrukce cest, kterou autor popisuje v článku a která je založena na iterativním vylepšování částečné cesty. Jedinci vytvořeni touto metodou jsou již poměrně kvalitním řešením, jejich poměr vůči náhodně vygenerovaným jedincům v populaci lze nastavit parametrem, v experimentech prováděných autorem byla jeho hodnota 0.9 (90% jedinců je náhodně vygenerovaných), aby byla zachována dostatečná diverzita v populaci.

Počítání hodnoty fitness funkce je ztíženo tím, že skutečná poptávka léků pro jednotlivé klienty není dopředu známa. K jejímu odhadu je použita iterativní stochastická metoda; následně je simulována celá cesta sestry a jsou vypočítány indikátory popisující míru zpoždění a dodatečné cesty do nemocnice, které jsou jsou způsobeny nedostatkem prostředků ve vozidle. Cílem modelu je minimalizovat délku cest všech sester.

Co se týče selekce, nepopisuje autor použitou techniku, pouze, že jsou vybíráni silnější jedinci s lepší hodnotou fitness funkce; nicméně podotýká, že je algoritmus *elitistický* – tedy že malé množství nejlepších jedinců z předchozí generace je v nezměněné podobě přesunuto do nové generace bez aplikování genetických operátorů. Ke křížení jsou použiti dva jedinci, přičemž z každého je náhodně vybrán jeden cyklus (posloupnost vrcholů, která začíná v nemocnici a končí v laboratoři, kde sestra vyzvedává léky) a tyto cykly jsou následně mezi jedinci navzájem prohozeny. Jelikož ostatní cykly mohou obsahovat nově přidané vrcholy, jsou všechny případné duplicity odstraněny ze všech ostatních cyklů jedince. Operátorů mutace bylo použito několik; jsou založeny na principu výměny vrcholů v rámci jedné cesty/mezi cestami, nebo na vkládání vrcholů na jiná místa. U všech je však riziko, že výsledný rozvrh nebude platný, neboť může být porušena platnost časového okna – proto je třeba po každém aplikování mutace zkontrolovat splnitelnost rozvrhu. Po mutaci autor navíc ještě aplikuje operátor lokálního vyhledávání na dva vybrané jedince a snaží se přesunout některé z vrcholů podobně, jako je tomu u mutace, avšak s tím rozdílem, že hledá nejlepší možné místo v cestě, kam může být vrchol vložen a změnu aplikuje pouze tehdy, pokud vylepší hodnotu fitness funkce.

Efektivita algoritmu byla měřena na různě velkých instancích s 25 – 200 pacienty rozdělených do tří kategorií: klasické instance VRPTW používané i v rámci jiných článků, instance HHCS s deterministickou poptávkou a instance HHCS s nedeterministickou poptávkou. V prvním zmiňovaném případě bylo dosaženo téměř stejně kvalitních výsledků jako nejlepší zveřejněné hodnoty v literatuře, a to pro všechny velikosti instancí. V případech HHCS byl algoritmus testován na upravených instancích (25 a 100 klientů); co se týče kvality, výsledky pro vstupy s deterministickou poptávkou byly porovnávány s optimálními výsledky vypočítanými pomocí ILP, avšak pouze pro menší instanci, protože exaktní výsledek instance se 100 klienty nebyl nalezen v časovém limitu. Hodnota nejlepšího řešení GA byla však stejná jako výsledek nalezený ILP solverem. Pro instance s nedeterministickou poptávkou autor

popisoval kvalitativní kritéria výsledných rozvrhů specifických pro tento typ problému, které však nejsou důležité pro potřeby této práce. Významné však je, že pomocí představeného genetického algoritmu bylo možné vyřešit problémy všech velikostí; řešení pro největší instanci VRPTW s 200 pacienty nalezl algoritmus za necelých 9 minut. Při porovnání s velikostmi největších instancí, které byly vyřešeny ILP solvery (viz kapitola 3.1), je patrné, proč jsou k řešení problému rozvrhování domácí péče používány heuristické algoritmy místo výpočtu pomocí ILP.

3.2.2 Neighborhood search

Mezi jedny z nejpopulárnějších metaheuristik používaných k řešení problému rozvrhování domácí péče patří algoritmy založené na prohledávání okolí (*neighborhood search*). Jejich stavebním kamenem je lokální vyhledávání (*local search*) – heuristická metoda, která prohledává prostor kandidátních řešení tím způsobem, že aplikuje lokální změny na aktuální řešení, které je přijato, pokud zlepšuje hodnotu kriteriální funkce. Schéma algoritmu lokálního prohledávání popisuje pseudokód 3.2 [32]. Algoritmus má tendenci se pohybovat ve směru gradientu, což pochopitelně přináší riziko konvergence do lokálního optima. Tento problém je řešen různými technikami, mezi které patří například adaptivní úprava velikosti kroku nebo opětovné spouštění algoritmu z jiných výchozích bodů. *Okolí bodu* o velikosti d lze definovat jako všechna řešení, jejichž vzdálenost od aktuálního bodu je menší než d. Samotné měření vzdálenosti může mít různé podoby, které závisí na typu řešeného problému, například euklidovská/Manhattonská vzdálenost pro reálné prostory, Hammingova vzdálenost pro řetězce nebo rozdíl hodnoty kriteriální funkce pro HHCS a ostatní optimalizační problémy.

Algoritmus 3.2 Schéma lokálního prohledávání

```
BSF ← initialize(P);                                   ▷ best so far
while not terminationCondition() do
    x ← perturb(BSF);
    if betterThan(x, BSF) then
        BSF ← x;
    end if
end while
return BSF;
```

Jedním z příkladů řešení problému rozvrhování domácí péče pomocí prohledávání okolí je představen v článku od D. S. Makowské et al. [27], který je již zmiňován v kapitole 3.1, kde je rozebírána její ILP formulace problému a analýza výsledků. Pomocí metaheuristického přístupu prohledávání okolí se podařilo získat mnohem kvalitnější řešení za výrazně kratší dobu, navíc na větších instancích. Algoritmus je rozdělen do dvou fází; v první je vygenerováno počáteční řešení, které je přípustné. To probíhá iterativním procesem, kdy jsou jednotlivé asistence seřazeny podle priority, tedy konce časového okna, a následně přiřazeny na základě požadavků pacientů. Ve druhé fázi je zaveden pojem okolí, které je popsáno typem operace, která je aplikována na aktuální řešení – těmi jsou posun asistence na pozdější termín se stejnou sestrou, prohození dvou asistencí dané sestry, přesunutí asistence k jiné sestře ve stejný čas, nebo přesun k jiné sestře v jiný čas. V článku jsou zmiňována ještě další čtyři okolí pro analogické operace týkající se směn, k jejichž vykonání je nutná přítomnost

dvou sester najednou; ta jsou zde pro přehlednost vynechána, jelikož takový případ není v definici problému pro tuto práci uvažován. Jednotlivá okolí jsou sekvenčně prohledávána a pokud již nelze najít zlepšení v rámci jednoho okolí, přesune se hledání do dalšího, dokud nebudou prohledána všechna okolí.

Jak je patrné, průběh algoritmu silně závisí na pořadí, ve kterém budou okolí prohledávána. Pro snížení tohoto vlivu jsou všechna čtyři okolí spojena do jednoho, nazývaného spojené okolí (*merged neighborhood*) a analogická operace je provedena pro okolí týkající se asistencí vyžadujících dvě sestry. Zároveň je však představen alternativní přístup pro řešení tohoto problému, který je nazýván *adaptive variable neighborhood search* (AVNS), adaptivní prohledávání širokého okolí. Při této metodě jsou jednotlivá okolí nejprve ohodnocena – v každém z nich je provedeno několik kroků algoritmu a podle optimality je určena pozice daného okolí. Následně je spuštěno samotné prohledávání, při kterém jsou procházena tato okolí v určeném pořadí a v každém z nich je hledán nejlepší soused aktuálního řešení. Pokud je tento soused kvalitnější, bude přijat jako současné nejlepší řešení a hledání bude pokračovat v prvním okolí, v opačném případě se přesune do následujícího. Algoritmus je ukončen v případě, že není nalezeno další zlepšení nebo po vypršení časového limitu.

Výsledky obou těchto přístupů byly měřeny, stejně jako ILP formulace, na instancích o velikosti 10 – 300 pacientů a 3 – 40 sester. Zatímco pomocí ILP solveru nebylo v limitu deseti hodin nalezeno řešení pro instance větší než 25 pacientů, každou ze tří implementovaných metod se podařilo vyřešit všechny testovací instance. Těmito třemi metodami jsou lokální prohledávání (LS), spojené okolí a adaptivní prohledávání širokého okolí. Nejlepším přístupem na největších instancích a i na většině menších instancí bylo AVNS, avšak všechny metody přinášely poměrně kvalitní řešení, nejvýše o 15% horší než nejlepší nalezené. Na některých instancích s 50 pacienty a v jednom případě se 100 pacienty bylo efektivnější hledání ve spojeném okolí, výsledek AVNS byl však pouze maximálně o 3% horší. Výsledků blíže optimu však bylo dosaženo prostřednictvím AVNS po delší době (když byl stanoven časový limit 2 hodiny); pokud bylo prohledávání zastaveno po stejné době, jako běželo hledání ve spojeném okolí, byly výsledky mírně horší. Co se obecně týče doby běhu, všechny algoritmy běžely pro instance do 50 klientů méně než 1 sekundu. Na největší instanci běželo LS i vyhledávání ve spojeném okolí 40 minut, AVNS pak 116 minut, aby nalezlo řešení o necelých 5% lepší oproti LS (o 8% lepší oproti výsledku AVNS po čtyřiceti minutách běhu). Jak je vidět, prohledáváním okolí lze nalézt poměrně kvalitní výsledky použitelné i pro reálná data, a to každou z testovaných metod.

Jinou verzi algoritmu založeného na prohledávání okolí zvolila ve svém článku také již zmiňovaná D. Guericke a kol. [22]; tímto algoritmem je metaheuristika *adaptive large neighborhood search* (ALNS), která byla poprvé představena pro řešení VRP [33]. Jejím prvním krokem je, stejně jako u AVNS, inicializace počátečního řešení, nicméně na způsobu jeho generování příliš nezáleží. Následně je spuštěn iterativní proces, při kterém je v každé iteraci vybrán jeden operátor ničení (*destroy*) a jeden operátor opravy (*repair*). Obě tyto heuristiky modifikují aktuální řešení; destroy operátor mění řešení tak, že není přípustné (např. odebráním asistencí) a repair operátor ho následně změní zpět do přípustné podoby (např. vložením nové asistence). Výběr konkrétních heuristik probíhá metodou ruletové selekce, která je známa z evolučních algoritmů, viz kapitola 3.1. Každý destroy i repair operátor má svou váhu, přičemž tyto hodnoty jsou na začátku inicializovány na stejnou hodnotu a v průběhu algoritmu jsou upravovány – pokud aplikování vybrané heuristiky vylepšilo ře-

šení, je váha zvýšena, v opačném případě dojde k jejímu snížení. V každé iteraci je vybrána jedna heuristika z každé kategorie a platí, že čím větší váha, tím větší pravděpodobnost výběru. Zároveň však mají všechny heuristiky v každé fázi nenulovou pravděpodobnost, že budou vybrány. Díky tomuto mechanismu algoritmus častěji používá takové destroy a repair operátory, které v dřívějších iteracích nejvíce zlepšily řešení.

Při přijímání nového řešení je možné s malou pravděpodobností přijmout i řešení horší kvality, než je aktuální, za účelem diverzifikace a snížení pravděpodobnosti konvergence do lokálního optima. Tato pravděpodobnost je udána *teplotou* algoritmu (*temperature*) a funguje obdobně jako v metodě simulovaného žíhání (*simulated annealing*) [39]. Celý proces ALNS je spuštěn paralelně s různými váhami heuristik, přičemž po každé iteraci je upraveno aktuální řešení – pokud je nejlepší globální řešení lepší než nejlepší řešení sub-heuristiky, dojde k jeho nahrazení, v opačném případě algoritmus pokračuje se současným řešením. Úprava vah probíhá lokálně.

Autoři implementovali několik různých typů destroy operátorů, mezi které patří například odebírání náhodných asistencí, odebírání všech asistencí objednaných jedním konkrétním klientem, odebírání asistencí přiřazených sestrám s kvalifikacemi, které jsou řídce zastoupené, nebo odebírání směn o víkendu či ve svátek. Implementované repair operátory přidávaly nepřiřazené asistence do částečných rozvrhů sester, a to prostřednictvím hladového (*greedy*) konstrukčního algoritmu, nebo metodou vkládání (*insertion heuristic*). Konstrukční algoritmus byl využíván pro tvorbu nových rozvrhů (den, kdy sestra nemá přiřazené žádné asistence), nebo pro přidávání nových asistencí na konec existujících cest. Při vkládání více asistencí záleží na jejich pořadí, tudíž byl jejich seznam nejprve seřazen různými způsoby podle aktuálně důležitých kritérií. Insertion heuristiky slouží ke vkládání jednotlivých asistencí; zde autoři implementovali náhodný hladový algoritmus a *look-ahead* heuristiku, která počítala negativní vliv vložení asistence na dané místo a vybrala takovou pozici, která jej minimalizovala.

Měření výsledků ALNS probíhalo porovnáním s nejlepšími nalezenými výsledky MILP solveru; pro paralelní verzi algoritmu byla použita 4 vlákna. Na instancích, které byly vygenerovány autorem, dosáhla metaheuristika lepších výsledků v téměř polovině případů, zbytek byl horší o maximálně 5%. Směrodatná chyba byla navíc pouze 0.5%, tudíž algoritmus vykazuje srovnatelné výsledky v každém běhu. Na druhé množině instancí byly výsledky ALNS horší o nejvýše 10% v polovině případů, na menších instancích pak nejvýše 2%. Na poslední množině, která obsahovala méně pracovních regulací a ve které chyběla časová okna, se výsledky heuristiky lišily maximálně o 3%. Hlavním přínosem je však rozdíl doby běhu algoritmu, který činí pouhých 5 minut (oproti MILP, které bylo pouštěno s časovým omezením 12h). Lze tedy říci, že pomocí ALNS lze najít srovnatelně kvalitní řešení za 0.7% doby běhu MILP solveru.

3.2.3 Hyperheuristiky

Jelikož použití metaheuristik k řešení rozvrhování domácí péče může být efektivní pouze pro vybrané instance či v určitých fázích tvorby rozvrhu, není výjimkou jejich kombinace do hyperheuristických algoritmů (viz kapitola 2.6). Tímto způsobem lze různé metaheuristiky například skládat sekvenčně za sebou podle jejich efektivity v dané fázi, nebo je pouštět paralelně a vybírat nejlepší výsledky. Proto lze paralelní verzi ALNS z článku D. Guericke [22]

popsanou v kapitole 3.2.2 považovat za hyperheuristiku, jelikož jsou současně spuštěny verze algoritmů s různým nastavením vah a po každé iteraci může být upraveno jejich současné řešení, pokud některý algoritmus našel nový nejlepší rozvrh.

Hyperheuristický přístup, který adaptivně mění použité metaheuristiky v průběhu výpočtu, ve svém článku popsal M. Misir et al. [29]. Jím formulovaný problém HHCS je charakterizován několika modifikacemi standardní verze, mezi které patří například rozdílný způsob přesunu mezi klienty (autem, na kole, pěšky), rozdílná startovní lokace zdravotních sester, časová okna, kvalifikace sester či preferovaná sestra pro jednotlivé klienty. Při porušení některé ze soft constraints je řešení příslušně penalizováno. Kriteriální funkce zahrnuje tuto penalizaci, navíc minimalizuje celkovou dobu strávenou na cestě a nevyužitý čas sester (*idle time*) – doba, kdy sestra není ani na cestě, ani u klienta.

Výběr heuristiky probíhal dvěma způsoby. Prvním, primitivním z nich, je metoda *simple random*, která jednoduše vybere náhodnou heuristiku. Tento přístup dává stejnou pravděpodobnost výběru všem heuristikám a při analýze výsledků vykazoval poměrně dobré hodnoty, proto byl autorem článku použit. V ostatních publikacích autora byla pro výběr použita také adaptivní dynamická množina heuristik (*adaptive dynamic heuristic set*) [30], která v každé iteraci vypočítá skóre pro jednotlivé heuristiky podle jejich efektivity a všechny heuristiky, jejichž skóre je menší než průměr, jsou na několik iterací suspendovány (dočasně odebrány z množiny). Pro jakoukoliv změnu rozvrhu je aplikována jedna ze tří heuristik, která rozhodne, zda bude změna přijata, nebo ne:

- **Improving or equal (IE)** – řešení je přijato, pokud je lepší nebo stejně kvalitní jako současné
- **Iteration limited threshold accepting (ILTA)** – každé lepší nebo stejně kvalitní řešení je přijato, navíc je přijato i horší řešení, pokud nedošlo k jeho zlepšení za posledních $k \geq 0$ iterací
- **Adaptive ILTA (AILTA)** – adaptivní verze ILTA, hodnota prahové hodnoty k je dynamicky upravována v průběhu algoritmu

Samotné upravování rozvrhů probíhá prostřednictvím vybrání jedné z šesti nízkoúrovňových heuristik (*low level heuristics*), které aplikují primitivní změny na současný rozvrh. Tuto množinu tvoří následující heuristiky:

1. Prohození asistencí mezi sestrami
2. Prohození pořadí asistencí pro jednu sestru
3. Přesun asistence na nejlepší pozici v rozvrhu jiné sestry
4. Výběr nejhorší směny a její přesun na nejlepší pozici v rozvrhu jiné sestry
5. Výběr nejhorší směny a její přesun na nejlepší pozici v rozvrhu jiné sestry tak, že dojde k největšímu zlepšení
6. Výběr nejhorší směny a její přesun do nejprázdnějšího rozvrhu

Měření výsledků probíhalo na šesti instancích, které obsahovaly 7 – 15 sester, 30 – 74 klientů a 61 – 154 asistencí. Testováno bylo několik různých hyperheuristik: SR-AILTA, SR-ILTA a SR-IE, přičemž měření prvních dvou jmenovaných probíhalo v několika různých konfiguracích, které se lišily hodnotami parametrů. Nejlepších výsledků bylo dosaženo ve všech případech hyperheuristikou SR-AILTA. Co se týče monitorování využití nízkoúrovňových heuristik, nejčastěji suspendovanými byly heuristiky č. 4 a 5 a nejvhodnější hodnota parametru udávající délku suspendování byla 1 – 4 iterace. Kvalitu výsledného rozvrhu lze však bohužel těžko posoudit, jelikož ve článku je uvedena pouze nejlepší hodnota kriteriální funkce a heuristiky jsou porovnávány pouze mezi sebou.

G. Hiermann a kol. [23] provedl ve svém článku porovnání čtyř různých heuristik; jednou z nich je hyperheuristika založená na konceptu *simulated annealing* (SA). Formulace problému zahrnuje kromě standardních požadavků HHCS také kvalifikace sester, odmítané atributy (např. sestra s alergií na kočky nemůže provádět asistenci u klienta, který má kočku, některé klientky mohou odmítat asistenta, který je muž apod.), multimodalitu (rozdílné váhy hran v grafu vyjadřující různé způsoby transportu, např. autem či veřejnou dopravou) a před-alokované asistence (5% asistencí je fixně přiřazeno vybraným sestrám před spuštěním algoritmu). Kriteriální funkce vyjadřuje eliminaci porušených hard constraints, minimalizaci porušených soft constraints a minimalizaci doby strávené přesunem a přesčasy (jelikož jsou vypláceny s vyšší hodinovou sazbou).

Hyperheuristický algoritmus SA probíhá ve svou fázích, stejně jako většina algoritmů z článků popsaných výše – tvorba počátečního rozvrhu a jeho následné iterativní zlepšování. V první fázi je řešení generováno buď náhodným konstrukčním algoritmem, nebo pomocí programování s omezujícími podmínkami (*constraint programming – CP*). Ve druhé fázi je rozvrh iterativně vylepšován; prvky hyperheuristiky jsou tvořeny výběrem jedné z šesti nízkoúrovňových heuristik, vytvořených kombinací tří typů úpravy rozvrhu – přesun jedné asistence na nejlepší pozici v rozvrhu jiné sestry, přesun asistence na nejlepší možné místo v rámci téže sestry a kompletní prohození rozvrhů dvou sester. Pravděpodobnost výběru heuristiky se dynamicky mění na základě úspěšnosti v předchozí iteraci. Prvky SA jsou zastoupeny zavedením teploty algoritmu, která ovlivňuje, zda bude nově vygenerované řešení přijato, nebo ne. Změna teploty je popsaná funkcí tak, aby na začátku algoritmu bylo přijímáno cca 5% řešení a na konci prohledávání 0.5%.

Výsledky všech čtyř autorem představených algoritmů (variable neighborhood search, memetický algoritmus, scatter search a SA) byly testovány na instancích s průměrně 500 sestrami a cca 700 asistencemi. Nejlepších výsledků bylo dosaženo pomocí memetického algoritmu, zatímco simulated annealing hyperheuristika našla řešení v průměru o 15% horší, co se týče hodnoty kriteriální funkce. K největšímu zlepšení rozvrhu došlo u všech algoritmů během prvních 100 – 300 sekund, přestože byl časový limit doby běhu stanoven na 75 minut.

3.3 Shrnutí

Přestože je problematika HHCS poměrně novým problémem, existuje mnoho článků, které se jím zabývají a představují širokou škálu přístupů k jeho řešení. Konkrétní formulace problému se vždy liší, avšak většinou jsou uvažována časová okna, různé podoby soft a hard constraints (nejčastěji preference klientů a kvalifikace sester) a často také stejný výchozí a

koncový bod sester (nemocnice, laboratoř apod.). Cílem algoritmu, který je vyjádřen prostřednictvím kriteriální funkce, je téměř vždy minimalizace doby cestování (tj. součet cen použitých hran v grafu), dále pak také minimalizace penalizace za porušené soft constraints a eliminace porušených hard constraints. Méně často je pak brán zřetel na počet využitých sester, nevyužitý čas (idle time) či pracovní regulace.

Co se týče konkrétních algoritmů, téměř vždy je k řešení použita heuristika nebo nějaká forma aproximačního algoritmu. Autoři často nejprve formulují problém jako (M)ILP, který předloží solveru a výsledky následně porovnávají s jejich heuristickým algoritmem. Všechny případy použití ILP k řešení problému rozvrhování domácí péče představené v kapitole 3.1 dosahovaly však velice slabých výsledků a řešení bylo nalezeno pouze pro velice malé instance. Důvodem je počet omezení a počet proměnných, které činí problém příliš složitým, aby byl vyřešen v přijatelném čase pro objem vstupních dat, který je běžný v praxi. Pro hledání suboptimálních rozvrhů autoři zpravidla používají metaheuristiky, jejichž výhodou je jejich schopnost adaptace na aktuální stav řešení, například přizpůsobením parametrů nebo výběrem použité heuristiky. Jednou z nejčastějších metod je prohledávání okolí, mezi které patří například AVNS, nebo ALNS, představené v kapitole 3.2.2. Největším rizikem tohoto přístupu je uvíznutí v lokálním optimu; to je řešeno různými způsoby, např. adaptací velikosti kroku, opětovným spuštěním algoritmu z jiného startovního bodu nebo paralelním spuštěním více instancí stejného algoritmu s různými počátečními hodnotami parametrů a průběžným porovnáváním dílčích výsledků.

Další používanou metaheuristikou jsou evoluční algoritmy, jejichž největší výhodou je deklarativní způsob řešení problému. Riziko konvergence do lokálního optima je řešeno implementací evolučních operátorů, zejména mutace, nebo nastavením parametrů algoritmu (které mohou být měněny i dynamicky). Méně častým, ale velice silným nástrojem pro řešení HHCS jsou hyperheuristiky, které kombinují různé heuristické přístupy a dynamicky vybírají nejvhodnější algoritmy, které nejvíce zlepší aktuální řešení. Jejich hlavní výhodou je malá závislost na řešeném problému, jelikož využívají nízkoúrovňové heuristiky jako black box a pouze hodnotí jejich efektivitu; lze je tedy použít k řešení široké škály problémů, čehož lze docílit pouhou výměnou těchto heuristik.

Všechny popisované metaheuristické přístupy byly na rozdíl od ILP schopny nalézt suboptimální řešení pro instance reálných velikostí, navíc za zlomek času a proto jsou vhodným kandidátem k řešení problému. Pro stanovení důležitosti jednotlivých soft constraints a ostatních kritérií autoři využívají konfigurovatelné parametry, které jsou součástí kriteriální funkce a s jejichž pomocí lze nastavit priority bez nutnosti změny algoritmu. Mezi tyto hodnoty patří zejména penalizační konstanty (za nepřiřazenou asistenci, porušení omezení apod.), ale také parametry jednotlivých heuristik. Jejich nespornou výhodou je schopnost adaptace algoritmu na změny priorit a obecně větší volnost co se týče úprav optimalizačního kritéria; na druhou stranu však s sebou přítomnost parametrů přináší problémy související s jejich počátečním či optimálním nastavením.

Kapitola 4

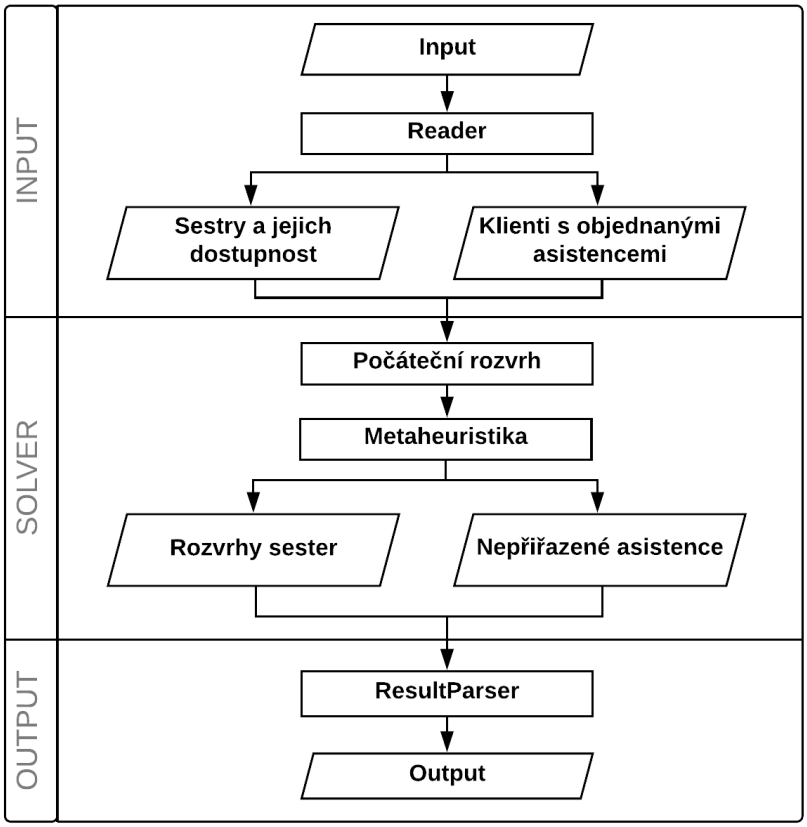
Návrh

Kapitola 3 byla zaměřena na rešerši stávajících přístupů k řešení problému rozvrhování domácí péče v literatuře. Pro každý algoritmus byl popsán řešený problém, přístup, který byl využit k nalezení suboptimálního rozvrhu, a byla provedena analýza efektivity. Cílem této kapitoly je navrhnout vlastní algoritmus, pomocí kterého bude řešen problém HHCS, jenž byl definován v kapitole 2.1. Návrh algoritmu bude vycházet ze znalostí získaných při analýze stávajících řešení, které budou přizpůsobeny požadavkům pro formulaci problému v této práci. Tato kapitola je zaměřena spíše na představení abstraktního frameworku pro řešení HHCS pomocí metaheuristik, jeho architekturu a popis komponent; konkrétní implementaci nízkoúrovňových heuristik a jednotlivých komponent se bude věnovat kapitola 5.

4.1 Architektura

Jedním z požadavků zmíněných v kapitole 2.1 je připravenost algoritmu na přidávání nových omezení či úpravu stávajících. Proto je při jeho návrhu kladen důraz na modifikovatelnost a nezávislost komponent, a to nejen co se týče soft a hard constraints, ale všech částí algoritmu v souladu se základními principy objektového návrhu – *high cohesion* a *loose coupling* [8]. Ty odkazují právě na stupeň závislosti jednotlivých komponent, který by měl být co nejnižší, a na sdružování funkcionality do logických celků zaměřených na řešení jednoho problému. Cílem této práce je nejprve vytvořit abstraktní framework pro řešení HHCS, nezávislý na použitém algoritmu, a následně implementovat konkrétní (meta)heuristiky, které bude tento framework využívat.

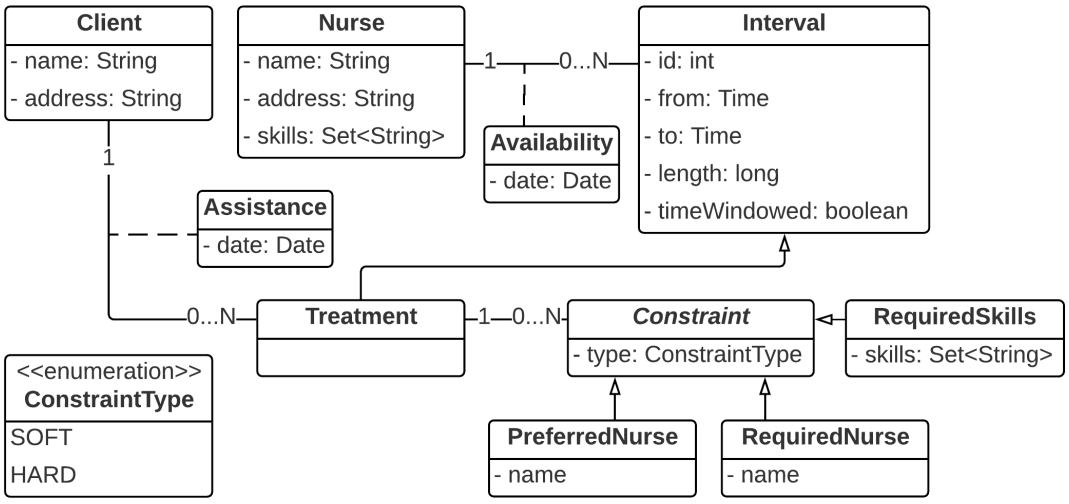
Přehled struktury frameworku je zobrazen na diagramu 4.1. Celý proces je rozdělen do tří částí – čtení uživatelského vstupu a transformace na vnitřní reprezentaci dat, samotné spuštění heuristiky tvořící výsledný rozvrh a zpracování výstupu. Implementace každého z procesů (které jsou znázorněny obdélníkem) může být snadno nahrazena (za předpokladu zachování stejného rozhraní) bez nutnosti zásahu do kódu ostatních komponent. Vstupními daty algoritmu je seznam zdravotních sester a jejich dostupnost na plánovací období (tím je v této práci následující měsíc, avšak algoritmus je schopen plánovat na libovolně dlouhé období) a dále seznam klientů a jimi objednané asistence. Spolu se vstupními daty jsou specifikována všechna omezení, časová okna apod. Komponenta `Reader` je zodpovědná za čtení těchto dat a jejich transformaci do vlastního modelu, který je popsán v kapitole 4.2.



Obrázek 4.1: Přehled struktury frameworku

V druhé části frameworku dochází k samotné tvorbě rozvrhu, která probíhá pomocí dvoufázového algoritmu, jenž je jedním z nejpoužívanějších přístupů v literatuře (viz kapitola 3.2). První fází je tvorba počátečního rozvrhu, jejímž výstupem je částečný rozvrh sester a seznam nepřiřazených asistencí. Druhou fází je metaheuristika, která iterativním procesem aplikuje změny na rozvrh, dokud není splněna ukončovací podmínka. Výstup je ve stejném formátu jako v předchozí fázi. Poslední část frameworku je zodpovědná za zpracování výsledku a transformaci vytvořeného rozvrhu a seznamu nepřiřazených asistencí do určité formy výstupu. Ten může mít libovolnou podobu, jako například výpis do konzole, vytvoření tabulky, nebo grafická reprezentace výsledného rozvrhu.

Framework však není pouze omezen na dvoufázový typ algoritmů použitých v této práci a lze snadno použít také na jednofázové algoritmy – implementace počátečního rozvrhu může totiž vytvořit pouze prázdný rozvrh a všechny asistence přesunout do seznamu nepřiřazených. Stejně tak je možné využití algoritmů, které nejsou založeny na iterativním přístupu, pomocí vhodné implementace nízkoúrovňových funkcí, mezi které patří ukončovací podmínka, nebo samotná heuristika. Struktura této části frameworku, tedy solveru, bude blíže představena v kapitole 4.3.



Obrázek 4.2: Class diagram modelu

4.2 Model

Struktura modelu, který je naplněn komponentou `Reader` a ve kterém jsou reprezentovány veškeré údaje o asistencích, dostupnosti klientů, současném rozvrhu a nepřiřazených asistencích, je znázorněna class diagramem 4.2. Nejvýznamnějšími entitami jsou `Client` a `Nurse`, které reprezentují klienty, respektive zdravotní sestry. Unikátním identifikátorem je v obou případech jméno; dále je vždy uvedena adresa, která slouží ke zjištění vzdálenosti mezi dvěma body z matice vzdálenosti (její vytvoření je popsáno v kapitole 5). Každá sestra může mít navíc libovolný počet kvalifikací, které můžou být vyžadovány pro některé objednané asistence.

Dostupnost sester a objednané asistence jsou v modelu popsány vazbami `Nurse-Interval` a `Client-Treatment`, přičemž obě definují atribut specifikující datum, ke kterému se vztahují. Dostupnost sester je popsána entitou `Interval`, která vždy obsahuje unikátní identifikátor pro snadnou referenci, začátek a konec intervalu, jeho délku a příznak, zda jde o interval definovaný časovým oknem. Pokud je jeho hodnota `true`, může být hodnota atributu `length` menší, než je doba trvání mezi začátkem a koncem intervalu, v opačném případě musí být roven této době. Asistence objednané klienty jsou reprezentovány téměř stejným způsobem, pouze jsou vyjádřeny entitou `Treatment`, která dědí od entity `Interval` a pouze přidává nepovinnou vazbu na omezení. Ta jsou popsána abstraktní třídou `Constraint`, která obsahuje pouze údaj, zda jde o soft, nebo hard constraint. Informace vztahující se ke konkrétnímu omezení jsou obsaženy v potomcích této entity; v diagramu jsou znázorněna tři omezení asistencí používaná v této práci, tj. preferovaná sestra, vyžadovaná sestra a nutné kvalifikace pro provedení ošetření. Díky tomuto abstraktnímu modelu lze snadno vytvořit nové typy omezení, nebo modifikovat stávající. Způsob transformace vstupních dat do tohoto modelu je popsán v kapitole 5.

4.3 Solver

Jak již bylo zmíněno výše, algoritmus je rozdělen do dvou fází, přičemž v první z nich je vytvořen částečný rozvrh a ve druhé je iterativním procesem vylepšován. Jelikož pro vstupní data nelze zaručit, že bude vždy možné každou asistenci přiřadit některé sestře, je v průběhu upravován kromě výsledného rozvrhu také seznam nepřiřazených prací.

Pro každou sestru jsou ukládány přiřazené asistence, jejich identifikace je určena *datem*, na které je objednána, *klientem*, který si danou asistenci objednal, *časem*, kdy sestra ke klientovi dorazí a unikátním *identifikátorem asistence* (atribut `id`, který je vygenerován pro každý `Interval`). Díky této struktuře lze snadno reprezentovat jak asistence s fixním intervalem, tak s definovaným časovým oknem; zároveň lze pomocí `id` získat informace o omezeních a dále také o klientovi (např. bydliště pro výpočet vzdálenosti). Nepřiřazené asistence jsou ukládány obdobným způsobem jako trojice datum asistence, její identifikátor a klient, kterým byla objednána.

4.3.1 Tvorba počátečního rozvrhu

Vstupními daty pro tvorbu počátečního rozvrhu je seznam sester a seznam klientů (tj. entit `Nurse` a `Client`), které obsahují objednané asistence a dostupnost sester. Konkrétní implementace této fáze algoritmu může mít libovolnou podobu, pouze musí zachovat kontrakt specifikovaný frameworkem, tedy musí objednané asistence přiřadit vybraným sestrám a zbytek uložit do seznamu nepřiřazených asistencí. Tímto způsobem lze implementovat libovolné požadavky, například je možné i zcela vynechat tuto fázi – stačí pouze všechny objednané asistence přesunout do seznamu nepřiřazených prací.

Výhodou rozdělení algoritmu a abstrakce obou fází je jejich tvárnost – vhodnou implementací lze snadno pokrýt mnoho dalších nových požadavků bez nutnosti zásahu do jakékoliv části existujícího kódu. Například v literatuře je často vyžadovanou funkcionalitou schopnost opětovného spuštění algoritmu s částečným rozvrhem nebo definice množiny před-přiřazených asistencí ke konkrétním sestrám. Oba požadavky lze ve fázi tvorby počátečního rozvrhu snadno implementovat například načtením těchto údajů ze souboru, případně přidáním omezení pro vyžadovanou sestru pro před-přiřazené asistence.

4.3.2 ALNS

Druhá fáze algoritmu slouží k implementaci metaheuristiky, jejímž prostřednictvím bude vylepšován rozvrh vytvořený v první fázi. Jednotlivé kroky algoritmu jsou popsány pseudokódem 4.1. První část, která je součástí frameworku, spočívání v iterativním zlepšování rozvrhu, které je vyjádřeno procedurou `enhance()`, jejíž implementace může být opět libovolná a pro jejíž změnu není nutné modifikovat kód mimo tuto proceduru. Iterativní proces probíhá do té doby, než je splněna ukončovací podmínka; její podoba je opět ponechána na uživateli. Mezi jednoduché příklady implementace patří časový limit, počet iterací nebo například stanovení prahové hodnoty kriteriální funkce, která může být interpretována jako "nejnižší přijatelná kvalita řešení". Tento způsob může být užitečný, pokud je známa optimální hodnota kriteriální funkce (nikoliv však podoba výsledného rozvrhu) a cílem je nalézt

Algoritmus 4.1 Druhá (iterativní) fáze algoritmu

```
schedule, unassignedJobs;                          ▷ output of the first phase
while not terminationCondition() do
    enhance(schedule, unassignedJobs);
end while

procedure ENHANCE(schedule, unassignedJobs)        ▷ ALNS implementation
    for each day ∈ planningHorizon do
        newSchedule, newUnassignedJobs ← copy(schedule, unassignedJobs);
        removalHeuristic(newSchedule, newUnassignedJobs);
        insertionHeuristic(newSchedule, newUnassignedJobs);
        if wasObjectiveFunctionImproved() then
            schedule = newSchedule;
            unassignedJobs = newUnassignedJobs;
        end if
    end for
end procedure
```

řešení, které je například alespoň z poloviny tak kvalitní. Pochopitelně je však zároveň třeba stanovit časový limit, jelikož heuristiky nezaručují, že rozvrh dané kvality bude nalezen.

Tělo procedury `enhance()` je příkladem možné implementace, která je inspirována heuristikou *adaptive large neighborhood search*, jež byla představena v kapitole 3.2.2. Tento přístup byl zvolen pro řešení problému v této práci; i samotný proces ALNS je však také napsán dostatečně abstraktně, aby bylo možné snadno změnit použité nízkoúrovňové heuristiky. Těmito heuristikami jsou operátory ničení (*destroy*) neboli *removal heuristics* a operátory opravy (*repair*) neboli *insertion heuristics*. Během každé iterace ve druhé fázi jsou pro každý den z plánovaného období aplikovány na aktuální řešení obě heuristiky a poté je spočítána nová hodnota kriteriální funkce. Pokud je nový rozvrh blíže optimu než původní, je tato změna přijata, v opačném případě algoritmus pokračuje dále beze změn v rozvrhu – jedná se tedy o *greedy* algoritmus. Jednou z výhod tohoto procesu je možnost paralelizace vnitřního cyklu přes všechny dny, neboť rozvrhy a objednané asistence pro jednotlivé dny tvoří disjunktní množiny, a tudíž mohou být zpracovávány ve stejný moment bez rizika souběžné modifikace.

Operátor ničení, který je aplikován na rozvrhy všech sester v jeden konkrétní den, je prvním krokem v této fázi. Zodpovědností konkrétních implementací, které budou představeny v kapitole 5, je analýza aktuálního rozvrhu a výběr asistencí, které budou odebrány a přidány do seznamu nepřiřazených asistencí. Ne nutně však musí dojít v každé iteraci k odebrání asistence (aby nedocházelo k destrukci kvalitních rozvrhů); stejně tak jich ale může být odebráno více najednou. Kritérium pro odebírání, priorita i počet iterací jsou vždy specifické pro konkrétní implementaci.

Druhý krok této fáze algoritmu je inverzní k předchozímu procesu, tedy dochází k přesunu nepřiřazených asistencí k jednotlivým sestrám. Stejně jako operátory ničení je tato heuristika ve frameworku použita abstraktně a konkrétní implementace může mít libovolnou podobu, pouze by měla zachovat specifikovaný kontrakt (tedy že se bude snažit přiřazovat asistence ze

seznamu jednotlivým sestrám). Důležitou vlastností tohoto způsobu implementace procesu ALNS je záruka splnitelnosti každého rozvrhu – jinými slovy, není přijata žádná změna, ve které je přiřazena asistence tak, že dojde k porušení některé hard constraint. Stejně tak je zachována pracovní doba sester, dojezdová vzdálenost apod. Na druhou stranu již ale není zaručeno, že budou všechny asistence přiřazeny; to však ale nemusí být možné ani v optimálním rozvrhu.

4.4 Interpretace výsledků

Poté, co bude splněna ukončovací podmínka (v pseudokódu 4.1 reprezentována funkcí `terminationCondition()`) se algoritmus přesune do poslední fáze, ve které dochází ke zpracování výsledného rozvrhu a nepřiřazených asistencí. Důvodem pro abstrakci i tohoto procesu je možnost adaptace na případné budoucí změny v požadavcích na výstup algoritmu – transformace výstupu není žádným způsobem vázán na algoritmus, pouze očekává na vstupu seznam nepřiřazených asistencí a výsledný rozvrh. Navíc lze také snadno použít vícero různých parserů, což je v praxi často požadováno – například tvorba samostatných souborů pro každou sestru, ve kterých bude pouze její rozvrh, dále stejná analogie s klientskými soubory (kdy dorazí která sestra, což je vhodné zejména v případě asistencí s časovými okny) a nakonec jeden velký soubor, kde bude celkový přehled všech sester a naplánovaných směn. Ten obsahuje všechna data a může významně zjednodušit proces manuálních ad hoc úprav rozvrhu, například v případě nemoci sestry, zrušení/přesunutí asistence apod.

4.5 Shrnutí

Návrh frameworku pro řešení problému rozvrhování domácí péče probíhal především s důrazem na modularitu, aby bylo možné snadno měnit implementaci jednotlivých komponent. Jejich vzájemné závislosti jsou minimalizovány v souladu s principy objektového návrhu (*high cohesion*, *loose coupling*), tudíž lze změny provádět bez zásahu do existujícího kódu. Konkrétním zvoleným přístupem pro hledání suboptimálního rozvrhu sester v této práci je metaheuristika založená na principu ALNS, avšak její výměna za jiný přístup je díky představenému návrhu velice snadná.

Výsledný rozvrh, dostupnost sester, objednané asistence a všechny ostatní informace jsou reprezentovány interním modelem, do kterého jsou transformována vstupní data. Jeho důležitým aspektem je abstrakce soft a hard constraints, které se vztahují k objednaným asistencím, díky čemuž je algoritmus připraven na snadnou integrací nových typů omezení, což bylo jedním z požadavků specifikovaných v kapitole 2.1. Transformace dosaženého výsledku algoritmu do čitelné podoby probíhá flexibilním procesem, s jehož pomocí lze jednoduše měnit formát výstupu, případně vytvořit více různých výstupů.

Samotná metaheuristika pro iterativní úpravu rozvrhu využívá dvou typů nízkoúrovňových operátorů (removal a insertion heuristiky), které ze současného rozvrhu odebírají, nebo do něj přidávají asistence za účelem jeho optimalizace. Několik konkrétních heuristik bude představeno v kapitole 5; jejich změna či nahrazení novou implementací je, stejně jako u ostatních komponent, velice snadná, jelikož i kostra ALNS metaheuristiky je navržena velice abstraktně.

Kapitola 5

Implementace

Framework představený v kapitole 4 tvoří modulární kostru pro řešení problému rozvrhování domácí péče. V této kapitole bude popsána implementace konkrétních heuristik, jejichž cílem je optimalizovat rozvrhy sester vzhledem k omezením asistencí a požadavkům, které byly formulovány v kapitole 2.1. Algoritmus je rozdělen do tří hlavních modulů – čtení vstupních dat, tvorba rozvrhu a transformace výsledků; v této kapitole budou postupně popsány různé způsoby implementace těchto částí, na kterých bude demonstrována modifikovatelnost algoritmu.

5.1 Použité technologie

Knihovna je implementována v programovacím jazyce Java, díky čemuž je program přenositelný mezi běžnými platformami. Pro spuštění je vyžadováno *JRE* (*Java Runtime Environment*) minimálně verze 8, jelikož v kódu je hojně využíváno funkcionalit, které nejsou dostupné ve starších verzích (zejména *Stream API* a lambda funkce [21]). Pro správu závislostí a sestavení (*build*) aplikace je využito nástroje *Apache Maven*[1], pomocí kterého lze snadno přidávat nové knihovny či upravovat jejich verze.

Mnoho komponent algoritmu vypisuje velké množství údajů, a to jak informativních, sloužící spíše pro debugování, tak statistických, které jsou velice důležité pro měření efektivity algoritmu, a díky nimž mohla být získána data pro kapitolu 7. K tomu je využita knihovna pro logování *Logback*[2], která poskytuje snadno konfigurovatelné API (*application programming interface* – aplikační programové rozhraní). Veškerá konfigurace knihovny, jako např. úroveň logování či typ výstupu (konzole či soubor) může být upravena v souboru `src/main/resources/logback.xml`.

Mimo samotný algoritmus byla v rámci této práce také implementována řada pomocných skriptů, které slouží například ke generování testovacích instancí nebo pro transformaci dat z reálného prostředí do požadovaného formátu vstupu, který algoritmus očekává. Implementace těchto skriptů je realizována v jazyce Python [3] a k jejich spouštění spolu s další logikou je využito také *bash* skriptů [4].

[1] https://maven.apache.org
[2] https://logback.qos.ch
[3] https://www.python.org
[4] https://www.gnu.org/software/bash

5.2 Načítání vstupních dat

Pro vytvoření interního modelu reprezentujícího konkrétní instanci problému je třeba předat algoritmu tři parametry – seznam zdravotních sester a jejich dostupnost v plánovaném období, seznam klientů a asistencí, které jimi byly objednány (spolu s případnými omezeními), a matici vzdáleností, která reprezentuje váhy hran mezi bydlišti sester a klientů.

Jako formát pro první dva parametry byl zvolen *.zip* soubor, který obsahuje seznam *.csv* souborů (*comma separated values*), přičemž každý z těchto souborů reprezentuje jednu sestru, případně klienta s objednanými asistencemi. Aby byl formát *.csv* souborů co nejvíce adaptibilní, jsou všechny informace uváděny dvojicí `<klíč><hodnota>`, kde klíč udává význam dat, které následují ve zbytku řádku a které tvoří hodnotu. Hodnota může být dále rekurzivně strukturována stejným způsobem, pomocí čehož lze reprezentovat např. omezení u objednaných asistencí. Klíče jsou implementovány pomocí enumu a k transformaci těchto dat do interního modelu slouží příslušné třídy využívající *Builder pattern* [41] (`ClientBuilder`, `NurseBuilder`). Ty mimo jiné také specifikují, jaké klíče mohou být použity pro danou entitu a provádějí validaci vstupních dat. Množina top-level klíčů je definována enumem `PersonData` a vypadá následovně:

- **NAME** – jméno sestry či klienta
- **ADDRESS** – adresa bydliště sestry či klienta
- **SKILLS** – množina kvalifikací/dovedností sestry
- **AVAILABILITY** – dostupnost sestry (následuje datum a interval, ve kterém může být dané sestře přiřazena nějaká asistence)
- **JOB** – asistence objednaná klientem (následuje datum a interval, který může být specifikován časovým oknem, a volitelně množina omezení)

Množinu omezení popisuje enum `ConstraintData`, jenž deleguje parsování omezení na příslušnou třídu podle hodnoty klíče. Aktuálně implementovaná omezení jsou následující:

- **PREFERRED_NURSE** – množina preferovaných sester (soft constraint)
- **REQUIRED_NURSE** – množina sester, kterým může být asistence přiřazena (hard constraint)
- **REQUIRED_SKILLS** – množina kvalifikací, které musí mít sestra, aby jí mohla být asistence přiřazena (hard constraint)

Díky tomuto oddělení je možné nejen snadno přidávat nová omezení, k čemuž stačí pouze přidání nového klíče a implementace odpovídajícího parseru, ale také jakýchkoliv dalších atributů sester či klientů. Samotné čtení ze *.zip* a *.csv* souborů zajišťují třídy `ZipReader` a `CsvReader`, které lze samotné považovat za knihovnu, jelikož nabízejí abstraktní kostru pro čtení souborů v libovolné podobě. `ZipReader`u stačí předat lambda funkci, která udává, jaká akce bude provedena s každým souborem ve vstupním *.zip* archivu a jaká je návratová hodnota; stejně tak `CsvReader` přijímá funkci, která bude zavolána pro každou řádku vstupního

souboru. Kompozicí těchto dvou tříd lze transformovat vstupní soubor do výsledného modelu pouze pomocí volání dvou metod. Univerzální použitelnost těchto dvou tříd podtrhává použití generických parametrů a lambda funkcí.

Posledním parametrem je matice vzdáleností. Jelikož jsou ve formulaci problému pro tuto práci váhy hran statické, je očekávána na vstupu matice o velikosti n^2, kde n je počet vrcholů (tj. sjednocení množin bydlišť sester a klientů). Matice je předávána ve formátu *.csv*, ve kterém má každý řádek podobu `<A><B><cost>`, kde `A` a `B` jsou jednotlivé vrcholy a `cost` je vzdálenost mezi nimi. V modelu je matice reprezentována třídou `DistanceMatrix`, která provádí základní validace, např. jestli platí, že vzdálenost $A \to A = 0$, nebo zda jsou hodnoty ekvidistantní (tj. vzdálenost $A \to B = B \to A$). K vytvoření této matice lze využít softwaru třetích stran, který může používat např. euklidovskou vzdálenost nebo průměrnou denní dobu cesty vybraným dopravním prostředkem.

5.2.1 Transformace ze stávajícího formátu

Jelikož stávajícím formátem pro specifikaci dostupnosti a objednaných asistencí byly *.xlsx* soubory, bylo nutné vytvořit transformační skripty pro testování algoritmu na reálných datech. Každý klient je aktuálně reprezentován jedním *.xlsx* souborem popisujícím objednané asistence spolu se sestrami, které byly k daným asistencím přiřazeny, jelikož rozvrhování bylo prováděno manuálně. Soubor pro sestry zase udává jména klientů a časy, ve které se má sestra ke klientovi dostavit.

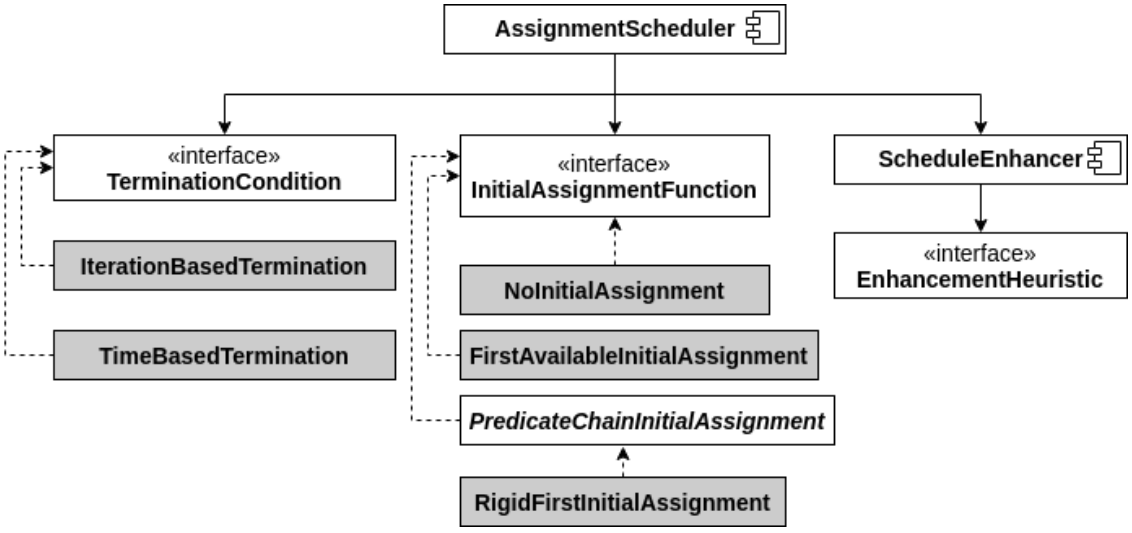
Pro účel transformace byl vytvořen skript `transformXlsx.py`, který z původního *.xlsx* souboru vytvoří *.csv* ve formátu popsaném výše, jenž je použitelný pro implementovaný algoritmus. Pro zabalení do *.zip* souborů slouží bash skript `transformData.sh`, který volá výše zmiňovaný Python skript a vytvoří *.zip* soubory, obsahující *.csv* pro všechny sestry a klienty. Tyto skripty jsou umístěny v adresáři `scripts` v kořeni projektu.

5.3 Solver

Druhou a nejvýznamnější fázi algoritmu zaštiťuje třída `AssignmentScheduler`. Schéma procesů a závislostí jejích komponent je zobrazeno na obr. 5.1. Šedě zvýrazněné třídy značí konkrétní implementaci, která může být snadno nahrazena; plná čára značí závislosti komponent a přerušovaná čára se šipkou značí dědičnost. Většina rozhraní je deklarována jako funkční rozhraní (*functional interface*) [21], což umožňuje použití lambda funkcí místo implementace třídy.

5.3.1 Tvorba počátečního rozvrhu

Základními kroky druhé fáze algoritmu jsou tvorba počátečního rozvrhu (rozhraní `InitialAssignmentFunction`) a iterativní vylepšování rozvrhu (třída `ScheduleEnhancer`). Výstupem první části je seznam nepřiřazených asistencí a částečný rozvrh, který zaručuje, že asistence v něm přiřazené neporušují žádnou z hard constraints. Aktuálně je implementováno několik strategií; první z nich je třída `NoInitialAssignment`, která pouze vytvoří prázdný rozvrh a všechny objednané asistence přesune do seznamu nepřiřazených. Využití této třídy



Obrázek 5.1: Diagram závislostí třídy AssignmentScheduler

je zejména pro měření efektivity druhé fáze algoritmu za cílem vhodného nastavení hodnot parametrů.

Druhá, reálně použitelná implementace je třída `FirstAvailableInitialAssignment`, jejíž logika je popsána pseudokódem 5.1. Ta vytváří rozvrh tím způsobem, že pro každý den iteruje přes seznam objednaných asistencí a snaží se najít první sestru, která je dostupná v daném termínu. Pokud je nalezena, je zkontrolováno, zda již nemá ve stejný čas přiřazenou jinou kolidující asistenci a je provedena validace, zda nedochází k porušení doby cesty z předchozí lokace, stejně tak jako doby cesty k dalšímu klientovi. Pokud lze i přesto asistenci přiřadit a není porušena žádná hard constraint, je asistence umístěna do rozvrhu a smazána ze seznamu nepřiřazených. V případě asistencí s časovými okny je však více možností, kam lze danou asistenci do rozvrhu umístit; v takovém případě algoritmus najde nejdřívější možný termín a přiřadí asistenci tam. V případě, že z některých uvedených důvodů nelze asistenci přiřadit nebo pokud není nalezena žádná sestra, která by byla v objednaném termínu dostupná, je asistence přidána do seznamu nepřiřazených asistencí. Aby mohly být výše popsané funkce použitelné i v ostatních algoritmech, je veškerá funkcionalita sloužící k přiřazování asistencí extrahována do třídy `AssignmentService`.

Třetí způsob implementace první části tvorby rozvrhu je realizována třídou `RigidFirstInitialAssignment`, jejímž cílem není přiřadit maximum možných asistencí v první fázi, ale soustředit se na přiřazování těch asistencí, které jsou co nejvíce omezovány a mají minimum míst v rozvrhu, na která mohou být umístěny. V rámci testování bylo vyzkoušeno několik různých přístupů k implementaci tohoto požadavku, nicméně nakonec se ukázal jako nejlepší způsob pokusit se přiřadit pouze ty asistence, které nejsou objednány s časovým oknem (tedy jejich časová pozice v rozvrhu je fixní) a navíc jsou omezeny alespoň jednou hard constraint (tudíž i množina sester, kterým může být přiřazena, je menší). Jelikož ostatní asistence, jejichž čas je flexibilní nebo které nemají hard constraint, mají více možností umístění do rozvrhu, dává smysl pokusit se o umístění rigidních asistencí v první fázi do prázdného rozvrhu. Pokud by jejich umístění nebylo ideální, stále mohou být odebrány

Algoritmus 5.1 Tvorba počátečního rozvrhu třídou FirstAvailableInitialAssignment

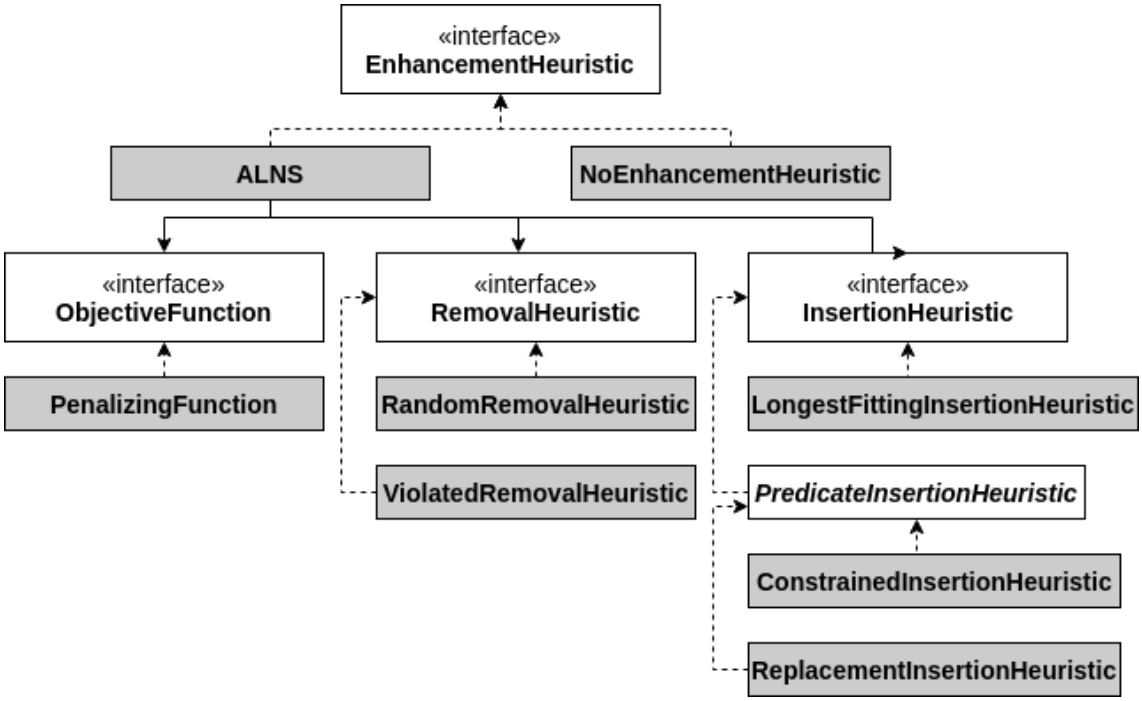
```
schedule, unassignedJobs, nurses                          ▷ schedule is empty
for each day ∈ planningHorizon do
    unassignedOnDay = unassignedJobs.get(day)
    for each unassignedJob ∈ unassignedOnDay do
        for each nurse ∈ nurses do
            if canBeAssigned(nurse, unassignedJob) then
                unassignedJobs.remove(unassignedJob)
                schedule.add(unassignedJob, nurse)
            end if
        end for
    end for
end for
```

ve druhé fázi algoritmu removal heuristikou.

Aby byla implementace co nejvíce znovupoužitelná pro případné další metody tvorby počátečního rozvrhu, byla většina logiky `RigidFirstInitialAssignment` přesunuta do abstraktní třídy `PredicateChainInitialAssignment`, díky níž má implementace potomka pouze jediný řádek. Myšlenka za touto abstrakcí byla vytvořit kostru pro iterativní tvorbu částečného rozvrhu na základě prioritizovaného seznamu vlastností asistencí, přičemž v každé iteraci by byla přiřazována pouze podmnožina nepřiřazených asistencí. Příkladem může být například následující požadavek – nejprve jsou přiřazeny asistence bez časových oken, které mají alespoň jednu hard constraint, následně jsou (již do částečného rozvrhu) umístěny zbývající asistence bez časových oken (které již nemusí mít omezení) a až nakonec všechny ostatní asistence. Implementace je založena na předání seřazeného seznamu predikátů (třída `Predicate` z Javy 8), který je postupně procházen a pro každý predikát je vyfiltrován seznam nepřiřazených asistencí, jež jsou následně přidělovány dostupným sestrám. Poté, co je zpracována celá množina, je celý proces opakován s následujícím predikátem s tím, že částečný rozvrh i seznam nepřiřazených asistencí se již mohl změnit. Tento proces je využit právě třídou `RigidFirstInitialAssignment`, avšak v tomto případě se ukázalo jako nejvýhodnější řešení použít pouze jediný predikát popsaný výše a nechat zbytek asistencí na zpracování pro heuristiky ve druhé fázi algoritmu.

5.3.2 Ukončovací podmínka

Ve druhé fázi algoritmu je metaheuristický proces opakován tak dlouho, dokud nedojde ke splnění ukončovací podmínky, která je reprezentována rozhraním `TerminationCondition`. Její dvě implementace, které jsou v této práci používány, jsou `IterationBasedTermination`, která stanovuje fixní počet iterací, po jehož překročení bude algoritmus ukončen, a `TimeBasedTermination`, pomocí níž je možno předem zadat maximální dobu běhu programu. Dalšími možnými implementacemi by mohla být například hodnota kriteriální funkce (tj. kvalita rozvrhu), počet nepřiřazených asistencí nebo procento porušených soft constraints.



Obrázek 5.2: Diagram závislostí třídy ALNS

5.3.3 ALNS

Samotné volání konkrétní metaheuristiky je iterativně vykonáváno třídou `ScheduleEnhancer` (dokud není splněna ukončovací podmínka). Pro řešení problému HHCS v této práci byla za implementaci zvolena metaheuristika inspirovaná procesem ALNS, která je reprezentována stejnojmennou třídou, přehled jejích závislostí je zobrazen na diagramu 5.2. Konfigurace této třídy je možná nastavením tří parametrů, kterými jsou implementace kriteriální funkce, heuristiky pro přidávání asistencí a heuristiky pro odebírání asistencí z rozvrhu. V samotném těle funkce, která je volána každou iteraci, je pro každý den zkopírován rozvrh všech sester a na tuto kopii je aplikována nejprve removal heuristika a následně insertion heuristika. Poté je spočítána hodnota kriteriální funkce, a pokud je po provedených změnách kvalita rozvrhu blíže optimu, je změna přijata. V opačném případě je výsledek ignorován a algoritmus pokračuje s další iterací.

Jelikož každá konkrétní implementace removal a insertion heuristiky se může soustředit na vylepšování rozvrhu z hlediska různého kritéria, může být v určitých případech vhodné kombinovat vícero heuristik dohromady, aby bylo dosaženo suboptimálního výsledku z hlediska více protichůdných kritérií. Z toho důvodu může být v konfiguraci ALNS uvedeno více removal i insertion heuristik s tím, že v každé iteraci je vybrána jedna heuristika, která bude použita. Pro výběr heuristiky je využita ruletová selekce, která je známa z evolučních algoritmů – pravděpodobnosti výběru jednotlivých heuristik lze specifikovat v konfiguračním souboru viz kapitola 5.4.

Tabulka 5.1: Proměnné pro kriteriální funkci

Proměnná	Popis
N	Množina všech sester
J	Asistence přiřazané vybrané sestře v daný den
S	Množina soft constraints
H	Množina hard constraints
U	Množina nepřiřazených asistencí
l_j	Délka asistence j
p_s	Penalizace za porušení soft constraint s
p_u	Penalizace za nepřiřazenou asistenci
v_s^j	1, pokud přiřazená asistence j porušuje soft constraint s, jinak 0
v_h^j	1, pokud přiřazená asistence j porušuje hard constraint h, jinak 0
i	Idle time – počet minut, kdy daná sestra není ani u klienta, ani na cestě
p_i	Penalizace za idle time
f_a	(Kladný) faktor za přiřazenou asistenci

5.3.3.1 Kriteriální funkce

Jednou z nejdůležitějších částí algoritmu určujících jeho kvalitu je kriteriální funkce. Její podoba je kritická mj. pro implementaci protichůdných kritérií, kterými je například vyvážení počtu nepřiřazených asistencí a porušených soft constraints. Z matematického pohledu je to zobrazení, které každému rozvrhu přiřadí jednu hodnotu (většinou číslo), které vyjadřuje jeho kvalitu. Rozhraní `ObjectiveFunction` umožňuje specifikovat, zda je funkce rostoucí, nebo klesající – tedy zda vyšší hodnota kriteriální funkce vyjadřuje kvalitnější rozvrh, nebo naopak.

Pro účely této práce byla implementována třída `PenalizingFunction`, jejímž cílem je ohodnotit rozvrh a penalizovat všechny porušené soft constraints či jiné negativní aspekty. Tento přístup je v literatuře poměrně standardní, zejména je využíván ve formulaci problému rozvrhování domácí péče pomocí ILP. `PenalizingFunction` je rostoucí funkce, která je konfigurovatelná několika parametry.

$$F_n = \sum\nolimits_{j \in \mathcal{J}} l_j \cdot f_a + \sum\nolimits_{s \in \mathcal{S}} v_s^j \cdot p_s + \sum\nolimits_{h \in \mathcal{H}} v_h^j \cdot -\infty + i \cdot p_i \tag{5.1}$$

$$F_u = \sum\nolimits_{u \in \mathcal{U}} l_u \cdot p_u \tag{5.2}$$

$$min(\sum\nolimits_{n \in N} F_n) + F_u \tag{5.3}$$

Kriteriální funkce pro vybraný den (5.3) se skládá ze dvou částí. První část (5.1) udává hodnotu kritéria pro přiřazené asistence, přičemž druhá část (5.2) penalizuje nepřiřazené asistence. Konfigurovatelné parametry z tabulky 5.1 jsou f_a, p_i, p_s a p_u. Hodnotu funkce zvyšují přiřazené asistence, přičemž čím je asistence delší, tím více zvýší hodnotu funkce; faktor zvýšení lze ovlivnit nastavením parametru f_a. Ke snížení hodnoty dojde při nepřiřazení asistence (parametr p_u), přičemž opět platí, že čím delší je nepřiřazená asistence, tím více

bude hodnota snížena. Dále je také počítán *idle time*, tedy čas, kdy sestra není ani u klienta, ani na cestě. Tento čas není efektivně využit, a proto je podle délky penalizován s faktorem p_i. Nakonec hodnotu kriteriální funkce snižuje porušení omezení – v případě soft constraints je to faktor p_s, který lze nastavit zvlášť pro každý typ omezení; v případě hard constraints je to hodnota $-\infty$ (v implementaci je použito vysoké záporné číslo, tento parametr je v ILP známý jako *big M*). Ta však slouží spíše jako bezpečnostní kontrola, jelikož heuristiky by neměly vytvářet rozvrhy, ve kterých jsou porušeny hard constraints. Hodnoty všech penalizačních faktorů s výjimkou f_a mají zpravidla zápornou hodnotu.

5.3.3.2 Removal heuristiky

Prvním krokem ALNS metaheuristiky je odebrání přiřazených asistencí ze stávajícího rozvrhu, k čemuž slouží funkční rozhraní `RemovalHeuristic`. První, přímočará implementace je odebírání náhodných asistencí, které je realizováno třídou `RandomRemovalHeuristic`. Ta z rozvrhu každé sestry odebere n náhodných přiřazených asistencí (pokud je rozvrh dostatečně dlouhý), přičemž tento počet lze konfigurovat. Odebrané asistence jsou přidány do seznamu nepřiřazených asistencí, odkud jsou následně vkládány zpět do rozvrhu insertion heuristikou. Přestože se může jevit tento přístup poněkud naivním, v literatuře je obdobný princip často používán a přináší poměrně uspokojivé výsledky [22].

Druhou, sofistikovanější removal heuristikou je třída `ViolatedRemovalHeuristic`. Ta odebírá taktéž n asistencí, avšak pořadí odebrání je dáno mírou porušení omezení. Jinými slovy, asistence jsou odebírány v pořadí od té, která je nejvíce penalizována. Využití této heuristiky je vhodné, pokud je primárním cílem minimalizovat počet porušených soft constraints. Tato skutečnost je již zahrnuta v implementaci kriteriální funkce, avšak třída `ViolatedRemovalHeuristic` aktivně míří k dosažení tohoto cíle.

5.3.3.3 Insertion heuristiky

Dalším, inverzním krokem k předchozímu je přesouvání asistencí ze seznamu nepřiřazených do současného rozvrhu. První z implementací rozhraní `InsertionHeuristic` je třída `LongestFittingInsertionHeuristic`, jež je popsána pseudokódem 5.2 a jejímž cílem je minimalizovat idle time zdravotních sester. Ta nejprve identifikuje všechna časová okna, ve kterých sestry nemají přiřazenou žádnou asistenci, a následně se je snaží vyplnit některou ze seznamu nepřiřazených asistencí. Pokud je nalezena taková asistence, která může být na dané místo přiřazena, dojde k přepočítání času přejezdu a uložení nového počtu idle minut, což je ta část identifikovaného časového okna, která nebyla tímto přiřazením zaplněna. Tento proces je opakován pro všechny sestry a celý seznam nepřiřazených asistencí a následně je přiřazena taková asistence, která danou mezeru v rozvrhu nejlépe vyplňuje. Nutno podotknout, že taková asistence nemusí být nejdelší – pokud by bylo rozhodováno mezi dvouhodinovou asistencí, po jejímž přiřazení z časového okna zůstane sestře ještě půl hodiny idle time, nebo mezi hodinovou asistencí, která na dané místo pasuje perfektně, tj. po dokončení předchozí asistence vyrazí sestra rovnou k dalšímu klientovi, bude preferována druhá možnost. Celý tento proces je, stejně jako u removal heuristik, opakován n krát.

Alternativní způsob vkládání asistencí do částečného rozvrhu sester nabízí třída `ConstrainedInsertionHeuristic`, která se, podobně jako `RigidFirstInitialAssignment`

Algoritmus 5.2 Insertion heuristika LongestFittingInsertionHeuristic

```
assignedJobs, unassignedJobs, nurses  ▷ assigned and unassigned jobs on a specific day
nurseToAssign, jobToAssign, idleMinutes = ∞
for each nurse ∈ nurses do
    assignedOnDay = assignedJobs.get(nurse)
    idleWindows = assignedOnDay.getIdleWindows()  ▷ available for new assignment
    for each unassignedJob ∈ unassignedJobs do
        windowLength = idleWindows.getForJob(unassignedJob)
        if canBeAssigned(unassignedJob, nurse) then
            newIdleMinutes = windowLength − unassignedJob.length
            if newIdleMinutes < idleMinutes then
                idleMinutes = newIdleMinutes
                nurseToAssign = nurse
                jobToAssign = unassignedJob
            end if
        end if
    end for
end for
if idleMinutes < ∞ then
    unassignedJobs.remove(jobToAssign)
    assignedJobs.add(jobToAssign, nurseToAssign)
end if
```

v případě tvorby počátečního rozvrhu, soustředí na přiřazování méně flexibilních asistencí. Konkrétně jsou přiřazovány ty směny, jež jsou omezeny alespoň jednou hard constraint, přičemž je aktivně vyhledáváno vhodné místo v rozvrhu a není pouze spoléháno na hodnotu kriteriální funkce. Ze seznamu nepřiřazených asistencí jsou nejprve vyfiltrovány ty, které mají alespoň jednu hard constraint, a následně je nalezena množina zdravotních sester, kterým by mohla být asistence přiřazena – tedy sestry, které jsou v době asistence dostupné a splňují definovaná omezení (patří mezi požadované sestry dané asistence, případně mají všechny nutné kvalifikace pro výkon péče). Z těchto sester je následně vybrána ta, která ve stejnou dobu již nemá přiřazenou jinou asistenci, která by časově kolidovala s asistencí, jež je právě zpracovávána, a pokud je taková sestra nalezena, dojde k přidání asistence do jejího rozvrhu pro daný den.

Pokud však nebude nalezena žádná sestra, které by mohla být přiřazena některá asistence s hard constraint, může být vynuceno nahrazení stávající asistence v případě, že byl pro heuristiku nastaven parametr `replace` na hodnotu `true` (ten lze nastavit v konfiguračním souboru viz kapitola 5.4). Pokud je tento parametr aktivován, proběhne nejprve výše popisovaný proces a pouze v případě, že touto cestou nelze asistenci přiřadit žádné ze sester, které splňují podmínky omezení, dojde k odebrání asistence, která časově koliduje a místo ní bude přiřazena nová. Způsob výběru asistence k odebrání probíhá následovně: pro každou sestru splňující podmínky je analyzován její rozvrh v daném dni a pokud s novou asistencí časově koliduje pouze jedna asistence, je uložena jako kandidát na odebrání. Pokud je nalezen další kandidát u jiné sestry, jsou porovnány délky asistencí a v případě, že by byla nalezena kratší asistence k odebrání, je kandidát nahrazen. Tímto způsobem je nalezena nejkratší

asistence, po jejímž odebrání bude možné validně přiřadit novou asistenci – díky tomu je minimalizován negativní vliv této změny na hodnotu kriteriální funkce.

5.4 Konfigurace

Jelikož mnoho částí algoritmu je silně závislých na hodnotách parametrů, je nutné, aby je bylo možné měnit bez nutnosti zásahu do kódu. Proto je veškerá konfigurace implementovaných tříd přesunuta do *.properties* souborů [5], které jsou logicky odděleny podle komponenty, ke které se vztahují. Konfigurovatelnými hodnotami nejsou pouze číselné konstanty, ale také samotné použité implementace v celém algoritmu. Díky tomuto flexibilnímu způsobu lze bez nutnosti zásahu do kódu měnit nejen penalizační konstanty v kriteriální funkci, ale také například vyměnit implementaci použité removal heuristiky či ukončovací podmínky. V současné chvíli jsou v adresáři `resources` tři konfigurační soubory:

- **csv.properties** – Zde je umístěna konfigurace různých typů oddělovačů pro *.csv* soubory či formáty data a času.
- **scheduler.properties** – V tomto souboru se nachází konfigurace použité implementace pro tvorbu počátečního rozvrhu, ukončovací podmínky a iterativní metaheuristiky (ALNS) spolu s požadovanými parametry těchto tříd (např. počet iterací či délka běhu pro ukončovací podmínku).
- **alns.properties** – Poslední ze souborů je specifický pro metaheuristiku ALNS a obsahuje největší množství parametrů. Na tomto místě lze konfigurovat použité implementace pro kriteriální funkci, removal a insertion heuristiky s pravděpodobnostmi jejich výběru při ruletové selekci a dalšími souvisejícími parametry. Mezi ty patří např. penalizační konstanty (pro porušení jednotlivých soft constraints, nepřiřazené asistence apod.) či počet iterací jednotlivých heuristik.

Pro přidání možnosti konfigurace nově vytvořené implementace z *.properties* souborů stačí pouze definovat konstruktor, který bude volán v příslušném rozhraní v metodě *createX* (např. *createTerminationCondition*, *createRemovalHeuristic* apod.). Zbytek toku programu již zajišťuje framework; tento způsob implementace je charakterizován návrhovým vzorem *Inversion of Control* [20].

5.5 Zpracování výsledků

Poslední fází algoritmu je zpracování výsledného rozvrhu a seznamu nepřiřazených směn a případná transformace do formátu použitelného pro cílové uživatele. K tomuto účelu slouží třída `ResultParser`, která jako parametr přijímá objekty typu `ResultTransformer`. Ty deklarují čtyři základní metody – `before()`, která je zavolána jednou před začátkem zpracování výsledku, `transformAssignedJobs()`, která slouží k transformaci rozvrhu jednotlivých sester, `transformUnassignedJobs()`, která transformuje nepřiřazené asistence a

[5] Standardně využívaný formát v jazyce Java pro externí nastavování proměnných, přístup pomocí třídy `java.util.Properties`

`after()`, jež je volána poté, co jsou zpracována všechna data. Díky této struktuře je poměrně snadné implementovat parser do libovolného výsledného formátu. V konstruktoru třídy `ResultParser` je parametrem předán seznam transformátorů, který udává pořadí, v němž se mají na výsledek aplikovat.

Pro účely testování algoritmu byl nejprve implementován jednoduchý `ConsolePrintingTransformer`, který výsledný rozvrh pouze vypisuje do konzole. Pro reálné použití je však využívána implementace `CsvTransformer`, která vytvoří jeden *.csv* soubor, ve kterém je přehled všech sester a jim přiřazeným asistencí v přehledném kalendářovém formátu. Seznam nepřiřazených asistencí je vypsán na konec souboru k příslušným dnům a u každé asistence je také poznamenáno jméno klienta pro snadnou identifikaci. Využití transformátorů však není limitováno pouze na tvorbu různých formátů výsledného rozvrhu – například třída `ValidatingTransformer` slouží k ověření správnosti výsledku a ke kontrole, zda nedošlo k porušení nějakého invariantu. Používání tohoto transformátoru výrazně zjednodušilo detekci chyb při vývoji algoritmu a úpravách heuristik; jeho detailnějšímu popisu se věnuje kapitola 6. Jiným příkladem alternativního využití transformátorů je `StatisticsTransformer`, pomocí kterého jsou získávány a vypočítávány relevantní statistiky z výsledků algoritmu. Mezi ty patří například analýza nepřiřazených asistencí za účelem zjišťování důvodů, proč nemohly být umístěny do rozvrhu, porušené soft constraints nebo měření utilizace zdravotních sester (poměr efektivně stráveného času vůči celkové dostupnosti). Tento transformátor byl hojně využíván nejen při testování, ale zejména při měření efektivity algoritmu (viz kapitola 7).

Kapitola 6

Testování

Návrh frameworku pro řešení problému rozvrhování domácí péče a jeho následné použití pro implementaci metaheuristiky ALNS byl představen v kapitolách 4 a 5. Cílem této kapitoly je popsat strukturu a typy testů, které byly použity pro ověření funkčnosti algoritmu.

6.1 Použité technologie

Jelikož je algoritmus implementován v jazyce Java, byl pro testování zvolen nástroj *JUnit* [1], který je standardně využíván pro implementaci jednotkových (*unit*) a integračních testů. Framework *JUnit* umožňuje tvorbu testů pomocí Java objektů a anotací a poskytuje pomocné metody pro ověřování predikátů, chybových stavů a přípravu testovacích dat. V jednotkových testech je hojně využívána knihovna *Mockito* [2], jež slouží k tvorbě *mock* objektů. Ty simulují funkcionalitu reálných objektů a lze kontrolovat jejich chování při definovaných interakcích (např. volání metody s konkrétním parametrem, nastavení návratové hodnoty apod.). Pokud jsou těmito objekty nahrazeny závislosti testované třídy, je funkčnost výkonného kódu ověřována izolovaně a výsledky testu nejsou ovlivněny případnou chybou v implementaci tříd, na kterých je testovaný objekt (*object under test*) závislý.

K automatizaci spouštění testů je využíván nástroj *Travis* [3] pro *continuous integration* (CI). Konfigurace této služby se nachází v souboru `.travis.yml`, spouštění jednotkových a integračních testů je prováděno pomocí nástroje *Apache Maven*. Po přidání nových změn do repozitáře, ve kterém je verzován zdrojový kód projektu, jsou automaticky spuštěny všechny testy a o případných chybách je autor informován emailem nebo přímo ve webovém rozhraní nástroje *Travis*, kam je možné přistoupit také pomocí ikony (*badge*) v souboru `README.md`.

6.2 Jednotkové testy

Záruka korektního chování základních komponent algoritmu je kritická pro celkovou funkčnost frameworku a jeho další vývoj. Granularita algoritmu výrazně usnadňuje testování prostřednictvím unit testů, neboť závislosti tříd mohou být nahrazeny *mock* objekty;

[1] https://junit.org
[2] http://site.mockito.org
[3] https://travis-ci.com

z toho důvodu je většina testů jednotkových, což je souladu s doporučením, které ilustruje testovací pyramida [10]. Díky intenzivnímu testování na nejnižší úrovni je snadné detekovat, lokalizovat a opravit chyby jak při počáteční implementaci, tak při pozdějších úpravách, kdy tyto testy slouží k odhalení regrese. Z celkových 3159 řádků zdrojového kódu (SLOC – *source lines of code*) je pokryto 62%, čehož je docíleno 2936 řádky jednotkových testů. U kritických komponent, mezi které patří například třída `AssignmentService`, pomocné metody pro parsování vstupu, heuristiky, kriteriální funkce a tvorba počátečního rozvrhu, se pokrytí jednotkovými testy pohybuje mezi 80-100%. Pro výpočet pokrytí zdrojového kódu byl použit integrovaný nástroj ve vývojovém prostředí *Intellij IDEA*[4], statistiky počtu řádků pak byly získaný pomocí pluginu *Statistic*[5].

Univerzální pomocné metody a konstanty jsou definovány v abstraktní třídě `AbstractTest`, od které dědí většina testů. Mezi její metody patří například vytváření klienta, zdravotní sestry nebo objektu objednané asistence, spolu s vytvořením *mock* metody pro třídy `TreatmentRepository` a `AssignmentService`. Soubory s testovacími daty a konfigurací se nachází v adresáři `/src/test/resources`; ty jsou využívány zejména pro testování čtení *.csv* a *.zip* souborů, nicméně mají využití i při testování efektivity algoritmu na generovaných a testovacích instancích viz kapitola 7. Struktura jednotkových testů odpovídá struktuře balíků (*package*) testovaných tříd, přičemž třídy testů mají stejný název se sufixem `Test`.

Intenzivní testy rozsáhlých a kriticky důležitých tříd, mezi které patří například `AssignmentService`, jsou navíc rozděleny do kategorií. Jednotlivé metody mají anotaci `@Category`, jejímž parametrem je libovolné rozhraní specifikované v balíku `cz.cvut.suite.category`. Toto rozdělení umožňuje nejen označit testy a shlukovat je do logických celků podle funkcionality, kterou testují, ale hlavně je možné tyto testy pouštět izolovaně pouze pomocí reference na kategorii i v případě, že by jednotlivé testy byly umístěny ve více třídách. K tomu může být využita integrovaná podpora vývojového prostředí (např. *Intellij IDEA* tuto funkci umožňuje), nebo nástroj *Apache Maven Surefire*[6], pomocí kterého lze spustit testy z vybrané kategorie příkazem `mvn test -Dgroups="${category}"`, kde `${category}` je plně kvalifikované jméno (FQN – *fully qualified name*) třídy dané kategorie. Poslední možností je vytvoření *test suite*, která bude pomocí anotace `@Categories.IncludeCategory` pouštět testy z vybrané kategorie. Tyto třídy jsou umístěny v balíku `cz.cvut.suite`.

6.3 Integrační testy

Pro ověření funkčnosti komunikace mezi jednotlivými komponentami byla implementována sada integračních testů. V nich je aplikován *black-box* přístup, tedy ke komponentám je přistupováno jako k černé skříňce bez znalosti zdrojového kódu a je ověřována korektnost výstupu na základě vstupních dat. Tento přístup však nemůže být aplikován na všechny procesy algoritmu, jelikož některé jeho části jsou založeny na nedeterministických procesech (např. removal heuristika `RandomRemovalHeuristic`, která odebírá náhodné asistence z rozvrhu). Proto je integračními testy pokryt například proces čtení vstupního souboru třídou

[4] https://www.jetbrains.com/idea
[5] https://plugins.jetbrains.com/plugin/4509-statistic
[6] https://maven.apache.org/surefire

`ZipReaderIntegrationTest`, která intenzivně testuje čtení *.zip* souborů s různým obsahem, nebo proces tvorby počátečního rozvrhu, pro který je ověřován výsledek první fáze algoritmu na bohaté sadě vstupních dat pokrývající možné kombinace dostupnosti sester, dovedností, omezení objednaných asistencí a časových oken.

6.4 Testovací scénáře

Validaci správného fungování implementovaného algoritmu a jeho jednotlivých komponent zajišťuje 29 testovacích tříd, které obsahují celkem 139 testů. Jelikož bylo v rámci testovací strategie vytvořeno mnoho scénářů, jsou v tabulce 6.1 vybrány jen nejdůležitější z nich.

6.5 End-to-end testy

Kromě jednotkových a integračních testů byly také průběžně prováděny end-to-end testy, které slouží pro validaci celého procesu od začátku, tj. čtení vstupních dat do konce, tedy vytvoření výstupu z výsledného rozvrhu. Zpočátku měly tyto testy převážně manuální podobu, kdy byl algoritmus spouštěn nad malou množinou testovacích dat, nicméně tento přístup není aplikovatelný na velká reálná data. Z toho důvodu byla implementována třída `ValidatingTransformer`, která využívá abstraktního návrhu parseru výsledku a možnosti využití více transformačních procesů najednou. Tento validátor iteruje nad výsledným rozvrhem a seznamem nepřiřazených asistencí a kontroluje, zda je výsledek v korektním stavu a nedošlo k porušení žádného invariantu. Kontrolované podmínky jsou následující:

- Id přiřazené/nepřiřazené asistence je uloženo ve třídě `TreatmentRepository`
- Žádná z přiřazených asistencí neporušuje hard constraint
- Čas přiřazené asistence souhlasí s objednaným časem (zejména pro kontrolu časových oken)
- Asistence není přiřazena sestře v čas, ve který není dostupná
- Doba přejezdu mezi klienty ve výsledném rozvrhu odpovídá hodnotám v matici vzdáleností
- Každá směna je přiřazena právě jednou, nebo je v seznamu nepřiřazených asistencí
- Všechny objednané asistence byly uvažovány při tvorbě výsledného rozvrhu

Díky strukturovanému přístupu k testování algoritmu a vhodné skladbě různých testovací strategií je významně redukována pravděpodobnost výskytu chyby. V případě, že by v novém kódu byla do systému zanesena chyba, která by porušila validitu výsledku, je poměrně pravděpodobné, že bude okamžitě objevena díky automatickému spouštění intenzivních jednotkových a integračních testů, případně na základě end-to-end testu v podobě `ValidatingTransformeru`.

Tabulka 6.1: Přehled vybraných testovacích scénářů

Testovaná metoda	**Popis scénáře**
`ZipReader #unzip`	Ze vstupních .zip souborů jsou korektně vytvářeni klienti a sestry
`ClientBuilder #build`	Ze vstupních dat je korektně vytvořen klient, v případě nevalidních dat je vyhozena výjimka
`NurseBuilder #build`	Ze vstupních dat je korektně vytvořena sestra, v případě nevalidních dat je vyhozena výjimka
`AssignmentService #assignIfPossible`	Asistence je sestře přiřazena jen tehdy, pokud není porušeno žádné omezení
`AssignmentService #getAssignableOption`	Pouze pokud není porušeno žádné omezení je nalezen čas, ve který může být asistence dané sestře přiřazena
`AssignmentService #selectFeasibleNurses`	Jsou nalezeny pouze dostupné sestry, které mohou pokrýt asistenci bez porušení omezení
`AssignmentService #replaceJob`	Nejkratší přiřazená asistence dostupné sestry je nahrazena pouze tehdy, pokud nedojde k porušení žádného omezení
`FirstAvailableInitialAssignment #apply`	Asistence je přiřazena dostupné sestře na nejdřívější možný termín, pokud není porušeno žádné omezení
`RigidFirstInitialAssignment #apply`	Pouze asistence specifikované časovým oknem a omezené hard constraints jsou přiřazeny
`PenalizingFunction #evaluateAllForDate`	Rozvrh v daný den je korektně ohodnocen vzhledem k penalizačním konstantám
`PenalizingFunction #penalizeIdle`	Penalizace je výsledkem součinu penalizační konstanty a počtu idle minut dané sestry
`LongestFittingInsertionHeuristic #insert`	Asistence je přiřazena na takové místo v rozvrhu sestry, na kterém nejvíce vyplní časové okno a nebude porušeno žádné omezení
`ConstrainedInsertionHeuristic #insert`	Pouze asistence s hard constraints jsou přiřazeny
`RandomRemovalHeuristic #remove`	n náhodných asistencí je přesunuto z rozvrhů sester do seznamu nepřiřazených asistencí
`ViolatedRemovalHeuristic #remove`	Prvních n asistencí je přesunuto z rozvrhů sester do seznamu nepřiřazených asistencí po seřazení od nejvíce penalizované asistence
`ALNS #enhance`	Pouze rozvrh s vyšší hodnotou kriteriální funkce nahradí stávající řešení

Kapitola 7

Benchmarking

V kapitole 5 byly představeny konkrétní způsoby implementace jednotlivých modulů algoritmu a heuristik, které jsou použity v metaheuristice ALNS (viz obr. 5.1). Cílem této kapitoly je změřit efektivitu algoritmu a jednotlivých heuristik a porovnat výsledky podle relevantních kritérií, vyhodnotit použitelnost heuristik pro různé množiny vstupních dat a analyzovat vliv hodnot parametrů na výsledný rozvrh.

7.1 Vstupní instance

Měření efektivity algoritmu probíhalo na několika sadách vstupních dat, které se lišily nejen počtem zdravotních sester, klientů a objednaných asistencí, ale také skladbou omezení, počtem směn s časovými okny, délkou směny sestry a mnoha dalšími aspekty.

První instancí jsou reálná data získaná od tuzemské firmy poskytující domácí péči; plánovací horizont je v jejich případě následující měsíc. Dostupnost zdravotních asistentů a pochopitelně i požadavky klientů se pro každé plánovací období mění. V získaných datech bohužel nejsou rozepsány tyto počáteční údaje; místo toho je dostupný pouze výsledný rozvrh na jeden konkrétní měsíc. I přesto je však možné získat alespoň částečná data a otestovat efektivitu algoritmu na instanci, která reprezentuje reálnou situaci. K této transformaci slouží skripty `transformXlsx.py` a `transformData.sh`, které jsou zmiňovány a částečně popsány v kapitole 5.2.1. Jelikož není známa původní dostupnost sester, je skriptem dopočítána z výsledného rozvrhu jako souvislý interval začínající časem příjezdu k prvnímu klientovi a končící odjezdem od posledního klienta, jehož asistenci má sestra v daný den přiřazenou. Aby mohla být také zohledněna doba strávená přesunem, je začátek dostupnosti posunut o 30 minut dříve. Dalšími informacemi, které již z výsledného rozvrhu nejde získat, jsou rozsah časových oken a případná omezení asistencí. Z toho důvodu jsou u této instance vynechány a analýza efektivity algoritmu z hlediska těchto kritérií je měřena na ostatních sadách vstupních dat. Reálná instance obsahuje 26 zdravotních sester, 68 klientů a celkem 678 objednaných asistencí různých délek, přičemž největší zastoupení mají hodinové (175 případů) a dvouhodinové asistence (282 případů).

Vzhledem k chybějícím informacím v množině reálných dat probíhalo další testování algoritmu na uměle vytvořených instancích. K tomuto účelu byl vytvořen skript

Tabulka 7.1: Parametry skriptu pro generování vstupních instancí

Parametr	Datový typ	Výchozí hodnota
num_nurses	int	-
num_clients	int	-
full_half_time_ratio	float	1
assistance_probability	float	1
time_windowed_probability	float	0
preferred_nurse_constraint_probability	float	0
required_nurse_constraint_probability	float	0
nurse_constraint_lb	int	1
nurse_constraint_ub	int	4
required_skills_constraint_probability	float	0
max_skills	int	3
skill_probability	float	0

`instanceGenerator.py` umístěný adresáři `scripts`, pomocí kterého jsou na základě přijatých parametrů vygenerována vstupní data v požadovaném formátu. Konfiguraci lze měnit pomocí parametrů předávaných z příkazové řádky – jejich přehled a výchozí hodnoty jsou popsány v tabulce 7.1. Předávání parametrů probíhá standardním způsobem, jak je zvykem v unixových systémech, tedy pro vygenerování vstupních dat s deseti sestrami a třiceti klienty stačí zavolat příkaz `python instanceGenerator.py --num-nurses 10 --num-clients 30` z adresáře, ve kterém je skript umístěn. Přehled dostupných parametrů a jejich popis lze zobrazit zavoláním skriptu s parametrem `--help`, nebo `-h`.

Jedinými povinnými parametry jsou `num_nurses` a `num_clients`, které udávají počet zdravotních sester a klientů; pokud bude některý z ostatních parametrů chybět při spouštění skriptu, bude automaticky nastaven na výchozí hodnou, která je uvedena v tabulce. Dostupnost sester je závislá na hodnotě parametru `full_half_time_ratio` – ten určuje, kolik procent sester bude dostupných na plný úvazek (tj. jejich dostupnost bude každý den od 9:00 do 17:00) a kolik bude zaměstnáno na poloviční úvazek (tedy budou dostupné každý den, ale pouze v intervalech 9:00 – 13:00 či 13:00 – 17:00, přičemž bude vybrán náhodně jeden z těchto dvou intervalů). Pokud bude mít tento parametr hodnotu např. 0.3, znamená to, že 30% sester bude pracovat na plný úvazek a zbylých 70% na poloviční. Jelikož jsou asistence generovány náhodně a jejich počet není deterministický, není tato hodnota zahrnuta v parametrech, nicméně množství vytvořených asistencí lze ovlivnit parametrem `assistance_probability`, který udává pravděpodobnost, se kterou bude klientem objednána asistence v jeden konkrétní den. Pro generování asistencí s časovým oknem slouží parametr `time_windowed_probability`, pomocí nějž lze specifikovat, jaké procento asistencí bude objednáno s časovým oknem (jeho rozsah je náhodný). Přidávání soft a hard constraints je možné nastavením pravděpodobnosti výskytu třem parametrům `preferred_nurse_constraint_probability`, `required_nurse_constraint_probability` či `required_skills_constraint_probability`. Velikost množiny sester, které můžou být klientem udány jako požadované či preferované je náhodné číslo v intervalu daným hodnotami parametrů `nurse_constraints_lb` a `nurse_constraints_ub`. Parametr `max_skills` udává maximální počet kvalifikací, které může sestra mít, případně kolik může být vyža-

Tabulka 7.2: Výsledky měření na reálných datech s dobou přesunu 30 minut

Metrika	**Hodnota**
Počet asistencí	678
Součet délek všech asistencí (v minutách)	109 738
Počet nepřiřazených asistencí po první fázi	111
Délka nepřiřazených asistencí po první fázi	34 318
Počet nepřiřazených asistencí po druhé fázi	85
Délka nepřiřazených asistencí po druhé fázi	28 504
Celková utilizace sester	68.86%
Procentuální vyjádření přiřazených asistencí	87.46%
Procentuální vyjádření přiřazených minut	74.03%
Doba běhu algoritmu	3000 ms

dováno pro přiřazení asistence s tímto omezením a nakonec, parametr `skill_probability` slouží ke specifikaci pravděpodobnosti, se kterou bude mít některé z těchto kvalifikací.

Pomocí skriptu `instanceGenerator.py` byla vygenerována sada instancí, které jsou v této kapitole využity pro měření efektivity algoritmu dle různých kritérií. Kompletní přehled používaných instancí a jejich atributů je uveden v příloze B.

7.2 Výsledky měření

Pro vyhodnocování kvality výsledků algoritmu je třeba brát v potaz různá kritéria, podle kterých bude výsledný rozvrh hodnocen. Jedním z nejdůležitějších kritérií je množství přiřazených asistencí; dále lze však výsledek hodnotit také z hlediska utilizace zdravotních sester (využití pracovní doby), počtu porušených soft constraints a řady dalších indikátorů. Z toho důvodu byly prováděny experimenty nejen na různých sadách vstupních instancí, ale také byly testovány různé konfigurace algoritmu, na jejichž základě byla následně provedena analýza za cílem výběru nejvhodnější konfigurace pro maximalizaci vybraného kritéria. Přehled hodnot jednotlivých parametrů použitých konfigurací je uveden v tabulkách v příloze C.

Všechny experimenty byly prováděny v pěti opakováních a výsledná data byla zprůměrována za cílem zvýšení spolehlivosti. Výsledky všech jednotlivých měření včetně výpočtů relevantních statistik jsou uvedeny v souboru `benchmarking/benchmarking.ods`, ze kterého jsou čerpána data uvedená v této kapitole. Použité vstupní instance a vybraná konfigurace zde budou referencovány jejich názvem, který je uveden v prvním řádku tabulky v odpovídající příloze. V neposlední řadě je třeba zmínit, že výsledky algoritmu jsou závislé na hodnotách v matici vzdáleností, která udává dobu přesunu mezi bydlišti jednotlivých klientů a sester; z důvodu absence těchto údajů v testovacích datech byly pro účely měření generovány a to buď randomizovaně, nebo nastavením fixních hodnot. Pro každé měření bude v textu vždy uvedeno, jakými hodnotami byla matice vzdáleností naplněna.

Všechny výsledky uvedených experimentů byly naměřeny na procesoru *Intel Core i7-3632QM* se čtyřmi 2.2GHz CPU s použitím 8GB RAM.

První experiment byl proveden na reálných datech (instance `real`), která byla dopočítána z výsledného rozvrhu viz kapitola 7.1, přičemž hodnoty v matici vzdáleností byly fixně

Tabulka 7.3: Výsledky měření na reálných datech se zanedbáním doby přesunu

Metrika	**Hodnota**
Počet asistencí	678
Součet délek všech asistencí (v minutách)	109 738
Počet nepřiřazených asistencí po první fázi	34
Délka nepřiřazených asistencí po první fázi	10 590
Počet nepřiřazených asistencí po druhé fázi	14
Délka nepřiřazených asistencí po druhé fázi	5 976
Celková utilizace sester	72.18%
Procentuální vyjádření přiřazených asistencí	97.88%
Procentuální vyjádření přiřazených minut	94.55%
Doba běhu algoritmu	2557 ms

nastaveny na 30 minut pro přesun mezi libovolnými lokacemi. Pro algoritmus byla využita konfigurace `conf_1`, tedy 100 iterací s rovnoměrným rozdělením pravděpodobnosti výběru všech heuristik. Podstatné výsledky měření jsou uvedeny v tabulce 7.2 – z celkového počtu 678 asistencí se podařilo přiřadit 593, což znamená 87.48%. Pro relevantnější výsledky je však třeba zohlednit také délky přiřazených a nepřiřazených asistencí a počítat poměr z těchto údajů, z výsledných dat vyplývá, že z celkových 109 738 minut objednaných klienty byly dostupným sestrám přiřazeny téměř tři čtvrtiny. Pro tuto instanci se ukázal jako nejefektivnější způsob tvorby počátečního rozvrhu použití třídy `FirstAvailableInitialAssignment`; jak je patrné z tabulky, většina asistencí byla přiřazena již v první fázi algoritmu (567 asistencí).

Co se týče druhé fáze algoritmu, byl zkoumán zejména efekt různých heuristik na výsledky algoritmu, v tomto konkrétním případě jde o jediná relevantní kritéria počet přiřazených asistencí (minut) a utilizaci sester, jelikož na asistencích nejsou definována žádná omezení. Tato statistika byla získána následujícím způsobem – pro všechny použité heuristiky byla vytvořena mapa četnosti použití, přičemž počáteční četnost byla nastavena na hodnotu 0. V případě, že došlo ke zvýšení hodnoty kriteriální funkce a tedy nalezení lepšího rozvrhu, došlo k inkrementaci četností insertion a removal heuristik, které byly v dané iteraci použity.

Pro instanci `real` byl rozvrh ve dvou třetinách případů zlepšen removal heuristikou `RandomRemovalHeuristic`, v jedné třetině pak `ViolatedRemovalHeuristic`. To je zapříčiněno vlastnostmi vstupní instance, konkrétně absencí omezení, tudíž druhá zmiňovaná heuristika může odebírat asistence pouze na základě jejich délky, čímž se její efektivita snižuje. Co se týče insertion heuristik, v podobném poměru (68.5%) byla efektivnější `LongestFittingInsertionHeuristic` oproti `ReplacementInsertionHeuristic` a to ze stejných důvodů – první zmiňovaná heuristika se soustředí na vyhledávání asistencí, které lze umístit do volných časových oken v rozvrzích sester tak, aby bylo co nejefektivněji vyplněno a následně pouze vyhodnotí, jestli není porušena nějaká z hard constraints, místo aby byly aktivně vyhledávala pouze sestry, které toto omezení splňují. To se ukázalo jako nejvhodnější způsob, ve zbytku případů byla použita `ReplacementInsertionHeuristic`, díky které mohl být narušen stávající rozvrh a neefektivně přiřazené asistence mohly být nahrazeny vhodnějšími. Kromě kritéria počtu přiřazených asistencí také stojí za pozornost celková utilizace sester, tedy poměr doby strávené u klienta a na cestě vůči celkové dostupnosti. Hodnota tohoto kritéria je 68.86%.

Tabulka 7.4: Výsledky měření na instanci 10_30 s konfigurací conf_1

Metrika	**Hodnota**
Počet asistencí	930
Součet délek všech asistencí (v minutách)	118 455
Počet nepřiřazených asistencí po první fázi	249
Délka nepřiřazených asistencí po první fázi	31 890
Počet nepřiřazených asistencí po druhé fázi	246
Délka nepřiřazených asistencí po druhé fázi	26 517
Celková utilizace sester	75.56%
Procentuální vyjádření přiřazených asistencí	73.46%
Procentuální vyjádření přiřazených minut	77.61%
Doba běhu algoritmu	5007 ms

Jelikož byly hodnoty v matici vzdáleností nastaveny fixně na 30 minut, což nereflektuje reálnou situaci (např. je vhodné, aby sestry jezdili za pacienty primárně v okolí svého bydliště), byly výsledky pro porovnání změřeny na stejné instanci s identickou konfigurací, pouze byla zanedbána doba přesunu (tj. všechny hodnoty v matici vzdáleností byly nastaveny na 0). Naměřené výsledky jsou popsány v tabulce 7.3. Nejvýraznějšího zlepšení bylo dosaženo z hlediska počtu přiřazených asistencí, tedy z nejdůležitějšího kritéria – pouhých 14 asistencí, tedy 2.12% nebylo po 100 iteracích algoritmu přiřazeno; při zohlednění jejich délek se ukázalo, že bylo přiřazeno celkem 94.44% objednaných minut. Zlepšení utilizace nebylo tolik výrazné (necelá 4%), nicméně to je zapříčiněno právě zanedbáním doby přesunu, jelikož ta se také započítávala do efektivně stráveného času sestry. I přesto však bylo několik sester utilizováno na 100%. V neposlední řadě byl také zkoumán vliv a efektivita jednotlivých heuristik na výsledky algoritmu; ty však byly ze stejných důvodů téměř identické s výsledky popsanými výše. Nejvhodnější dvojice heuristik byla `RandomRemovalHeuristic` a `LongestFittingInsertionHeuristic` (frekvence použití obou heuristik vůči ostatním heuristikám stejného typu byla zhruba 70%).

Další měření probíhalo na generované instanci `10_30` s identickou konfigurací `conf_1` a fixní třicetiminutovou dobou cesty mezi libovolnými klienty; výsledky lze nalézt v tabulce 7.4. Parametry skriptu pro vytvoření této instance byly zvoleny tak, aby poměr sester vůči pacientům a délce objednaných asistencí vůči celkové dostupnosti byl co nejvíce podobný hodnotám reálné instance, přičemž asistence budou moci být navíc zadány časovým oknem nebo mít omezení. Výsledné statistiky jsou z hlediska procenta přiřazených asistencí podobné hodnotám naměřených na reálné instanci – 77.61% přiřazených objednaných minut, avšak pokud není rozlišována délka asistencí, je tato hodnota nižší (73.46%), než na reálných datech. Důvodem je fakt, že v reálných datech převažovaly zejména hodinové a dvouhodinové asistence; v instanci `10_30` byly délky rovnoměrněji rozděleny a analýza nepřiřazených asistencí ukázala, že nejvíce nepřiřazených asistencí mělo právě délku do hodiny a půl. Naopak utilizace sester byla zvýšena na 75% s tím, že i utilizace nejméně vytížené sestry vždy přesahuje hodnotu 60%.

Na této instanci vstupních dat již lze navíc analyzovat také výsledná data z hlediska omezení – co se týče soft constraints, z celkových 119 asistencí s omezením preferované sestry bylo do rozvrhu umístěno 87, přičemž pouze necelých 30% z nich bylo splněno. To je však

Tabulka 7.5: Výsledky měření na instanci 10_30_tw s konfigurací conf_1

Metrika	**Hodnota**
Počet asistencí	930
Součet délek všech asistencí (v minutách)	108 705
Počet nepřiřazených asistencí po první fázi	139
Délka nepřiřazených asistencí po první fázi	17 895
Počet nepřiřazených asistencí po druhé fázi	123
Délka nepřiřazených asistencí po druhé fázi	13 008
Celková utilizace sester	80.58%
Procentuální vyjádření přiřazených asistencí	86.77%
Procentuální vyjádření přiřazených minut	88.03%
Doba běhu algoritmu	4211 ms

zapříčiněno nízkou penalizací za porušení tohoto omezení viz přehled konfigurace `conf_1` v příloze C. Procento přiřazených asistencí s hard constraints `RequiredNurseConstraint` je 60% a pro `RequiredSkillsConstraint` 43%, ostatní asistence s těmito omezeními zůstaly nepřiřazeny. V této konfiguraci však kriteriální funkce spíše upřednostňuje větší množství přiřazených asistencí před zohledňováním jejich omezení (samozřejmě však nemohou být porušeny hard constraints). Efektivita jednotlivých heuristik je podobná, jako v případě reálné instance – nejvhodnějšími implementacemi jsou `RandomRemovalHeuristic` a `LongestFittingInsertionHeuristic` s podobnou četností využití.

Pro analýzu způsobu zlepšení rozvrhu z hlediska různých kritérií byl algoritmus s výše zmíněnými daty spouštěn opakovaně s mírnými změnami ve vstupních datech i v konfiguraci a byl zkoumán vliv jednotlivých aspektů na kvalitu výsledného rozvrhu. V první řadě byla vygenerována nová data se stejnými parametry, pouze asistence byly učiněny více flexibilními (pravděpodobnost časového okna byla zvýšena z hodnoty 0.3 na 0.8, jeho délka je náhodně velká), díky čemuž mají asistence více možností, kam mohou být umístěny v rozvrhu. Atributy této instance lze nalézt v příloze B pod názvem `10_30_tw`, výsledky měření pak v tabulce 7.5. Díky tomuto relaxování časových požadavků se podařilo o polovinu zmenšit nepřiřazené asistence a to jak z hlediska jejich počtu, tak celkového součtu jejich délky. Procentuálně vyjádřeno bylo dosaženo zlepšení o 15% na celkových téměř 87% přiřazených asistencí a utilizace sester navíc stoupla na 80%.

Pokud je však důležitějším kritériem splnění soft constraints, nebylo pouhým přidáním časových oken výrazně zlepšeno. Proto byl algoritmus spuštěn navíc s pozměněnou konfigurací `conf_2`, ve které byla zvýšena penalizace za porušení tohoto typu omezení (viz tabulka C.2 v příloze C). Procento přiřazených asistencí se splněným omezením `PreferredNurseConstraint` bylo zvýšeno téměř dvojnásobně s tím, že celkový počet přiřazených asistencí byl snížen pouze o 5% a utilizace sester klesla o 4%. Z těchto výsledků je patrné, že vliv penalizačních konstant v kriteriální funkci má výrazný vliv na vlastnosti výsledného rozvrhu, přičemž není vyžadován žádný zásah do samotného algoritmu či výměna použitých heuristik. Za zmínku stojí také změna statistik použitých heuristik – značně stoupl počet úspěšných použití `ViolatedRemovalHeuristic` (dosáhl stejné četnosti, jako `RandomRemovalHeuristic`) a z insertion heuristik vynikla `ReplacementInsertionHeuristic`, která byla dokonce používána častěji, než `LongestFittingInsertionHeuristic`.

Tabulka 7.6: Výsledky měření na instanci 50_150 s konfigurací conf_1

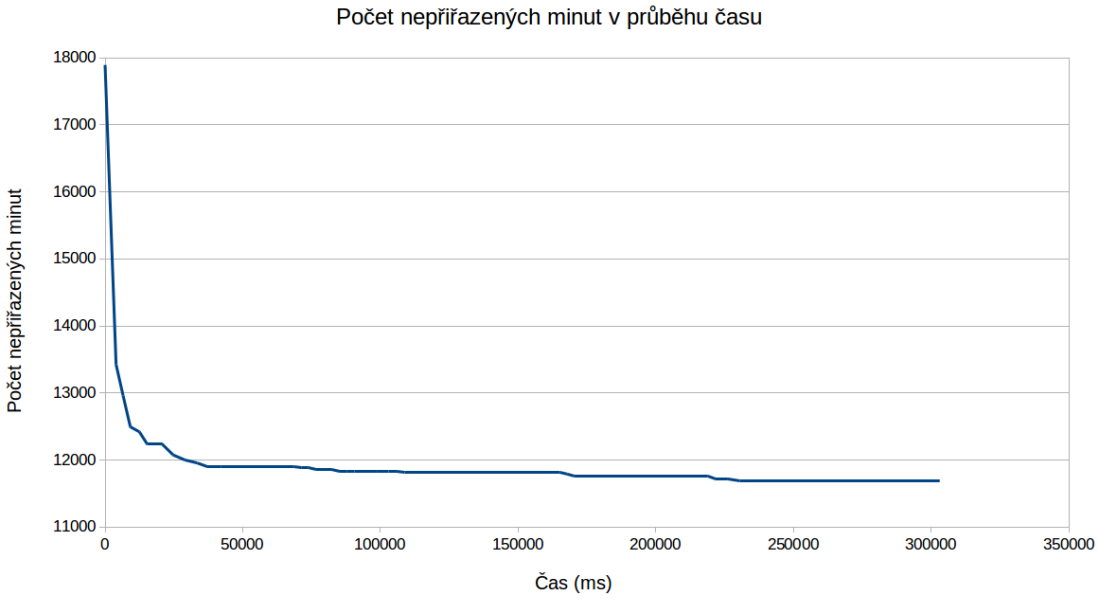
Metrika	**Hodnota**
Počet asistencí	4650
Součet délek všech asistencí (v minutách)	596 805
Počet nepřiřazených asistencí po první fázi	786
Délka nepřiřazených asistencí po první fázi	101 475
Počet nepřiřazených asistencí po druhé fázi	772
Délka nepřiřazených asistencí po druhé fázi	99 522
Celková utilizace sester	82.48%
Procentuální vyjádření přiřazených asistencí	83.40%
Procentuální vyjádření přiřazených minut	83.32%
Doba běhu algoritmu	216 954 ms

Aby mohla být efektivita algoritmu změřena i na velkém objemu vstupních dat, byla vygenerována instance `50_150`, která je zhruba pětkrát větší, než výše zmiňovaná instance z hlediska počtu sester, klientů a asistencí. Její ostatní atributy jsou popsány v tabulce B.4 a výsledky měření v tabulce 7.6; konfigurace použitá pro tento experiment byla `conf_1`, aby mohly být výsledky porovnány s předchozím měření instance `10_30_tw`. Z hlediska počtu přiřazených asistencí a utilizace sester jsou výsledky srovnatelné s touto instancí – obě tato kritéria přesahují 80%, stejně jako procentuální vyjádření počtu přiřazených minut objednaných asistencí. Za pozornost však stojí četnost heuristik, které byly použity v případě, že došlo ke zlepšení hodnoty kriteriální funkce; co se týče removal heuristik, tak jediná efektivní implementace byla `ViolatedRemovalHeuristic`, přičemž `RandomRemovalHeuristic` nebyla vybrána ani v jednom případě. To je zásadní změna oproti menším instancím, kde byla jednoznačně nejefektivnější právě druhá zmiňovaná, což naznačuje, že pro komplexnější rozvrhy je vhodnější vybírat asistence k odebrání informovaným způsobem, spíše než náhodně. Podobný rozdíl je i v insertion heuristikách, kde `LongestFittingInsertionHeuristic` sice byla v pár případech úspěšná, avšak `ReplacementInsertionHeuristic` byla efektivnější téměř šestkrát častěji.

7.2.1 Analýza doby běhu algoritmu

Co se týče doby běhu algoritmu, menší instance (`real` a `10_30`) byly v konfiguraci `conf_1`, tedy s ukončovací podmínkou `IterationBasedTerminationCondition` se 100 iteracemi spočítány do 5000 ms. Většina času byla pochopitelně strávena ve druhé fázi algoritmu, kdy je iterativně vylepšován částečný rozvrh, jelikož heuristiky musí procházet přiřazené asistence a vybírat nejvhodnějšího kandidáta na smazání, nebo nejvhodnější místo na vložení nové asistence. Počáteční přiřazování do prázdných rozvrhů je výrazně snazší, proto se doba běhu první fáze pohybovala mezi 40 – 70 ms. Na pětkrát větší instanci `50_150` trvalo vykonání 100 iterací 217 sekund, přičemž první fáze algoritmu byla dokončena po 786 ms. Velikost této instance však výrazně převyšuje velikost reálných dat nejen v instanci `real`, ale i ve většině reálných instancí uváděných v literatuře.

Za účelem měření vývoje kritéria počtu nepřiřazených asistencí, respektive počtu nepřiřazených minut, které je přesnější, byl algoritmus spuštěn s konfigurací `conf_3` na instanci



Obrázek 7.1: Vývoj počtu nepřiřazených minut v čase

`10_30_tw` s tím, že po každých 100 iteracích byla zkoumána hodnota měřeného kritéria. Na výsledném grafu vývoje počtu nepřiřazených minut v průběhu času 7.1, který začíná hodnotou 17895 (výsledek po dokončení tvorby počátečního rozvrhu) lze vidět, že prvních 100 iterací mělo největší vliv na změnu kvality rozvrhu. Počet nepřiřazených minut se snížil o 4470, přičemž počet nepřiřazených asistencí zůstal stejný. To znamená efektivnější výběr a preference delších asistencí před krátkými, díky čemuž je zvyšována utilizace zdravotních sester. Během dalších 900 iterací bylo měřené kritérium zlepšeno ještě o přibližně 1500 minut, avšak tempo se výrazně zpomalovalo v průběhu času, jelikož kratší asistence byly postupně přesouvány do seznamu nepřiřazených. Ve zbývajících 9000 iterací se počet nepřiřazených minut sice stále snižoval (výsledná hodnota byla 11685 nepřiřazených minut), avšak z globálního hlediska je již toto zlepšení poměrně zanedbatelné.

7.3 Porovnání s výsledky v literatuře

V kapitole 3 byla představena a analyzována řada různých přístupů v literatuře pro řešení problému rozvrhování domácí péče, včetně vyhodnocení publikovaných výsledků. Jejich porovnání s výsledky naměřenými v této práci je však velice problematické z několika důvodů – za prvé, formulace řešeného problému se liší, stejně jako kritérium, podle kterého probíhá optimalizace rozvrhu; problém řešený v této práci například nepočítá s přiřaditelností všech asistencí a i částečný rozvrh je považován za platný, hlavní optimalizovaná kritéria jsou právě počet přiřazených asistencí a utilizace sester, případně porušená omezení. V článku D. Guericke a kol. [22] je řešený problém například založen na pracovních regulacích, mezi které patří rotace služeb, víkendových směn či nutné přestávky v průběhu dne; genetický algorit-

mus představený v článku Y. Shi a kol. [34] zase zohledňuje rozvoz léků a případné návraty do laboratoře v případě, že jich sestra nemá dostatek k dispozici. I přesto by však bylo možné částečně porovnat atributy výsledných rozvrhů, pokud by byly autory popsány; to je však druhým problémem, proč je porovnání problematické. Ve výsledcích měření v publikovaných článcích je téměř vždy uváděna pouze hodnota kriteriální funkce, ze které je zpětně nemožné dopočítat relevantní atributy rozvrhu a slouží pouze k vzájemnému porovnání výsledků v rámci publikace, případně autoři porovnávají rozdíl kritérií vůči výsledkům nalezeným ILP solverem na velmi malých instancích.

Výsledky algoritmu však mohou být alespoň částečně porovnány z hlediska velikosti vyřešených instancí a době běhu algoritmu. Oproti publikovaným výsledkům ILP solverů bylo heuristické řešení v této práci dle očekávání schopné nalézt řešení pro výrazně vyšší instance za zlomek času – ILP solver v článku D. Guericke a kol. [22] nalezl optimální řešení pouze pro instance do patnácti klientů, na větších datech nebyl ve většině případů schopný nalézt jakékoliv přípustné řešení v limitu dvanácti hodin. Stejně tak pomocí ILP přístupu z článku D. S. Mankowské et al. [27] nebylo možné nalézt žádné řešení pro instance s více, než dvaceti pěti klienty během deseti hodin.

Publikované heuristické přístupy však byly schopné řešit výrazně větší instance. Algoritmus D. S. Mankowské et al. [27] byl testován mj. na instancích o velikostech 100, 200 a 300 klientů a 20, 30 a 40 sester, přičemž počet asistencí autoři neuvádí. Řešení nejmenší instance bylo nalezeno do čtyřiceti sekund, střední do půl hodiny a v případě největší instance do dvou hodin. Výsledků na instanci `50_150` bylo dosaženo během 217 sekund, což se jeví po interpolaci hodnot jako mírně lepší výsledek, avšak jak již bylo řečeno na začátku této kapitoly, není možné objektivně porovnat kvalitu výsledného rozvrhu těchto dvou řešení. Výsledky genetického algoritmu prezentovaného v článku Y. Shi a kol. [34] byly měřeny na různých instancích o velikosti 25 klientů, na kterých se doba výpočtu pohybovala mezi 200 a 600 sekundami; dále na instancích se 100 klienty, které byly vyřešeny v rozmezí deseti až dvaceti pěti minut a nakonec dvakrát tak velké instance, pro které trvalo nalezení výsledného rozvrhu tři čtvrtě hodiny až hodinu a čtvrt. Podle těchto výsledků je představené řešení výrazně rychlejší, nicméně stejně jako v předchozím případě bohužel není možné kvalifikovaně porovnat výsledné rozvrhy. Výsledky měření představené v ostatních článcích v kapitole 3 bohužel nemohou být porovnány, neboť neuvádějí dobu běhu algoritmu, nebo pro porovnání používají pouze hodnotu kriteriální funkce. Podle výše popsaných kritérií se tedy zdá, že doba výpočtu strávená hledáním suboptimálního rozvrhu algoritmem představeném v této práci nijak výrazně nepřevyšuje výsledky publikované v literatuře.

7.4 Shrnutí

Výsledky měření efektivity algoritmu v této kapitole byly měřeny na řadě vygenerovaných instancí s různými parametry a na reálné instanci, přičemž byly použity různé konfigurace a byl zkoumán vliv nastavených parametrů na kvalitu výsledného rozvrhu. Počet přiřazených asistencí byl ve všech případech vyšší, než tři čtvrtiny celkově objednaných asistencí. Co se týče reálných dat, ukázalo se, že výrazný vliv na výsledek mají hodnoty v matici vzdáleností, které popisují čas cesty mezi jednotlivými lokacemi. Pokud byly zanedbány, byl algoritmus schopen přiřadit téměř všechny objednané asistence; to sice není reálná situace, avšak v případě, že by ve vstupních datech byly uvedeny reálné adresy klientů a sester a matice byla

naplněna odpovídajícími hodnotami, bylo by možné použít heuristik, které se budou aktivně snažit přiřazovat sestrám do rozvrhu primárně ty klienty, kteří se nacházejí v okolí bydliště sestry. Díky tomu by byla doba přesunů minimalizována a mohlo by být možné přiřadit více asistencí.

Z hlediska efektivity použitých implementací pro tvorbu počátečního rozvrhu a heuristik používaných ve druhé fázi se ukázalo, že nejlepším přístupem je použít `FirstAvailableInitialAssignment` v první fázi; efektivita heuristik se však již lišila v závislosti na atributech vstupních dat a kritériích, kterých má být dosaženo. Pokud je hlavním cílem maximalizovat počet přiřazených minut objednaných asistencí, vyplatí se na menších datech více použít removal heuristiku `RandomRemovalHeuristic` a insertion heuristiku `LongestFittingInsertionHeuristic`. Na větších datech však již nebyla tato removal heuristika vůbec účinná, proto je lepší využít `ViolatedRemovalHeuristic`, stejně tak pro vkládání asistencí heuristiku `ReplacementInsertionHeuristic`. Ve všech případech je však vhodné nakonfigurovat algoritmus tak, aby používal všechny zmiňované heuristiky a nastavit pravděpodobnosti jejich výběru v závislosti na typu vstupní instance, jelikož například obě insertion heuristiky se velice dobře doplňují. `ReplacementInsertionHeuristic` dokáže narušit existující rozvrh a nahradit přiřazené asistence vhodnějšími, což může vytvořit nová volná okna, která jsou následně vyplněna `LongestFittingInsertionHeuristic`. Pokud je možné zasahovat do vstupních dat, ukázalo se, že zvýšením flexibility klientů přidáním časových oken k asistencím lze výrazně zlepšit výsledný rozvrh při zachování stejného počtu a délky objednaných asistencí.

V poslední řadě je také třeba věnovat dostatek času konfiguraci kriteriální funkce, zejména nastavení penalizačních konstant pro jednotlivá kritéria. Toto je velice flexibilní způsob změny chování algoritmu bez nutnosti úpravy zdrojového kódu a jak ukázaly výsledky měření, mají výrazný dopad na kvalitu výsledného rozvrhu. Co se týče ukončovací podmínky, z výsledků měření změn hlavního kritéria (viz obr. 7.1) se jeví jako vhodný poměr doby běhu algoritmu a kvality rozvrhu ukončit algoritmus po 1000 iteracích. K drobným zlepšením sice docházelo i v následujících iteracích, avšak vzhledem k celkovému počtu přiřazených minut byly tyto změny zanedbatelné.

Kapitola 8

Závěr

Domácí péče, jejíž poskytování je v posledních letech umožňováno čím dál častěji, je alternativní způsob zdravotní péče cílený zejména na seniory a osoby se sníženou schopností pohybu či jiným postižením. Pobyt v domácím prostředí má pozitivní vliv na psychiku a redukuje často problematický transport pacienta do nemocnice.

Cílem této práce bylo analyzovat problematiku rozvrhování domácí péče a navrhnout a implementovat algoritmus, který vytvoří rozvrh na základě specifikovaných nároků sester, objednaných asistencí a jejich případných omezení. Tento problém patří do třídy $\mathcal{NP}$ hard problémů, jakožto kombinace problému obchodního cestujícího a nurse rostering problému. Vhodným přístupem k řešení podobných úloh z této třídy složitosti jsou heuristiky, které se snaží najít suboptimální řešení, na rozdíl od exaktních algoritmů, pro které jsou již poměrně malé instance nevyřešitelné v přijatelném časovém limitu.

Rešerše byla zaměřena na současné přístupy k řešení problému rozvrhování domácí péče, zejména na analýzu formulace řešeného problému, konkrétního algoritmu pro hledání rozvrhu a vyhodnocení výsledků. Všechny analyzované články popisovaly algoritmy založené na heuristickém přístupu, přičemž v některých byl navíc problém formulován také jako úloha pro celočíselné programování a výsledky na malých vstupních instancích byly porovnány s řešením nalezeným ILP solverem. Autoři navíc následně demonstrovali vlastní implementovanou heuristiku, pomocí které byli schopni vyřešit několikanásobně větší instance v přijatelném čase. Často používaným přístupem je použití metaheuristických algoritmů, což se na základě publikovaných výsledků ukázalo jako vhodný nástroj.

Na základě provedené analýzy a rešerše byl v rámci této práce nejprve navržen abstraktní framework pro řešení problému, přičemž byl kladen velký důraz na konfigurovatelnost a modularizaci, díky které jsou jednotlivé komponenty algoritmu na sobě nezávislé, což umožňuje snadné úpravy či nahrazení jednotlivých částí. Komponenta `Solver`, která je zodpovědná za samotnou tvorbu rozvrhu je rozdělena do dvou fází – nejprve je vytvořen částečný rozvrh a ten je poté iterativním procesem vylepšován *greedy* algoritmem. Tento framework byl následně využit pro implementaci několika heuristik, pomocí kterých byl řešen problém formulovaný na začátku analýzy v této práci. Ten zahrnoval kromě standardních atributů problému rozvrhování domácí péče také asistence s časovými okny a řadu omezení. Ta byla rozdělena do dvou kategorií; první jsou soft constraints, tedy omezení, jejichž porušení neovlivňují platnost rozvrhu (např. sestra preferovaná klientem), avšak při jejich splnění je zvýšena hodnota kriteriální funkce. Druhou kategorií jsou hard constraints, tedy omezení,

která musí být při umístění asistence do rozvrhu splněna, v opačném případě nebude rozvrh platný. Příkladem tohoto typu omezení je požadovaná sestra nebo požadované kvalifikace pro výkon péče. Veškerá konfigurace algoritmu, která zahrnuje například penalizační konstanty pro porušená omezení použité v kriteriální funkci, ale také použité implementace heuristik či způsob tvorby počátečního rozvrhu, jejich parametry a mnoho dalších je striktně oddělena od zdrojového kódu. Velké množství změn chování algoritmu může být tedy upraveno a přizpůsobeno konkrétním požadavkům pouze úpravou těchto konfiguračních souborů.

Implementovaný algoritmus byl následně testován na několika sadách vstupních dat (reálných i generovaných) a na základě naměřených výsledků byla vyhodnocena jeho efektivita. Při experimentech byly použity nejen různé vstupní instance s rozdílnými atributy, ale také různé konfigurace algoritmu. Po analýze výsledných dat bylo nakonec určeno, které heuristiky jsou vhodné k použití na konkrétní vstupní data a jaká je nejvhodnější konfigurace parametrů algoritmu. Na generovaných datech bylo dosaženo více než 80% přiřazených asistencí na malých i velkých instancích, na reálných datech pak 75 – 95% v závislosti na hodnotách v matici vzdáleností. Správnost implementace algoritmu je kontrolována intenzivními jednotkovými a integračními testy a řadou validací, které jsou prováděny nad výslednými rozvrhy, jež jsou výstupem algoritmu.

8.1 Budoucí rozvoj

Cílem implementovaného algoritmu bylo nalézt rozvrh pro úlohu rozvrhování domácí péče, který by vyhovoval objednaným požadavkům a neporušoval žádná omezení a ostatní invarianty. Jak je ukázáno v kapitole 7, výsledky měření efektivity algoritmu jsou poměrně uspokojivé na všech testovaných vstupních instancích, nicméně je zde dostatek prostoru pro budoucí rozvoj algoritmu v různých ohledech.

Jedním z možných kroků je přidání podpory nových typů omezení, která by učinila algoritmus aplikovatelný na větší množství reálných problémů. Jelikož byl při návrhu algoritmu brán zřetel na budoucí rozšiřitelnost, přidávání nových omezení je výrazně usnadněno – pouze postačuje implementovat způsob načítání tohoto omezení ze vstupního souboru, logiku vyhodnocování, zda došlo k jeho porušení, či nikoliv, a případně přidat penalizaci do výpočtu kriteriální funkce. Příkladem omezení, které by rozšířilo použitelnost algoritmu, může být minimální a maximální utilizace sestry, která by umožnila zaručení plnění smluveného úvazku a zlepšila by predikovatelnost výsledného rozvrhu z pohledu sester. Tento aspekt je v literatuře často zmiňován a věnoval se mu např. článek D. Guericke a kol. [22].

Jinou oblastí pro zlepšení kvality algoritmu je implementace nových insertion a removal heuristik, díky kterým by bylo možné ve druhé fázi lépe identifikovat místa v rozvrhu, na která by měly být umístěny nepřiřazené asistence, případně kam přesunout asistence, které jsou již některé sestře přiřazeny. Jak se ukázalo při měření efektivity, algoritmus velice dobře funguje při kombinaci několika heuristik s použitím ruletové selekce. Díky tomu lze logiku nových heuristik zaměřit na užší skupinu asistencí, například pouze pro asistence s omezením s tím, že bude poté použita v kombinaci s další, obecnější heuristikou (např. `LongestFittingInsertionHeuristic`). Jednou z možností je implementace heuristiky, která by byla schopná efektivněji zacházet s asistencemi s časovým oknem, jelikož v současnosti je hlavní snahou jejich přiřazení na nejdřívější možný termín, což nemusí být vždy ideální postup.

Další vhodnou úpravou, která by výrazně zlepšila reálnou použitelnost algoritmu, je vytvoření grafického rozhraní, které by usnadnilo zadávání vstupních dat, případně konfiguraci parametrů algoritmu pro koncové uživatele, kterými budou zaměstnanci zdravotnických firem odpovědní za tvorbu rozvrhu. Jelikož je pravděpodobné, že alespoň část objednaných asistencí se nebude mezi jednotlivými měsíci měnit, bylo by užitečné například ukládat poslední vložená data či jiným způsobem usnadnit konfiguraci. Formát výsledného rozvrhu by také mohl mít grafickou reprezentaci, což by mohlo zjednodušit případné manuální změny v rozvrhu, například v případě nemoci či při změně požadavků.

8.2 Shrnutí

Cílem práce bylo analyzovat problematiku rozvrhování domácí péče a navrhnout, implementovat a otestovat algoritmus pro řešení tohoto $\mathcal{NP}$ hard problému. Pro hledání řešení bylo implementováno několik různých heuristik, jejichž efektivita byla následně měřena na řadě vstupních instancí a podle vlastností vstupních dat a měřeného kritéria bylo vyhodnoceno, kterou z heuristik či jakou jejich kombinaci je nejvhodnější použít. Pomocí implementovaného algoritmu se podařilo najít výsledný rozvrh jak pro reálná data, tak pro generovaná data v řádu vteřin, v případě větších dat pak v řádu minut, což je čas srovnatelný s výsledky publikovanými v literatuře. Všechny cíle, které byly stanoveny pro tuto práci, byly splněny.

Literatura

[1] AUSIELLO, G. et al. *Complexity and approximation: Combinatorial optimization problems and their approximability properties.* Berlin : Springer Science & Business Media, 2012.

[2] BEKTAS, T. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega.* 2006, 34, 3, s. 209–219.

[3] BERRADA, I. – FERLAND, J. A. – MICHELON, P. A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Economic Planning Sciences.* 1996, 30, 3, s. 183 – 193. ISSN 0038-0121. doi: https://doi.org/10.1016/0038-0121(96)00010-9. Dostupné z: <`http://www.sciencedirect.com/science/article/pii/0038012196000109`>.

[4] BONDY, J. A. – MURTY, U. S. R. et al. *Graph theory with applications.* 290. Princeton : Citeseer, 1976.

[5] BRUCATO, C. *The Traveling Salesman Problem.* PhD thesis, University of Pittsburgh, 2013.

[6] BURKE, E. et al. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of metaheuristics.* Berlin: Springer, 2003. s. 457–474.

[7] BURKE, E. K. et al. The state of the art of nurse rostering. *Journal of scheduling.* 2004, 7, 6, s. 441–499.

[8] CHIDAMBER, S. R. – KEMERER, C. F. A metrics suite for object oriented design. *IEEE Transactions on software engineering.* 1994, 20, 6, s. 476–493.

[9] CHRISTOFIDES, N. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[10] COHN, M. *Succeeding with agile: software development using Scrum.* London : Pearson Education, 2010.

[11] COOK, W. J. *In pursuit of the traveling salesman: mathematics at the limits of computation.* Princeton, NJ, USA : Princeton University Press, 2011.

[12] CORMEN, T. H. et al. *Introduction to Algorithms.* Cambridge, Massachusetts London, England : The MIT Press, 3rd edition, 2009.

[13] DANTZIG, G. *Linear programming and extensions.* Princeton, NJ, USA : Princeton university press, 2016.

[14] DANTZIG, G. B. – RAMSER, J. H. The truck dispatching problem. *Management science.* 1959, 6, 1, s. 80–91.

[15] DE CAUSMAECKER, P. – BERGHE, G. V. A categorisation of nurse rostering problems. *Journal of Scheduling.* 2011, 14, 1, s. 3–16.

[16] DEMLOVÁ, M. Teorie algoritmů, 2017. Dostupné z: <`http://math.feld.cvut.cz/demlova/teaching/tal/predn_tal.html`>.

[17] HARTOG, S. On the Complexity of Nurse Scheduling Problems. Master's thesis, Utrecht University, Utrecht, Netherlands, 2016.

[18] DU, G. – LIANG, X. – SUN, C. Scheduling optimization of home health care service considering patients' priorities and time windows. *Sustainability.* 2017, 9, 2, s. 253.

[19] FESTA, P. A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems. In *Transparent Optical Networks (ICTON), 2014 16th International Conference on*, s. 1–20. IEEE, 2014.

[20] FOWLER, M. Inversion of control containers and the dependency injection pattern. 2004.

[21] GOSLING, J. et al. *The Java Language Specification, Java SE 8 Edition (Java Series).* Boston, USA : Addison-Wesley Professional, 2014. ISBN: 013390069X.

[22] GUERICKE, D. – SUHL, L. The home health care problem with working regulations. *OR Spectrum.* 2017, 39, 4, s. 977–1010.

[23] HIERMANN, G. et al. Metaheuristics for solving a multimodal home-healthcare scheduling problem. *Central European Journal of Operations Research.* 2015, 23, 1, s. 89–113.

[24] JOUBERT, J. W. *An integrated and intelligent metaheuristic for constrained vehicle routing.* PhD thesis, University of Pretoria, 2007.

[25] LAPORTE, G. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research.* 1992, 59, 2, s. 231–247.

[26] LÓPEZ-IBÁÑEZ, M. et al. The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization. *Applied Soft Computing.* 2013, 13, 9, s. 3806–3815.

[27] MANKOWSKA, D. S. – MEISEL, F. – BIERWIRTH, C. The home health care routing and scheduling problem with interdependent services. *Health care management science.* 2014, 17, 1, s. 15–30.

[28] MILLER, C. E. – TUCKER, A. W. – ZEMLIN, R. A. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM).* 1960, 7, 4, s. 326–329.

[29] MISIR, M. et al. Hyper-heuristics with a dynamic heuristic set for the home care scheduling problem. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, s. 1–8. IEEE, 2010.

[30] MISIR, M. et al. Security personnel routing and rostering: a hyper-heuristic approach. In *Proceedings of the 3rd International Conference on Applied Operational Research*, 3, s. 193–205. Tadbir, 2011.

[31] POP, P. C. et al. Heuristic algorithms for solving the generalized vehicle routing problem. *International Journal of Computers Communications & Control.* 2011, 6, 1, s. 158–165.

[32] POŠÍK, P. Evolutionary optimization algorithms. *Optimization, Local search, Evolutionary methods.* 2016.

[33] ROPKE, S. – PISINGER, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science.* 2006, 40, 4, s. 455–472.

[34] SHI, Y. – BOUDOUH, T. – GRUNDER, O. A hybrid genetic algorithm for a home health care routing problem with time window and fuzzy demand. *Expert Systems with Applications.* 2017, 72, s. 160–176.

[35] SOLOS, I. P. – TASSOPOULOS, I. X. – BELIGIANNIS, G. N. A generic two-phase stochastic variable neighborhood approach for effectively solving the nurse rostering problem. *Algorithms.* 2013, 6, 2, s. 278–308.

[36] STEEG, J. – SCHRÖDER, M. A hybrid approach to solve the periodic home health care problem. In *Operations Research Proceedings 2007.* Berlin: Springer, 2008. s. 297–302.

[37] TORRES-RAMOS, A. et al. Mathematical model for the home health care routing and scheduling problem with multiple treatments and time windows. In *Proceedings of the 1st International Conference on Mathematical Methods & Computational Techniques in Science & Engineering*, s. 140–145, 2014.

[38] TOTH, P. – VIGO, D. *The vehicle routing problem.* 3600 Market Street, Philadelphia, PA 19104-2688 USA : SIAM, 2002.

[39] VAN LAARHOVEN, P. J. – AARTS, E. H. Simulated annealing. In *Simulated annealing: Theory and applications.* Berlin: Springer, 1987. s. 7–15.

[40] VAN LEEUWEN, J. *Handbook of theoretical computer science (vol. A): algorithms and complexity.* Cambridge, MA, USA : MIT Press, 1991.

[41] VLISSIDES, J. et al. Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley.* 1995, 49, 120, s. 11.

[42] YUAN, Z. – FÜGENSCHUH, A. *Home health care scheduling: a case study.* Hamburg : Helmut-Schmidt-Univ., Professur für Angewandte Mathematik, 2015.

[43] ZWEIFEL, P. – FELDER, S. – MEIERS, M. Ageing of population and health care expenditure: a red herring? *Health economics*. 1999, 8, 6, s. 485–496.

[44] Český statistický úřad. Senioři v mezinárodním srovnání. 2017. Dostupné z: <https://www.czso.cz/documents/10180/46239581/310034171.pdf/73c5195d-8162-41ea-b7c2-a7b64ecd01a3?version=1.0>.

Příloha A

Seznam použitých zkratek

AILTA	Adaptive iteration limited threshold accepting
ALNS	Adaptive large neighborhood search
API	Application programming interface
AVNS	Adaptive variable neighborhood search
CI	Continuous integration
CP	Constraint programming
EA	Evoluční algoritmus
FQN	Fully qualified name
GA	Genetický algoritmus
HHCS	Home health care scheduling
HHC	Home health care
IE	Improving or equal
ILP	Integer linear programming
ILTA	Iteration limited threshold accepting
JRE	Java Runtime Environment
LP	Lineární programování
LS	Local search
m-TSP	Multiple travelling salesman problem
MILP	Mixed integer linear programming
NRP	Nurse rostering problem

PVRPTW Periodic vehicle routing problem with time windows

SA Simulated annealing

SLOC Source lines of code

TSPTW Travelling salesman problem with time windows

TSP Travelling salesman problem

VRPTW Vehicle routing problem with time windows

VRP Vehicle routing problem

Příloha B

Přehled vstupních instancí

Testování algoritmu a měření efektivity (viz kapitola 7) probíhalo na několika sadách vstupních instancí. V této příloze jsou popsány všechny použité reálné i generované instance, společně s relevantními statistikami, jako je např. počet asistencí s omezením, celková délka objednaných asistencí apod.

Tabulka B.1: Atributy reálné instance

Atribut	**Hodnota**
Název instance	`real`
Počet sester	26
Počet klientů	68
Počet asistencí	678
Celková délka asistencí (min)	109 738
Celková dostupnost sester (min)	143 758
Počet asistencí s omezením pref. sestra	0
Počet asistencí s omezením pož. sestra	0
Počet asistencí s omezením pož. kvalifikace	0
Počet asistencí s časovým oknem	0

Tabulka B.2: Atributy generované instance 1

Atribut	**Hodnota**
Název instance	`10_30`
Počet sester	10
Počet klientů	30
Počet asistencí	930
Celková délka asistencí (min)	118 455
Celková dostupnost sester (min)	148 800
Počet asistencí s omezením pref. sestra	119
Počet asistencí s omezením pož. sestra	152
Počet asistencí s omezením pož. kvalifikace	49
Počet asistencí s časovým oknem	253

Tabulka B.3: Atributy generované instance 2

Atribut	**Hodnota**
Název instance	`10_30_tw`
Počet sester	10
Počet klientů	30
Počet asistencí	930
Celková délka asistencí (min)	118 705
Celková dostupnost sester (min)	148 800
Počet asistencí s omezením pref. sestra	122
Počet asistencí s omezením pož. sestra	143
Počet asistencí s omezením pož. kvalifikace	67
Počet asistencí s časovým oknem	750

Tabulka B.4: Atributy generované instance 3

Atribut	**Hodnota**
Název instance	`50_150`
Počet sester	50
Počet klientů	150
Počet asistencí	4650
Celková délka asistencí (min)	596 805
Celková dostupnost sester (min)	744 000
Počet asistencí s omezením pref. sestra	566
Počet asistencí s omezením pož. sestra	672
Počet asistencí s omezením pož. kvalifikace	327
Počet asistencí s časovým oknem	3672

Příloha C

Přehled konfigurací algoritmu

V kapitole 7 jsou analyzovány výsledky měření efektivity algoritmu na různých instancích a s různými hodnotami parametrů algoritmu. Přehled všech použitých konfigurací a hodnot relevantních parametrů je zobrazen v následujících tabulkách. V tabulkách však nejsou uvedeny všechny použité parametry, jelikož řada z nich byla stejná ve všech případech – například ukončovací podmínka je vždy založena na počtu iterací, nebo používaná implementace kriteriální funkce je ve všech případech `PenalizingFunction`. Kopie konfiguračních souborů s kompletním přehledem všech parametrů lze nalézt v adresáři `benchmarking/config` v kořenovém adresáři projektu.

Tabulka C.1: Konfigurace 1

Atribut	**Hodnota**
Název konfigurace	`conf_1`
Tvorba počátečního rozvrhu	`FirstAvailableInitialAssignment`
Počet iterací	100
Pravděpodobnosti removal heuristik	`RandomRemovalHeuristic` – 0.5 `ViolatedRemovalHeuristic` – 0.5
Pravděpodobnosti insertion heuristik	`LongestFittingInsertionHeuristic` – 0.5 `ReplacementInsertionHeuristic` – 0.5
Počet odebraných asistencí (`RandomRemovalHeuristic`)	2
Počet odebraných asistencí (`ViolatedRemovalHeuristic`)	2
Počet iterací (`LongestFittingInsertionHeuristic`)	50
Počet iterací (`PredicateInsertionHeuristic`)	3
Nahrazování asistencí (`PredicateInsertionHeuristic`)	`true`
Faktor penalizace za nepřiřazenou asistenci	-5
Faktor penalizace za idle time	-1
Faktor penalizace pro omezení pref. sestry	-2

Tabulka C.2: Konfigurace 2

Atribut	**Hodnota**
Název konfigurace	`conf_2`
Tvorba počátečního rozvrhu	`FirstAvailableInitialAssignment`
Počet iterací	100
Pravděpodobnosti removal heuristik	`RandomRemovalHeuristic` – 0.5 `ViolatedRemovalHeuristic` – 0.5
Pravděpodobnosti insertion heuristik	`LongestFittingInsertionHeuristic` – 0.5 `ReplacementInsertionHeuristic` – 0.5
Počet odebraných asistencí (`RandomRemovalHeuristic`)	2
Počet odebraných asistencí (`ViolatedRemovalHeuristic`)	2
Počet iterací (`LongestFittingInsertionHeuristic`)	50
Počet iterací (`PredicateInsertionHeuristic`)	3
Nahrazování asistencí (`PredicateInsertionHeuristic`)	`true`
Faktor penalizace za nepřiřazenou asistenci	-5
Faktor penalizace za idle time	-1
Faktor penalizace pro omezení pref. sestry	-50

Tabulka C.3: Konfigurace 3

Atribut	**Hodnota**
Název konfigurace	`conf_3`
Tvorba počátečního rozvrhu	`FirstAvailableInitialAssignment`
Počet iterací	10000
Pravděpodobnosti removal heuristik	`RandomRemovalHeuristic` – 0.5 `ViolatedRemovalHeuristic` – 0.5
Pravděpodobnosti insertion heuristik	`LongestFittingInsertionHeuristic` – 0.5 `ReplacementInsertionHeuristic` – 0.5
Počet odebraných asistencí (`RandomRemovalHeuristic`)	2
Počet odebraných asistencí (`ViolatedRemovalHeuristic`)	2
Počet iterací (`LongestFittingInsertionHeuristic`)	50
Počet iterací (`PredicateInsertionHeuristic`)	3
Nahrazování asistencí (`PredicateInsertionHeuristic`)	`true`
Faktor penalizace za nepřiřazenou asistenci	-5
Faktor penalizace za idle time	-1
Faktor penalizace pro omezení pref. sestry	-2

Příloha D

Obsah přiloženého CD

```
|-- home-health-care/              zdrojové kódy algoritmu
| |-- benchmarking/                výsledky benchmarkingu
|   |-- config/                    konfigurační soubory
|     |-- generator-params.txt     parametry použité pro generování instancí
|   |-- benchmarking.ods           výsledky měření
| |-- dist/                        zkompilované zdrojové kódy
| |-- docs/                        dokumentace
|   |-- javadoc/                   dokumentace zdrojového kódu v HTML
| |-- scripts/                     pomocné skripty (Python a Bash)
| |-- src/                         zdrojové kódy
| |-- test/                        zdrojové kódy testů
|   |-- resources/                 vygenerované instance
| |-- README.md                    návod k instalaci
|
|-- text/                          text diplomové práce
| |-- src/                         zdrojové kódy pro LaTeX
| |-- master.pdf                   soubor s textem diplomové práce
|
|-- README.txt                     tento obsah CD
```