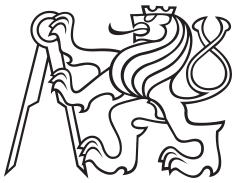
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Optimalizace logistiky pro recyklaci odpadu

Bc. Ondřej Mareš

Školitel: doc. Ing. Přemysl Šůcha, Ph. D.
Obor: Otevřená informatika – Softwarové inženýrství
Květen 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Mareš** Jméno: **Ondřej** Osobní číslo: **474776**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Optimalizace logistiky pro recyklaci odpadu

Název diplomové práce anglicky:

Logistics Optimization for Waste Recycling

Pokyny pro vypracování:

Logistika související s odpadem určeným k recyklaci je téma které je v současnosti považováno za prioritní. Efektivní tok odpadu od producentů odpadu k zpracovávatelským závodům je klíčovým aspektem pro funkčnost systému recyklace odpadu. Cílem práce je popsat problém transportu odpadu v návaznosti na jeho recyklaci, který je v literatuře označován jako úloha reverzní logistiky. Pro tento problém vytvořte algoritmus, pro nalezení efektivního transportu odpadu. Zadání má tyto části:
1) proveďte rešerši existující literatury,
2) zmapujte dostupnost dat týkajících se recyklace odpadu v České republice,
3) formálně popište úlohu optimalizace svozu odpadu,
4) navrhněte a implementujte algoritmus pro optimalizaci svozu odpadu,
5) vyhodnoťte výkonnost algoritmu a na získaných datech posuďte jeho dopad na efektivitu svozu odpadu.

Seznam doporučené literatury:

[1] Elodie Suzanne, Nabil Absi, Valeria Borodin, Towards circular economy in production planning: Challenges and opportunities, European Journal of Operational Research,
Volume 287, Issue 1, 2020, Pages 168-190.
[2] Mor, A., Speranza, M.G. Vehicle routing problems over time: a survey. 4OR-A Quarterly Journal of Operations Research, 18, 2020, Pages 129–149.
[3] Reza Moghdani, Khodakaram Salimifard, Emrah Demir, Abdelkader Benyettou,
The green vehicle routing problem: A systematic literature review, Journal of Cleaner Production, Volume 279, 2021.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Přemysl Šůcha, Ph.D. katedra řídicí techniky FEL

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

Datum zadání diplomové práce: **21.01.2022** Termín odevzdání diplomové práce: **20.05.2022**

Platnost zadání diplomové práce: **30.09.2023**

doc. Ing. Přemysl Šůcha, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. 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.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Chtěl bych poděkovat vedoucímu mé diplomové práce doc. Ing. Přemyslu Šůchovi, Ph. D. za věcné připomínky, cenné rady a vstřícnost při konzultacích.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu. V Praze 4. května 2022

Abstrakt

Práce se zabývá logistikou v oblasti zpracování odpadů. Konkrétně se jedná o problém nalezení efektivního způsobu přepravy odpadu od jeho producentů k jeho zpracovatelům. Klíčovým aspektem problému je zvýhodnění zpracovatelských závodů, které odpad efektivně recyklují, oproti těm závodům, které odpad pouze likvidují. Tato práce popisuje současný stav v oblasti nakládání s odpadem, mapuje dostupnost dat týkajících se zpracování odpadu a shrnuje existující literaturu vážící se k řešenému problému. Jádro práce se věnuje návrhu algoritmu pro optimalizaci logistické úlohy s odpady. Vyhodnocení chování navrženého algoritmu je založeno na reálných datech z České republiky.

Klíčová slova: optimalizace, cirkulární ekonomika, reverzní logistika, algoritmus, guided local search

Školitel:
doc. Ing. Přemysl Šůcha, Ph. D.

Abstract

The thesis deals with logistics in the field of waste processing. Specifically, it deals with the problem of finding an efficient way of transporting waste from its producers to its processors. A key aspect of the problem is the preference for processing facilities that efficiently recycle waste over those facilities that only dispose of waste. This paper describes the current state in waste management, maps the availability of data related to waste processing, and summarizes the existing literature related to the problem. The core of the thesis is devoted to the design of an algorithm for the optimization of the waste logistics problem. The evaluation of the behaviour of the proposed algorithm is based on real data from the Czech Republic.

Keywords: optimization, circular economy, reverse logistics, algorithm, guided local search

Obsah

Seznam použitých zkratek 1

1 Úvod 3

Část I
Průzkum

2 Cirkulární ekonomika 7
2.1 Hlavní témata CE 9
2.1.1 Od recyklace produktu k recyklaci surovin 9
2.1.2 Doprovodné (co-product) a vedlejší produkty (by-product) . 10
2.1.3 Pilíře udržitelnosti 10
2.1.4 Reverzní logistika a nakládání s odpady 10
2.2 Reverzní logistika 11

3 Problém směřování vozidel (VRP) 13
3.1 Úvod 13
3.2 Historický pohled 15
3.3 Lokální prohledávání (angl. Local Search) 15
3.4 Meta-heuristiky 16
3.4.1 Založené na jednom řešení . . 18
3.4.2 Založené na množině řešení . . 19

4 Současný stav v České republice 21
4.1 Systém a obce 21
4.2 Dostupná data 22
4.2.1 Místa nakládání s odpadem . 22
4.2.2 Kapacity, množství odpadu a flotila 23
4.2.3 Shrnutí 24

Část II
Problém

5 Formulace problému 27
5.1 Zařazení problému 29
5.2 Příklad 30
5.2.1 Instance 30
5.2.2 Možné řešení 32

6 Řešení problému (návrh algoritmu) 35
6.1 Guided Local Search (GLS) 35
6.1.1 Vlastnosti 36
6.1.2 Penalizace vlastností 37
6.1.3 Pseudokód 39
6.2 Konstrukce iniciálního řešení . . . 39
6.2.1 Mapování producentů ke konzumentům 39
6.2.2 Vytvoření základních tras . . . 40
6.2.3 Mapování základních tras k vozidlům 42
6.2.4 Spojení základních tras na vozidlech 46
6.3 LS operátory 47
6.3.1 Jedna trasa 47
6.3.2 Dvojice tras 51

Část III
Prototyp

7 Implementace 59
7.1 Generátor instancí 59
7.2 Implementace GLS 62
7.3 Filtrace dat z ISOH 66

8 Experimenty 69
8.1 Instance 69
8.1.1 Reálné instance 69
8.1.2 Abstraktní instance 75
8.2 Parametry stroje pro měření . . . 76
8.3 Pořadí operátorů 76
8.3.1 Vyhodnocení 77
8.4 Přínos operátorů 81
8.4.1 Vyhodnocení 81
8.5 Vliv parametru λ 84
8.5.1 Vyhodnocení 84
8.6 Velikost flotily 87
8.6.1 Vyhodnocení 87

9 Závěr 91
9.1 Návrhy na budoucí postup 91

Přílohy

A Literatura 95

B Seznam digitálních součástí 99

Obrázky

2.1 Rozdíl mezi lineární a cirkulární ekonomikou (převzato z [2]) 8
2.2 Překryvy cílů CE (přeloženo z [4]) 9

3.1 Ukázka možného řešení VRP s 6 zákazníky 14
3.2 Průběh LS v prostoru řešení (přeloženo z [21]) 16
3.3 Přehled meta-heuristických algoritmů (převzato z [17]) 18

4.1 Chybná pozice zařízení v ISOH . 23

5.1 Legenda ke grafům 30
5.2 Zadání problému s 3 producenty a 2 konzumenty 31
5.3 Ukázkové řešení tras vozidel 32

6.1 Příklad z části 5.2 – základní trasy pro producenty 1, 2 a 3 41
6.2 Příklad z části 5.2 – trasy pro vozidlo 2 před spojením 44
6.3 Příklad z části 5.2 – iniciální trasa pro vozidlo 2 46
6.4 Ukázka 2-opt 48
6.5 Operátor přesunu producenta . . 54
6.6 Operátor výměny konzumentů . . 56

8.1 Graf velikosti flotily – průměry bodů 88

Tabulky

2.1 Přehled definic reverzní logistiky podle [5] 11

3.1 Přehled definic meta-heuristik podle [25] 17

5.1 Přehled producentů v ukázkovém zadání 31
5.2 Přehled konzumentů v ukázkovém zadání 32
5.3 Přehled vozidel v ukázkovém zadání 32

7.1 Argumenty generátoru instancí . 61
7.2 Argumenty přijímané implementací GLS 63
7.3 Odhadnutí hlavních XML tagů z ISOH 66

8.1 Počet třídíren a zpracovatelů podle typů odpadu v ČR 69
8.2 Počty zařízení Středočeský kraj a Praha 71
8.3 Počty zařízení Jihočeský kraj . . . 71
8.4 Počty zařízení Jihomoravský kraj 71
8.5 Počty zařízení Moravskoslezský kraj 72
8.6 Počty zařízení Plzeňský kraj . . . 72
8.7 Počty zařízení Kraj Vysočina . . . 72
8.8 Počty zařízení Liberecký kraj . . . 72
8.9 Počty zařízení Ostrava 73
8.10 Velikost flotil pro oblasti 73
8.11 Souhrn argumentů generátoru pro reálné instance 74
8.12 Označení reálných instancí 75
8.13 Přehled vrcholů abstraktních instancí 75
8.14 Přehled argumentů abstraktních instancí pro generování 76
8.15 Parametry stroje pro měření . . 76
8.16 Výsledky měření cen různého pořadí operátorů 79
8.17 Výsledky měření času různého pořadí operátorů (ve vteřinách) . . . 80
8.18 Výsledky měření cen s vynecháním operátoru 82

8.19 Výsledky měření času s vynecháním operátoru (ve vteřinách) 83
8.20 Výsledky měření cen s různým parametrem λ 85
8.21 Výsledky měření času s různým parametrem λ (ve vteřinách) 86
8.22 Výsledky měření vlivu velikostí flotil na výsledek 89

Seznam použitých zkratek

ACO	Ant-colony optimization
ALNS	Adaptive Large Neighborhood Search
CE	Cirkulární ekonomika
CEP	Circular Economy Package
CSP	Constraint Satisfaction Problems
CVRP	Problém kapacitního směřování vozidel (angl. Capacitated Vehicle Routing Problem)
ČR	Česká republika
EU	Evropská unie
GA	Genetický algoritmus (angl. Genetic Algorithm)
GLS	Guided Local Search
HVRP	VRP s heterogenní flotilou vozidel (angl. Heterogeneous Vehicle Routing Problem)
ISOH	Registr zařízení, obchodníků a spisů
ISSaR	Informační systém statistiky a reportingu v životním prostředí
LS	Lokální prohledávání (angl. Local Search)
MDVRP	VRP s více depy (angl. Multi-depot Vehicle Routing Problem)
MŽP	Ministerstvo životního prostředí
PSO	Particle-swarm optimization
PVRP	VRP s periodickým plánováním (angl. Periodic Vehicle Routing Problem)
SA	Simulated Annealing
SAT	Problém splnitelnosti booleovské formule (angl. Boolean satisfiability problem)
TS	Tabu Search
TSP	Problém obchodního cestujícího (angl. Traveling Salesman Problem)
VNS	Variable Neighborhood Search
VRP	Problém směřování vozidel (angl. Vehicle Routing Problem)
VRPPD	VRP s vyzvedáváním a dodávkami (angl. Vehicle Routing Problem with Pickups and Deliveries)

VRPTW	VRP s časovými okny (angl. Vehicle Routing Pronlem with Time Windows)
XML	Extensible Markup Language
ZEVO	Zařízení pro energetické využití odpadů

Kapitola 1
Úvod

Podle [22] v Evropské unii (dále EU) rezonuje čím dál větší snaha o ochranu životního prostředí. Vyspělé země produkují na jednoho obyvatele více odpadu, než země rozvojové – růst HDP přímo souvisí s větší produkcí odpadu (viz [23]). Objevuje se tedy snaha přechodu k cirkulární ekonomice (dále CE). Jedná se o ekonomický systém, který vzniká jako protiklad k lineárnímu systému s otevřeným koncem (vyrobit, spotřebovat a zlikvidovat). V CE dochází k uzavírání materiálových smyček, například pomocí recyklace, více o CE se dozvíte v kapitole 2. V květnu roku 2018 vstoupil v platnost balíček Evropské komise k CE, tzv. CEP (Circular Economy Package). Balíček dává členským státům EU cíle a způsoby jak realizovat v daném časovém horizontu přechod k CE, obsah tohoto balíčku by měl být implementován do legislativ států co nejdříve. Jedním z hlavních cílů CEP je navyšování podílu recyklovaného komunálního odpadu. Existuje také Směrnice o skládkování odpadů, která si klade za cíl pro rok 2035 skládkovat pouze 10 % produkce odpadu [22].

V odpadovém hospodářství, zejména pak v EU, tedy dochází k významným změnám, ke kterým je potřeba vyvíjet nástroje pro podporu rozhodování, plánování investic a udržitelnou ekonomickou stabilitu. Plánování logistických procesů ve zpracování odpadu tedy nabývá více na významu [22].

Z logistických úloh se nabízí návrh svozových tras pro sběr komunálního odpadu, tímto problémem se zabývá například práce Ing. Vlastimíra Nevrlého z VUT v Brně (viz [22]). Nejedná se ale o jedinou logistiku, která je při zpracování odpadu potřebná. Komunální odpad se sváží zpravidla do třídíren (na cestě může předcházet ještě sběrný dvůr, pokud není sběrný dvůr sloučen s třídírnou). V třídírnách vznikají balíky odpadů, různých typů – železné/neželezné kovy, papíry různých kvalit, plasty apod, které mají svoji poptávku, ale také z procesu třídění vzejdou odpady, které další využití postrádají a končí v lepších případech v ZEVO (spalovny s energetickým využitím odpadu), nebo v obyčejných spalovnách, případně na skládkách. Odběratelé těchto komodit mají zpravidla nějakou kapacitu, kterou jsou schopni pojmout.

Tato práce se zabývá návrhem a implementací prototypu algoritmu řešící logistiku svozu odpadu právě v oblasti od třídíren (producentů odpadu), k jejich zpracovatelům (konzumentům odpadu) upřednostňující recyklaci před ZEVO a skládkováním odpadu. Takovýto logistický problém je úzce spjat s úlohou směřování vozidel (Vehicle Routing Problem, dále VRP, viz

kapitola 3), která je zobecněním problému obchodního cestujícího (Traveling Salesman Problem, dále TSP) [22]. Práce se může stát podkladem pro budoucí rozvoj tohoto logistického problému a budoucím konkrétním využitím v praxi na reálných situacích.

Nejprve je v práci proveden průzkum, kdy je provedena rešerše literatury související s CE (viz kapitola 2) a VRP (viz kapitola 3). Následně je prozkoumávána současná situace v České republice a zmapována dostupnost dat (viz kapitola 4). Potom se dostáváme k samotnému problému a jeho formulaci (viz kapitola 5), po které následuje návrh řešení (viz kapitola 6). Na závěr jsou na prototypu navrženého řešení provedeny experimenty (viz kapitoly 7 a 8).

Část I

Průzkum

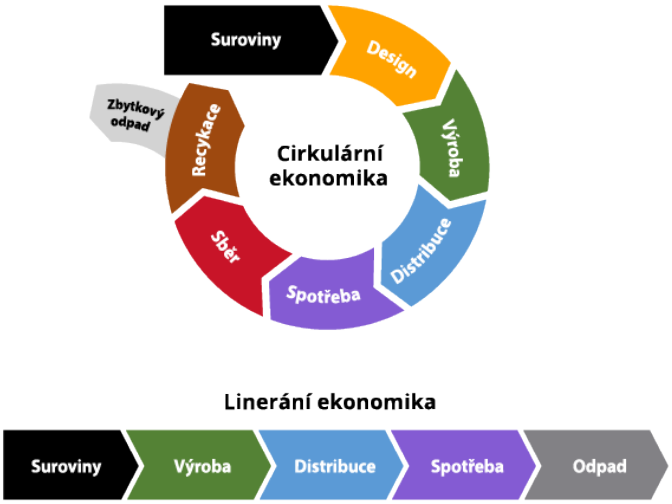
Kapitola 2

Cirkulární ekonomika

V současné době mezinárodního obchodu, globálního oteplování a vyčerpávání přírodních zdrojů Země nás ochota vytvářet udržitelné a konkurenceschopné výhody nutí přestat uvažovat lineárně (vyrobit, spotřebovat a zlikvidovat) a přejít k oběhovému přístupu uzavíráním materiálových smyček, který se nazývá cirkulární ekonomika (též oběhové hospodářství, dále CE) [1]. Jedná se o ekonomický systém, který vzniká jako protiklad k lineárnímu systému s otevřeným koncem.

V klasickém lineárním systému se produkt vyrábí z nově získaných, vytěžených surovin, produkt se po výrobě distribuuje a následně spotřebuje, přičemž po spotřebě se stává odpadem bez dalšího využití a končí na skládkách, případně ve spalovnách.

Oproti tomu v CE se spotřebovaný produkt nestává odpadem bez dalšího využití, nýbrž je využit k výrobě nového produktu, nebo energie. Takto uzavřený systém přidává nové fáze do života produktu, jak je ilustrováno na obrázku 2.1. Produkty je potřeba navrhovat tak, aby se myslelo na využití některých recyklovaných zdrojů, které mohou mít jiné vlastnosti oproti nově získaným surovinám, stejně tak je důležité myslet na recyklovatelnost takového produktu (viz design na obrázku 2.1). Po výrobě produktu, jeho distribuci a spotřebě je také důležitější sběr spotřebovaných produktů, který musí být vzhledem k následnému třídění položek jinak organizovaný, než je tomu v lineárním systému. Po sběru a třídění přichází na řadu proces recyklace, ve kterém dojde k rozložení výrobku na základní suroviny, případně repasování, při kterém se vrátí produkt zpátky do používání [4].



Obrázek 2.1: Rozdíl mezi lineární a cirkulární ekonomikou (převzato z [2])

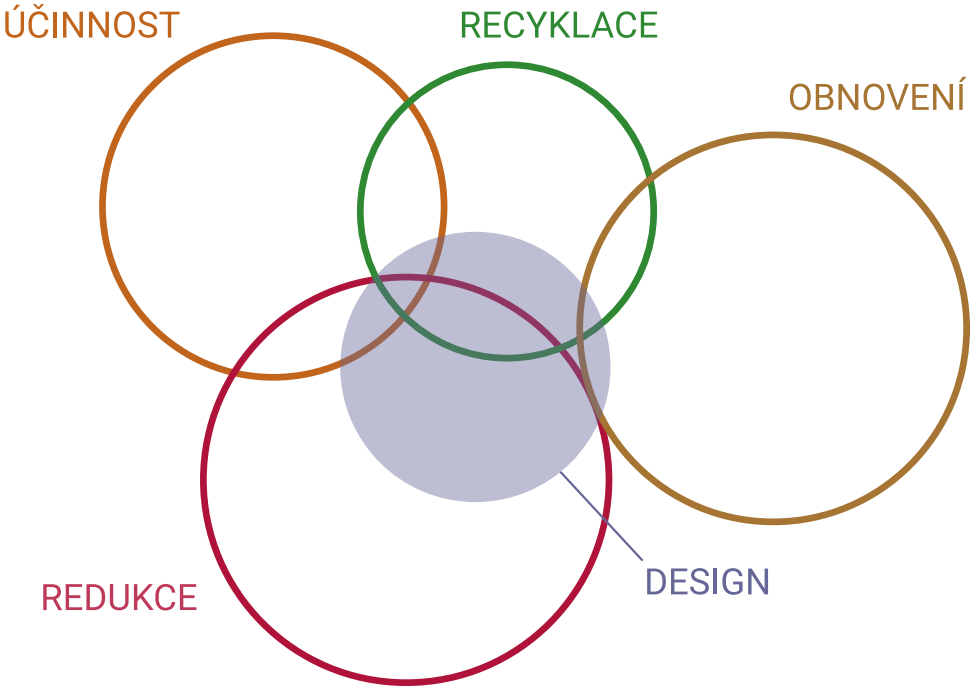
Cílem CE je dosáhnout udržitelného rozvoje, který současně vytváří kvalitu životního prostředí, ekonomickou prosperitu a sociální spravedlnost ve prospěch současných i budoucích generací [1]. Například město Praha na svém webu [2] uvádí: *„Odpady mají velký potenciál – ať už pro opětovné využití, recyklaci či pro naplňování cílů ochrany životního prostředí. Každý z nás je jejich producentem. S tím pracuje tzv. cirkulární ekonomika neboli oběhové hospodářství. Základní myšlenkou je vznik odpadů minimalizovat, a když už odpady vzniknou, tak je přeměnit na zdroje. Inspirací pro město mohou být přírodní procesy. V přírodě neexistuje odpad – veškeré suroviny kolují v nekonečných cyklech, bez ztráty na kvalitě.“*

Podle [4] můžeme cíle CE rozdělit do 5 oblastí – účinnost, recyklace, redukce, obnovení a design, ve kterých bychom mohli nalézt některé úlohy vhodné pro optimalizaci. Účinnost se týká cílů, které vedou k menšímu využívání zdrojů, tedy například vody, materiálů a energií, při stejně velké produkci. Recyklace potom vede k využívání recyklovaných surovin namísto nově získaných. Redukce se zabývá cíli, které vedou k menšímu množství produkovaného odpadu a emisí. Obnovení vede k využití odpadu jiným způsobem než přeměnou na základní suroviny, příkladem může být použití odpadu jako zdroje energie pomocí spalování.

Design se týká všech 4 zbylých cílů, jak bylo zmíněno na začátku kapitoly, týká se samotného návrhu produktu, při kterém se musí myslet na ekologické aspekty (tzv. eko-design), to znamená zohlednit v procesu výroby produktu recyklované materiály a také počítat s recyklovatelností takového výrobku (případně i s opravitelností).

Jednotlivé cíle CE se překrývají, a to z důvodu jejich vysoké míry vzájemné provázanosti, například cíle snižování množství odpadu se mohou týkat materiálové účinnosti, tyto překryvy jsou znázorněny na obrázku 2.2. Překryvy

jednotlivých cílů, podle [4], nám ukazují, že správně formulovaná optimalizační logistická úloha, kterou bychom mohli zařadit na průnik účinnosti a redukce (například z důvodu kritéria jako je kratší ujetá vzdálenost, která má vliv jak na efektivitu, tak na redukci emisí, produkující vozidla zahrnutá v logistice), má vliv i na samotnou recyklaci. Design výrobků zde díky efektivnější logistice může také hrát svou roli.



Obrázek 2.2: Překryvy cílů CE (přeloženo z [4])

2.1 Hlavní témata CE

Dále podle [1] můžeme CE rozdělit na 4 hlavní témata, ve kterých se dají nalézt optimalizační úlohy. Tato hlavní témata si nyní představíme.

2.1.1 Od recyklace produktu k recyklaci surovin

Podle článku 3 rámcové směrnice o odpadech [1, 32] se recyklací rozumí: *„jakýkoli způsob využití, jímž je odpad znovu zpracován na výrobky, materiály nebo látky, ať pro původní nebo pro jiné účely“*[1]. V literatuře se lze s možnostmi recyklace setkat pod různými pojmy, do popředí se ale dostávají především dva termíny z oblasti recyklace, a to renovace a repasování. Renovace se objevuje jako proces obnovy, při němž se odpad shromažďuje, testuje, opravuje, čistí a znovu prodává jako použitý výrobek v provozuschopném stavu, aniž by byl demontován. Na renovované výrobky se často vztahuje záruka. Repasování je nejčastěji označováno jako operace obnovy použitých výrobků, zahrnující sběr, opravu, demontáž a výměnu opotřebovaných součástí za účelem obnovy výrobků na kvalitativní úroveň nově vyrobených. Hlavní zvláštnost repasování

spočívá v demontáži výrobku, která je prvním a nejdůležitějším krokem na trzích s náhradními díly nebo v operacích opětovného zpracování ve výrobě.

2.1.2 Doprovodné (co-product) a vedlejší produkty (by-product)

Pojmy vedlejší a doprovodné produkty se v poslední době objevují v problémech optimalizace dodavatelského řetězce. Doprovodné produkty které mají podobný význam jako hlavní produkt, vznikají společně s hlavním produktem a mají vlastní poptávku, zatímco vedlejší produkty jsou obvykle neočekávané produkty vznikající ve výrobním procesu a mají menší ekonomickou hodnotu než kontrolovatelné výstupy výroby.

2.1.3 Pilíře udržitelnosti

Udržitelnost má 3 pilíře – ekonomický, environmentální a sociální. Oběhové hospodářství zahrnuje na základě své definice udržitelný rozvoj, který je definován jako *„rozvoj, který uspokojuje potřeby současnosti, aniž by ohrožoval schopnost budoucích generací uspokojovat své vlastní potřeby“* [3]. 3 pilíře udržitelnosti se vzájemně prolínají, například ekonomický pilíř je implicitně zohledněn prostřednictvím ekonomicky orientovaných účelových funkcí problémů. Například v logistické úloze, kde je hlavním cílem snížit ujetou vzdálenost vozidel, se snížením ujeté vzdálenosti sníží spotřeba paliva a tím pádem se ušetří i náklady na dopravu. To samozřejmě vede i k environmentálnímu pilíři tím, že vozidla budou méně znečišťovat své okolí.

2.1.4 Reverzní logistika a nakládání s odpady

Toto poslední uváděné téma se týká všech operací šetrných k životnímu prostředí, které souvisejí s opětovným použitím výrobků a surovin. Např. evropská komise stanovila článkem 4 rámcové směrnice o odpadech[1] pořadí priorit operací využití (tzv. pětistupňová hierarchie nakládání s odpady), počínaje přednostní možností předcházení vzniku odpadů, následovanou přípravou odpadů k opětovnému použití, recyklací a jiným způsobem využití (tj. zpětným zasypáním), přičemž odstranění (tj. skládkování) je až poslední možností. Opětovné využití odpadu je po jeho předcházení další nejžádanější možností v hierarchii možností nakládání s odpady, která je stanovena v rámci legislativy Evropské komise.

Tuto pětistupňovou hierarchii nakládání s odpady můžeme později zohlednit v návrhu logistické optimalizační úlohy. Vzhledem k tomu, že se práce zabývá logistickou optimalizační úlohou, která spadá primárně do této kategorie, budeme se dále zabývat právě definicí reverzní logistiky.

[1] `https://eur-lex.europa.eu/legal-content/CS/TXT/PDF/?uri=CELEX:32008L0098&from=LV`

2.2 Reverzní logistika

Myšlenka zpětného odběru použitých výrobků a komponentů, které mají být znovu použity v dodavatelském řetězci, je známá jako reverzní logistika [5]. Tento název se nabízí proto, že se vztahuje k logistickým činnostem uvnitř organizací, ale v opačném směru, než jsou běžné činnosti dodavatelského řetězce. Uvádí se také, že ziskové rozpětí společnosti do značné míry závisí na činnostech reverzní logistiky.

Pojem „reverzní logistika“ je čas od času definován mnoha autory v jejich vlastním pojetí (viz tabulka 2.1 s přehledem definic) [5]. Zjednodušeně jej lze definovat jako zpětný chod konvenčního výrobního toku. Původně byla reverzní logistika definována v pojmech řízení využití výrobků jako dosažení maximální ekologické a finanční hodnoty a zároveň snížení maximálního množství odpadu. Později byla časem modifikována a zahrnuje efektivní tok výrobků a informací opačně než tradiční dodavatelský řetězec.

Autor	Definice
[6]	Obor související s dovednostmi a činnostmi, které se podílejí na řízení odpadu, pohybu a likvidaci výrobků a obalů.
[7]	Proces plánování, realizace a řízení efektivního, nákladově efektivního toku surovin, zásob v procesu výroby, hotových výrobků a souvisejících informací z místa spotřeby do místa původu za účelem zpětného získání hodnoty nebo řádné likvidace.
[8]	Proces plánování, realizace a řízení účinného, efektivního vázaného toku a skladování druhotného zboží a souvisejících informací v opačném směru než v tradičním dodavatelském řetězci za účelem zpětného získání hodnoty nebo řádné likvidace.

Tabulka 2.1: Přehled definic reverzní logistiky podle [5]

Kapitola 3
Problém směřování vozidel (VRP)

V této kapitole si nejprve představíme problém směřování vozidel (angl. Vehicle Routing Problem, dále VRP), jeho varianty a podíváme se na historii. Poté se zaměříme na meta-heuristiky, před kterými si představíme lokální prohledávání, které je často součástí meta-heuristik.

3.1 Úvod

VRP je jeden z nejstudovanějších problémů z oblasti kombinatorické optimalizace [10]. VRP spočívá v návrhu nejméně nákladných tras pro doručování přes množinu geograficky rozptýlených zákazníků s ohledem na řadu vedlejších omezení [9]. Podle [9] tento problém zaujímá ústřední místo v řízení distribuce a denně se s ním potýkají desítky tisíc dopravců po celém světě.

Problém má mnoho podob, z důvodu různorodosti omezení, s nimiž se v praxi setkáváme. Do této kategorie problémů můžeme zařadit i problém obchodního cestujícího (angl. Traveling Salesman Problem, dále TSP). Ve VRP máme k dispozici flotilu vozidel, kdy každé vozidlo vykonává právě jednu trasu a začíná a končí v depu. Rozhodnutí, která je v VRP potřeba učinit, se týkají přiřazení zákazníků k vozidlům (jejich trasám) a pořadí těchto zákazníků v přiřazených trasách, tak aby byly minimalizovány celkové náklady na směřování (ukázka na obrázku 3.1) [10].

Za nejzákladnější variantu je považováno kapacitní směřování vozidel (angl. Capacitated Vehicle Routing Problem, dále CVRP) [10], kdy zákazníci, z nichž každý má danou poptávku, mají být obslouženi vozovým parkem stejných vozidel s uniformní kapacitou (Dantzig a Ramser 1959 [11]). Variantu CVRP budeme i my považovat za základní a dále ji tedy budeme označovat jako základní VRP.

Nyní si představíme některé další varianty VRP podle [17]. Pokud základní VRP rozšíříme o časová okna, ve kterých každé vozidlo dodává zboží zákazníkům, jedná se o variantu VRPTW (angl. Vehicle Routing Problem with Time Windows).

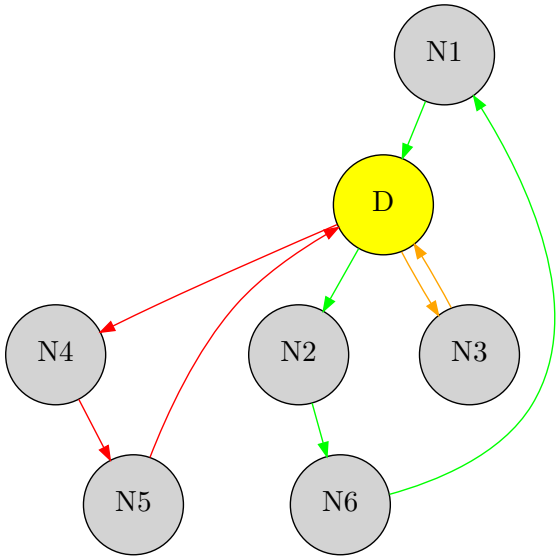
Varianta VRPPD (angl. Vehicle Routing Problem with Pickups and Deliveries) zase rozděluje zákazníky na místa pro vyzvedávání a doručení. Každé místo, určené pro vyzvedávání má dané množství, které je potřeba přemístit do míst určených pro doručení.

HVRP (angl. Heterogeneous Vehicle Routing Problem) je variantou, kdy předpokládáme, že naše flotila vozidel je heterogenní, tedy nemáme jen jeden typ vozidla se stejnou kapacitou, ale různou škálu vozidel, s různými kapacitami.

Přirozeně další základní variantou, je taková, kde budeme mít více než jen jedno depo. Taková varianta je označována jako MDVRP (angl. Multi-depot Vehicle Routing Problem).

Poslední uváděnou variantou v [17] je PVRP (angl. Periodic Vehicle Routing Problem). V této variantě neplánujeme pouze jednu rozvážku zboží, ale plánujeme na delší časový horizont, tedy více rozvážek, tím pádem mohou být zákazníci navštíveni více než jen jednou.

Varianty se samozřejmě můžou různě kombinovat a vytvářet nové přidáním dalších podmínek, protože v praxi se setkáváme se spousty omezeními, a z tohoto důvodu má VRP mnoho podob. Za více než padesátiletou historii tohoto problému vznikají další a další varianty a problém přitahuje velkou pozornost velké části komunity operačního výzkumu [9]. Je to částečně dáno ekonomickým významem tohoto problému, ale také metodologickými výzvami, které představuje. Například podle [9] lze TSP, který je speciálním případem VRP, nyní řešit pro tisíce, a dokonce desetitisíce vrcholů. Naproti tomu je řešení VRP mnohem obtížnější. Například v relativně jednoduchém případě, kdy jsou přítomna pouze kapacitní omezení (CVRP), je stále obtížné řešit instance s jedním nebo dvěma sty zákazníky pomocí exaktních algoritmů. Proto se v posledních letech velká část výzkumného úsilí zaměřila na vývoj výkonných meta-heuristik.



Uzel D představuje depo, uzly Nx představují zákazníky a hrany trasy vozidel oddělené barvami (máme 3 vozidla a každé má svoji barvu).

Obrázek 3.1: Ukázka možného řešení VRP s 6 zákazníky

3.2 Historický pohled

Formálně bylo VRP, konkrétně CVRP, představeno v roce 1959 Dantzigem a Ramserem [11] jako tzv. Truck Dispatching Problem. Zároveň tito autoři navrhli jednoduchou heuristiku pro její řešení založenou na párování zákazníků a ilustrovali ji na instanci velmi malé velikosti [9].

V následujících letech se objevilo několik heuristik založených na různých principech, například geografické blízkosti uzlů, párování zákazníků a také kroků pro zlepšení uvnitř trasy a mezi trasami [9]. Asi nejznámější heuristikou z této kategorie je heuristika Clark and Wright z roku 1964 [12], jedná se o poměrně jednoduchou heuristiku, která obstála díky své rychlosti a poměrně dobré přesnosti (dnes se často používá na generování přípustných iniciálních řešení).

Vývoj exaktních algoritmů pro VRP odstartoval v roce 1981, kdy Christofides, Mingozzi a Toth publikovali dva články v časopisech Networks a Mathematical programming. První z těchto článků [13] navrhl algoritmus založený na dynamickém programování s relaxací stavového prostoru, zatímco druhý [14] navrhl dvě matematické formulace využívající q-cesty a k-minimálních koster grafu. Podle [9] byla od té doby navržena řada exaktních algoritmů založených na formulacích matematického programování. Některé formulace obsahují toky a jsou často řešeny pomocí branch-and-cut nebo branch-and-price metody [9, 18]. VRP lze také formulovat jako set partition problem, ke kterému jsou přidané nějaké validní nerovnosti (podmínky, které neovlivní přípustné celočíselné řešení), na této metodice jsou založeny některé z nejúspěšnějších implementací [9].

Vývoj moderních heuristik pro řešení problémů typu VRP začal až po roce 1990 s příchodem meta-heuristik. VRP je NP-těžký problém a jeho aplikace do reálného světa zpravidla vyžadují velké instance, proto jsou pro praktické využití často vhodnější meta-heuristiky [17]. Zpočátku byl výzkum v této oblasti značně roztříštěný s velkým příklonem věnující se přístupům založených na Tabu Search technice (viz Taillard 1993 [15], Gendreau a spol. 1994 [16]). Podle [9] jsou nejlepší meta-heuristiky ty, které současně provádějí široké a hluboké prohledávání prostoru řešení a mohou řešit několik variant problému. Obvykle buď používají několik operátorů (např. Adaptive Large Neighborhood Search – ALNS), nebo kombinují genetické prohledávání s lokálním (hybridní genetický algoritmus).

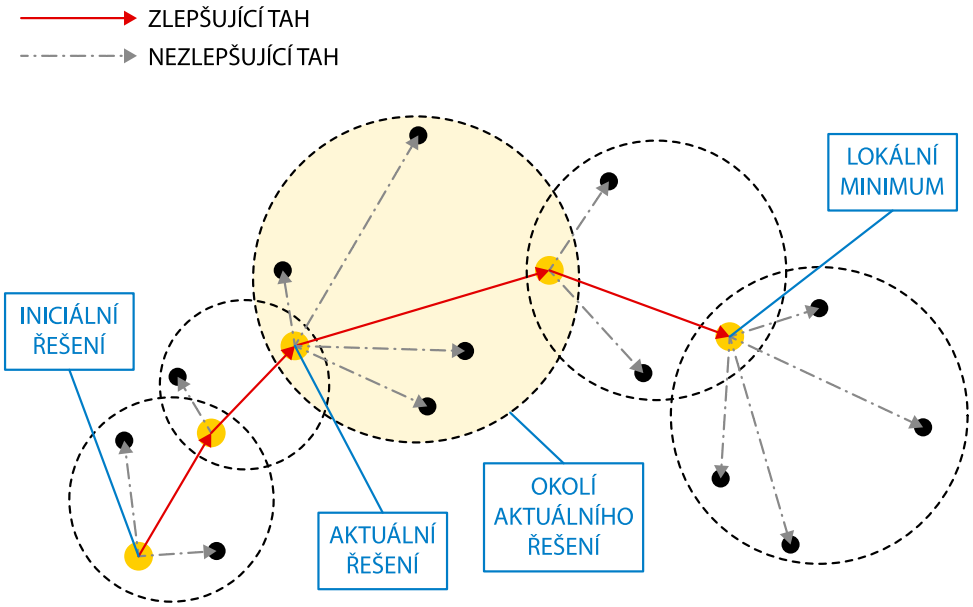
3.3 Lokální prohledávání (angl. Local Search)

Lokální prohledávání (dále LS) je technika pro řešení problémů z oblasti kombinatorické optimalizace [20]. Důvod, proč jej zde uvádíme je ten, že je zpravidla základem meta-heuristik (viz část 3.4). Podle [20] je hlavní myšlenkou, že se prostor řešení prohledává pomocí lokálních pertubací, díky kterým se z jednoho řešení přechází do sousedního. V každém kroku hledání algoritmus zvažuje sérii perturbací nazývaných tahy. Každý tah spočívá ve změně

malé části aktuálního řešení, aby se získalo lepší alternativní řešení, které je blízké předchozímu. Takové alternativní řešení se nachází v okolí předchozího řešení (viz obrázek 3.2). Okolí se obvykle vytváří pomocí operátorů, z nichž každý definuje typ perturbace, která se má použít.

Dalšími dvěma důležitými pojmy v lokálním prohledávání jsou podle [20] intenzifikace a diverzifikace. Intenzifikace se týká zlepšování současného řešení směrem k lokálnímu minimu (nazývané také jako lokální optimum), zatímco diverzifikace má za cíl prozkoumat různé části prohledávaného prostoru, aby se zabránilo uváznutí v lokálním minimu. Tyto dva aspekty se obvykle realizují prostřednictvím heuristik a meta-heuristik. Jednou z odlišností heuristik a meta-heuristik je právě to, že heuristika se zabývá intenzifikací výběrem dalšího zlepšujícího se tahu, zatímco meta-heuristika zavádí diverzifikaci implementací strategií k prozkoumání různých částí prohledávaného prostoru a vymanění se z lokálního minima (viz část 3.4) [20].

LS je výhodné především díky své výkonnosti, na úkor optimality, nicméně v mnoha případech může LS poskytnou téměř optimální, nebo dokonce optimální řešení, a to v krátkém čase. Hodí se tedy na velké instance, kde je prohledávaný prostor příliš velký na to, aby mohl být prozkoumán v rozumném čase. Výkonnost ale není jedinou výhodou, kterou LS disponuje. Další výhodou je nízká spotřeba paměti, protože okolí uvažované v každém kroku je podstatně menší, než celý prohledávaný prostor [20].



Obrázek 3.2: Průběh LS v prostoru řešení (přeloženo z [21])

3.4 Meta-heuristiky

V práci [25] se nachází následující definice: *„Meta-heuristiky jsou třídou výpočetních metod používaných na optimalizační problémy, jejichž řešitelé*

nevyžadují nutně nalezení optimálního řešení. Pomocí meta-heuristických algoritmů lze prohledávat stavový prostor úlohy, avšak bez záruky nalezení dostatečně kvalitního řešení v rozumném čase.“ Existuje mnoho definic meta-heuristik, které spolu korelují a pokrývají podobné třídy algoritmů, avšak v dílčích detailech se liší [25]. Podle [25] termín meta-heuristika sám označuje přesah, něco nad kořenovým pojmem heuristika. Pojem heuristika označuje metody, které vznikly pozorováním a nejsou exaktně prokazatelné [25]. První použití výrazu meta-heuristika se přisuzuje Fredovi Gloverovi z roku 1986, kde jím označuje nadvrstvu nad běžnou heuristikou [25]. V tabulce 3.1 můžeme vidět některé různé definice podle [25].

Autor	Definice
[26]	*„Meta-heuristika je proces iterativního generování, který řídí podřízenou heuristiku kombinováním rozdílných konceptů pro prohledávání a využívání stavového prostoru pomocí učících se strategií k strukturování informací pro efektivní nalezení téměř optimálního řešení.“*
[27]	*„Meta-heuristika je množina algoritmických konceptů, které mohou být využity k definování heuristických metod aplikovatelných na široké spektrum různých problémů. Jinak řečeno, meta-heuristiky mohou být chápány jako zobecněné heuristické metody navržené k určování postupu problémově závislých heuristik (lokální prohledávání, konstrukční heuristiky) směrem k oblastem ve stavovém prostoru obsahující vysoce kvalitní řešení. Meta-heuristika je tedy obecný algoritmický rámec aplikovatelný na mnoho různých problémů s relativně malým počtem úprav nutných pro přizpůsobení se konkrétnímu problému.“*
[28]	*„Meta-heuristický algoritmus může být definován jako vysokoúrovňová obecná metodologie (šablona), která je používána jako návod k odvození heuristik pro řešení specifických optimalizačních problémů.“*

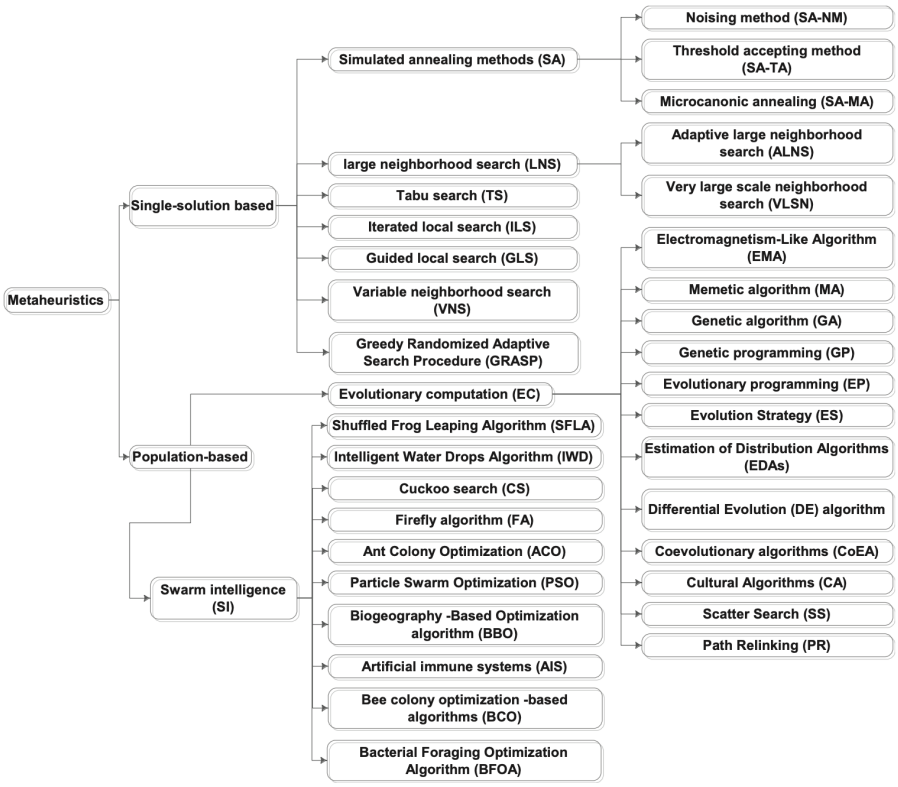
Tabulka 3.1: Přehled definic meta-heuristik podle [25]

Podle [29] neexistuje žádná obecně uznávaná definice pojmu meta-heuristika a trendem poslední doby je nazývat meta-heuristikou jakýkoliv stochastický algoritmus s randomizací a LS. Jednoduchým příkladem meta-heuristiky je opakování LS, pokud je prohledávání prováděno hladově, takovému algoritmu se říká iterativní Hill-climbing (viz část 3.4.1). Opakovaní LS můžeme také vylepšit tím, že při prohledávání zohledníme předchozí řešení, metody které toto využívají jsou například algoritmy Tabu Search (TS, viz část 3.4.1) a Guided Local Search (GLS, viz část 3.4.1). Dalším způsobem jak vylepšit LS je umožnit tahy, které nepovedou ke zlepšení, avšak nám umožní opustit lokální minimum. Na tomto základě jsou postavené například algoritmy Simulated Annealing (SA, viz část 3.4.1) a Variable Neighborhood Search (VNS) [18].

Meta-heuristiky založené na množině řešení (angl. population-based), na-

místo jednoho výchozího řešení, neustále zlepšují v každé iteraci množinu proveditelných řešení. Příkladem takovýchto algoritmů jsou například Genetický algoritmus (GA, viz část 3.4.2), Ant-colony optimization (ACO, viz část 3.4.2) a Particle-swarm optimization (PSO) [18].

Na obrázku 3.3 můžeme vidět členění jednotlivých meta-heuristických algoritmů. Jako meta-heursitický je označováno velké množství algoritmů. Podle [17] je můžeme rozdělit na 2 skupiny – založené na jednom řešení (například pouze předchozím řešení, angl. single-solution based) a na množině řešení (angl. population-based). Dále si představíme některé zástupce obou skupin.



Obrázek 3.3: Přehled meta-heuristických algoritmů (převzato z [17])

3.4.1 Založené na jednom řešení

Hill-climbing

Algoritmus je také známý jako Greedy descent (gradientní sestup). Podle [18] se jedná o LS postup, který provádí hladový výběr v okolí aktuálního přípustného řešení, aby našel nové lepší řešení. Výběrem se snaží optimalizovat kriteriální funkci, která se zpravidla skládá ze součtu vzdáleností použitých hran, ale i jiných podmínek (záleží na variantě VRP). Tato LS metoda zaručuje nalezení lokálního nikoli však globálního optima. Aby LS neuváznul

v jednom lokálním minimu, můžeme hill-climbing iterovat s různými počátečními hodnotami. Přestože je tento algoritmus poměrně jednoduchý, tak se ukázalo, že funguje dobře na některých reálných problémech [18].

Tabu Search (TS)

Podle [18] se iterativní Hill-climbing algoritmus dá upravit tak, aby již prozkoumaná řešení znovu nevyhledával. TS se právě snaží o to, aby se vyhnul prohledávání, které vede k předchozímu lokálnímu minimu, a to tak, že určí určité atributy těchto řešení, nazývané tabu omezení. Z důvodu výkonu algoritmu se určí také doba uložení těchto tabu omezení.

Guided Local Search (GLS)

Dle [18] identifikuje GLS na řešení vlastnosti, kde každá vlastnost má nějakou cenu (např. u TSP, že je určitá hrana v řešení a cena hrany je i cena této vlastnosti). Vytváří se seznam těchto vlastností z předchozích řešení. Při hledání nového řešení pomocí LS se poté cena vypočítává pomocí rozšířené nákladové funkce. Počet vlastností, které by nové řešení sdílelo s předchozími řešeními, se pak používá jako penalizační člen v této rozšířené nákladové funkci (jedná se vlastně o kriteriální funkci rozšířenou o penalizace). Penalizace ale neprobíhá u všech vlastností, které by nové řešení sdílelo s předchozími, ale využívá se utilizační funkce, která závisí na ceně vlastnosti a na počtu penalizací této vlastnosti. Díky tomuto přístupu je GLS schopno uniknout z lokálního minima (využije pro to modifikace řešení, které reálně nevedou ke zlepšení). GLS je více do hloubky vysvětleno v části 6.1.

Simulated Annealing (SA)

Podle [18] se v každé iteraci SA algoritmu z velkého okolí aktuálního řešení vybere další řešení náhodně. Je-li nové řešení lepší než aktuální, stane se z něj řešení aktuální. Pokud je nové řešení horší než aktuální, je s určitou pravděpodobností přijato jako aktuální řešení, v opačném případě je zamítnuto – pravděpodobnost závisí na tom, jak moc je nové řešení horší než aktuální a na tzv. teplotním parametru, který klesá v průběhu času. Na začátku je teplotní parametr vysoký, a proto algoritmus zpočátku častěji vybírá jakékoliv nově nalezené řešení, i když je horší než to aktuální. V různých implementacích se mohou měnit parametry, jako je počáteční teplota, pokles teploty pro novou iteraci, počet kontrolovaných řešení pro danou teplotu a pravděpodobnostní funkce pro přijetí nových řešení.

3.4.2 Založené na množině řešení

Genetický algoritmus (GA)

Podle [19] se jedná o pravděpodobně nejrozšířenější a nejznámější meta-heuristiku, které je dnes věnována mimořádná pozornost po celém světě.

Je založený na procesu biologické evoluce, kdy se provádí přirozený výběr. Výsledkem je nejlepší řešení, které bylo nalezeno na množině řešení (v populaci) během evoluce. Evoluce se imituje díky operátorům kopírující přírodní evoluci, nejběžnější jsou operátory jako křížení a mutace. Tato technika se tedy snaží směřovat k optimálnímu řešení problému pomocí vývoje různých množin proveditelných řešení.

Ant-colony optimization (ACO)

Tyto algoritmy se inspirují reálným chováním mravenců v koloniích, kteří hledají dobré zdroje potravy pro své hnízdo. Řešení jsou konstruována díky parametrizovanému pravděpodobnostnímu rozdělení nad prostorem řešení [19].

Kapitola 4

Současný stav v České republice

Podle [30] musí Česká republika pro splnění evropských cílů do roku 2025 recyklovat veškerý svůj komunální odpad z 55 %. Dle dat z roku 2019 se jí to daří plnit pouze ze 41 %. Do roku 2030 se musí recyklovat 60 % komunálního odpadu a o dalších 5 let později dalších 5 %. Nový zákon o odpadech, schválený poslaneckou sněmovnou v roce 2020, by v tomto měl pomoci. V práci [30] se o tomto zákoně píše: *„Nový zákon o odpadech zavádí k podpoře třídění v obcích tzv. třídicí slevu. Poslanecká sněmovna ji ještě upravila tak, aby se obce zaměřily na třídění a nebyly – potažmo ani jejich obyvatelé – významně dotčeny zvýšením skládkovacího poplatku.“*

U obcí, které v českém systému zpracovávání odpadu hrají důležitou roli, zůstaneme, a v této kapitole si nejprve představíme, systém zpracovávání odpadu v České republice, především z pohledu obcí. Následně zmapujeme, jaká jsou veřejně dostupná data, která by se mohla hodit pro testování algoritmu logistické úlohy z této práce.

4.1 Systém a obce

Dne 3. 3. 2022 proběhla v centrále společnosti Pražské služby, a. s. schůzka a následná ukázka fungování třídíren a sběrného dvora s panem Ing. Janem Svátkem, MPA. Z této schůzky vyplynulo, že v českém systému svozu odpadu hrají důležitou roli obce, jakožto původce odpadu. V momentě vhození položky do kontejneru pro sběr odpadu se vlastníkem této položky stává obec, která má ve smyslu zákona o odpadech povinnost odpady shromažďovat, sbírat, svážet, přepravovat, třídit, zajistit využití nebo odstranění.

Obce zpravidla ke svozu odpadu využívají svozové firmy. Ty odpad dopravují na třídící linky. Na ukázce fungování třídíren nám byla ukázána třídička, která vytřídí železné a neželezné kovy (např. hliníkové plechovky), tyto kovy se poté svazují do balíků (přibližně 1 m^3 velké). Následně nám byla ukázána třídírna papíru, kde jsme se dozvěděli, že se papíry třídí rovnou podle budoucího zpracování na 3 úrovně kvality. Papíry se poté opět svazují do balíků (přibližně 1 m^3) stejných kvalit.

Balíky vytříděných odpadů mají poté poptávku (např. papíry sháněj papírny pro recyklaci). Po vytřídění odpadů zbude také nějaký zbytkový odpad, například takový odpad, který se nerecykluje. Co se týče Pražských

služeb, a. s., tak provozují vlastní ZEVO (spalovnu s energetickým využitím odpadu) v Malešicích, kam vhodnou zbytkovou část odpadu po vytřídění odvážejí a přeměňují ji na energii. Bylo nám řečeno, že měření v horní části komína a na silnici před ZEVO v Malešicích ukázala, že vzduch, který z tohoto ZEVA vychází obsahuje méně zplodin, než se vyskytuje v okolí příjezdové cesty.

Každá taková činnost generuje nějaká data, z nichž některá jsou veřejně dostupná. V další části se podíváme právě na veřejně dostupná data z oblasti zpracovávání odpadu.

4.2 Dostupná data

Český systém nabízí některá veřejně dostupná data, která bychom mohli použít při formulování naší VRP úlohy, nebo pro následné testování prototypu. Určitě bude potřeba znát místa, kde se nějakým způsobem nakládá s odpadem, pokud budou k dispozici i souřadnice, můžeme simulovat i vzdálenosti těchto míst. V následující části se tedy nejprve budeme zabývat veřejně dostupnými daty o místech, která nakládají s odpadem. Dále se budeme zabývat kapacity takových míst a také flotilou vozidel (včetně jejich kapacit), které obsluhují nějakou oblast.

4.2.1 Místa nakládání s odpadem

Veřejně dostupná data, poskytující místa, kde se nakládá s odpadem, která se hodí pro naši logistickou úlohu poskytuje Ministerstvo životního prostředí (dále MŽP), které ze zákona provozuje registr zařízení a obchodníků (dále ISOH). Registr si klade za cíl umožnit veřejnosti přístup k potřebným informacím o zařízeních určených pro nakládání s odpady a obchodnících s odpady.

ISOH[1] je rozdělen na veřejnou a neveřejnou část [24]:

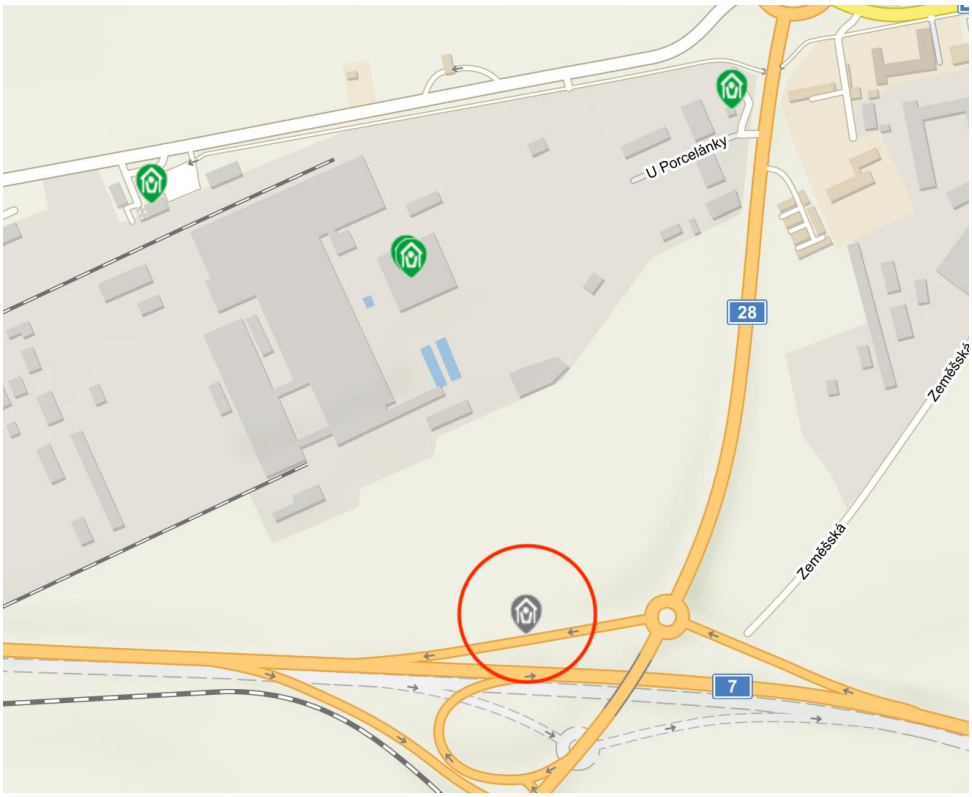
- „*Ve své veřejné části ve smyslu § 128 zákona č. 541/2020 Sb., o odpadech poskytuje aktuální informace o činnosti obchodníků podle § 26 a seznam vydaných povolení k činnosti podle § 26.*“
- „*Ve své neveřejné části umožňuje přístup k detailním informacím pro účely státní správy. Přístup do neveřejné části Registru zařízení a obchodníků mají pouze kontrolní orgány.*“

Data v tomto registru jsou k dispozici v interaktivní mapě a jako XML soubor. XML soubor je jednou denně aktualizovaný a bohužel není k dispozici žádná dokumentace, která by objasnila, co jednotlivé XML tagy určují. Podle názvu tagů se dá alespoň odhadnout, co který určuje. K dispozici jsou pouze souřadnice a adresy míst, s typem zařízení a typem odpadu, které zpracovávají/přijímají (jsou zde i například sběrné dvory). Údaje o kapacitách a provozu míst zde nenalezneme. Tato data získaná z tohoto registru se

[1] `https://isoh.mzp.cz`

dají použít pro experimenty na nějakém konkrétním městě/oblasti, avšak bude potřeba vytvořit nástroj, který data z XML souboru vyfiltruje podle požadavků.

Ukázalo se také, že data nebudou příliš přesná, protože se zde nacházejí místa, u kterých jsou špatné souřadnice, příkladem může být zařízení provozující společnost PATOK, a. s. s adresou U Porcelánky 2903, Louny, kde jsou souřadnice v mapě tohoto místa zaneseny doprostřed sjezdu na silnici 7 (viz obrázek 4.1), což ani neodpovídá uvedené adrese (navíc jsou zde další zařízení na stejné adrese, které mají jiné souřadnice).



Obrázek 4.1: Chybná pozice zařízení v ISOH

4.2.2 Kapacity, množství odpadu a flotila

Údaje o jednotlivých kapacitách zařízení a velikosti flotil bohužel nejsou veřejně dostupné. K dispozici jsou data o množství produkce odpadu v jednotlivých krajích a v celé ČR v rámci Informačního systému statistiky a reportingu v životním prostředí (ISSaR[2]), avšak tyto údaje nejsou rozděleny ani podle typů odpadu.

Dále poskytuje data také Český statistický úřad, který vydává jednou za rok přehledy[3]. Zde se opět dozvídáme množství odpadu podle jednotlivých

[2] `https://issar.cenia.cz/`

[3] `https://www.czso.cz/csu/czso/produkce-vyuziti-a-odstraneni-odpadu-2020`

regionů i někde rozdělené na typy odpadů. Navíc jsou u statistického úřadu k dispozici celorepubliková data o nakládání s odpady, tedy kde jaké množství odpadu skončilo.

Data také evidují samosprávy krajů, obcí a případně komerční sféra. Zde záleží na konkrétní samosprávě, jaké typy dat poskytuje/neposkytuje veřejnosti. Vesměs se ale jedná o přehledy množství odpadů v dané oblasti působnosti.

4.2.3 Shrnutí

Při experimentech na algoritmu by bylo dobré alespoň přibližně zmapovat nějakou oblast nebo město a testovat na takto zjištěných datech. K tomu může posloužit registr ISOH, ze kterého budeme muset ale například odhadnout, kde se nalézá depo pro danou oblast. Bohužel veřejně nejsou k dispozici data o flotilách a kapacitách, které budeme muset odhadnout na základě množství komunálního odpadu ve správních oblastech.

Podle [22] se v dostupných datech často vyskytují chyby, to vedlo výzkumníky z VUT v Brně k vytvoření nástroje Justine[4]. Jedná se o výpočetní nástroj pro eliminování chyb v datech. Justine představuje nástroj pro současné předpovídání množství a parametrů odpadů v různých územních jednotkách. Z obecného hlediska lze nástroj použít na jakýkoli problém, kde se prognózy provádějí na základě prostorově rozložených dat z předchozích let. Předpokládá se, že tato data jsou neúplná, někdy dokonce nejistá. Nástroj bohužel není veřejně přístupný, tedy jeho využití pro naší logistickou úlohu není reálné.

[4] `www.upi.fme.vutbr.cz/justine`

Část II

Problém

Kapitola 5

Formulace problému

Náš VRP se zabývá logistikou svozu odpadu v oblasti od třídíren ke zpracovatelům odpadu. Při tvorbě problému byla brána v úvahu i možnost využití veřejně dostupných dat, která jsou k dispozici a jedinečnost problému. Třídírny budeme dále nazývat producenty a zpracovatele jako konzumenty odpadu. Máme depo, ze kterého vyjíždí heterogenní flotila vozidel, která převáží odpad z producentů ke konzumentům, přičemž každého producenta obsluhuje právě jedno vozidlo. Při výběru do kterého konzumenta odvezeme odpad, budeme zohledňovat, jakým způsobem je u daného konzumenta odpad zpracováván. Budeme zvýhodňovat například recyklaci před ZEVO a ZEVO před skládkováním (vycházíme z článku 4 rámcové směrnice o odpadech, viz sekce 2.1.4). V úvahu musíme brát také kapacity vozidel a konzumentů, přičemž chceme odvézt veškeré množství odpadu, které producenti nabízejí. Nyní tedy formulujme tento problém formálněji.

Mějme **množinu vrcholů** $V = \{0, ..., n\}$, která se dále dělí na 3 celky:

- $0 \in V$, označující vrchol reprezentující depo,
- $V_p \subset V$, množina označující producenty odpadu,
- $V_c \subset V$, množina označující konzumenty odpadu.

Dále máme **množinu hran** $E = \{(i, j) \mid i, j \in V\}$. Nechť **úplný orientovaný graf** $G = (V, E)$, který neobsahuje smyčky, reprezentuje geografickou část našeho problému. G tedy představuje místa našeho problému a vzdálenosti mezi nimi. Vzdálenost hrany $(i, j) \in E$ určuje parametr $d_{ij} \in \mathbb{N}_0$. Každá hrana $(i, j) \in E$ představuje nejkratší možnou vzdálenost z vrcholu i do vrcholu j. Neexistuje tedy kratší cesta přes nějaký třetí vrchol (platí zde tzv. trojúhelníková nerovnost).

Uvažujme, že $V_p \cap V_c = \varnothing$, $\{0\} \cap V_c = \varnothing$ a $\{0\} \cap V_p = \varnothing$, tedy vrchol nemůže být zároveň producentem a konzumentem a depo není producent, ani konzument. Potřebujeme-li mít vrcholy, které jsou zároveň producentem a konzumentem (nebo depem), můžeme daný vrchol zdvojit a hraně mezi těmito zdvojenými vrcholy dát délku 0.

Flotila vozidel je reprezentována množinou $F = \{1, ..., m\}$, kde $k \in F$ označuje vozidlo. Každé vozidlo $k \in F$ má parametr $c_{f_k} \in \mathbb{N}_0$ značící jeho kapacitu. Každou hranu $(i, j) \in E$ je možné přejet libovolným vozidlem k.

Množství odpadu, které je potřeba z producenta $i \in Vp$ převézt do některého z konzumentů odpadu $j \in V_c$, je reprezentováno parametrem $q_i \in \mathbb{N}_0$. Každý konzument odpadu $j \in V_c$ má parametr $c_j \in \mathbb{N}_0$, určující kapacitu daného konzumenta (jaké množství odpadu je schopen přijmout). Odpad z právě jednoho producenta odváží právě jedno vozidlo, tedy nemůže se stát, že by jednoho producenta obsluhovalo více než jen jedno vozidlo.

Skóre konzumenta představuje, jak daný konzument zpracovává odpad. Je určené vektorem $\vec{s} = (0, 1, 2)$, každý vrchol $j \in V_c$ má parametr $s_j \in \vec{s}$, který značí jeho skóre, přičemž:

- 0 reprezentuje recyklaci,
- 1 reprezentuje přeměnu odpadu na energii (např. ZEVO),
- 2 reprezentuje ostatní neekologické zpracování odpadu (např. skládkování).

Cíl úlohy je nalezení optimálních tras pro vozidla $k \in F$, přičemž každé k začíná a končí v depu a jsou dodržovány kapacity vozidel a konzumentů. Každé vozidlo vyváží odpad z vrcholů V_p a dováží do vrcholů z V_c tak, aby na začátku cesty bylo vozidlo prázdné a na konci taktéž. Po skončení všech tras musí každý vrchol z V_p nabízet nulové množství odpadu. Součet kapacit konzumentů musí být tedy větší nebo roven součtu nabízeného množství odpadu producenty.

Kriteriální funkce (5.1) je definovaná třemi členy, z nichž každý představuje jedno významné kritérium problému. Pro představení prvního členu (5.3), který přestavuje celkovou ujetou vzdálenost všech vozidel, si zadefinujme funkci (5.2).

$$K = \sum_{k=1}^{|F|}\sum_{i=0}^{|V|}\sum_{j=0}^{|V|}(r(k,i,j)\cdot d_{ij}) + \alpha\cdot\sum_{i=0}^{|V|}(a(i)\cdot s(i)) + \beta\cdot\max_{k=1}^{|F|}(\sum_{i=0}^{|V|}\sum_{j=0}^{|V|}(r(k,i,j)\cdot d_{ij})) \tag{5.1}$$

Funkce (5.2) má tři argumenty. Prvním je vozidlo $k \in F$, druhým vrchol $i \in V$ a třetím také vrchol $j \in V$. Funkce vrátí počet výskytu hrany $(i, j) \in E$ na trase vozidla k. Díky tomu jsme schopni spočítat celkovou ujetou vzdálenost pomocí sumy přes všechna vozidla a všechny vrcholy, jak je uvedeno ve výrazu (5.3). Vzdálenost hrany (i, j) je reprezentována parametrem d_{ij} (viz úvod formulace).

$$r(k,i,j) = x \in \mathbb{N}_0 \tag{5.2}$$

$$\sum_{k=1}^{|F|}\sum_{i=0}^{|V|}\sum_{j=0}^{|V|}(r(k,i,j)\cdot d_{ij}) \tag{5.3}$$

Druhý člen (5.6) představuje penalizaci za využití neekologických konzumentů pro zpracování odpadu. Pro jeho formulaci musíme zadefinovat dvě

funkce (5.4) a (5.5). První funkce $a(i)$ (5.4) má jeden argument, jímž je libovolný vrchol $i \in V$, pokud je i konzumentem ($i \in V_c$), potom vrátí množství odpadu, které je do i převáženo, jinak vrátí 0. Druhá funkce $s(i)$ (5.5) má taktéž jeden argument, jímž je vrchol $i \in V$, a pokud se jedná o konzumenta ($i \in V_c$), tak vrátí jeho skóre. Není-li i konzument, poté vrátí funkce (5.5) 0. Díky těmto dvěma funkcím jsme schopni penalizovat konzumenta na základě množství odpadu, které do něj převážíme. Celý člen je vynásoben parametrem $\alpha \in \mathbb{R}$, který slouží ke korigování vlivu tohoto kritéria na celou kriteriální funkci.

$$a(i) = \begin{cases} x \in \mathbb{N}_0, & \text{pokud } i \in V_c \\ 0, & \text{jinak} \end{cases} \tag{5.4}$$

$$s(i) = \begin{cases} x \in \vec{s}, & \text{pokud } i \in V_c \\ 0, & \text{jinak} \end{cases} \tag{5.5}$$

$$\alpha \cdot \sum_{i=0}^{|V|} (a(i) \cdot s(i)) \tag{5.6}$$

Třetí člen (5.7) představuje penalizaci za celkovou dobu svozu odpadu. Opět se skládá z funkce $r(k, i, j)$ (5.2) a délky hrany d_{ij}. Člen hledá maximum z délek všech tras v řešení. Pokud bychom toto kritérium nezahrnuli do kriteriální funkce, tak by bylo vždy optimální využít pouze jedno vozidlo, které by mohlo svážet odpad prakticky po neomezeně dlouhou dobu. Podobně jako u druhého členu je i zde parametr $\beta \in \mathbb{R}$, který slouží ke korigování vlivu tohoto členu na celkovou kriteriální funkci. Zvýšením hodnoty β dosáhneme rovnoměrnějšího rozložení svozu na jednotlivé vozy, zatímco snížením můžeme více zatížit některé vozy na úkor jiných.

$$\beta \cdot \max_{k=1}^{|F|} \left(\sum_{i=0}^{|V|} \sum_{j=0}^{|V|} (r(k, i, j) \cdot d_{ij}) \right) \tag{5.7}$$

Celkovou kriteriální funkci, zahrnující všechny 3 definované členy, potom reprezentuje výraz (5.1). Cílem je kriteriální funkci K minimalizovat při dodržení stanovených podmínek.

5.1 Zařazení problému

Je zřejmé, že náš problém spadá do kategorie problémů směřování vozidel (VRP, viz kapitola 3). Vzhledem k tomu, že máme uzly, ze kterých potřebujeme vyzvednout náklad a dopravit ho do jiných uzlů určených pro doručování nákladu, má problém nejblíže k variantě VRPPD (viz sekce 3.1). K dispozici máme také heterogenní flotilu, tudíž se jedná i o variantu HVRP (viz sekce 3.1).

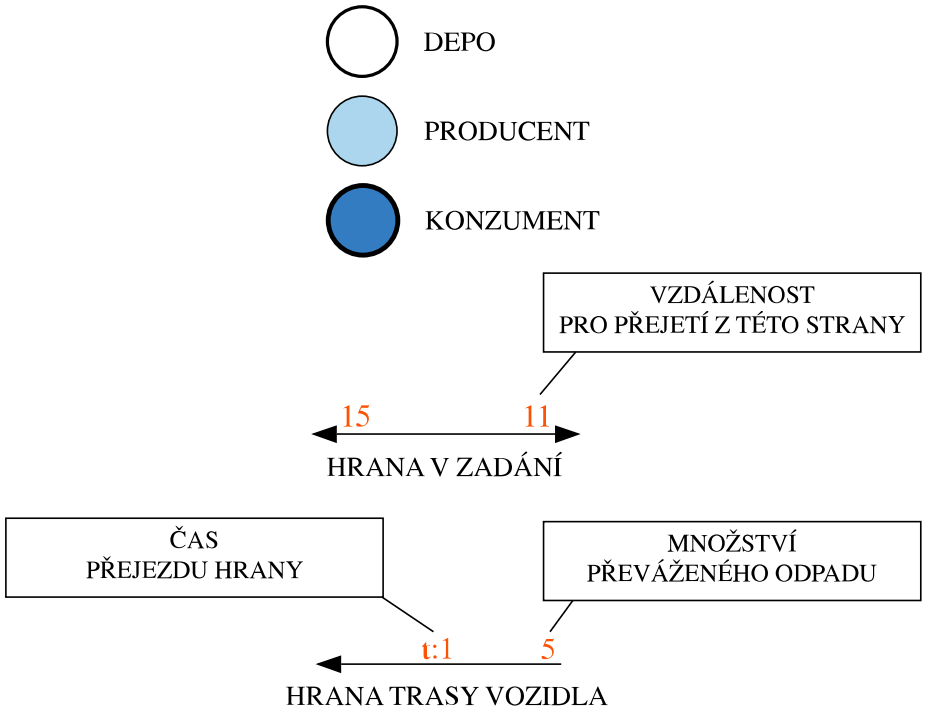
VRPPD je NP-těžký problém [18]. Náš problém obyčejné VRPPD ještě rozšiřuje o to, že má heterogenní flotilu vozidel. Navíc jsou přítomna i kapacitní omezení konzumentů a zvýhodňování konzumentů na základě jejich skóre. Můžeme tedy předpokládat, že i náš problém je NP-těžký.

Autoři práce [18] tvrdí, že VRPPD v kombinaci s variantou VRPTW (viz sekce 3.1) má NP-těžké dokonce i nalezení přípustného řešení. Z tohoto důvodu nebyla časová okna při formulaci problému brána v úvahu a byla pouze přidána do kriteriální funkce penalizace za nejdelší trasu, s možností korigování této penalizace pomocí parametru β.

5.2 Příklad

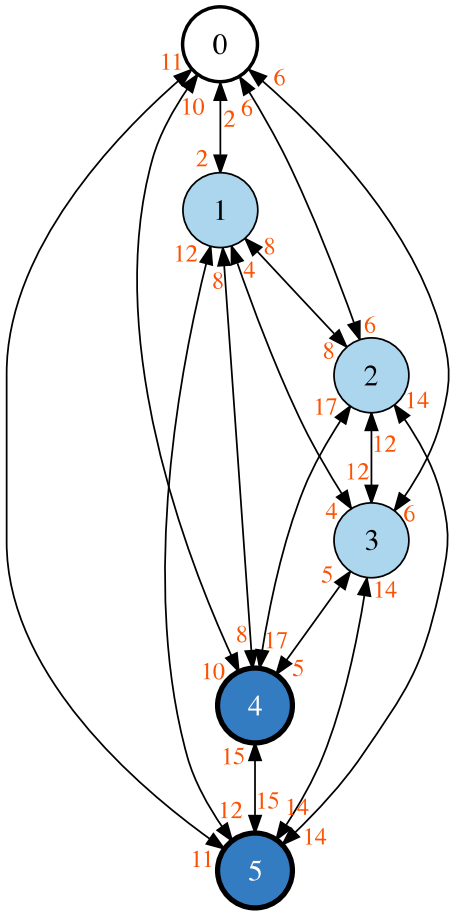
5.2.1 Instance

Pro lepší představu o tom, jak problém vypadá, jej ilustrujeme na malém příkladu, včetně možného řešení. Předtím si ještě zadefinujeme, jak budeme ilustrovat jednotlivé grafy. Budeme mít 2 typy ilustrací – zadání a trasy vozidel. Pro přehlednost nebudeme v obrázcích uvádět všechny parametry. Legenda k ilustracím je znázorněna obrázkem 5.1. Budeme-li ilustrovat zadání, tak u hran uvedeme oranžová čísla značící vzdálenost, kterou je vozidlo nutno překonat, vydá-li se právě po této hraně. Čísla budou u hran vždy 2x, protože vzdálenost například z vrcholu 0 do 1 může být jiná než z 1 do 0. Vzdálenost nutná pro přejetí hrany bude vždy uvedena u zdrojového vrcholu této hrany. Budeme-li ilustrovat trasu vozidla, potom na hranách takové trasy uvedeme množství odpadu, které vozidlo v okamžiku přejetí hrany převáží, a čas přejetí. Převážené množství uvedeme číslem u zdroje, zatímco čas uvedeme uprostřed hrany (začínající t:). Tyto typy obrázků pro ilustraci zadání a tras budeme v práci používat i dále.



Obrázek 5.1: Legenda ke grafům

Obrázek 5.2 představuje grafické zadání problému (vyjma kapacit, skóre konzumentů a vozidel). Máme 3 producenty $V_p = \{1, 2, 3\}$ (na obrázku 5.2 znázorněné světle modře) a 2 konzumenty $V_c = \{4, 5\}$ (na obrázku 5.2 znázorněné tmavě modře), přičemž konzument 4 spaluje odpad pro zisk energie (má skóre o hodnotě 1) a 5 recykluje odpad (má skóre o hodnotě 0). Dále máme 2 vozidla $F = \{1, 2\}$.



Obrázek 5.2: Zadání problému s 3 producenty a 2 konzumenty

V tabulce 5.1 můžeme vidět přehled množství odpadů, které poskytují jednotlivý producenti a je potřeba odvézt ke konzumentům. Dále v tabulce 5.2 je přehled konzumentů s jejich kapacitami a skóre. Poslední tabulkou 5.3 je přehled vozidel a jejich kapacit. Veškeré údaje jako kapacity a množství odpadu mají stejné jednotky (např. m^3).

Producent ($i \in V_p$)	**Množství odpadu** ($q_i \in \mathbb{N}$)
1	51
2	42
3	20

Tabulka 5.1: Přehled producentů v ukázkovém zadání

Konzument ($j \in V_c$)	Kapacita ($c_j \in \mathbb{N}_0$)	Skóre ($s_j \in \vec{s}$)
4	73	1
5	120	0

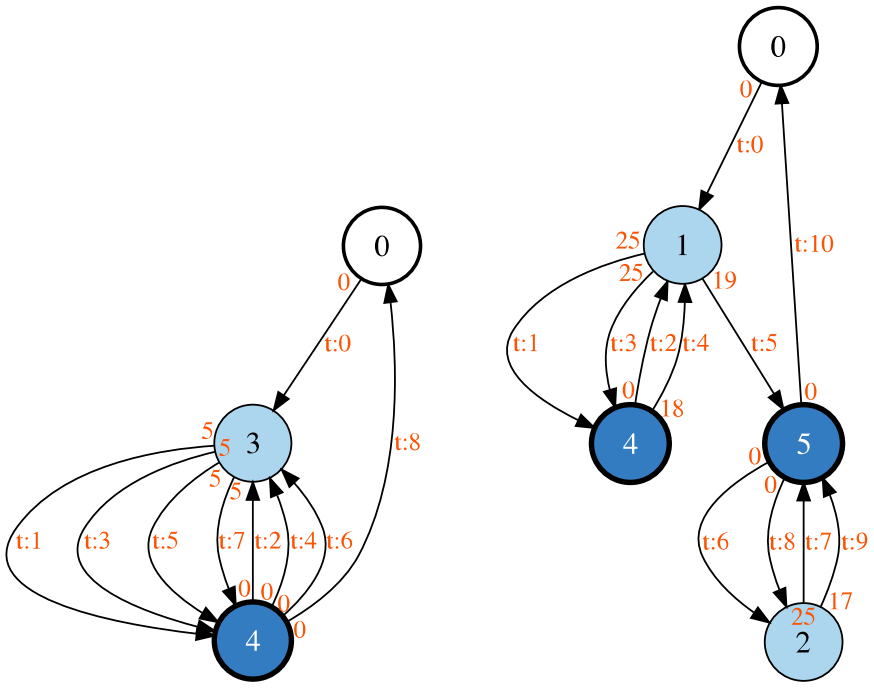
Tabulka 5.2: Přehled konzumentů v ukázkovém zadání

Vozidlo ($k \in F$)	Kapacity ($c_{f_k} \in \mathbb{N}_0$)
1	5
2	25

Tabulka 5.3: Přehled vozidel v ukázkovém zadání

5.2.2 Možné řešení

Řešení úlohy se bude skládat z vozidel, které budou mít přiřazenou svou trasu. Jedno z možných přípustných řešení je ukázáno na obrázku 5.3. Obrázek 5.3a reprezentuje trasu vozidla 1 a obrázek 5.3b trasu vozidla 2. Každé vozidlo začíná a končí v depu ($0 \in V$).



(a) : Vozidlo 1 **(b) :** Vozidlo 2

Obrázek 5.3: Ukázkové řešení tras vozidel

Určíme-li pro kriteriální funkci parametr α jako 1 a β jako 2, potom cenu takového řešení získáme po dosazení do kriteriální funkce (5.8). Jak jsme zmínili při formulaci problému, kriteriální funkci můžeme rozdělit na 3 členy (5.9), (5.10) a (5.11). První výraz (5.9) je pouze součet vzdáleností tras,

výsledkem je tedy 164 jednotek (vozidlo 1 překonává vzdálenost 51 jednotek a vozidlo 2 113 jednotek).

$$K_p = \sum_{k=1}^{2}\sum_{i=0}^{5}\sum_{j=0}^{5}(r(k,i,j)\cdot d_{ij}) + 1\cdot\sum_{i=0}^{5}(a(i)\cdot s(i)) + 2\cdot\max_{k=1}^{2}(\sum_{i=0}^{5}\sum_{j=0}^{5}(r(k,i,j)\cdot d_{ij})) \tag{5.8}$$

$$\sum_{k=1}^{2}\sum_{i=0}^{5}\sum_{j=0}^{5}(r(k,i,j)\cdot d_{ij}) = 51 + 113 = 164 \tag{5.9}$$

Pro výpočet druhého výrazu (5.10) potřebujeme znát, kolik odpadu jsme odvezli do kterého konzumenta. V ukázkovém případě musíme odvézt z producentů 113 jednotek odpadu a v konzumentech máme k dispozici kapacitu o celkovém součtu 193 jednotek odpadu. Odvezli jsme do konzumenta 4 52 jednotek odpadu a do konzumenta 5 61 jednotek odpadu. Konzument 4 využívá odpad k tvorbě energie a jeho skóre je 1, zatímco konzument 5 odpad recykluje, tedy skóre má 0. Výsledkem druhého členu je tedy 52 jednotek, viz výpočet (5.10).

$$1\cdot\sum_{i=0}^{5}(a(i)\cdot s(i)) = (52\cdot 1) + (61\cdot 0) = 52 \tag{5.10}$$

Třetí člen je pouze maximum ze vzdáleností tras, vynásobený parametrem β, který je v našem případě roven číslu 2. Nejdelší trasu překonává vozidlo 2. Výsledkem je tedy 226 jednotek, viz výpočet (5.11).

$$2\cdot\max_{k=1}^{2}(\sum_{i=0}^{5}\sum_{j=0}^{5}(r(k,i,j)\cdot d_{ij})) = 2\cdot\max(51,113) = 226 \tag{5.11}$$

Pokud všechny 3 členy sečteme, získáme cenu řešení rovnou 442 jednotkám, viz výpočet (5.12). Na výsledku si můžeme všimnout, že kvůli tomu, že jsme zvolili parametr α pouze jako 1, tak jsme trochu upozadili význam nakládání s odpady v ceně našeho řešení. Volba parametrů, ať už to je α nebo β bude tedy v budoucnu velmi důležitá, a to především pro praktické využití. Například aby vliv skóre konzumentů neztratil význam (nebo abychom jej nepřecenili), budeme jej muset volit úměrně ke vzdálenostem, které jsou vozidla v dané instanci schopna překonat, a také vzhledem k množství převáženého odpadu.

$$K_p = 164 + 52 + 226 = 442 \tag{5.12}$$

Kapitola 6
Řešení problému (návrh algoritmu)

V této kapitole uvedeme možné řešení problému, formulovaného v kapitole 5. Exaktní metody, jako branch-and-cut, nebo branch-and-price jsou schopny řešit pouze malé instance VRPPD, s kapacitními omezeními, a to do přibližně 130 zákazníků [18]. Jak již bylo zmíněno v sekci 3.2, tak aplikace VRP do reálného světa zpravidla vyžaduje velké instance. Z tohoto důvodu jsou pro praktické využití často vhodnější meta-heuristiky, tudíž i v našem případě navrhneme meta-heuristický algoritmus.

Nabízelo by se také využití Google OR-Tools[1], jedná se o optimalizační nástroj, který nabízí možnosti řešení VRP pomocí meta-heuristik. Při detailnějším průzkumu tohoto nástroje bylo ale zjištěno, že by pravděpodobně bylo obtížné, ne-li nemožné zanést do tohoto nástroje veškerá kritéria, která náš problém obnáší (příkladem může být skóre konzumentů). Navíc je předmětem práce návrh algoritmu, tvorba prototypu a experimenty na něm.

Kapitola nejprve vysvětlí, jak funguje zvolený meta-heuristický GLS algoritmus. Vysvětlíme, proč jsme zvolili právě GLS, a určíme parametry pro náš problém, které GLS vyžaduje. Následně se podíváme na to, jak můžeme konstruovat přípustné iniciální řešení, které je pro funkci GLS nezbytné. Na závěr se podíváme na LS algoritmus a zvolené operátory.

6.1 Guided Local Search (GLS)

GLS algoritmus byl už krátce představen v části 3.4.1. Podle [31] má své kořeny v neuronové architektuře nazvané GENET. GENET je použitelný pro třídu problémů známých jako CSP (angl. Constraint Satisfaction Problems), které úzce souvisejí s třídou problémů SAT (splnitelnost booleovské formule). GLS zobecňuje některé prvky přítomné v architektuře GENET a aplikuje je na obecnou třídu kombinatorických úloh.

V práci [18] je porovnáván GLS na problému VRPPD s algoritmy TS, iterativní Hill-climbing a SA (algoritmy jsou krátce popsány v sekci 3.4). Porovnávala se kvalita řešení jednotlivých algoritmů, přičemž jako zastavující kritérium každého algoritmu byla určena doba běhu. Měřilo se 5 dob – 5, 10, 30 minut, 1 a 2 hodiny. Pro každý čas se každý algoritmus spustil desetkrát

[1] `https://developers.google.com/optimization/routing`

a výsledek se zprůměroval. Výsledkem bylo, že GLS a TS dopadly lépe, než iterativní Hill-climbing a SA. GLS a TS dosahovaly přibližně stejných výsledků, nicméně TS dosáhlo lehce lepších výsledků v 1 a 2 hodinovém běhu. Oproti tomu GLS dopadlo lépe v 5, 10 a 30 minutovém běhu.

Google OR-Tools, optimalizační nástroj, tvrdí, že GLS je obecně nejefektivnější meta-heurstikou pro VRP[2]. Vzhledem k těmto poznatkům jsme se rozhodli pro využití GLS algoritmu, jakožto frameworku pro náš algoritmus.

GLS je iterativní algoritmus, který nemá předem dané zastavující kritérium. Můžeme tedy určit jako zastavující kritérium například počet iterací tohoto algoritmu, anebo zvolit úplně jiné. Algoritmus využívá pro vylepšování řešení LS algoritmus (viz sekce 3.3), který je volán v každé iteraci a vylepšuje řešení, které bylo nalezené v předchozí iteraci (návrh LS algoritmu uvedeme později v sekci 6.3). Účelem GLS oproti tradičnímu iterativnímu LS, který se zasekne v lokálním minimu, je právě to, abychom dokázali uniknout z lokálního minima.

6.1.1 Vlastnosti

Únik z lokálního minima je v GLS realizován pomocí penalizací vlastností, které identifikujeme na řešení. Podle [31] vlastností řešení může být cokoliv, co splňuje jednoduché omezení, které není triviální. To znamená, že ne všechna řešení mají všechny vlastnosti. Některá řešení mají nějakou vlastnost, kterou jiná řešení nemají. Tyto vlastnosti jsou závislé na problému a slouží jako rozhraní mezi algoritmem a konkrétní aplikací. Každá taková vlastnost má cenu, která představuje přímý nebo nepřímý vliv na náklady na řešení. Příklad vlastnosti můžeme ukázat na TSP, kde vlastností nějakého řešení může být to, zda je určitá hrana v onom řešení. Cenou takové vlastnosti je potom délka této hrany [31].

Zadefinujme následující parametry:

- $M = \{m_1, m_2, ..., m_{|M|}\}$, označující množinu vlastností,
- $\vec{c_{m_i}} = (c_{m_1}, c_{m_2}, ..., c_{m_{|M|}})$, vektor označující ceny jednotlivých vlastností, tedy $m_i \in M$ má cenu $c_{m_i} \in \mathbb{R}$,
- S, označuje množinu řešení ($s \in S$ je právě jedno řešení).

Potom můžeme uvést indikační funkci $I_{m_i}(s)$ (6.1), která má jediný argument, jímž je řešení $s \in S$. Funkce (6.1) vrátí 1, pokud dané řešení má vlastnost m_i, jinak 0.

$$I_{m_i}(s) = \begin{cases} 1, & \text{řešení } s \text{ má vlastnost } m_i \\ 0, & \text{jinak} \end{cases} \tag{6.1}$$

[2] https://developers.google.com/optimization/routing/routing_options

Vlastnosti řešení našeho problému

Na našem problému můžeme identifikovat 2 typy vlastností:

- **Hrany** – Stejně jako u TSP, tuto vlastnost má řešení, pokud zahrnuje určitou hranu. Cenou vlastnosti je délka takové hrany.
- **Konzumenti** – Tuto vlastnost má řešení, zahrnuje-li určitého konzumenta. Cenou takové vlastnosti je modifikované skóre konzumenta.

Pro vlastnosti typu konzumenti musíme cenu vypočítávat modifikací skóre konzumenta. Nemůžeme použít pouze prosté skóre konzumenta, protože by zde mohla vzniknout výrazná nevyváženost s vlastnostmi typu hrany. Cena délky hrany totiž může být výrazně vyšší než skóre konzumenta. Skóre konzumenta musíme také modifikovat tak, že přičteme 1, abychom mohli penalizovat i konzumenty, které recyklují. Ty mají totiž v zadání skóre 0. Abychom cenu vlastností typu konzumenti vyrovnali s cenou vlastností typu hrany, vynásobíme skóre, po přičtení 1, parametrem $\gamma \in \mathbb{R}$. Celá modifikace je uvedena výrazem (6.2), přičemž $j \in V_c$ a c_{m_i} je cena vlastnosti m_i a m_i je vlastnost vztahující se k j.

$$c_{m_i} = \gamma \cdot (s(j) + 1) \tag{6.2}$$

Můžou nás napadnout i další vlastnosti řešení, jako například přiřazení určitého producenta k nějakému vozidlu. Vozidla mají jediný parametr, kterým je jejich kapacita. Pokud bude producenta s vyšším množstvím odpadu obsluhovat vozidlo s nízkou kapacitou, tak se bude muset do takového producenta vracet víckrát než vozidlo s vyšší kapacitou. Například budeme-li mít producenta, který nabízí 30 jednotek odpadu, a budeme jej chtít obsloužit vozidlem o kapacitě 5 jednotek, tak potom se vozidlo musí do takového producenta vrátit šestkrát. Pokud bychom použili vozidlo o kapacitě 10 jednotek, potom se bude vracet třikrát. Kdybychom tedy do našeho algoritmu zahrnuli i vlastnosti přiřazení určitého producenta k nějakému vozidlu, cenu bychom museli zvolit na základě množství odpadu, které nabízí producent a kapacity daného vozidla. Taková vlastnost se ale jeví nadbytečně, pokud již při konstrukci iniciálního řešení budeme volit pro vozidla s vyšší kapacitou producenty, které nabízejí větší množství odpadu.

Vlastností můžeme na řešení identifikovat spousty, ale následně musíme převést jejich cenu tak, aby se vyrovnala cenám ostatních vlastností (např. podobně jako modifikujeme skóre konzumenta). Pro zjednodušení tedy budeme uvažovat jen námi představené 2 typy vlastností – hrany a konzumenti. Nicméně v budoucnu je možné algoritmus rozšířit o další vlastnosti.

6.1.2 Penalizace vlastností

Pro pochopení následujícího textu musíme nejprve zadefinovat nákladovou funkci $g(s)$. Tato funkce má jediný argument, kterým je libovolné řešení $s \in S$. Výstupem $g(s)$ je potom cena řešení s, která je vypočítána pomocí kriteriální funkce, představené v kapitole 5.

Aby GLS unikl z lokálního minima (kde uvázne LS algoritmus, viz sekce 3.3), tak vlastnosti uvedené v předchozí části 6.1.1 penalizuje. Podle [31] se to děje díky rozšířené nákladové funkci, kterou v každé iteraci předává LS místo toho, aby předával prostou nákladovou funkci $g(s)$. Díky tomu může LS zvolit řešení, které není nutně lepší než řešení předchozí, tím pádem unikne z lokálního minima. V příštích iteraci potom prozkoumá jiné části prostoru řešení, kam by se za normálních okolností LS nedostal z důvodu uváznutí. Tedy GLS zavolá LS, získá nějaké vylepšené řešení a následně penalizuje vlastnosti tohoto řešení. Při příští iteraci LS vyhledává další řešení s již penalizovanými některými vlastnostmi, a tak se pravděpodobně těmto vlastnostem vyhne.

Mějme tedy následující penalizační vektor:

- $\vec{p_m} = (p_{m_1}, p_{m_2}, ..., p_{m_{|M|}})$, kde $(p_{m_i} \in \vec{p_m}) \in \mathbb{N}_0$ reprezentuje penalizaci vlastnosti $m_i \in M$.

Rozšíření nákladové funkce $g(s)$ je poté reprezentováno výrazem (6.3). Přítomen je parametr $\lambda \in \mathbb{R}$ (tj. síla penalizací), díky kterému můžeme relativizovat vliv penalizací vzhledem k hodnotě nákladové funkce $g(s)$, a také můžeme zvýšit, nebo naopak snížit vliv těchto penalizací.

$$h(s) = g(s) + \lambda \cdot \sum_{i=1}^{|M|} p_{m_i} \cdot I_{m_i}(s) \tag{6.3}$$

V závislosti na hodnotě λ může být zapotřebí jedna nebo více iterací, než dojde k přesunu z lokálního minima. Vysoké hodnoty λ způsobují, že algoritmus je agresivnější a rychle uniká z lokálních minim, zatímco nízké hodnoty λ způsobují, že algoritmus je opatrnější a vyžaduje větší nárůst penalizace, než je dosaženo úniku.

Modifikace penalizací

V úvodu algoritmu jsou všechny penalizace p_{m_i} nastavené na hodnotu 0. Jak již bylo zmíněno, v každé iteraci GLS dochází k vylepšování předchozího řešení pomocí LS, který porovnává provedené tahy pomocí rozšířené nákladové funkce. V každé iteraci se vyčkává, dokud se LS nezasekne v lokálním minimu. Dojde-li k uváznutí, tak nastupuje modifikace penalizací. Ta probíhá na základě utilizační funkce (6.4), s^* zde představuje řešení nalezené LS, když uváznulo v lokálním minimu [31].

$$util(s^*, m_i) = I_{m_i}(s^*) \cdot \frac{c_{m_i}}{1 + p_{m_i}} \tag{6.4}$$

Penalizovány jsou ty vlastnosti, které maximalizují utilizační funkci (6.4). Tato funkce má dva argumenty, jedním je řešení s^* a druhým je vlastnost $m_i \in M$. Po uváznutí LS dojde k zavolání této funkce pro každou vlastnost na nalezeném řešení. Výstupem je utilizační hodnota takové vlastnosti vzhledem k řešení s^*. U právě těch vlastností, kde je tato utilizační hodnota nejvyšší,

dojde k penalizaci. K penalizování dochází tak, že jejich p_{m_i} je zvýšeno o 1. V utilizační funkci je vidět, že je zde parametr p_{m_i} použit, aby omezil vícenásobné penalizování stejných vlastností, navíc vlastnosti s vyšší cenou jsou penalizovány častěji než vlastnosti s nižší.

6.1.3 Pseudokód

Nejlepší ilustrací, jak GLS pracuje, je pseudokód 1. v tomto pseudokódu jsou znázorněny všechny parametry algoritmu, vysvětlené v předchozích částech.

Pseudokód 1 Guided local search (převzato z [31])

function (S, g, λ, $[I_{m_1}, ..., I_{m_{|M|}}]$, $[c_{m_1}, ..., c_{m_{|M|}}]$, $|M|$)
  $k \leftarrow 0$
  $s_0 \leftarrow$ generované heuristické iniciální řešení v S
  for $i \leftarrow 1$, $|M|$ **do**
    $p_i \leftarrow 0$
  end for
  while zastavující kritérium **do**
    $h \leftarrow g + \lambda \cdot \sum_{i=1}^{|M|} p_{m_i} \cdot I_{m_i}(s_k)$
    $s_{k+1} \leftarrow LocalSearch(s_k, h)$
    for $i \leftarrow 1$, $|M|$ **do**
      $util_i \leftarrow I_{m_i}(s_{k+1}) \cdot \frac{c_{m_i}}{1+p_{m_i}}$
    end for
    for každé i takové, že $util_i = \max_{j=1}^{|M|}(util_j)$ **do**
      $p_{m_i} \leftarrow p_{m_i} + 1$
    end for
    $k \leftarrow k + 1$
  end while
  $s^* \leftarrow$ nejlepší nalezené řešení s ohledem na nákladovou funkci g
  return s^*
end function

6.2 Konstrukce iniciálního řešení

Jak můžeme vidět v GLS pseudokódu (viz pseudokód 1), na počátku potřebujeme heuristicky generované přípustné řešení. Takové řešení by mělo být generováno rychle, abychom měli více času na prohledávání prostoru řešení pomocí GLS algoritmu. Námi navržená konstrukce se skládá ze 4 fází, z nichž každou v této části práce rozebereme.

6.2.1 Mapování producentů ke konzumentům

První fází konstrukce iniciálního řešení je přiřazení producentů ke konzumentům. Každý producent, jakožto původce odpadu, získá seznam konzumentů,

do kterých by potencionálně mohl uložit svůj odpad. Těchto seznamů využijeme v další fázi při konstrukci základních tras. Vzhledem k tomu, že se zajímáme o konstrukci přípustného řešení, tak můžeme některé parametry zanedbat (např. skóre konzumenta). Již v této fázi se ale můžeme pokusit o to, aby iniciální řešení mělo některé rozumné parametry, například v tomto konkrétním případě můžeme seznam konzumentů pro každého producenta řadit podle vzdálenosti, od nejbližších po nejvzdálenější. Definujme tedy následující:

- $clist_i = [x, ..., y]$, kde $i \in V_p$, $x, y \in V_c$, označující seřazený seznam všech konzumentů. Konzumenti jsou v seznamu seřazeni podle vzdálenosti k producentovi i, od nejbližších po nejvzdálenější.
- $[clist_i, ..., clist_j]$, kde $i, j \in V_p$, označující seznam všech seznamů konzumentů daných producentů.

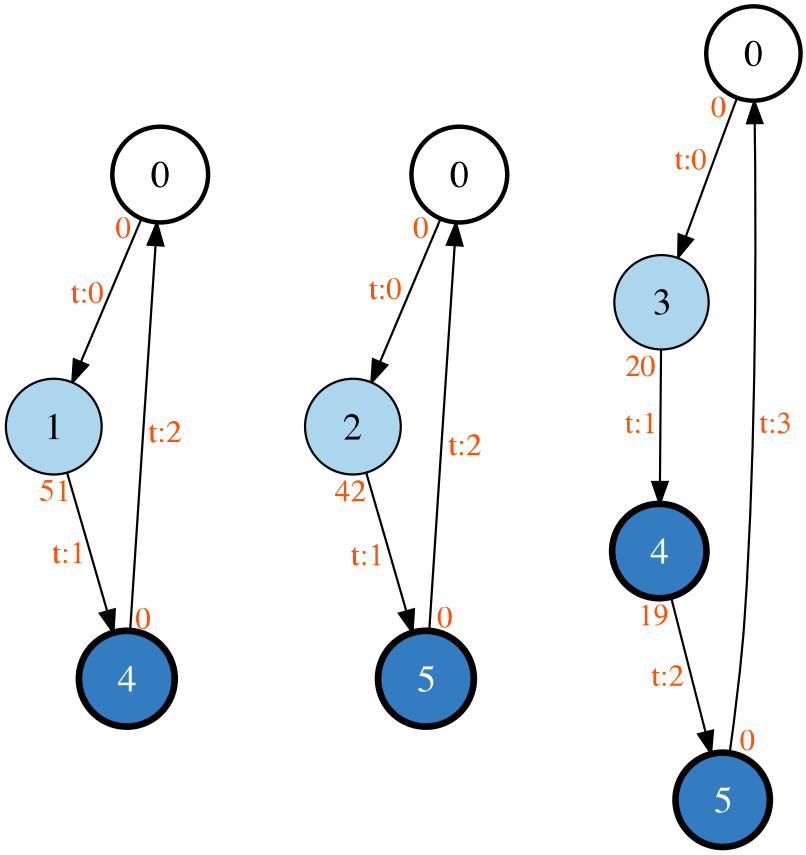
6.2.2 Vytvoření základních tras

V další fázi určíme každému producentu konzumenty, do kterých uloží svůj odpad. Budeme v této fázi plnit podmínku, aby každý producent na konci neměl k dispozici žádný odpad a aby byly dodrženy kapacitní omezení konzumentů. K výběru konzumentů využijeme právě seznamy vytvořené v předchozí části 6.2.1. K tomuto účelu vytvoříme každému producentovi trasu, kterou budeme označovat jako základní trasu.

Tato trasa obsahuje hrany, zahrnující množství převáženého odpadu v okamžiku cestování po této hraně. Základní trasa, podobně jako finální trasa vozidla, bude začínat a končit v depu ($0 \in V$). Po depu bude vždy následovat hrana k producentovi, jehož se tato základní trasa týká. Z producenta budeme odvážet veškerý odpad, který nabízí. To znamená, že následující hrana bude mít množství převáženého odpadu stejné, jako je nabízené množství producentem. Jelikož vytváříme základní trasu pro každého producenta, tak tím plníme podmínku, abychom odvezli veškerý odpad, který producenti nabízejí.

Označíme-li takového producenta jako i, potom hrana z i povede již do některého z konzumentů, kterého vybereme jako nejbližšího z $clist_i$. Při konstrukci základní trasy snižujeme kapacity konzumentů, do kterých odpad po hranách převážíme. A to tak, že se snažíme veškerý odpad, který po hraně do konzumenta převážíme, u něj vyložit. Naplníme-li celou kapacitu konzumenta, ale nevyložíme veškerý odpad, který po hraně transportujeme, vyjmeme z $clist_i$ dalšího nejbližšího konzumenta. Do dalšího konzumenta již po příchozí hraně převážíme množství odpadu snížené předchozím konzumentem. Následně při konstrukci dalších základních tras již máme sníženou kapacitu konzumentů, do kterých jsme převáželi odpad v předchozích trasách. Tím pádem se může stát, že pokud vyjmeme z $clist_i$ konzumenta, tak bude mít již nulovou kapacitu. V takovém případě konzumenta přeskočíme a vybereme dalšího. Díky tomuto snižování kapacit konzumentů je zajištěno dodržování kapacit konzumentů. Výsledné základní trasy by již mohly být přímo předány vozidlům, pokud bychom nedodržovali kapacity vozidel a měli dostatek vozidel

pro všechny základní trasy. Obrázek 6.1 ukazuje možné základní trasy pro příklad uvedený v sekci 5.2.



Obrázek 6.1: Příklad z části 5.2 – základní trasy pro producenty 1, 2 a 3

Pseudokód 2 popisuje tvorbu těchto základních tras. Proměnná *capacities* představuje mapu. Klíče v této mapě reprezentují jednotlivý konzumenti a hodnotami jsou jejich aktuální kapacity. Tím pádem *capacities*`.get(`l`)` získá aktuální kapacitu $l \in V_c$ a *capacities*`.set(`l`,` c_l`)` do mapy uloží novou kapacitu c_l konzumenta l. Dále je v pseudokódu 2 reprezentována základní trasa proměnnou `route`, přičemž `route.addEdge(`$m \in V$`,` a`)` přidá hranu z posledního vrcholu na trase do vrcholu m a uloží, že po této hraně se převáží množství odpadu, které je uvedené v parametru a. Pseudokód 2 nepočítá s tím, že na vstupu dostane nepřípustnou instanci.

Pseudokód 2 Tvorba základních tras iniciálního řešení

```
1:  function ([clist_i, ..., clist_j], capacities, [q_i, ..., q_j], V_p, 0)
2:      allBaseRoutes ← []
3:      for each k ∈ V_p do
4:          baseRoute ← trasa začínající depem
5:          for each l ∈ clist_k do
6:              consumerCapacity ← capacities.get(l)
7:              if consumerCapacity > 0 then
8:                  baseRoute.addEdge(l, q_k)
9:                  diff ← consumerCapacity − q_k
10:                 if diff ≥ 0 then
11:                     baseRoute.addEdge(0, 0)          ▷ 0 ∈ V je depo
12:                     capacities.set(l, diff)
13:                     break          ▷ V tento moment je trasa hotová.
14:                 else
15:                     q_k ← q_k − consumerCapacity
16:                     capacities.set(l, 0)
17:                 end if
18:             end if
19:         end for
20:         allBaseRoutes.add(baseRoute)
21:     end for
22:     return allBaseRoutes
23: end function
```

6.2.3 Mapování základních tras k vozidlům

V předchozí fázi jsme získali základní trasy, následující fáze tyto trasy přiřadí k jednotlivým vozidlům. Ke každému vozidlu si uložíme seznam tras, které jsme vozidlu přiřadili. Jak můžeme vidět v pseudokódu 3, vytvoříme si prioritní frontu s vozidly `plannedVehicles`. Každé vozidlo $l \in F$ bude mít parametr `finishAt`, který označuje, kdy vozidlo l končí (je to součet ujeté vzdálenosti). V prioritní frontě `plannedVehicles` se nám poté vozidla řadí právě podle parametru `finishAt`. Z prioritní fronty `plannedVehicles` budeme chtít nejprve získat vozidla, která končí nejdříve, tedy mají nižší `finishAt` parametr. Na počátku prioritní frontu naplníme všemi vozidly, která máme (viz řádek 4 a 5 v pseudokódu 3), v tomto okamžiku mají všechna vozidla parametr `finishAt` roven 0.

Do proměnné `plannedRoutes` budeme ukládat trasy, které jsme přiřadili ke konkrétnímu vozidlu. Jedná se tedy o mapu, kde klíčem je vozidlo a hodnotou je seznam tras.

Následně projdeme všechny základní trasy. Na začátku každého cyklu vyjmeme z prioritní fronty vozidlo, které končí nejdříve. Poté pomocí funkce `createVehicleRoute` (pseudokód 4 představíme později) vytvoříme pro dané vozidlo trasu ze základní trasy. Tato vytvořená trasa již bude brát ohled na kapacitu vozidla, bude tedy obsahovat otočky zpět ke producentovi za účelem

odvezení odpadu z něj daným vozidlem s danou kapacitou.

Dále je potřeba zvýšit parametr `finishAt` u vozidla tak, že k němu přičteme délku nově určené trasy (viz řádek 11 v pseudokódu 3). Vozidlo se zvýšeným parametrem opět vrátíme do prioritní fronty a trasu uložíme k vozidlu. Díky použití prioritní fronty a vracení vozidel zpět do stejné fronty je zajištěno, že pro každou základní trasu budeme mít k dispozici vozidlo. Pokud totiž nemáme dostatek vozidel, aby každé vozidlo obsluhovalo právě jednoho producenta (jednu základní trasu), tak použijeme vozidlo, které má zatím nejkratší trasu/trasy a přiřadíme mu další základní trasu. Vozidla tedy v tuto chvíli mohou mít přiřazených více tras, ne jen jednu.

Pseudokód 3 Mapování základních tras k vozidlům

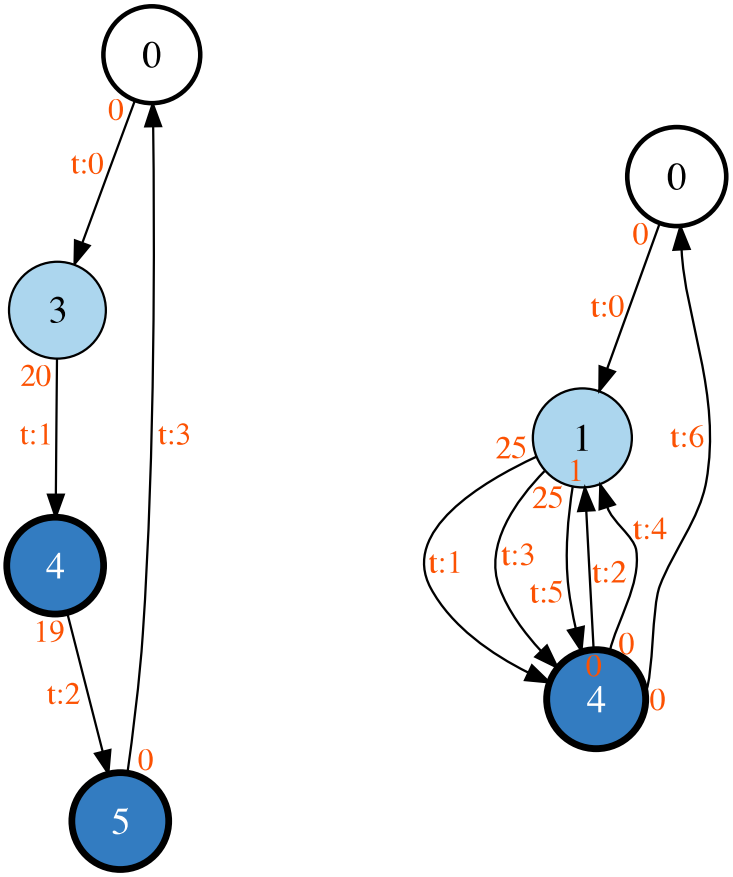
```
1:  function (allBaseRoutes, F, 0)
2:      plannedVehicles ← prioritní fronta, seřazená podle toho,
3:                        kdy naplánované vozidlo končí, od nejdřívějších
4:      for each l ∈ F do
5:          plannedVehicles.add(l)
6:      end for
7:      plannedRoutes ← Map<f ∈ F, routes[]>
8:      for each r ∈ allBaseRoutes do
9:          l ← plannedVehicles.poll()              ▷ Vyjme vozidlo z fronty.
10:         resultRoute ← createVehicleRoute(l, r, 0)
11:         l.finishAt ← l.finishAt + resultRoute.distance
12:         plannedVehicles.add(l)                  ▷ Vrátí vozidlo zpátky.
13:         plannedRoutes.get(l).add(resultRoute)
14:     end for
15:     return plannedRoutes
16: end function
```

Pseudokód 4 představuje funkci `createVehicleRoute` zmíněnou v pseudokódu 3. Jedná se o tvorbu trasy vozidla, splňující kapacitní omezení vozidla, ze základní trasy. Je tedy potřeba procházet celou základní trasu (v pseudokódu 4 je realizováno procházení po hranách). Objeví-li se na základní trase producent (kvůli povaze základní trasy víme, že to bude ihned v první iteraci), tak producenta uložíme do proměnné `prod` a do výsledné trasy přidáme hranu, po které bude vozidlo převážet nulové množství odpadu do producenta `prod` (trasa bude nyní tedy obsahovat cestu depo → producent). Do proměnné `cargo` uložíme, kolik odpadu je vozidlo schopno z producenta v daný moment odvézt. Jedná se o minimum z kapacity vozidla, nebo množství odpadu, které nabízí producent (viz řádek 11 v pseudokódu 4).

V příští iteraci bude na základní trase konzument. Nejprve musíme zjistit, jaké množství odpadu do tohoto konzumenta máme odvézt. To zjistíme tak, že odečteme od transportovaného množství odpadu po aktuální hraně základní trasy transportované množství hrany následující (viz řádek 16, pseudokód 4). Toto množství odpadu uložené v proměnné `stored` teď musíme obsloužit vozidlem tak, že se vozidlo bude stále vracet do producenta, dokud `stored`

nebude 0 (viz řádky 18 až 38, pseudokód 4). V parametru `wasteQuantity` proměnné `prod` je uloženo aktuální, zatím neodvezené množství odpadu producenta ze základní trasy.

Obrázek 6.2 ukazuje, jak můžeme vytvořit trasy pro vozidlo 2 z ukázkového příkladu uvedeného v sekci 5.2. Můžeme si povšimnout, že tyto dvě trasy vznikly z tras základních (viz obrázek 6.1). Trasa obsluhující producenta 3 je dokonce totožná se základní trasou pro producenta 3 (viz obrázek 6.1), zatímco u trasy pro producenta 1 se musí vozidlo několikrát vrátit do producenta 1. Další fází bude spojení těchto tras do jedné. Jak již bylo zmíněno, tak při mapování základních tras k vozidlům nejprve využijeme všechna vozidla, která máme k dispozici. Je-li ale přítomno více základních tras, než je počet vozidel, tak vyjmeme z prioritní fronty `plannedVehicles` (viz pseudokód 3) vozidlo, které končí nejdříve. Takovému vozidlu poté přiřadíme další základní trasu, která obsluhuje dalšího producenta. Z tohoto důvodu jsou v tomto okamžiku u vozidla 2 přítomny trasy dvě.



Obrázek 6.2: Příklad z části 5.2 – trasy pro vozidlo 2 před spojením

Pseudokód 4 Vytvoření trasy ze základní trasy

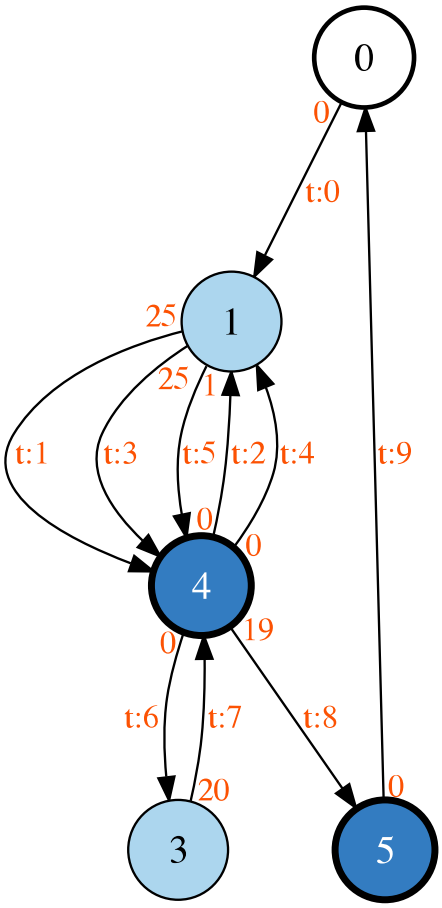
```
1:  function createVehicleRoute(vehicle, baseRoute, 0)
2:      route ← trasa začínající depem
3:      prod ← ∅
4:      cargo ← 0
5:      for each edge ∈ baseRoute do    ▷ Postupné procházení po hranách.
6:          dest ← edge.destination                          ▷ Cílový vrchol.
7:
8:          if dest je producent then
9:              prod ← dest
10:             route.addEdge(dest, 0)
11:             cargo ← min(vehicle.capacity, dest.wasteQuantity)
12:         end if
13:
14:         if dest je konzument then
15:             nextEdge ← baseRoute.nextEdge       ▷ Hrana vedoucí z dest.
16:             stored ← edge.transport − nextEdge.transport
17:
18:             first ← true
19:             while true do
20:                 if (route.lastNode je producent) nebo (first je true) then
21:                     route.addEdge(dest, cargo)
22:
23:                     if stored ≤ cargo then
24:                         cargo ← cargo − stored
25:                         prod.wasteQuantity ← prod.wasteQuantity − stored
26:                         break          ▷ S konzumentem dest jsme skončili.
27:                     else
28:                         stored ← stored − cargo
29:                         prod.wasteQuantity ← prod.wasteQuantity − cargo
30:                         cargo ← 0
31:                     end if
32:
33:                     first ← false
34:                 else if route.lastNode je konzument then
35:                     route.addEdge(prod, cargo)
36:                     cargo ← min(vehicle.capacity, prod.wasteQuantity)
37:                 end if
38:             end while
39:         end if
40:     end for
41:
42:     route.addEdge(0, 0)                          ▷ Trasu zakončíme depem.
43:     return route
44: end function
```

6.2.4 Spojení základních tras na vozidlech

Máme sice už trasy, které jsou přiřazené k jednotlivým vozidlům, splňující kapacitní omezení vozidel a konzumentů odvážející veškerý odpad z producentů. Vozidla ale mohou mít více než jednu trasu. Chtěli bychom tedy dosáhnout právě jedné trasy pro každé vozidlo. Toho dosáhneme tak, že trasy na vozidlech spojíme.

Spojení tras bude probíhat poměrně přirozeně tak, že postupně projdeme všechny trasy u vozidla a u první trasy odstraníme poslední hranu, která vede do depa. Následně u každé trasy kromě poslední, odstraníme počáteční a závěrečnou hranu (hrany z a do depa). U poslední trasy odstraníme počáteční hranu. Takto modifikované trasy spojíme postupným zřetězením.

Po splnění této poslední fáze již máme k dispozici přípustné řešení, kde každé vozidlo má svou trasu a jsou splněny všechny podmínky. Příkladem spojení může být obrázek 6.3, který opět vychází z příkladu uvedeného v části 5.2. Jedná se o výslednou iniciální trasu pro vozidlo 2. Můžeme si všimnout, že první trasa při spojování byla ta, která obsluhuje producenta 1 (viz obrázek 6.2). Jakmile tato trasa končí, začíná druhá obsluhující producenta 3.



Obrázek 6.3: Příklad z části 5.2 – iniciální trasa pro vozidlo 2

6.3 LS operátory

V této části se podíváme na LS algoritmus. Především na zvolené LS operátory pro řešení našeho problému. Nejprve se podíváme na to, jak můžeme vylepšit jednu konkrétní trasu a následně jak dvojice tras.

Jak je vidět v pseudokódu 1, LS je nedílnou součástí GLS. Náš LS algoritmus bude probíhat do té doby, dokud se nezasekne v lokálním minimu, jako je to znázorněno na obrázku 3.2. Sousední řešení aktuálního budeme konstruovat pomocí tahů, které buď modifikují jednu konkrétní trasu, nebo dvojici tras. Pořadí takovýchto tahů, nebo jejich vynechání bude předmětem experimentů v kapitole 8.

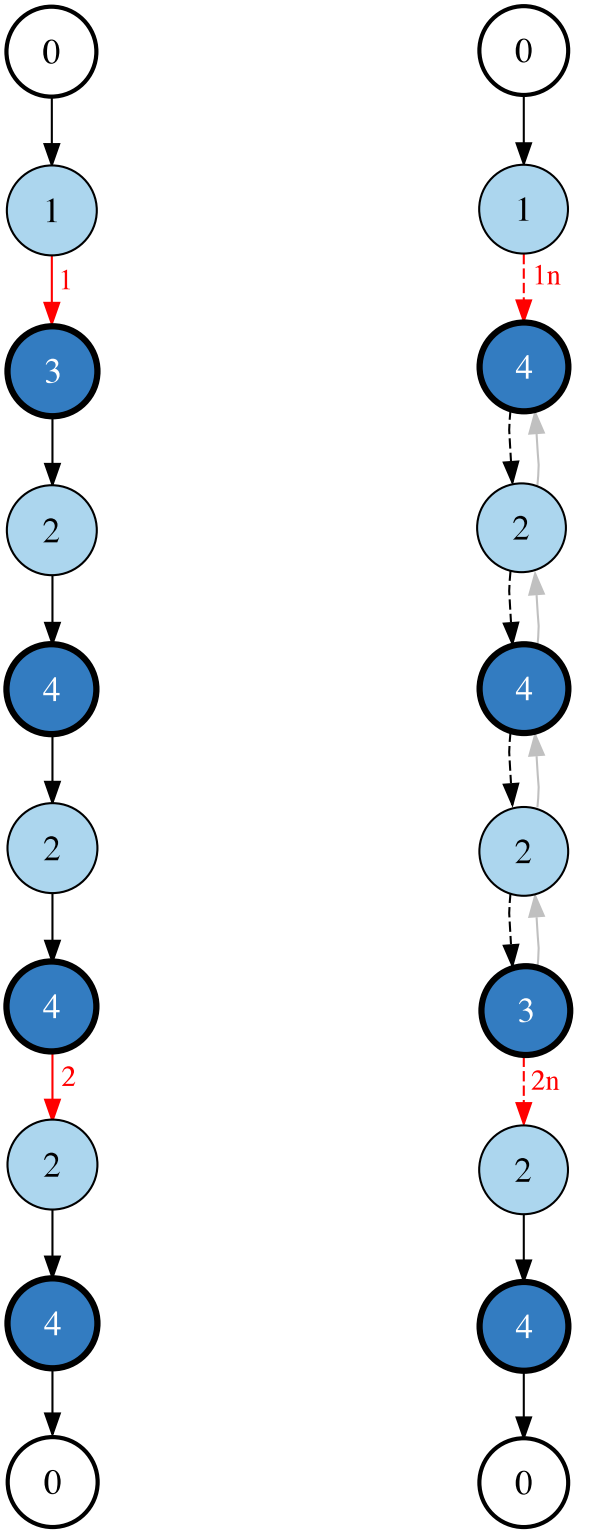
6.3.1 Jedna trasa

2-opt

Využití tzv. k-opt operátoru pro logistické úlohy je časté. Například práce [20], která se zabývá LS pro VRP, provádí vylepšování jedné trasy pomocí k-opt operátoru. Podle [20] spočívá k-opt v tom, že se na dané trase vynechá k hran a následně se vzniklé segmenty opět spojí hranami, ale jinak než tomu bylo původně. V důsledku toho mohou mít některé segmenty opačné pořadí, než měly původně. Tento operátor lze definovat pro libovolný počet hran počínaje dvěma. Avšak kvůli tomu, že by výpočet všech možných výměn mohl být náročný, tak se obvykle omezuje k na 2 nebo 3.

V našem případě jsme se rozhodli pro využití 2-opt operátoru, tedy budeme z jedné trasy vynechávat dvě hrany a vzniklé segmenty následně opět spojíme. Kvůli tomu, že trasa je orientovaná, tak prostřední segment bude muset otočit pořadí. To přináší některé významné komplikace, jelikož musíme stále dodržovat kapacity a odvézt požadované množství z producentů ke konzumentům. Toto je jeden z důvodů, proč pokud odstraníme nějaké dvě hrany, tak přehozením pořadí můžeme získat nepřípustnou trasu.

Obrázek 6.4 ilustruje přípustnou modifikaci trasy provedenou pomocí 2-opt. Původní trasa (viz obrázek 6.4a) znázorňuje přípustnou trasu pro neurčené vozidlo. Ačkoliv se tam nachází více vrcholů označených jako 0, 4, nebo 2 tyto vrcholy jsou vždy jeden tentýž, jen pro přehlednost je obrázek znázorněn jako řada vrcholů. Červeně znázorněné hrany v obrázku 6.4a označené jako 1 a 2 jsou takové, které jsme se rozhodli vynechat. Jediným způsobem, jakým můžeme po vynechání hrany 1 určit novou hranu je, že ji povedeme z producenta 1 do konzumenta 4. Tato nová hrana je znázorněna červeně přerušovaně jako 1n na obrázku 6.4b. V takovém případě musíme otočit pořadí vrcholů v následném segmentu. V segmentu, kde otáčíme hrany, jsou nové hrany znázorněny černě přerušovaně, zatímco původní hrany jsou znázorněny šedě. Po otočení pořadí vrcholů končí segment vrcholem 3, kterým jsme u původní trasy segment začínali. Následuje nová hrana označená v obrázku 6.4b jako 2n, vedoucí z 3 do 2. Pokračování až do depa je poté shodné s původní trasou.



(a) : Původní trasa **(b) :** Nová trasa

Obrázek 6.4: Ukázka 2-opt

Ukázali jsme si případ přípustné modifikace. Některé nepřípustné modifikace se dají určit v okamžiku určení hran, které vynecháme. Za tímto účelem byly identifikovány 3 následující pravidla:

- **R1** – Po depu může následovat pouze producent.
- **R2** – Po producentovi může následovat konzument. Pokud má vozidlo volnou kapacitu, tak může následovat i producent. Depo následovat nemůže nikdy.
- **R3** – Po konzumentovi může následovat producent. Pokud vozidlo ještě něco převáží, tak může následovat i konzument. Depo může následovat jen, pokud vozidlo už nic nepřeváží.

Po vynechané hraně jsme mohli převážet nějaký odpad. Toto množství si musíme zapamatovat, protože podle něj určíme, zda má vozidlo volnou kapacitu a tu potřebujeme pro posouzení pravidel **R1**, **R2** a **R3**. Tyto **R** pravidla nám vyfiltrují některé nepřípustné modifikace, nejedná se ale o všechny.

Náš 2-opt operátor zachovává konzistenci kapacit konzumentů a množství převáženého odpadu z producentů. V nově vzniklé trase budeme tedy z producentů odvážet stále stejná množství odpadu. Stejné množství odpadu budeme i dovážet do konzumentů. Tím zaručíme dodržování kapacit konzumentů a udání veškerého odpadu z producentů. Při tvorbě nové trasy si budeme hlídat odpovídající nakládky v původní trase. To znamená, že při konstrukci nové trasy se podíváme u každého producenta na to, jaké množství odpadu jsme z něj v odpovídající okamžik v původní trase odváželi. Toto množství převedeme do stejného okamžiku v nové trase. U konzumentů si před tvorbou nové trasy zjistíme, jaká množství odpadu jsme do jednotlivých konzumentů v původní trase odvezli. Nebudeme tedy, podobně jako u producentů, kopírovat vykládky v původní trase. Budeme se snažit aktuálně naložený odpad vždy v konzumentovi udat s ohledem na to, kolik odpadu přijímal v trase původní. To přináší nové problémy, které je potřeba hlídat a díky kterým může nově vzniklá trasa, která vznikla modifikací původní, být nepřípustná.

Prvním problémem, který může nastat, je stav, kdy po sobě následují dva, či více producentů. Kvůli kopírování nakládek z původní trasy se potom může stát, že by došlo k překročení kapacity vozidla. Tuto skutečnost tedy musíme ohlídat, protože poté by se jednalo o nepřípustnou trasu.

Situace, kdy vznikne souslednost producentů, přináší ještě jeden problém. Může nastat situace, kdy se v úseku po sobě jdoucích producentů opakuje tentýž producent, například $i \to j \to i$, kde $i, j \in V_p$. Postrádá smysl, proč by se vozidlo mělo naložit nějakým odpadem v i, potom v j a pak se opět vrátit do i a naložit zbytek. Pokud by tedy taková situace nastala, tak v dané souslednosti producentů sloučíme naložení odpadu v producentovi, který se opakuje. V uvedeném příkladu by tedy došlo ke sloučení nakládek ve vrcholu i a souslednost producentů by byla upravena na $i \to j$.

Při konstrukci nové trasy se může objevit i souslednost konzumentů. V reálném světě není moc pravděpodobné, že by vůz naložený nějakým sypkým

odpadem nechal část odpadu v jednom zpracovateli odpadu a zbytek odvezl do dalšího. Náš problém ale toto nezakazuje. Pokud bychom takovou situaci zakázali, tak by se nám obtížně konstruovalo přípustné řešení s ohledem na kapacity konzumentů. Mohlo by se totiž stát, že aby bylo vyhověno kapacitám konzumentů, tak by muselo dojít právě k souslednosti konzumentů v některé z tras. Tím pádem by se obtížně určovalo, zda je daná instance přípustná, či nikoli.

Pokud by v nové trase došlo k souslednosti konzumentů, tak oproti souslednosti producentů nemusíme kontrolovat, zda je splněno kapacitní omezení vozidla. Vozidlo se totiž pouze vykládá, tím pádem může nastat opačná situace, kdy by vozidlo mířilo do konzumenta prázdné. Nastane-li taková situace, potom konzumenta, ve kterém bychom nevyložili žádný odpad, přeskočíme a přejdeme rovnou k dalšímu producentovi. Kvůli tomu, že převážíme z producentů stejná množství odpadu a stejná množství i dováží me do konzumentů, tak pokud vynecháme konzumenta, tak část odpadu nebudeme mít kde udat. Na konci trasy by potom nastala situace, kdy by se vozidlo do depa vracelo naplněné a jednalo by se o nepřípustnou trasu. Tím pádem musíme na konec nové trasy přidat konzumenty, které jsme v průběhu trasy vynechali.

Obdobně jako u souslednosti producentů může nastat situace, kdy po sobě v souslednosti konzumentů bude následovat stejný konzument. Například trasa bude obsahovat úsek $k \to l \to k$, kde $k, l \in V_c$. Vykládky ve stejném vrcholu v souslednosti konzumentů sloučíme. V uvedeném příkladu tedy vznikne úsek $k \to l$.

Komplexnost vylepšování trasy našeho problémů pomocí k-opt byla důvodem, proč jsme nezvolili k větší než dva. Dvojice hran budeme pro 2-opt vybírat tak, že na trase zkusíme všechny kombinace, abychom vyzkoušeli všechny možnosti. Pokud se nám podaří získat přípustnou trasu, tak její cenu porovnáme s nejlepší dosud nalezenou, a pokud je lepší, tak si ji uložíme jako dosud nejlepší trasu. Samotný výběr hran probíhá stále na původní trase.

Pro výpočet ceny trasy nemusíme započítávat všechny členy kriteriální funkce (5.1), stačí nám pouze pouze člen vyjadřující vzdálenost trasy, viz člen (5.3). To protože 2-opt operátorem zachováváme převážené množství odpadu do konzumentů, tím pádem víme, že nedojde ke změně členu (5.6), který vyjadřuje skóre konzumenta. Člen (5.7) vyjadřující maximální vzdálenost zde také nehraje roli, protože nám jde o to cestu zkrátit. Pokud by trasa, kterou vylepšujeme, byla v řešení nejdelší trasou, tak bychom zkrácením trasy docílili i zmenšením tohoto členu (tedy vylepšení celkové ceny). Toho docílíme ale i v případě, pokud tento člen vynecháme, protože budeme považovat kratší trasu za lepší. Rozhodneme-li se takto vylepšenou trasu použít, musíme samozřejmě cenu celého řešení následně přepočítat, protože budeme mít novou nejdelší trasu. Pokud trasa, kterou vylepšujeme, nebude nejdelší trasou v řešení, tak se z členu (5.7) stane konstanta. Stále ovšem musíme započítávat penalizace hran plynoucí z rozšířené nákladové funkce (6.3). Operátor 2-opt budeme využívat uvnitř dalších operátorů, které uvedeme dále, nebude se tedy vyskytovat samostatně.

Výměna s nepoužívanými konzumenty

Další způsob, jak můžeme vylepšovat jednu konkrétní trasu, je takový, kdy se podíváme na konzumenty, které nejsou vůbec součástí našeho řešení. Takové konzumenty můžeme zkusit vyměnit s konzumenty, které máme na trase. Musíme se samozřejmě podívat, zda kapacita nepoužívaného konzumenta vyhovuje množství odpadu, které jsme převáželi do konzumenta původního. Po výměně zavoláme na trasu 2-opt (viz sekce 6.3.1), který nám tuto trasu může ještě vylepšit díky přeskládání pořadí. Abychom docílili co největšího potenciálu tohoto operátoru, tak jej budeme používat tak, že v iteraci LS algoritmu jej zavoláme na všechny trasy aktuálního řešení.

Nalezneme-li díky tomuto operátoru lepší trasu, tak ji okamžitě uložíme do řešení. Uchovávat všechna nalezená vylepšení pro trasy by totiž bylo náročné. Museli bychom se rozhodnout, která vylepšení využijeme. Při rozhodování bychom totiž museli brát ohled na kapacitu nepoužívaných konzumentů. Pravděpodobně bychom tedy nemohli využít všechna vylepšení a stejně bychom se rozhodovali postupně, které vylepšené trasy promítneme do řešení. Kvůli okamžitému promítnutí vylepšené trasy, bude záležet na pořadí výběru tras. Může se totiž stát, že nepoužívaný konzument nebude mít kapacitu obsloužit další trasu v pořadí, protože byl právě využit pro předchozí trasu. Bude tedy záležet na implementaci algoritmu, zda bude zachovávat stejné pořadí při výběru tras, nebo ne. Pokud konkrétní implementace algoritmu zachovávat pořadí při výběru nebude, tak se může stát, že budeme v různých bězích získávat rozdílné výsledky.

6.3.2 Dvojice tras

V této části se podíváme na způsoby, kterými budeme nalézat sousední řešení výběrem dvou tras a následnou jejich vzájemnou modifikací. Výběr tras můžeme realizovat několika způsoby. My budeme buď tvořit variace, nebo kombinace všech dvojic (bez opakování).

V práci [20] jsou uvedeny tři nejběžnější operátory pro VRP, které se týkají změn provedených mezi dvěma trasami. Jedná se o relocate operátor, kdy z jedné trasy přesuneme vrchol do druhé trasy. Potom je uveden swap operátor, kdy se vymění jeden vrchol z jedné trasy s nějakým vrcholem z trasy druhé. Poslední uvedený je cross operátor, který obě trasy rozdělí a vzniklé segmenty prohodí.

Přesun producenta

Tento operátor vychází z relocate operátoru uvedeného v předchozím odstavci. Nebudeme však přesouvat jakýkoliv vrchol z jedné trasy do druhé, ale pouze producenty. Konzumenty přesouvat nemůžeme. V trase, ze které bychom přesouvali konzumenta, do něj ukládáme nějaké množství odpadu, které bychom poté neměli kam uložit.

Při přesunu producenta z trasy A do trasy B přijdeme v trase A o množství odpadu, které jsme z tohoto producenta odváželi. To znamená, že v trase

A budeme muset odstranit i některé konzumenty. Naopak v trase B budeme muset přidat konzumenty, do kterých odpad z přesunutého producenta odvezeme. To znamená, že přesuneme producenta i s tím, že využijeme stejné konzumenty z původní trasy pro odvoz odpadu z tohoto producenta. Po dokončení přesunu ještě zkusíme vylepšit vzdálenosti na každé trase pomocí 2-opt operátoru (viz sekce 6.3.1).

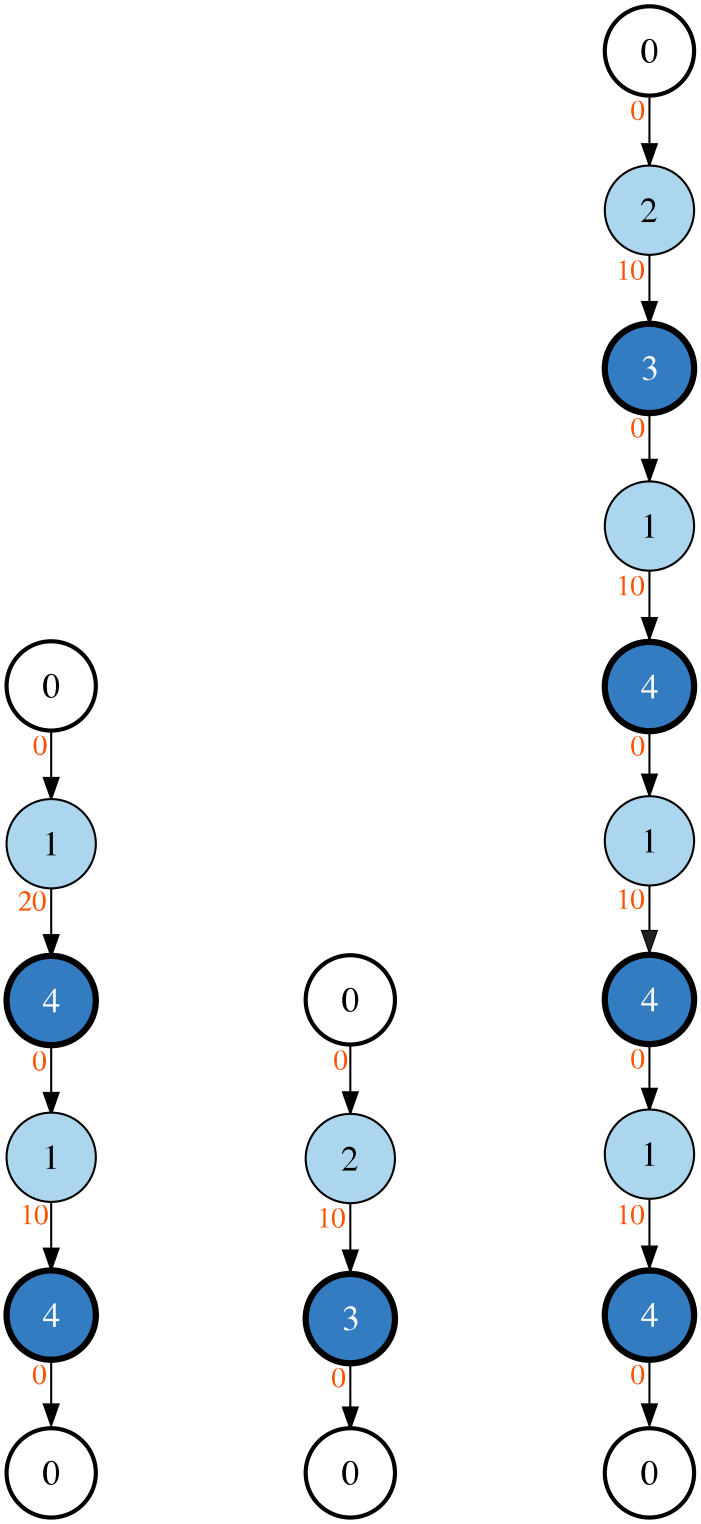
Abychom využili největšího potenciálu tohoto operátoru, tak jej zkusíme realizovat pro všechny možnosti. Vždy budeme mít jednu trasu, ze které zkoušíme přesouvat všechny producenty do druhé (tím otestujeme všechny možnosti přesunu z jedné trasy do druhé). Kvůli tomu nám záleží na pořadí ve výběru dvojice, protože zkoušíme přesouvat jen z první trasy do druhé. Pokud bychom zkoušeli přesouvat producenty z druhé trasy do první, dostali bychom jiný výsledek. Využijeme tedy variace bez opakovaní pro výběr dvojic tras. Budeme do dvojic vybírat i vozidla, která nemají přiřazenou žádnou trasu. Díky tomu můžeme vytvořit novou trasu pro vozidla, která jsme v předchozím řešení nevyužívali.

Jak již bylo zmíněno, tak s producentem přesouváme i konzumenty, do kterých producent převážel odpad v původní trase. To je realizováno tak, že před vyjmutím producenta se spočítá jaké množství v jeho trase každý konzument přijímá. Po vyjmutí producenta v této trase opět spočítáme dovážené množství do jednotlivých konzumentů. Následně se podíváme na rozdíly převážených objemů do jednotlivých konzumentů. Tyto rozdíly nám poté určují, kam a v jakém množství máme odvézt odpad z přesunutého producenta v nové trase. Díky tomu jsou zachovávána množství odpadů, která převážíme do konzumentů. Tím pádem při výpočtu ceny modifikace můžeme vynechat z kriteriální funkce člen (5.6), který představuje penalizaci konzumentů.

Obdobně jako u operátoru výměny používaných konzumentů za nepoužívané, i v tomto případě, nalezneme-li lepší situaci pro dvojici tras, tak ji okamžitě promítneme do řešení. Bylo by totiž obtížné uchovávat všechna nalezená vylepšení a poté se rozhodnou, které použít, protože jedna trasa je párována se všemi ostatními (využíváme variace). Řešili bychom poté párovací problém, kdy bychom se museli rozhodnout u každé trasy, které vylepšení zvolit, a tím bychom ovlivnili i druhou trasu (u které bychom se mohli také rozhodovat). Kvůli tomu, že vylepšení okamžitě promítneme do řešení, tak bude záležet na pořadí výběru dvojic. Bude-li implementace algoritmu takové pořadí zachovávat, budeme pro různé běhy dostávat stejné výsledky, jinak mohou být výsledky rozdílné.

Ukázku tohoto operátoru si ilustrujme na obrázku 6.5. Máme dvě vozidla, 1 a 2, vozidlo 1 má kapacitu 20 jednotek a vozidlo 2 10 jednotek. Z původní trasy vozidla 1, viz obrázek 6.5a, přemístíme producenta 1 do trasy vozidla 2, viz obrázek 6.5b a 6.5c. Přesuneme producenta na konec trasy, tedy po konzumentovi 3 přijde okamžitě producent 1. V trase vozidla 1 se odpad z producenta 1 převážel do konzumenta 4, takže i v trase vozidla 2 se bude převážet do 4. Vozidlo 2 má menší kapacitu, než vozidlo 1, to znamená, že zatímco se vozidlo 1 stačilo naložit dvakrát v producentovi 1, tak vozidlo

2 se musí naložit třikrát. Z trasy vozidla 1 po odstranění producenta 1 nic nezbude, tedy vozidlo 1 v tento moment nemá přiřazenou žádnou trasu.



(a) : Původní trasa voz. 1 **(b) :** Původní trasa voz. 2 **(c) :** Nová trasa voz. 2

Obrázek 6.5: Operátor přesunu producenta

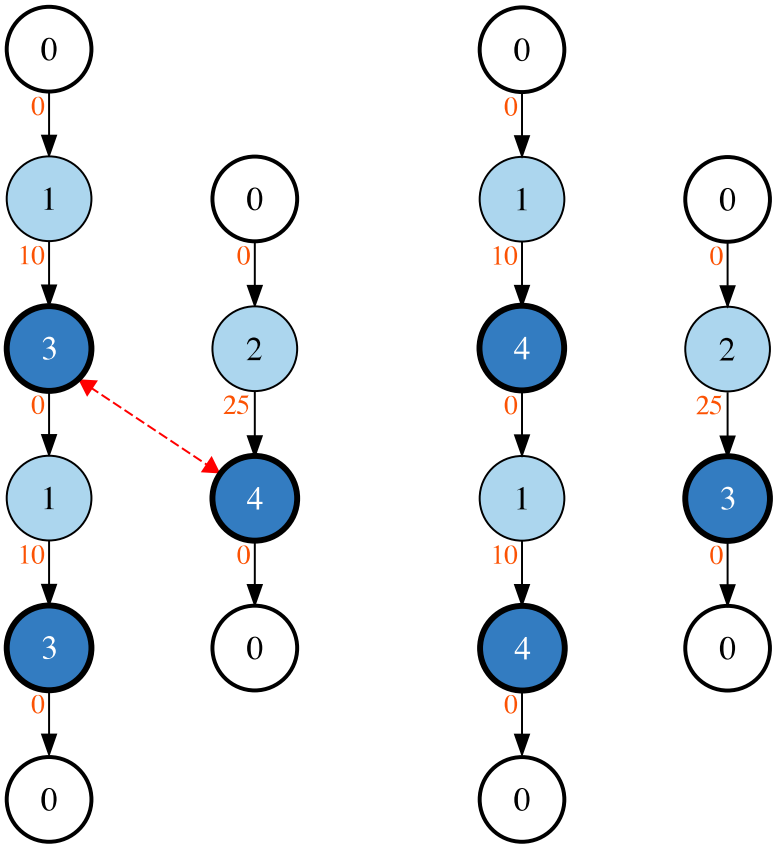
Výměna konzumentů

Další operátor, který použijeme, je operátor vycházející ze swap operátoru uvedeného na začátku této sekce 6.3.2. Operátor spočívá v tom, že zkusí prohodit konzumenty mezi dvěma trasami. Samozřejmě se musíme podívat, zda vyhovují kapacitní omezení konzumentů. Opět abychom využili maximální potenciál tohoto operátoru, tak budeme zkoušet prohazování všech konzumentů z obou tras.

Podobně jako u 2-opt operátoru (viz sekce 6.3.1) může nastat případ, kdy po sobě bude následovat tentýž konzument. Například představme si situaci, kdy v jedné trase je úsek $i \to j \to k$, kde $i, j, k \in V_c$, a my budeme chtít i nahradit za k. V takové situaci by v trase následoval úsek $k \to j \to k$, kdy je nežádoucí, aby se auto vracelo do k a vyložilo zbytek nákladu, když už v k bylo a znovu se nikde nenaložilo. Tuto situaci musíme pohlídat a vyložit veškerý odpad pro k rovnou při první návštěvě k (příští návštěvu přeskočíme). Po modifikaci obou tras zavoláme ještě na obě 2-opt operátor, abychom vylepšili vzdálenosti v každé trase (viz sekce 6.3.1).

Budeme-li chtít otestovat všechny možnosti, tak výběr dvojic tras zde budeme realizovat pomocí kombinací. V tomto případě, na rozdíl od operátoru přesunu producenta, nám totiž nezáleží na pořadí ve dvojici. Pro porovnávání, zda je po výměně konzumentů dvojice tras lepší, než původní, již musíme počítat s celou kriteriální funkcí (5.1). Výměnou konzumenta totiž docílíme jak změny množství převáženého odpadu do konkrétního konzumenta, tak i změny délky trasy a možná i změny nejdelší trasy v řešení. U dvojice tras si můžeme ukládat jednotlivé výsledky výměny konzumentů a do řešení promítnou nejlepší nalezenou. I přes tuto skutečnost nám stále bude záležet na pořadí výběru dvojic tras, kterou může implementace algoritmu zaručovat stejnou, ale také nemusí. Opět nebude-li implementace algoritmu zachovávat pořadí výběru dvojic tras, můžeme pro různé běhy algoritmu získat rozdílné výsledky.

Na obrázku 6.6 je ilustrován tento operátor. Obrázek 6.6a ukazuje dvě trasy, přičemž červenou přerušovanou čarou je znázorněno realizované prohození konzumentů. Na obrázku 6.6b jsou potom tyto dvě trasy s prohozenými konzumenty.



(a) : Trasy před výměnou **(b) :** Trasy po výměně

Obrázek 6.6: Operátor výměny konzumentů

Část III

Prototyp

Kapitola 7
Implementace

V předchozí kapitole jsme si ukázali řešení problému z kapitoly 5 v podobě GLS algoritmu. Pro účely experimentů uvedených v kapitole 8 bylo zapotřebí implementovat algoritmus a některé další aplikace. Vyjma implementace samotného algoritmu to je generátor instancí a nástroj pro filtraci dat z registru ISOH (o registru se dočtete v kapitole 4).

Tato kapitola popisuje implementované aplikace a jejich rozhraní pro budoucí využití. Nejprve se podíváme na generátor instancí, následně na implementaci GLS algoritmu a na konec na nástroj pro filtraci dat z ISOH.

7.1 Generátor instancí

Pro účely experimentů je důležité umět vygenerovat instance. Generovat instance manuálně by bylo nepraktické a u větších instancí prakticky nemožné. Za tímto účelem byl stvořen nástroj, který umí instance našeho problému vygenerovat. Nástroj byl vytvořen v jazyce Python, verze 3.9. Python byl zvolen především kvůli své jednoduchosti a přenositelnosti mezi různými operačními systémy.

Výstupem generátoru je soubor v rozšířeném trivial graph formátu (dále rozšířený TGF). V případě prostého TGF se jedná o textový soubor se spojovým seznamem pro popis matematických grafů [33]. Formát se skládá z definice vrcholů, následované definicí hran (viz ukázka, kód 7.1). Každý vrchol je zapsán na samostatném řádku, nejprve je uvedeno ID vrcholu, po kterém následuje označení vrcholu, které od ID odděluje mezera. Po zapsání všech vrcholů následuje na samostatném řádku znak `#` a na řadu přichází zápis hran. Každá hrana je opět zapsána na samostatném řádku, přičemž nejprve je uvedeno ID výchozího vrcholu, následované mezerou a ID cílového vrcholu. Je-li nutné zapsat označení hrany, tak je následováno po další mezeře.

Kód 7.1: Ukázka prostého TGF

```
1 První vrchol
2 Druhý vrchol
3 Třetí vrchol
#
1 2 Hrana mezi prvním a druhým vrcholem
```

V práci [33] je uvedeno, že TGF nemá dostatečnou standardizaci, a tak existuje mnoho podob. To je případ i našeho formátu, kdy je prostý TGF nedostatečný, protože potřebujeme v souboru oddělit konzumenty od producentů, zapsat kapacity, skóre konzumentů a množství odpadů v producentech. Dále potřebujeme zapsat vozidla s jejich kapacitami a vzdálenosti hran. Z tohoto důvodu si prostý TGF rozšíříme.

Budeme mít 4 typy vrcholů, které do souboru zapíšeme. Prvním je vrchol reprezentující depo, který se ve validním souboru bude nacházet pouze jeden. Druhým typem jsou konzumenti a dalším producenti a posledním vozidla. Jelikož TGF zahrnuje jen část pro vrcholy a hrany, tak v tomto případě budeme reprezentovat vozidla také jako vrcholy, ale jen z hlediska reprezentace v TGF souboru. Depo bude mít jako ID uvedený znak `D` (viz řádek 1 kódu 7.2). ID producentů bude začínat znakem `P`, přičemž bude bezprostředně následovat celočíselný identifikátor mezi producenty. Po ID producenta následuje mezera následovaná celým číslem, které označuje množství odpadu v daném producentovi (viz řádky 2 a 3 kódu 7.2). ID konzumentů bude začínat znakem `C`, opět bezprostředně následované celočíselným identifikátorem mezi konzumenty. Dále uvedeme kapacitu konzumenta, která je od ID oddělena mezerou. Poslední údaj, který potřebujeme u konzumenta uvést, je jeho skóre jako zpracovatele odpadu, které následuje po kapacitě a od kapacity jej dělí opět mezera (viz řádek 4 kódu 7.2). Vozidla budou mít identifikátor začínající znakem `A`, bezprostředně následovaný celým číslem, které vozidlo odlišuje od ostatních vozidel. Kapacita vozidla je potom zapsána celým číslem po mezeře, která jej odděluje od ID (viz řádek 5, kód 7.2).

Stejně jako v prostém TGF následuje na samostatném řádku znak `#`, po kterém zapíšeme hrany. Hrany budeme zapisovat stejným způsobem jako jsou zapisovány v prostém TGF, jen místo označení uvedeme celočíselnou hodnotu, která reprezentuje vzdálenost hrany. Tato reprezentace je vidět v ukázce kódu 7.2 na řádcích 7 až 18, můžeme si všimnout, že máme úplný orientovaný graf, tzn. že vede hrana z každého vrcholu do každého dalšího (vyjma vrcholů reprezentující v tomto zápisu vozidla).

Kód 7.2: Ukázka námi rozšířeného TGF

```
1  D
2  P1 51
3  P2 42
4  C1 124 1
5  A1 25
6  #
7  C1 P1 8
8  C1 P2 17
9  C1 D 10
10 D C1 10
11 P1 C1 8
12 P1 P2 8
13 P1 D 2
14 D P1 2
15 P2 C1 17
16 P2 P1 8
17 P2 D 6
```

```
18 D P2 6
```

Jelikož pro zadání musí platit tzv. trojúhelníková nerovnost (tj. hrana $(i, j) \in E$ je nejkratší cestou jak se dostat z $i \in V$ do $j \in V$), nástroj si vytvoří uvnitř virtuální 2D prostor, do kterého náhodně umístí jednotlivé vrcholy. Díky tomu může poté určit vzdálenost hran mezi vrcholy tak, aby byla trojúhelníková nerovnost dodržena. Tento princip ale neumožňuje rozdílné vzdálenosti mezi hranou $(i, j) \in E$ a hranou $(j, i) \in E$. Nicméně pro účely experimentů tuto skutečnost můžeme zanedbat. V zadání problému jsou vzdálenosti uvedeny celočíselně, musíme tedy vzdálenosti zaokrouhlovat. Generátor využívá euklidovskou vzdálenost, kterou následně zaokrouhlí vždy dolů.

Generátor přijímá při spuštění některé povinné a nepovinné argumenty, podle kterých následně výslednou instanci vygeneruje. Tyto argumenty jsou uvedeny v tabulce 7.1.

Argument	Popis	Povinné	Typ
-np	Počet producentů.	✓	$\mathbb{N}$
-nc	Počet konzumentů.	✓	$\mathbb{N}$
-nv	Počet vozidel.	✓	$\mathbb{N}$
-ncr	Počet konzumentů, kteří mají skóre 0.	✓	$\mathbb{N}_0$
-nce	Počet konzumentů, kteří mají skóre 1.	✓	$\mathbb{N}_0$
-x	Velikost osy X 2D prostoru pro generování vrcholů.	✗	$\mathbb{N}$
-y	Velikost osy Y 2D prostoru pro generování vrcholů.	✗	$\mathbb{N}$
-wql	Spodní hranice množství odpadu v producentovi.	✗	$\mathbb{N}$
-wqu	Horní hranice množství odpadu v producentovi.	✗	$\mathbb{N}$
-vcl	Spodní hranice kapacity vozidla.	✗	$\mathbb{N}$
-vcu	Horní hranice kapacity vozidla.	✗	$\mathbb{N}$
-ccd	Maximální odchylka kapacit konzumentů.	✗	$\mathbb{N}_0$

Tabulka 7.1: Argumenty generátoru instancí

Není-li uvedená velikost osy X 2D prostoru pro generování vrcholů, je tento parametr určen tak, že se počet producentů vynásobí koeficientem rovným deseti. Obdobně pokud není definována velikost osy Y, vynásobí se deseti počet konzumentů.

Další otázkou může být, jak uživatel nastaví počet konzumentů, kteří mají skóre 2. Počet konzumentů se skóre 2 se určí odečtením argumentů `-ncr` a `-nce` od celkového počtu konzumentů.

Co se týče množství odpadu jednotlivých producentů, to je určováno ná-

hodně z určitého rozmezí. Ve výchozím stavu je spodní hranice tohoto rozmezí stanovena na 10 a horní na 60. Argumenty `-wql` a `-wqu` slouží k manuálnímu stanovení tohoto rozmezí, `-wql` omezí náhodný výběr zespoda, zatímco `-wqu` zeshora. Obdobně je určena kapacita vozidel, kdy ve výchozím stavu je náhodně vybírána kapacita pro konkrétní vozidlo z intervalu $\langle 5, 25 \rangle$, přičemž tento interval se dá přenastavit argumenty `-vcl` a `-vcu` ($\langle$`-vcl`, `-vcu`$\rangle$).

Kapacity konzumentů jsou poté určovány tak, že se celkové množství odpadu, které je potřeba odvézt ze všech producentů, podělí počtem konzumentů. Tím se získá, kolik by přibližně mělo odpadu připadnout na jednoho konzumenta, aby bylo rozložení rovnoměrné (zaokrouhluje se vždy nahoru). Poté se pro každého konzumenta vygeneruje náhodné číslo, které budeme nazývat odchylkou konzumenta. Odchylka se přičte k vypočítanému rovnoměrného rozložení pro jednoho konzumenta. Odchylku přičítáme, abychom zajistili, že bude možnost výběru mezi jednotlivými konzumenty. Pokud by se totiž součet kapacit konzumentů rovnal celkovému množství odpadu nabízeného producenty, potom by se nedalo rozhodovat mezi tím, do jakého konzumenta odpad odvezeme. Museli bychom využít kapacity konzumentů na maximum. Ve výchozím stavu se odchylka určí jako 1/3 množství odpadu, které by při rovnoměrném rozdělení mělo připadnout právě jednomu konzumentovi. Označíme-li maximální možnou hodnotu této odchylky jako x, potom výběr náhodného čísla, tedy odchylky konkrétního konzumenta, probíhá na intervalu $\langle 0, x \rangle$. Proměnnou x reprezentuje argument `-cdd`. Argumentem `-cdd` tedy můžeme modifikovat maximální možnou hodnotu této odchylky.

Díky navrženým argumentům pro generování instance budeme moci vytvořit takovou instanci, která bude simulovat reálnou oblast. Parametry oblasti zase určíme pomocí nástroje pro filtraci dat z ISOH (viz sekce 7.3). Z důvodu toho, že souřadnice v ISOH nemusí být zrovna přesné (viz kapitola 4) jsme se rozhodli, že budeme souřadnice vrcholů generovat vždy náhodně na námi definované velikosti 2D prostoru. Před nástrojem na filtraci dat z ISOH se ale ještě podíváme na implementaci GLS algoritmu.

7.2 Implementace GLS

Implementace algoritmu, který řeší instance našeho problému probíhala v jazyce Java, verze 17. Java byla zvolena z důvodu rychlosti běhu aplikace, dobrou znalostí jazyka autorem a především jednoduššímu debuggování oproti například jazyku C++. Tuto aplikaci, která přijímá instanci na vstupu a vygeneruje řešení, budeme dále nazývat jako implementace GLS.

Přijímaná instance má formát rozšířeného TGF, který jsme uvedli v sekci 7.1. Na vstupu se zdrojový TGF soubor předává argumentem `-i`, který označuje cestu k tomuto souboru. Přehled argumentů implementace GLS naleznete v tabulce 7.2. Dalším důležitým argumentem je argument `-o`, který určuje cestu do adresáře, kde bude vygenerovaný adresář s řešením instance. Posledním argumentem je `-p`, který představuje cestu k souboru formátu *.properties, ve kterém se nastavuje běh aplikace a některé další parametry. Na tento soubor se nyní podíváme podrobněji.

Argument	Popis	Povinné
-i	Cesta k souboru s instancí.	✓
-o	Cesta do adresáře, kde bude vygenerován adresář s výsledkem.	✓
-p	Cesta k souboru s nastavením běhu aplikace.	✓

Tabulka 7.2: Argumenty přijímané implementací GLS

V ukázce kódu 7.3 je ukázáno vzorové nastavení parametrů v souboru *.properties pro běh aplikace. Nastavení je rozděleno do 2 oblastí, první se týká základních vlastností algoritmu, které byly popsány v kapitole 5 a 6. Zatímco druhá se týká nastavení toho, jak budou vypadat výsledné obrázky, které z aplikace vzejdou. Pro vykreslení obrázků je zapotřebí mít na počítači nainstalovaný software Graphviz[1] s možností generování obrázků přes příkazovou řádku. Není-li na počítači tento nástroj k dispozici, implementace GLS při vykreslování oznámí chybu, avšak dodá do výsledné složky s řešením *.txt soubory se zápisem grafů pro vykreslení nástrojem Graphviz. Jednotlivé vizualizace mají stejný formát, jaký je zobrazován při vizualizací grafů a tras problému v této práci.

Na ukázce jsou znázorněny všechny parametry pro běh algoritmu. Vidíme, jak je možné nastavit parametr α, β z kriteriální funkce (5.1). Dále pak parametr λ, který slouží ke korigování vlivu GLS penalizací na nákladovou funkci (viz sekce 6.1). Jako zastavující kritérium GLS algoritmu je využíván počet iterací, který se nastaví parametrem `gls.number-of-iterations`.

Tři celočíselné parametry, začínající `gls.operator`, slouží k určení pořadí operátorů. Parametr `producer-relocation` slouží k určení pořadí operátoru přesunu producenta (viz sekce 6.3.2). Pro určení pořadí operátoru pro výměnu konzumentů je k dispozici parametr `consumer-exchange`. Pořadí operátoru pro výmenu použitých konzumentů s nepoužívanými se poté nastavuje parametrem `unused-consumers`. Pořadí se určuje číslem od 1 do 3, přičemž operátor 1 bude první a 3 poslední. Uvedeme-li číslo, které není v rozmezí od 1 do 3, tak tím operátor vypneme. Tyto parametry jsou připravené pro experimenty (viz kapitola 8).

Poslední důležitý parametr pro běh algoritmu je takový, kterým určíme hodnotu parametru γ (viz sekce 6.1.1). K nastavení hodnoty γ slouží parametr `gls.features.consumers.GAMMA`. Není-li parametr γ uveden, zvolí se jako průměr vzdáleností všech hran.

Parametr `intermediate-results` je potom seznam celých kladných čísel, který slouží především pro účely experimentů v kapitole 8. Tímto parametrem získáme výsledek z určité iterace běhu GLS algoritmu. Díky možnosti získat mezivýsledky potom nemusíme pro každý počet iterací pouštět aplikaci znovu, ale stačí pouze jeden běh. Příklad uvedený v kódu 7.3 tedy dodá mezivýsledky při 100., 250. a 500. iteraci GLS algoritmu.

[1] `https://graphviz.org/`

Navíc naše implementace GLS nepoužívá při řešení žádnou náhodu a zachovává pořadí při různých výběrech, která jsou nutná realizovat v algoritmu (např. výběr dvojic tras). Různá pořadí výběrů by znamenala různé výsledky pro různé běhy, jak jsme zmínili v sekci 6.3. To znamená, že pro konkrétní instanci bude určitá iterace se stejnými vstupními parametry vždy stejná.

Posledním základním parametrem je parametr `time-measurement`, tímto parametrem zapneme měření času při běhu algoritmu. Součástí řešení potom bude i čas v milisekundách, za který jsme řešení při daných parametrech získali.

Kód 7.3: Ukázka nastavení řešitele

```
### BASICS

cost-function.ALPHA=1
cost-function.BETA=100
gls.LAMBDA=3
gls.number-of-iterations=4000
gls.operator.producer-relocation=2
gls.operator.consumer-exchange=3
gls.operator.unused-consumers=1
intermediate-results=100,250,500

# gls.features.consumers.GAMMA=10
# unset means the average of the edge distances

time-measurement=true

### VISUALIZATION

vis=true

# depot, consumer or producer
# vis.depot.fill="yellow"
# vis.depot.fontColor="black"

# vis.edge.distance.fontColor="purple"
# vis.ratio=1
```

U parametrů pro vizualizaci vidíme, že můžeme nastavit barvu výplně jednotlivým typům vrcholů, stejně tak barvu jejich popisků. Navíc můžeme nastavit barvu popisků u hran a poměr výsledných obrázků[2]. Booleovským parametrem `vis`, nastaveným na hodnotu false, můžeme vykreslování grafů do obrázků úplně vypnout.

Dále se podíváme na výstup implementace GLS. Výstupem je adresář s následujícím názvem: solution-*[název souboru s instancí]*-it*[počet iterací GLS]*-*[datum a čas]*. Uvnitř adresáře najdeme 5 souborů – assignment.tgf (instance problému), assignmentGraph.svg (vizualizace grafu instance problému), assignmentGraph.txt (kód pro vygenerování grafu instance problému pomocí Graphviz), characteristics.properties (nastavení běhu aplikace) a result.txt (řešení instance). Ukázku obsahu result.txt souboru můžeme vidět

[2] O poměru se dozvíte více na `https://graphviz.org/docs/attrs/ratio/`

v kódu 7.4. Získáme trasy pro jednotlivá vozidla s množstvím odpadu převáženého na jednotlivých hranách v trase. Můžeme si všimnou i celkové ceny řešení a kontrolních dat, která sloužila hlavně při vývoji k ověření, zda je výsledek doopravdy přípustné řešení. Kromě souborů se ve výsledném adresáři bude také nalézat adresář routes. V tomto adresáři nalezneme vizualizace tras pro jednotlivá vozidla opět i s textovým souborem reprezentující zápis pro vygenerování vizualizace pomocí Graphviz nástroje (např. A1.svg a A1.txt).

Kód 7.4: Ukázka řešení instance

```
1  ############ ROUTES
2  # [VEHICLE]: N -[quantity transported]-> N, dist: [route distance]
3
4  A2: D -0-> P1 -25-> C1 -0-> P1 -25-> C1 -18-> P1 -19-> C2 -0-> P2 -25->
       C2 -0-> P2 -17-> C2 -0-> D, dist: 113
5  A1: D -0-> P3 -5-> C1 -0-> P3 -5-> C1 -0-> P3 -5-> C1 -0-> P3 -5-> C1
      -0-> D, dist: 51
6
7
8  ############ SUMMARY
9
10 Total cost: 11466.0 (alpha: 1.0, beta: 100.0)
11 The longest route: A2 (distance: 113)
12 Total distance: 164
13
14
15 ############ GLS PARAMETERS
16
17 Number of iterations: 100
18 Lambda: 3.0
19 Gamma: avg
20
21
22 ############ CONTROL DATA
23
24 # Current amount of waste in producers
25 P1 (amount: 0, origin amount: 51)
26 P2 (amount: 0, origin amount: 42)
27 P3 (amount: 0, origin amount: 20)
28
29 # Vehicle capacities
30 A1 (capacity: 5)
31 A2 (capacity: 25)
32
33 # Current free capacity of consumers
34 C1 (capacity: 0, origin capacity: 52, score: 1)
35 C2 (capacity: 0, origin capacity: 61, score: 0)
36
37 # Unused consumers
```

Při běhu naší implementace GLS jsou do konzole vypisovány některé aktuální informace o běhu, například kolikátá iterace GLS algoritmu právě probíhá. Aplikace implementuje principy, které byly představeny v kapitole 6. Než přejdeme k experimentům, které budou prováděny právě na této aplikaci, tak si ještě představíme nástroj pro filtraci dat z ISOH.

7.3 Filtrace dat z ISOH

Pro účely experimentů by bylo vhodné zmapovat nějakou oblast a informace o místech, která zpracovávají odpad v této oblasti převést na instanci našeho problému. K tomuto účelu nám poslouží registr zpracovatelů odpadů ISOH (více o registru kapitole 4). Jelikož registr ISOH nabízí pouze XML soubor s daty bez jakékoliv dokumentace, tak musíme odhadnout, co které XML tagy znamenají. Hlavní tagy, které byly v souboru identifikovány, jsou znázorněny v tabulce 7.3.

Tag	Pravděpodobný účel
`EXP_PARAMS`	Informace o vytvoření souboru, například datum a čas a také o verzi ISOH, ze které byly data vytvořena.
`KATALOG`	Přehled jednotlivých typů odpadu, včetně jejich kódů.
`ZARIZ`	Jednotlivá zařízení, včetně GPS souřadnic.
`ZKATALOG`	Procesy/oblasti, ve kterých se jednotlivá zařízení angažují (např. skládkování, spalování, recyklace atd.).
`ZRZHISTORIEPROVOZU`	Historie provozu (dle toho se dá zjistit, zda je konkrétní zařízení ještě v provozu).
`ZRZPOVOLODP`	Povolení ke zpracování daného typu odpadu od do určitého data.
`ZRZPROVOZOVATEL`	Provozovatelé jednotlivých zařízení.
`ZRZROZSAH`	Spojení oblastí zpracování odpadu (tag `ZKATALOG`) s provozovateli.
`KRAJ`	Kódy, názvy a zkratky krajů.
`ZEME`	Kódy, názvy (i anglicky) států.

Tabulka 7.3: Odhadnutí hlavních XML tagů z ISOH

Aplikaci pro filtraci dat z ISOH je implementovaná v Javě, verze 17. Podobně jako u implementace GLS byla zvolena Java z důvodu její dobré znalosti autorem této práce. Při spuštění aplikace automaticky stáhne XML soubor s daty z registru ISOH. Z důvodu možností různých filtrací dat pro aplikaci nebylo implementováno žádné rozhraní. To, co aplikace vypíše, je na samotném programátorovi.

Ve třídě App je připraven objekt třídy Reader, díky kterému je možné z XML přečíst jednotlivá data, která jsou následně převedena do Java objektů. Například jednotlivá místa mohou být převedena do objektů třídy Place. V rámci čtení navíc dochází k provázání s dalšími entitami uloženými v tomto XML souboru. Takto přečtená data poté může programátor pomocí Java streamů filtrovat a nechat si je vypsat, případně přečíst pomocí debuggovacího nástroje.

Díky tomuto nástroji pro čtení dat z ISOH si můžeme vyfiltrovat určité typy

zařízení pro nějakou oblast a zjistit jejich počet. Zjištění můžeme následně použít pro nastavení parametrů do nástroje pro generování instancí (viz sekce 7.1) a nechat si vygenerovat reálné zadání pro náš problém, na kterém budeme zkoušet experimenty v následující kapitole 8.

Kapitola 8

Experimenty

V předchozí kapitole 7 jsme si krátce představili 3 aplikace, které využijeme v této kapitole pro účely experimentů. Jelikož nebyl v literatuře nalezen obdobný problém, který by se dal s naším problémem porovnávat, tak jsme se rozhodli, že budeme provádět experimenty jen na samotném algoritmu. Nebudeme tedy náš algoritmus s ničím dalším srovnávat. Nejprve si zadefinujeme a označíme instance našeho problému, které budeme využívat pro účely experimentů. Následně experimenty určíme nejvhodnější pořadí operátorů a podíl každého operátoru na výsledném řešení. Ukážeme, že určení parametru λ má velký vliv na výsledné řešení. Na závěr porovnáme výsledky pro různě velké flotily.

8.1 Instance

8.1.1 Reálné instance

Jak již bylo zmíněno, pro účely experimentů by bylo vhodné mít instance, které budou alespoň částečně vycházet z reálné situace určité oblasti. Tyto instance budeme nazývat jako reálné instance. Za tímto účelem byl stvořen nástroj pro filtraci dat z registru ISOH (viz sekce 7.3). První věcí, kterou musíme určit, je typ odpadu, pro který budeme hledat třídírny (producenty odpadu) a zpracovatele (konzumenty odpadu).

V registru ISOH se nachází 973 typů odpadů, z nichž každý typ má kód. Výběr si omezíme pouze na komunální odpady, protože u nich se dá předpokládat nejvíce třídíren a zpracovatelů. K dispozici je 30 typů komunálních odpadů. Podíváme se na 3 nejzákladnější – papír (kód 200101), plast (kód 200139) a sklo (kód 200102). U těchto 3 základních typů byla provedena celorepubliková analýza, jejíž výsledek je zanesen v tabulce 8.1.

Typ	Třídírny	Recyklace	Energie	Skládka
Papír	436	57	14	67
Plast	398	89	16	80
Sklo	264	26	3	82

Tabulka 8.1: Počet třídíren a zpracovatelů podle typů odpadu v ČR

V analýze byly zahrnuty činnosti jednotlivých třídíren a zpracovatelů odpadu. Činností, která jednotlivá zařízení vykonávají bylo u aktivních stacionárních zařízení identifikováno 78, každá činnost má svůj kód. Druhý sloupec tabulky 8.1 ukazuje zjištěný počet třídíren pro daný typ odpadu (činnost s kódem 3.4.0). Třetí sloupec ukazuje počet zařízení, která recyklují odpad (činnosti s kódy 5.2.0, 5.14.2 a 5.14.3), tedy v našem problému se může jednat o konzumenty se skórem 0. Čtvrtý sloupec energie představuje počet zařízení, která přeměňují daný typ komunálního odpadu na energii (činnosti s kódy 4.1.0, 4.1.1, 4.2.0, 4.3.0, 4.4.0 a 4.4.1), můžeme je tedy označit jako konzumenty se skórem 1. Poslední sloupec ukazuje zařízení, kde dochází ke skládkování daného typu odpadu (činnosti s kódy 10.1.0, 10.2.0, 10.3.0, 10.5.0, 8.1.0, 8.2.0 a 8.3.0), v našem případě tedy konzumenty se skórem 2.

Kódy činností se dělí do 2 skupin, podle starého zákona 185/2001 Sb. a nového 541/2020 Sb. Protože autor není dobře obeznámen se zákony o odpadovém hospodářství ČR a není to předmětem této práce, byly činností kódů určeny tak, aby vyhovovaly našemu problému. Není tedy vyloučené například to, že do některých z uvedených 4 skupin (třídírny, recyklace, energie a skládkování) můžou připadnou další kódy. Navíc jak již bylo zmíněno, v datech jsou uvedeny chyby, tudíž se musí přesnost dat brát s rezervou. Například ZEVO v pražských Malešicích má pouze kód 4.1.0 – *„Využití odpadu jako paliva nebo k výrobě energie“* podle zákona 185/2001 Sb. V datech je ale uvedeno, že přijímá komunální odpad. To by znamenalo, že by měl být i přítomen kód 4.1.1 – *„Energetické využití komunálních odpadů“* podle zákona 541/2020 Sb. Tento kód ale u ZEVA v Malešicích přítomen není. Některá zařízení jsou v registru také uvedena víckrát, pravděpodobně jich tedy ve skutečnosti bude méně.

Po identifikaci činností jednotlivých zařízení se musíme rozhodnout, který odpad využijeme pro konstrukci reálných instancí. Vzhledem k tomu, že sklo má poměrně malý počet míst, kde se přeměňuje na energii, tak tento typ odpadu pro vytvoření reálné instance nepoužijeme. Pravděpodobně to bude tím, že sklo se jen tak nedá v ZEVO spálit. Mezi plastem a papírem je rozdíl v počtech zařízení minimální. Nicméně na schůzce s panem Ing. Janem Svátkem, MPA (viz kapitola 4) jsme viděli, že papír má po třídění okamžitý odběr u zpracovatele, zatímco u plastu si tím nemusíme být úplně jistí. Z tohoto důvodu jsme se rozhodli stavět naši reálnou instanci na papírovém odpadu.

Máme tedy k dispozici celorepubliková data o počtech zařízení, které nějakým způsobem nakládají s papírovým odpadem. Celorepubliková data jako instance by byla pro experimenty příliš velká. Na vyřešení bychom pravděpodobně čekali příliš dlouho a nemohli bychom snadno ilustrovat výsledné řešení. Proto si určíme regiony/města, na základě kterých vygenerujeme instance.

Největší vygenerovanou reálnou instanci budeme stavět na zjištěných datech ze Středočeského kraje a Prahy. Zjištěné počty zařízení na tomto území jsou zaneseny do tabulky 8.2. Tato instance bude mít celkem 110 vrcholů (počet zařízení + depo). Pro další instance budeme volit kraje ČR. Nezvolíme ale všechny kraje, protože některé nemají zástupce všech typů zařízení. Například

Karlovarský kraj má pouze 15 třídíren a 2 zařízení na recyklaci, zatímco zástupce ze skupiny přeměny na energii nebo skládkování chybí.

Další instance jsou odstupňované sestupně podle počtu zařízení (neboli výsledných vrcholů instance). Připravili jsme data pro dalších 6 instancí, na základě krajů – Jihočeský kraj (tabulka 8.3), Jihomoravský kraj (tabulka 8.4), Moravskoslezský kraj (tabulka 8.5), Plzeňský kraj (tabulka 8.6), Kraj Vysočina (tabulka 8.7) a Liberecký kraj (tabulka 8.8).

Poslední nejmenší reálnou instanci postavíme na základě města. Města zpravidla pochopitelně neobsahují zástupce z kategorie skládkování, nicméně našli jsme výjimku v podobě Ostravy (tabulka 8.9). Ostrava tedy reprezentuje naši nejmenší reálnou instanci.

	Počet
Třídírna	84
Recyklace	10
Přeměna na energii	3
Skládkování	12
Celkem zpracovatelů	25
Počet zařízení	**109**

Tabulka 8.2: Počty zařízení Středočeský kraj a Praha

	Počet
Třídírna	62
Recyklace	3
Přeměna na energii	1
Skládkování	14
Celkem zpracovatelů	18
Počet zařízení	**80**

Tabulka 8.3: Počty zařízení Jihočeský kraj

	Počet
Třídírna	44
Recyklace	3
Přeměna na energii	3
Skládkování	2
Celkem zpracovatelů	8
Počet zařízení	**52**

Tabulka 8.4: Počty zařízení Jihomoravský kraj

	Počet
Třídírna	33
Recyklace	7
Přeměna na energii	3
Skládkování	7
Celkem zpracovatelů	17
Počet zařízení	**50**

Tabulka 8.5: Počty zařízení Moravskoslezský kraj

	Počet
Třídírna	27
Recyklace	4
Přeměna na energii	2
Skládkování	5
Celkem zpracovatelů	11
Počet zařízení	**38**

Tabulka 8.6: Počty zařízení Plzeňský kraj

	Počet
Třídírna	30
Recyklace	5
Přeměna na energii	1
Skládkování	1
Celkem zpracovatelů	7
Počet zařízení	**37**

Tabulka 8.7: Počty zařízení Kraj Vysočina

	Počet
Třídírna	14
Recyklace	5
Přeměna na energii	1
Skládkování	2
Celkem zpracovatelů	8
Počet zařízení	**22**

Tabulka 8.8: Počty zařízení Liberecký kraj

	Počet
Třídírna	9
Recyklace	2
Přeměna na energii	1
Skládkování	1
Celkem zpracovatelů	4
Počet zařízení	**13**

Tabulka 8.9: Počty zařízení Ostrava

Pro reálné instance máme tedy připravené počty jednotlivých typů vrcholů. U konzumentů máme k dispozici i jejich skóre. Souřadnice vrcholů necháme vygenerovat generátorem řešení, včetně velikosti 2D prostoru pro generování (viz sekce 7.1). Implementace přepočtu souřadnic do 2D osy by byla náročná a nepřinesla by nám žádné benefity, protože pro experimenty nebude rozdíl mezi přepočtem a náhodným rozmístěním vrcholů.

Další počty, které musíme určit, je velikost flotil vozidel pro každou reálnou instanci. Nemáme žádná veřejně dostupná data, ze kterých bychom mohli čerpat. Musíme tedy sami odhadnou, jak velké flotily by mohla každá oblast mít. Odhad budeme realizovat na základě počtu producentů v dané oblasti, protože v našem problému obsluhuje jednoho producenta jen jedno vozidlo (viz kapitola 5). Nebudeme ale určovat pro každou oblast velikost flotil tak, aby každé vozidlo mělo přiřazeno právě jednoho producenta. Zvolíme číslo menší, abychom měli zaručeno, že některá vozidla budou muset obsloužit více producentů než jen jednoho. Velikost flotily pro každou reálnou instanci tedy určíme jako $\lfloor 0,6 \cdot |V_p| \rfloor$ (60 % počtu producentů zaokrouhlených dolů). Výsledné velikosti flotil jsou zaneseny do tabulky 8.10.

Oblast	**Počet vozidel**
Středočeský kraj a Praha	50
Jihočeský kraj	37
Jihomoravský kraj	26
Moravskoslezský kraj	19
Plzeňský kraj	16
Kraj Vysočina	18
Liberecký kraj	8
Ostrava	5

Tabulka 8.10: Velikost flotil pro oblasti

Co se týče množství odpadu v producentech, tak bohužel nemáme veřejně dostupná data, která bychom pro toto mohli použít. Máme celorepubliková data o množství vyprodukovaného komunální odpadu, stejně tak podle krajů. Nevíme ale, jakou složku z toho tvoří papír. Jednotlivá množství a kapacity konzumentů a vozidel tedy necháme vygenerovat námi vytvořeným generátorem instancí (viz část 7.1). Do generátoru můžeme předat interval, ve kterém

má hledat množství odpadu v producentech. Na základě tohoto intervalu poté můžeme určit velikost dalších parametrů, které do generátoru předáme. Jelikož třídírny odpadu zpravidla zajišťují obce (viz kapitola 4), dá se předpokládat, že některé třídírny budou nabízet velká množství odpadu, zatímco jiné malá (z důvodu velkých rozdílů v počtech obyvatel obcí). Interval, ze kterého bude generátor náhodně vybírat množství odpadu pro producenta, tedy zvolíme od 10 do 100 včetně, a to pro všechny oblasti.

Abychom otestovali lépe schopnosti algoritmu, tak budeme chtít mít i celkem rozdílné kapacity u vozidel (malá a velká). Zároveň by bylo dobré, aby se vozidlo mockrát nevracelo zpátky pro opětovné naložení stejného producenta. V reálném světě by se pravděpodobně například desetkrát vozidlo zpátky nevracelo. Máme-li tedy množství odpadu v producentech v rozmezí $\langle 10, 100 \rangle$, potom zvolíme kapacity vozidel v rozmezí $\langle 10, 80 \rangle$.

Posledním parametrem, který pro generátor určíme, je maximální odchylka u kapacit konzumentů (viz sekce 7.1). Aby byla instance přípustná, tak se množství odpadu, které je potřeba odvézt z konzumentů rovnoměrně rozloží mezi konzumenty, přičemž ke každému se ještě přičte odchylka (více v sekci 7.1). Odchylka se vybírá z intervalu od 0 do parametru, který právě určíme. Chceme získat různou škálu kapacit konzumentů, proto tento parametr určíme jako 100 jednotek.

Parametry pro určení kapacit vozidel, konzumentů a množství odpadu v producentech zvolíme při generování stejné pro všechny reálné instance. Stále se ovšem jedná o náhodný výběr, pokud by se nám některé parametry instance nelíbily, můžeme je ručně pozměnit ve výsledném souboru. Souhrn argumentů, které sdílejí jednotlivé reálné instance při generování naleznete v tabulce 8.11. Abychom od sebe reálné instance v budoucnu jednoduše oddělili, tak jim přiřadíme označení, viz tabulka 8.12.

Argument	**Hodnota**
Množství odpadu v producentech $\langle$`-wql`, `-wqu`$\rangle$	
`-wql`	10
`-wqu`	100
Kapacity vozidel $\langle$`-vcl`, `-vcu`$\rangle$	
`-vcl`	10
`-vcu`	80
Maximální odchylka konzumenta	
`-ccd`	100

Tabulka 8.11: Souhrn argumentů generátoru pro reálné instance

Oblast instance	Označení
Středočeský kraj a Praha	SP
Jihočeský kraj	J
Jihomoravský kraj	B
Moravskoslezský kraj	M
Plzeňský kraj	P
Kraj Vysočina	V
Liberecký kraj	L
Ostrava	O

Tabulka 8.12: Označení reálných instancí

8.1.2 Abstraktní instance

Osm reálných instancí, které jsme si představili v předchozí části 8.1.1, není pro účely experimentů dostatečných. Reálné instance tedy doplníme nějakými dalšími, u kterých můžeme zvolit i nějaké extrémní nereálné případy, např. jednoho producenta na několik konzumentů. Takovéto instance nazveme abstraktními a jsou znázorněny v tabulce 8.13 a 8.14 a označeny kódy. V tabulce 8.13 můžeme vidět počty vrcholů a velikost flotily, zatímco v tabulce 8.14 jsou zaneseny argumenty, které byly zvoleny při generování těchto instancí (o argumentech se dočtete v části 7.1). Celkem máme tedy připravených 20 instancí pro experimenty.

Označení	$\|V_p\|$	$\|V_c\|$	$\|F\|$	$\vec{s}=0$	$\vec{s}=1$	$\vec{s}=2$
AB1	1	10	1	3	3	4
AB2	1	10	1	1	8	1
AB3	10	1	2	0	1	0
AB4	3	2	2	1	1	0
AB5	5	3	3	1	1	1
AB6	9	9	9	1	0	8
AB7	9	9	1	6	3	0
AB8	61	15	20	5	15	5
AB9	50	3	25	1	1	1
AB10	32	10	5	4	3	3
AB11	16	1	16	0	0	1
AB12	14	20	10	15	0	5

Tabulka 8.13: Přehled vrcholů abstraktních instancí

Označení	⟨-wql, -wqu⟩	⟨-vcl, -vcu⟩	-ccd
AB1	⟨100, 200⟩	⟨200, 200⟩	0
AB2	⟨100, 200⟩	⟨50, 150⟩	100
AB3	⟨10, 50⟩	⟨50, 50⟩	0
AB4	⟨10, 100⟩	⟨50, 80⟩	50
AB5	⟨10, 100⟩	⟨50, 80⟩	50
AB6	⟨10, 100⟩	⟨50, 80⟩	100
AB7	⟨10, 20⟩	⟨20, 20⟩	20
AB8	⟨10, 100⟩	⟨10, 100⟩	100
AB9	⟨20, 50⟩	⟨20, 50⟩	10
AB10	⟨40, 50⟩	⟨20, 30⟩	40
AB11	⟨10, 30⟩	⟨20, 30⟩	0
AB12	⟨10, 100⟩	⟨10, 80⟩	0

Tabulka 8.14: Přehled argumentů abstraktních instancí pro generování

8.2 Parametry stroje pro měření

V předchozí části jsme vygenerovali instance a nyní se přesouváme k samotným experimentům. Experimenty zaznamenané v následujících částech byly prováděny na fyzickém stroji, jehož parametry jsou uvedeny v tabulce 8.15.

Počítač	Apple MacBook Pro 16″ 2019
Procesor	2,6 GHz 6jádrový Intel Core i7
Paměť	16 GB 2667 MHz DDR4

Tabulka 8.15: Parametry stroje pro měření

8.3 Pořadí operátorů

Máme 3 hlavní operátory – přesun producenta, výměna konzumentů a výměna používaných konzumentů s nepoužívanými (viz kapitola 6.3). Každý z těchto hlavních operátorů si označíme písmenem pro přehlednější orientaci v tabulkách:

- **R** – přesun producenta,
- **E** – výměna konzumentů,
- **U** – výměna používaných konzumentů s nepoužívanými.

Experimenty na určení nejvhodnějšího pořadí budeme provádět tak, že zkusíme každé pořadí operátorů. Máme tedy 6 případů pro každou instanci představenou v části 8.1. Budeme měřit ceny řešení a dobu běhu. Vzhledem k dlouhé době běhu algoritmu, jsme se rozhodli náš vzorek omezit jen na

těchto 20 instancí, ačkoliv by bylo lepší mít instancí více. Ze stejného důvodu jsme nastavili počet iterací GLS pro každou instanci na 500, větší počet se ukázal příliš časově náročný. Při měření nastavíme ostatní parametry fixně, tedy $\alpha = 1, \beta = 2, \lambda = 3$ a parametr γ necháme nastavit algoritmus jako aritmetický průměr všech vzdáleností hran pro každou instanci.

8.3.1 Vyhodnocení

Výsledky měření cen řešení jsou zaneseny do tabulky 8.16 a korespondující doby běhu k zisku řešení do tabulky 8.17. Kromě výsledků měření jsou v tabulkách také zaneseny průměry a odhady odchylek. Budeme-li dále uvádět průměry, budeme tím myslet průměry aritmetické. Obdobně budeme-li uvádět odchylky, myslíme tím odhady směrodatných odchylek.

Jelikož je doba běhu výrazně závislá na ostatních okolnostech, jako je vytížení a výkon počítače, tak by bylo vhodnější provést měření více a výsledky zprůměrovat. Vzhledem k náročnosti a době běhu velkých instancí (několik minut) jsme se rozhodli zanedbat měření času na více bězích a zaokrouhlit výsledky na vteřiny. Uvedení časových výsledků ve vteřinách minimalizuje nepřesnosti, které by byly zmírněny při měření přes více běhů. Navíc nás stejně zajímá přibližná časová hodnota, takže pro vyhodnocení času nepotřebujeme znát hodnotu přesnou na milisekundy.

V tabulce 8.16 jsou tučně znázorněny nejlepší dosažené výsledky. Z výsledků vyplývá, že nejvhodnější pořadí operátorů je ERU, tedy nejprve výměna používaných konzumentů, následovanou přesuny producentů a končící výměnou používaných konzumentů za nepoužívané. Toto pořadí získalo největší počet nejlepších nalezených řešení, průměrně dosahuje nejnižší cenové hodnoty a má i nejnižší odhad odchylky. U odchylek pro jednotlivé instance můžeme v závorce vidět i variační koeficient, tedy procentuální vyjádření odchylky k průměru. Nejvyšších procentuálních hodnot odchylek dosahují instance AB5, V a P, avšak žádná souvislost mezi těmito instancemi oproti jiným zřejmá není.

Tabulka 8.17 ukazuje, že pochopitelně největší instance jako SP a J dosahují mezi měřenými instancemi nejdelších běhů. Porovnáme-li jednotlivá pořadí operátorů a jejich časy, tak nějaké významné rozdíly vidět nejsou. Z řady trochu vyčnívají pořadí UER a RUE, které mají nejvyšší průměry a i nejvyšší odchylky. Pravděpodobně to je způsobeno největší reálnou instancí SP, kdy těmto dvěma pořadím trvá získat řešení déle než ostatním. Nejspíše se tato pořadí u instance SP vydávají časově náročnější cestou. Výrazně nejvyšší rozdíly v časech jsou u instance AB12 (34,6 % od průměru). Za povšimnutí stojí, že u této instance trvají nejdéle běhy, kde začínáme operátorem R, nebo operátorem U, po kterém následuje R. Pravděpodobně to znamená, že pokud je tento operátor první, tak pracuje na počátku v této instanci déle než ostatní. Zajímavé je i to, že v těchto déle trvajících pořadích dosáhneme horšího, ale stejného výsledku než u pořadí ostatních. Nejspíše to znamená, že k těmto výsledkům vedla stejná cesta, která byla časově náročnější. Dále mají také vyšší odchylky instance O, AB4, AB5, M a AB1, žádná spojitost mezi těmito instancemi ale není patrná. Můžeme si všimnout, že instance

s více producenty, které mají přibližně stejný počet vrcholů jako instance s méně producenty, mají průběh delší. Například instance AB2 oproti AB3, nebo instance B oproti M.

Instance	REU	ERU	UER	EUR	RUE	URE	Průměr	Odchylka
O	2077	**1928**	2182	2099	**1928**	2099	2052,17	102,66 (5,0 %)
AB3	**1872**	**1872**	**1872**	**1872**	**1872**	**1872**	1872,00	0,00 (0,0 %)
AB4	349	**325**	**325**	**325**	349	349	337,00	13,15 (3,9 %)
AB5	**532**	629	629	629	**532**	**532**	580,50	53,13 (9,2 %)
AB2	**540**	**540**	**540**	**540**	**540**	**540**	540,00	0,00 (0,0 %)
AB12	2929	**2643**	**2643**	**2643**	2929	2929	2786,00	156,65 (5,6 %)
V	4257	**3676**	**3676**	**3676**	4257	4257	3966,50	318,23 (8,0 %)
J	18331	**17514**	**17514**	**17514**	18331	18331	17922,50	447,49 (2,5 %)
M	**6462**	6762	6540	7128	6526	6663	6680,17	244,36 (3,7 %)
P	5141	4332	5136	**4315**	4611	5087	4770,33	399,06 (8,4 %)
B	8494	**7639**	**7639**	**7639**	8494	8494	8066,50	468,30 (5,8 %)
L	**1715**	1849	1849	1849	**1715**	**1715**	1782,00	73,39 (4,1 %)
AB9	18007	**17841**	**17841**	**17841**	18007	18007	17924,00	90,92 (0,5 %)
AB7	**1385**	**1385**	**1385**	**1385**	**1385**	**1385**	1385,00	0,00 (0,0 %)
AB1	**1103**	**1103**	**1103**	**1103**	**1103**	**1103**	1103,00	0,00 (0,0 %)
AB6	1528	**1413**	1514	1429	1536	1551	1495,17	58,90 (3,9 %)
AB8	**13915**	15029	15029	15029	**13915**	**13915**	14472,00	610,16 (4,2 %)
SP	22976	**20965**	24246	21934	21665	21470	22209,33	1200,24 (5,4 %)
AB11	**2875**	**2875**	**2875**	**2875**	**2875**	**2875**	2875,00	0,00 (0,0 %)
AB10	8463	**8039**	**8039**	**8039**	8463	8463	8251,00	232,23 (2,8 %)
Průměr	6147,55	**5917,95**	6128,85	5993,20	6051,65	6081,85		
Odchylka	6843,38	**6580,67**	6982,40	6696,87	6687,65	6653,79		
Počet nejlepších	9	**15**	12	13	9	8		

Tabulka 8.16: Výsledky měření cen různého pořadí operátorů

Instance	REU	ERU	UER	EUR	RUE	URE	Průměr	Odchylka
O	6,3	5,1	5,7	7,2	5,6	7,4	6,23	0,91 (14,7 %)
AB3	0,6	0,6	0,8	0,6	0,7	0,6	0,65	0,06 (9,9 %)
AB4	0,2	0,2	0,2	0,1	0,2	0,1	0,17	0,03 (15,9 %)
AB5	0,4	0,3	0,4	0,3	0,4	0,3	0,37	0,05 (14,4 %)
AB2	0,2	0,2	0,2	0,2	0,2	0,2	0,19	0,02 (12,5 %)
AB12	22,0	12,3	11,5	11,7	22,6	23,6	17,27	5,98 (34,6 %)
V	21,1	20,4	20,4	20,4	22,8	22,2	21,19	1,06 (5,0 %)
J	247,6	241,9	259,9	241,4	233,9	231,2	242,64	10,30 (4,2 %)
M	40,9	34,3	41,7	42,0	31,8	44,3	39,18	4,94 (12,6 %)
P	21,4	25,5	22,7	20,7	21,3	22,7	22,37	1,72 (7,7 %)
B	71,5	86,7	88,6	91,5	73,3	72,7	80,72	9,15 (11,3 %)
L	4,8	4,6	4,8	4,5	5,0	4,8	4,77	0,17 (3,7 %)
AB9	40,3	46,5	47,3	47,7	40,2	40,5	43,76	3,76 (8,6 %)
AB7	2,4	2,5	2,5	2,3	2,5	2,3	2,43	0,09 (3,7 %)
AB1	0,1	0,1	0,1	0,1	0,1	0,1	0,09	0,01 (16,7 %)
AB6	2,5	2,4	2,2	2,2	2,7	1,8	2,29	0,30 (12,9 %)
AB8	138,4	146,4	151,8	148,3	143,4	156,2	147,42	6,26 (4,2 %)
SP	404,3	385,7	418,2	364,4	435,0	319,5	387,85	41,58 (10,7 %)
AB11	1,9	1,9	2,1	2,0	1,9	1,9	1,95	0,08 (4,0 %)
AB10	115,6	98,6	101,5	104,2	117,4	115,6	108,82	8,32 (7,6 %)
Průměr	57,13	55,80	59,13	55,59	58,05	53,40		
Odchylka	103,08	99,55	107,32	96,13	107,71	87,90		

Tabulka 8.17: Výsledky měření času různého pořadí operátorů (ve vteřinách)

8.4 Přínos operátorů

V těchto experimentech měříme přínos jednotlivých hlavních operátorů. Označení operátorů bude shodné jako v experimentech na pořadí operátorů (viz sekce 8.3). Měření bude probíhat tak, že postupně každý operátor vypneme a ostatní operátory necháme zapnuté. Algoritmus pustíme na všech instancích představených v sekci 8.1. Pořadí zbylých dvou operátorů zachováme takové, jaké jsme jako nejlepší určili v sekci 8.3, tedy ERU. Uvedeme i výsledky pro pořadí bez vypnutého jakéhokoliv operátoru, bude se jednat o stejná data, která jsme naměřili u ERU pořadí. Při měření nastavíme ostatní parametry fixně, tedy $\alpha = 1, \beta = 2, \lambda = 3$, počet iterací 500 a parametr γ necháme nastavit algoritmus jako aritmetický průměr všech vzdáleností hran pro každou instanci.

8.4.1 Vyhodnocení

Výsledky cen jsou znázorněny v tabulce 8.18. V posledních třech sloupcích je uveden procentuální rozdíl oproti zapnutým všem operátorům. Můžeme si všimnout, že pro některé instance jsme získali lepší řešení s nějakým vypnutým operátorem, než když byly všechny zapnuté. To se týče jen měření, kde jsme vypnuli operátor U nebo E. Při vypnutém operátoru R jsme lepší řešení, než se všemi zapnutými nezískali v žádném případě. Jak z měření totiž vyplývá, tak nejsilnějším operátorem je operátor R, po jehož vypnutí jsme dosáhli zdaleka nejhorších řešení. Operátory U a E se zdají být na přibližně podobné úrovni, ačkoliv vypnutí operátoru E přineslo o trochu horší ceny, než vypnutí operátoru U. Pokud se ale podíváme na procentuální rozdíly oproti zapnutým všem operátorům, tak zde vychází operátor U lépe, než operátor E. Tyto skutečnosti se promítají i do dob běhu. V tabulce 8.19 si můžeme všimnout, že pokud vypneme operátor R, tak algoritmus doběhne zdaleka nejdříve. Jedná se tedy nejenom o nejlepší operátor, co se ceny týče, ale zároveň i o nejnáročnější operátor, co se času týče. Stejná situace je i mezi operátorem E a U, kdy lehce cenově lepší operátor E je i zároveň druhým nejnáročnějším operátorem podle času.

Instance	Všechny operátory	Bez U	Bez R	Bez E	Rozdíl U	Rozdíl R	Rozdíl E
O	**1928**	2167	2683	1980	12,4 %	39,2 %	2,7 %
AB3	**1872**	**1872**	2032	**1872**	0,0 %	8,5 %	0,0 %
AB4	**325**	**325**	386	344	0,0 %	18,8 %	5,8 %
AB5	**629**	**629**	727	666	0,0 %	15,6 %	5,9 %
AB2	**540**	738	**540**	**540**	36,7 %	0,0 %	0,0 %
AB12	2643	2643	4511	**2640**	0,0 %	70,7 %	-0,1 %
V	**3676**	**3676**	9213	3829	0,0 %	150,6 %	4,2 %
J	**17514**	**17514**	41794	17883	0,0 %	138,6 %	2,1 %
M	6762	**6663**	11421	6874	-1,5 %	68,9 %	1,7 %
P	**4332**	4370	9449	5296	0,9 %	118,1 %	22,3 %
B	**7639**	**7639**	18577	9027	0,0 %	143,2 %	18,2 %
L	1849	1849	3327	**1785**	0,0 %	79,9 %	-3,5 %
AB9	**17841**	**17841**	30409	17856	0,0 %	70,4 %	0,1 %
AB7	**1385**	2033	**1385**	**1385**	46,8 %	0,0 %	0,0 %
AB1	**1103**	**1103**	**1103**	**1103**	0,0 %	0,0 %	0,0 %
AB6	**1413**	1539	1865	1526	8,9 %	32,0 %	8,0 %
AB8	15029	15029	28366	**14279**	0,0 %	88,7 %	-5,0 %
SP	**20965**	22632	57666	23251	8,0 %	175,1 %	10,9 %
AB11	**2875**	**2875**	4965	**2875**	0,0 %	72,7 %	0,0 %
AB10	8039	8039	11654	**7853**	0,0 %	45,0 %	-2,3 %
Průměr	**5917,95**	6058,80	12103,65	6143,20	5,61 %	66,80 %	3,54 %
Odchylka	**6580,67**	6745,60	15781,37	6858,11	12,99 %	54,90 %	6,88 %
Počet nejlepších	**15**	10	3	9	-5	-12	-6

Tabulka 8.18: Výsledky měření cen s vynecháním operátoru

Instance	Všechny operátory	Bez U	Bez R	Bez E
O	5,1	6,8	2,0	5,3
AB3	0,6	0,7	0,1	0,6
AB4	0,2	0,2	0,1	0,2
AB5	0,3	0,4	0,1	0,4
AB2	0,2	0,1	0,2	0,2
AB12	12,3	12,0	2,4	6,2
V	20,4	20,6	3,8	16,8
J	241,9	235,0	33,0	153,5
M	34,3	35,9	8,9	29,8
P	25,5	24,1	4,2	17,9
B	86,7	92,7	13,3	56,1
L	4,6	4,9	2,1	3,8
AB9	46,5	46,5	3,2	40,3
AB7	2,5	0,2	2,5	2,5
AB1	0,1	0,1	0,1	0,1
AB6	2,4	2,7	0,5	2,2
AB8	146,4	149,2	40,8	93,9
SP	385,7	350,7	96,9	283,5
AB11	1,9	1,9	0,5	1,8
AB10	98,6	100,9	25,9	58,9
Průměr	55,80	54,27	12,03	38,71
Odchylka	99,55	93,20	23,20	69,83

Tabulka 8.19: Výsledky měření času s vynecháním operátoru (ve vteřinách)

8.5 Vliv parametru λ

V této části experimentů se budeme zabývat vlivem parametru λ na řešení. Jedná se o tzv. sílu penalizací, má-li λ příliš nízkou hodnotu, tak může trvat několik iterací, než algoritmus unikne z lokálního minima, naopak při vyšších utíká z lokálních řešení algoritmus rychleji (může utéct i dříve, než narazí na lokální minimum). Více o tomto parametru je v části 6.1. Experimenty budeme opět provádět na všech instancích představených v části 8.1. Hodnoty λ určíme exponenciální řadou se základem 2. To kvůli tomu, abychom se rychle dostali až k vysoké hodnotě, přičemž nejvyšší naměřenou hodnotou bude $\lambda = 1024$. Ostatní parametry nastavíme fixně, tedy $\alpha = 1, \beta = 2$, počet iterací 500 a parametr γ necháme nastavit algoritmus jako aritmetický průměr všech vzdáleností hran pro každou instanci. Pořadí operátorů bude opět ERU (tedy nejlepší zjištěné).

8.5.1 Vyhodnocení

Stejně jako u předchozích experimentů jsme kromě cen (tabulka 8.20) měřili i časy (tabulka 8.21), přičemž jednotlivé hodnoty λ jsou uvedeny ve sloupcích. V tabulce 8.20 je patrné, že nejvyšší hodnota λ dosáhla zároveň nejlepšího výsledku. Sestupný trend cen se zvyšující se hodnotou λ je vidět u největších instancí SP, J a AB8. To je zapříčiněno pravděpodobně tím, že tyto velké instance s velkými počty vrcholů mají ohromně velké prostory s řešeními. Tím pádem je lepší rychle utíkat z lokálních oblastí a přecházet do dalších. Uváznout v lokálním minimu několik iterací je pro tyto instance velmi drahé. Sestupné tendence s rostoucí hodnotou λ jsou ale vidět i u některých menších instancí. Závěrem tedy můžeme dodat, že podle našich měření se na našem problému vyplatí vysoká hodnota λ, protože máme velké prostory řešení a s nižší hodnotou bychom se k některým typům řešení ani nemuseli dostat.

Podle měření roste s vyšší hodnotou λ i doba běhu algoritmu. Jak můžeme vidět v tabulce 8.21, tak rozdíl mezi $\lambda = 1$ a $\lambda = 1024$ je, co se průměru týče, přibližně 80 vteřin. S rostoucím parametrem λ roste i odchylka, avšak zde to znamená kladnou odchylku, protože mezi instancemi máme i velmi malé instance, které i s vysokým λ parametrem doběhnou za méně jak vteřinu. Rozdíl se projeví až u větších instancí a to velmi výrazně. Například největší instance SP doběhla s $\lambda = 1$ za přibližně 347 vteřin, zatímco s $\lambda = 1024$ za 1109 vteřin. Tento rozdíl bude pravděpodobně způsoben tím, že pokud se algoritmus dostane do nové části prostoru řešení (díky vyšší hodnotě λ), tak nalezne více možností, jak dané řešení vylepšit. Pokud máme nižší hodnoty λ, tak algoritmus často zůstává v lokálních minimech, a tím pádem nemá příliš možností, jak řešení vylepšit. Zamítnutí možného řešení často probíhá na počátcích operátorů (např. zamítnutí z důvodu kapacitních omezení). Uvázne-li algoritmus v lokálním minimu, tak operátory budou nalézat více nepřípustných řešení, které zamítnou ještě předtím, než začnou s časově náročnými operacemi, jakými jsou například modifikace tras.

Instance\λ	1	2	4	8	16	32	64	128	256	512	1024
O	2075	2077	2077	1714	1990	2009	**1669**	1702	1684	1871	1804
AB3	1900	1872	1872	1872	1872	1872	1872	1872	**1868**	**1868**	**1868**
AB4	**325**	**325**	**325**	**325**	**325**	**325**	**325**	**325**	**325**	**325**	**325**
AB5	678	678	555	678	678	**418**	456	443	573	675	675
AB2	**540**	**540**	**540**	**540**	**540**	**540**	**540**	**540**	**540**	**540**	**540**
AB12	3000	2930	2883	2851	2579	2542	2447	2573	**2206**	2363	2378
V	3627	3866	3648	3598	3862	3562	3422	3857	**3225**	3473	3556
J	18771	19336	18310	17383	16405	17610	16762	16598	16654	16616	**15752**
M	6780	6660	6744	6130	5952	5366	5565	5889	5879	**4593**	5529
P	4440	4353	4204	4355	**3910**	4383	3945	4038	4461	3941	4129
B	9528	8349	9285	8337	7489	7884	6958	7086	6928	7671	**6387**
L	2048	1928	1613	2021	1913	1760	**1473**	1599	1792	1597	1547
AB9	17884	18282	18024	17359	**15956**	17258	16606	16248	17284	16678	15963
AB7	1454	1364	1367	1364	**1310**	1355	1361	1319	1334	1358	1414
AB1	**1103**	**1103**	**1103**	**1103**	**1103**	**1103**	**1103**	**1103**	**1103**	**1103**	**1103**
AB6	1551	1434	1557	1397	1385	1510	1530	1340	1539	**1157**	**1157**
AB8	15613	15353	14412	14013	12735	13557	14006	11997	**11457**	11982	11852
SP	21712	21554	20067	19669	20390	19183	19444	18690	18765	18974	**17585**
AB11	2909	2909	2875	2875	2853	2853	2853	2847	2847	2841	**2839**
AB10	9243	9243	8204	8129	7481	7461	6915	6823	6200	6508	**6169**
Průměr	6259,05	6207,80	5983,25	5785,65	5536,40	5627,55	5462,60	5344,45	5333,20	5306,70	**5128,60**
Odchylka	6863,59	6903,96	6560,32	6304,31	6060,74	6223,91	6144,04	5863,04	5940,58	5951,30	**5591,19**
Počet nejlepších	3	3	3	3	6	4	5	3	7	6	**10**

Tabulka 8.20: Výsledky měření cen s různým parametrem λ

Instance\λ	1	2	4	8	16	32	64	128	256	512	1024
O	6,1	6,1	6,3	6,5	6,8	5,9	3,3	3,6	4,5	3,6	2,9
AB3	0,6	0,6	0,6	0,7	0,7	0,9	0,9	0,9	0,8	0,8	0,8
AB4	0,2	0,2	0,2	0,1	0,2	0,2	0,1	0,1	0,1	0,1	0,1
AB5	0,4	0,4	0,4	0,5	0,4	0,4	0,3	0,3	0,4	0,4	0,3
AB2	0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,3	0,2	0,2	0,2
AB12	10,7	12,4	18,6	25,7	23,4	34,9	30,1	33,2	27,8	52,9	40,3
V	19,7	19,4	20,5	22,8	28,0	32,2	34,7	41,0	43,8	44,4	49,3
J	231,9	256,6	254,7	304,3	265,9	418,4	368,2	426,3	506,3	451,5	451,3
M	33,3	31,2	36,3	41,8	43,3	57,5	61,8	72,1	67,9	65,5	82,6
P	21,0	22,1	24,2	26,2	33,4	41,1	52,5	49,6	50,5	51,4	64,3
B	64,2	64,1	77,8	83,3	94,8	124,5	116,6	141,5	144,8	160,1	215,9
L	4,8	5,0	4,5	5,6	6,0	6,9	7,5	9,0	11,9	8,3	10,6
AB9	43,8	40,5	47,6	51,8	63,9	58,8	88,5	77,6	74,2	91,6	127,8
AB7	2,2	2,4	2,2	2,7	3,4	3,9	4,1	4,6	4,3	4,3	4,3
AB1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
AB6	2,8	2,8	3,0	3,4	3,4	4,4	5,9	4,9	4,7	3,3	3,1
AB8	133,9	133,3	136,8	169,0	199,7	240,5	277,7	314,2	297,5	304,1	374,7
SP	347,3	365,4	415,8	446,2	505,0	543,2	694,0	671,3	861,8	723,2	1109,0
AB11	1,9	1,8	1,9	2,2	2,3	2,5	2,8	2,7	3,0	3,0	2,9
AB10	109,9	97,0	100,3	138,9	166,9	161,5	148,8	145,9	156,1	198,7	171,9
Průměr	51,75	53,08	57,59	66,60	72,40	86,90	94,90	99,97	113,04	108,37	135,61
Odchylka	91,49	96,79	105,43	117,90	126,95	150,33	172,68	176,09	216,79	187,76	262,95

Tabulka 8.21: Výsledky měření času s různým parametrem λ (ve vteřinách)

8.6 Velikost flotily

V této části experimentů provedeme měření za účelem určení vlivu velikosti flotily na výsledné řešení. Při měření nastavíme parametry opět fixně, tedy $\alpha = 1$, $\beta = 2$ a γ necháme nastavit algoritmus jako aritmetický průměr všech vzdáleností hran pro každou instancí. Pořadí operátorů bude opět ERU (tedy nejlepší zjištěné). Stejně tak nám vyšlo, že s $\lambda = 1024$ získáme nejlepší výsledky, tak při těchto měřeních necháme parametr λ na této hodnotě.

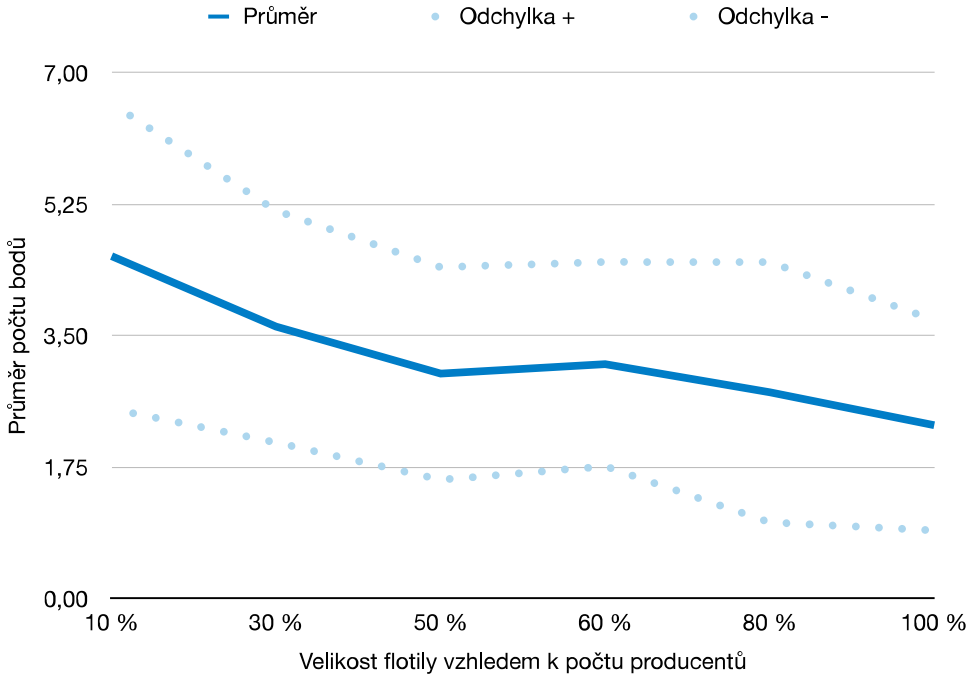
Jelikož jednoho producenta může obsluhovat pouze jedno vozidlo, tak velikost flotily budeme určovat procenty z počtu producentů v dané instanci. Maximální velikost flotily bude tedy odpovídat počtu producentů (100 %). Budeme měřit 10 %, 30 %, 50 %, 60 %, 80 % a 100 % počtu producentů. Vzhledem k tomu, že některé instance, které jsme představili, mají málo producentů (některé jen jednoho), tak měření neprovedeme na všech instancích. Vyřadíme instance AB1 (1 producent), AB2 (1 producent), AB4 (3 producenti) a AB5 (5 producentů), protože u nich by některá měřená procenta měla stejné počty producentů. Experiment tedy provedeme na zbývajících 16 instancích. Zvolíme-li u reálných instancí velikost flotily jako 60 % počtu producentů, získáme původní instanci, protože při vytváření těchto instancí jsme zvolili velikost flotily právě jako 60 % počtu producentů (viz sekce 8.1). Při vytváření variant, které mají menší flotilu než původní počet vozidel, byla vozidla postupně odebírána. U instancí s více jak původním počtem vozidel byla přidána vozidla nová. Instance s větší flotilou obsahuje vždy stejná vozidla jako instance s nižší a přidává další nová. Při generování nových vozidel je rozmezí pro určení kapacity vozidel stejné, jako bylo při generování původní instance.

Na výsledcích nemůžeme porovnávat ceny řešení, protože ceny řešení jsou závislé na každé instanci zvlášť. Pro každou instanci porovnáme získané ceny pro různě velké flotily a ohodnotíme je body od 1 do 6 (měříme 6 velikostí flotil). 1 bod získá nejlepší výsledek z hlediska ceny, přičemž u následujícího horšího výsledku přičteme 1 bod. Budeme-li mít u některých velikostí flotil stejné ceny řešení, tak bude i bodováno stejně. Nejvyšší možný počet bodů je 6, a to v případě, kdy pro veškeré velikosti flotil získáme odlišné ceny řešení. Takto získané výsledky poté můžeme promítnout do tabulky a spočítat sumy, průměry, odchylky a výsledek vyhodnotit napříč jednotlivými instancemi.

8.6.1 Vyhodnocení

Výsledek experimentu je zanesen do tabulky 8.22. Z výsledků jsme sestavili graf (obrázek 8.1), ve kterém jsou znázorněny průměry získaných bodů pro jednotlivé velikosti flotil. Tečkovanou čarou jsou vidět odchylky od průměru. Na grafu je vidět, že průměry postupně klesají s rostoucí flotilou. Výjimkou je rozdíl mezi 50% a 60% velikostí flotily, kde 60% flotila je o trochu horší, než 50% (má ale lehce nižší odchylku). Tato výjimka bude pravděpodobně způsobena tím, že mezi těmito dvěma velikostmi je menší rozdíl, na kterém se zlepšení ještě neprojeví (u ostatních je rozdíl vždy 20 %). Nejlepšího výsledku

jsme dosáhli, pokud máme stejně velkou flotilu, jako je počet producentů. Neznamená to ale, že by algoritmus využil všechna vozidla. To, zda algoritmus využije více vozidel, je dáno parametrem β z kriteriální funkce, který jsme pro měření zvolili na poměrně nízké hodnotě oproti měřítkům ostatních částí kriteriální funkce. Tento výsledek je nejspíše dán tím, že jelikož máme operátory, které vylepšují vždy dvojice tras dvou vozidel, tak při větší flotile mají operátory více možností jak řešení změnit.



Obrázek 8.1: Graf velikosti flotily – průměry bodů

Podíváme-li se na součty získaných bodů, tak mezi 10 % a 30 % je rozdíl 15 bodů. Mezi 30 % a 50 % 10 bodů. Jak již bylo zmíněno, tak mezi 50 % a 60 % došlo k lehkému zhoršení a získali jsme o 2 body více. Následně mezi 50 % a 80 % je rozdíl 4 body (od 60 % 6 bodů). Na konec mezi 80 % a 100 % dělá rozdíl 7 bodů. Z výsledků je tedy patrné, že pro flotily velikostí menších jak 50 % klesá počet bodů rychleji, než u flotil velikostí větších jak 50 % počtu producentů. To znamená, že uděláme dobře, budeme-li mít k dispozici flotilu větší jak 50 % počtu producentů. Zvolením 60% flotily u reálných instancí jsme tedy neudělali chybu.

Instance\Flotila	10 %	30 %	50 %	60 %	80 %	100 %
O	6	5	4	3	2	1
AB3	3	2	1	1	1	1
AB12	6	3	1	4	5	2
V	6	5	2	4	1	3
J	6	5	2	3	1	4
M	6	4	3	5	2	1
P	1	5	2	4	6	3
B	5	4	6	2	3	1
L	6	5	4	3	2	1
AB9	1	6	4	2	3	5
AB7	5	4	3	2	1	4
AB6	6	1	2	4	5	3
AB8	6	1	3	5	2	4
SP	6	3	4	2	5	1
AB11	3	2	2	1	1	1
AB10	1	3	5	5	4	2
Suma	73	58	48	50	44	**37,0**
Průměr	4,56	3,63	3,00	3,13	2,75	**2,31**
Odchylka	2,03	1,54	1,41	**1,36**	1,73	1,40

Tabulka 8.22: Výsledky měření vlivu velikostí flotil na výsledek

Kapitola 9
Závěr

V práci byl formulován problém, který se zabývá svozem odpadu mezi třídírnami a zpracovateli odpadu. Na tento problém bylo navrženo možné řešení v podobě algoritmu, který je postaven na základě GLS rámce. Následně byl implementován prototyp, který byl použit pro účely experimentů. Experimenty ověřily funkčnost navrženého řešení a přinesly další poznatky, například vysokou závislost dobrého řešení na hodnotě λ.

Řešení problému může přispět k větší efektivitě svozu odpadu v části mezi třídírnami a zpracovateli. Vzhledem ke stále většímu trendu v EU, akcentující ochranou životního prostředí, je náš algoritmus schopen zvýhodňovat ekologičtější zpracovatele odpadu na úkor méně ekologických. Díky tomu může naše řešení přispět ke splnění Směrnice o skládkování odpadu, která si klade za cíl pro rok 2035 skládkovat pouze 10 % produkce odpadu. Jak nám bylo ukázáno při schůzce s panem Ing. Janem Svátkem, MPA (více v kapitole 4), tak nynější systém prakticky nevyužívá moderních technologií a svozy zpravidla nejsou optimalizovány. To se snaží námi navržené řešení změnit.

Problém může být ale aplikovatelný i na problémy z jiné oblasti. Může se jednat o jakýkoliv problém, kde je vyzvedáváno zboží, které je následně potřeba někam odvézt, přičemž můžeme zvýhodnit některá doručovací místa před jinými. Do zvýhodňování míst pro doručování můžeme zahrnout například aktuální vytížení a rozložit tak zatížení těchto míst.

9.1 Návrhy na budoucí postup

Kromě využití i v jiných oblastech je možnost současný problém rozšířit. Pro zjednodušení jsme uvažovali, že právě jednoho producenta může obsloužit jen jedno vozidlo. Návrhem pro budoucí práce může být tedy to, že producenta bude obsluhovat vozidel více. Nabízí se také možnost přidání časových oken, kdy ale již vytváření iniciálního řešení bude NP-těžký problém (viz práce [18]). Další možností je například rozšíření problému o další typy odpadu. V kapitole 4 bylo zmíněno, že po vytřídění odpadu vzniká zbytkový odpad, který může být například spálen pro energetické účely (ZEVO). Současná formulace našeho problému ale s dělením na několik typů odpadů, případně právě na využitelnou a nevyužitelnou část konkrétního odpadu, nepočítá.

Zajímavá by byla i spolupráce přímo s nějakým provozovatelem zařízení

zpracovávající odpad a využití navrženého řešení v praxi. Bohužel nám se nepovedlo s nikým navázat spolupráci. Při takovéto spolupráci by mohlo být zajímavým rozvojem dynamické plánování cest využívající aktuální data o kapacitách jednotlivých konzumentů odpadu a nabízeného odpadu producenty. Za tímto účelem by ale pravděpodobně musel být vybudován nějaký celorepublikový systém, nebo alespoň lokální, shromažďující taková aktuální data.

Opustíme-li možnosti rozvoje našeho problému a podíváme-li se přímo na námi představený algoritmus, tak ten může být rozšířen o další GLS vlastnosti. Například můžeme implementovat již zmíněnou vlastnost řešení, která bude zkoumat přiřazení určitého producenta k nějakému vozidlu, více v sekci 6.1.1. V řešení našeho problému také chybí operátor, který by dělil množství odpadu, které převážíme do určitého konzumenta. Nabízí se tedy možnost vytvoření nějakého komplexnějšího operátoru, který by například náhodně, nebo podle nějakého kritéria, zkusil část odpadu převáženého do nějakého konzumenta odvézt do jiného.

Možností rozvoje, nebo vylepšení našeho problému by se určitě našlo více. Představený problém je poměrně komplexní a má spoustu parametrů. Navíc samotný VRP má velmi pestrou škálu různých variant. Díky tomu je do budoucna možnost náš problém rozvíjet různými směry.

Přílohy

Příloha A

Literatura

[1] SUZANNE, Elodie, Nabil ABSI a Valeria BORODIN. Towards circular economy in production planning: Challenges and opportunities. European Journal of Operational Research [online]. 2020, 287(1), 168-190 [cit. 2021-12-26]. ISSN 03772217
Dostupné z: doi:10.1016/j.ejor.2020.04.043

[2] Cirkulární ekonomika. Pražská mise nulové emise [online]. Praha: Město Praha, 2021 [cit. 2021-12-26]. Dostupné z: `https://klima.praha.eu/cs/cirkularni-ekonomika.html`

[3] KEEBLE, Brian R. The Brundtland report: 'Our common future'. Medicine and War [online]. 2007, 4(1), 17-25 [cit. 2021-12-26]. ISSN 0748-8009. Dostupné z: doi:10.1080/07488008808408783

[4] MORSELETTO, Piero. Targets for a circular economy. Resources, Conservation and Recycling [online]. 2020, 2020(153), 104553 [cit. 2022-03-24]. ISSN 0921-3449. Dostupné z: doi:10.1016/j.resconrec.2019.104553

[5] PRAJAPATI, Himanshu, Ravi KANT a Ravi SHANKAR. Bequeath life to death: State-of-art review on reverse logistics. Journal of Cleaner Production [online]. 2019, 2019(211), 503-520 [cit. 2022-03-24]. ISSN 0959-6526. Dostupné z: doi:10.1016/j.jclepro.2018.11.187

[6] KOPICKI, R., M. J. BERG a L. LEGG. Reuse and recycling - reverse logistics opportunities [online]. United States, 1993, 1993 [cit. 2022-03-24]. Dostupné z: https://www.osti.gov/biblio/133268

[7] DALE S., Rogers, TIBBEN-LEMBKE a Ronald S. Going backwards: reverse logistics trends and practices. 2. USA: Reverse Logistics Executive Council Pittsburgh, PA, 1999.

[8] FLEISCHMANN, Moritz. Quantitative Models for Reverse Logistics. 2001. Berlin: Springer; 2001st edition (March 20, 2001), 2001. ISBN 978-3540417118.

[9] LAPORTE, Gilbert, Paolo TOTH a Daniele VIGO. Vehicle routing: historical perspective and recent contributions. EURO Journal on Trans-

portation and Logistics [online]. 2013, 2(1-2), 1-4 [cit. 2022-03-25]. ISSN 21924376. Dostupné z: doi:10.1007/s13676-013-0020-6

[10] MOR, A. a M. G. SPERANZA. Vehicle routing problems over time: a survey. 4OR [online]. 2020, 18(2), 129-149 [cit. 2022-03-25]. ISSN 1619-4500. Dostupné z: doi:10.1007/s10288-020-00433-2

[11] DANTZIG, G. B. a J. H. RAMSER. The Truck Dispatching Problem. Management Science [online]. 1959, 6(1), 80-91 [cit. 2022-03-25]. ISSN 0025-1909. Dostupné z: doi:10.1287/mnsc.6.1.80

[12] CLARKE, G. a J. W. WRIGHT. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. Operations Research [online]. 1964, 12(4), 568-581 [cit. 2022-03-25]. ISSN 0030-364X. Dostupné z: doi:10.1287/opre.12.4.568

[13] CHRISTOFIDES, Nicos, A. MINGOZZI a P. TOTH. State-space relaxation procedures for the computation of bounds to routing problems. Networks [online]. 1981, 11(2), 145-164 [cit. 2022-03-25]. ISSN 00283045. Dostupné z: doi:10.1002/net.3230110207

[14] CHRISTOFIDES, N., A. MINGOZZI a P. TOTH. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. Mathematical Programming [online]. 1981, 20(1), 255-282 [cit. 2022-03-25]. ISSN 0025-5610. Dostupné z: doi:10.1007/BF01589353

[15] TAILLARD, É. Parallel iterative search methods for vehicle routing problems. Networks [online]. 1993, 23(8), 661-673 [cit. 2022-03-25]. ISSN 00283045. Dostupné z: doi:10.1002/net.3230230804

[16] GENDREAU, Michel, Alain HERTZ a Gilbert LAPORTE. A Tabu Search Heuristic for the Vehicle Routing Problem. Management Science [online]. 1994, 40(10), 1276-1290 [cit. 2022-03-25]. ISSN 0025-1909. Dostupné z: doi:10.1287/mnsc.40.10.1276

[17] ELSHAER, Raafat a Hadeer AWAD. A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. Computers & Industrial Engineering [online]. 2020, 140, 106242 [cit. 2022-03-26]. ISSN 03608352. Dostupné z: doi:10.1016/j.cie.2019.106242

[18] MOHAN, Aravind, Anandhu DILEEP, Sreesankar AJAYAN, Georg GUTJAHR a Prema NEDUNGADI. Comparison of Metaheuristics for a Vehicle Routing Problem in a Farming Community. THAMPI, Sabu M., Ljiljana TRAJKOVIC, Kuan-Ching LI, Swagatam DAS, Michal WOZNIAK a Stefano BERRETTI, ed. Machine Learning and Metaheuristics Algorithms, and Applications [online]. 1203. Singapore: Springer Singapore, 2020, 2020-04-05, s. 49-63 [cit. 2022-03-26]. Communications in Computer and Information Science. ISBN 978-981-15-4300-5. Dostupné z: doi:10.1007/978-981-15-4301-2_5

[19] CHANDANA KAJA, Sai. A New Approach for Solving the Disruption in Vehicle Routing Problem During the Delivery: A Comparative Analysis of VRP Meta-Heuristics [online]. Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden, 2020 [cit. 2021-12-26]. Diplomová. Blekinge Institute of Technology. Vedoucí práce Dr. Julia Sidorova. Dostupné z: http://bth.diva-portal.org/smash/record.jsf?pid=diva2%3A1436175&dswid=-4259

[20] SMET, Laurent a Charles THOMAS. Local Search for the Vehicle Routing Problem. Louvain, Belgie, 2016. Magisterská práce. École polytechnique de Louvain. Vedoucí práce Yves DEVILLE. Dosupné z: https://dial.uclouvain.be/memoire/ucl/en/object/thesis%3A4615/datastream/PDF_01/view

[21] LIU, Shuai, Zhong-Kai FENG, Wen-Jing NIU, Hai-Rong ZHANG a Zhen-Guo SONG. Peak Operation Problem Solving for Hydropower Reservoirs by Elite-Guide Sine Cosine Algorithm with Gaussian Local Search and Random Mutation. Energies [online]. 2019, 12(11) [cit. 2022-03-26]. ISSN 1996-1073. Dostupné z: doi:10.3390/en12112189

[22] NEVRLÝ, Vlastimír. KOMPLEXNÍ MODELY SVOZU ODPADŮ. Brno, 2020. Dizertační práce. VUT v Brně – Fakulta strojního inženýrství. Vedoucí práce Prof. Ing. Petr Stehlík, CSc., dr. h. c. Dostupné z https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=218059

[23] FU, Hui-zhen, Zhen-shan LI a Rong-hua WANG. Estimating municipal solid waste generation by different activities and various resident groups in five provinces of China. Waste Management [online]. 2015, 41, 3-11 [cit. 2022-03-27]. ISSN 0956053X. Dostupné z: doi:10.1016/j.wasman.2015.03.029

[24] Registr zařízení a obchodníků. Isoh [online]. 2021 [cit. 2022-03-31]. Dostupné z: https://isoh.mzp.cz/RegistrZarizeni/Main/ORegistru

[25] ŠANDERA, Čeněk. Hybridní model metaheuristických algoritmů. Brno, 2013. Dizertační práce. VUT v Brně, FAKULTA STROJNÍHO INŽENÝRSTVÍ. Vedoucí práce Prof. RNDr. Ing. Miloš Šeda, Ph.D. Dostupné z: https://dspace.vutbr.cz/bitstream/handle/11012/35853/final-thesis.pdf?sequence=1&isAllowed=y

[26] OSMAN, I H a J P KELLY. Meta-Heuristics Theory and Applications. Journal of the Operational Research Society [online]. 2017, 48(6), 657-657 [cit. 2022-04-07]. ISSN 0160-5682. Dostupné z: doi:10.1057/palgrave.jors.2600781

[27] DORIGO, Marco a Thomas G. STÜTZLE. Ant colony optimization. Cambridge: MIT Press, 2004. ISBN 9780262042192.

[28] TALBI, El-Ghazali. Metaheuristics: from design to implementation. Hoboken: Wiley, c2009. ISBN isbn978-0-470-27858-1.

[29] YANG, Xin-She. Nature-inspired metaheuristic algorithms. 2nd ed. Frome,: Luniver Press, 2010. ISBN isbn978-1905986286.

[30] ČR nastupuje trend: od skládkování ke třídění, recyklaci a materiálovému využití na maximum. Ministerstvo životního prostředí [online]. Ministerstvo životního prostředí: Tiskové oddělení MŽP, 2020 [cit. 2022-04-07]. Dostupné z: https://www.mzp.cz/cz/news_20201201-CR-nastupuje-trend-od-skladkovani-ke-trideni-recyklaci-a-materialovemu-vyuziti-na-maximum

[31] VOUDOURIS, Christos a Edward TSANG. Guided local search and its application to the traveling salesman problem. European Journal of Operational Research [online]. 1999, 113(2), 469-499 [cit. 2022-04-11]. ISSN 03772217. Dostupné z: doi:10.1016/S0377-2217(98)00099-Xz

[32] SMĚRNICE EVROPSKÉHO PARLAMENTU A RADY (ES) č. 98/2008: o odpadech a o zrušení některých směrnic. In: . Úřední věstník Evropské unie: Evropská unie, 2008, ročník 2008, 98/2008. Dostupné také z: https://eur-lex.europa.eu/legal-content/CS/TXT/PDF/?uri=CELEX:32008L0098&from=LV

[33] ROUGHAN, Matthew a Jonathan TUKE. The Hitchhikers Guide to Sharing Graph Data. In: 2015 3rd International Conference on Future Internet of Things and Cloud [online]. IEEE, 2015, 2015, s. 435-442 [cit. 2022-04-19]. ISBN 978-1-4673-8103-1. Dostupné z: doi:10.1109/FiCloud.2015.76

Příloha B

Seznam digitálních součástí

Název	***Typ***	**Popis**
isohRetrieval	*Adresář*	Nástroj pro filtraci dat z registru ISOH.
generator	*Adresář*	Nástroj pro generování instancí.
measurement	*Adresář*	Nástroje pro spouštění experimentů.
solver	*Adresář*	Implementace GLS algoritmu.