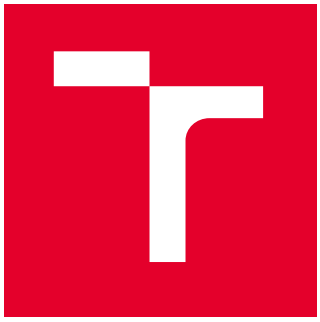
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta strojního inženýrství

DIPLOMOVÁ PRÁCE

Brno, 2022 Bc. Jiří Kučera



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MATEMATIKY

INSTITUTE OF MATHEMATICS

MODELOVÁNÍ LOGISTIKY MEZIOBECNÍ PŘEPRAVY ODPADU

MODELLING OF LOGISTICS OF INTER-MUNICIPAL WASTE TRANSPORT

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jiří Kučera**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Vlastimír Nevrlý, Ph.D.**

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav matematiky
Student:	**Bc. Jiří Kučera**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Matematické inženýrství
Vedoucí práce:	**Ing. Vlastimír Nevrlý, Ph.D.**
Akademický rok:	2021/22

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Modelování logistiky meziobecní přepravy odpadu

Stručná charakteristika problematiky úkolu:

Zpracování dat z oblasti odpadového hospodářství představuje důležitou roli pro výpočty logistických řetězců. Na úrovni obcí se za účelem svozu odpadu zakládají svazky, které si zajišťují svoz všech typů odpadu pomocí vlastních zdrojů (vozidla, pracovníci). Pro nová uskupení obcí je nutné navrhnout svozové plány a definovat potřebný vozový park. Absence historických dat a zkušeností otevírá prostor pro definování základních logistických řetězců. Práce se bude zabývat tvorbou infrastrukturní sítě pomocí poznatků z teorie grafů pro silniční přepravu (vzdálenosti, čas přepravy) na meziobecní úrovni. S ohledem na potřeby pro jednotlivé typy odpadu budou aplikovány vícerozměrné statistické metody (např. shlukovací algoritmy) pro agregaci adresních bodů. Nad definovanou sítí proběhne vývoj a testování modelů operačního výzkumu pro svoz odpadu. Výsledek práce umožní vyhodnotit základní ekonomickou rozvahu svazků obcí. Závěrečná práce bude v průběhu konzultována s Ing. Dušanem Hrabcem, Ph.D.

Cíle diplomové práce:

– Seznámení s logistikou odpadu na meziobecní úrovni.
– Implementace algoritmů pro tvorbu dopravní sítě.
– Aplikace statistických metod a algoritmů pro agregaci adresních míst.
– Testování a vývoj modelů svozu odpadu – případová studie.
– Závěrem budou dosažené výsledky patřičně interpretovány. Budou kriticky diskutovány omezení a limity zvoleného přístupu a budou vymezeny směry pro navazující výzkum.

Seznam doporučené literatury:

GHIANI, G., LAPORTE, G., MUSMANNO, R. Introduction to Logistics systems planning and control. Wiley-interscience series in systems and optimization, John Wiley & Sons, Chichester, 2004.

WILLIAMS, H. P. Model Building in Mathematical Programming. London School of Economics, UK: Wiley, 5th ed., 432, 2013. ISBN 978-1-118-44333-0.

VANDERBEI, R. J. Linear Programming: Foundations and Extensions. International Series in Operations Research & Management Science, Springer, 2007.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2021/22

V Brně, dne

L. S.

prof. RNDr. Josef Šlapal, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Tato diplomová práce se zaměřuje na problematiku meziobecní přepravy odpadu. Stěžejní část práce se zaměřuje na vývoj výpočetního jádra určeného k vytvoření svozového plánu pro vybrané obce tvořící svazek. Výpočtové jádro bylo tvořeno tak, aby mohlo dojít k jeho integraci do uživatelsky přívětivé webové aplikace. Tvorba výpočetního nástroje vychází z teoretických základů teorie grafů, matematického programování a heuristických algoritmů. V rámci práce byly vyvinuty dílčí, na sebe navazující algoritmy pro clusterování adresních míst, návrh svozu pro více typů odpadu s heterogenním vozovým parkem a návrh svozu do několikatýdenního svozového plánu. Představený přístup byl testován na reálných datech z existujícího svazku obcí v Jihomoravském kraji.

KLÍČOVÁ SLOVA

Přeprava odpadu, operační výzkum, heuristické algoritmy, shlukování, svozová úloha, plánování.

ABSTRACT

This diploma thesis focuses on the issue of inter-municipal waste transport. The main part of the thesis focuses on the development of a computing core designed to create a collection plan for selected municipalities forming a union. The computing core was created so that it could be integrated into a user-friendly web application. The creation of a computational tool is based on the theoretical foundations of graph theory, mathematical programming and heuristic algorithms. In this thesis, consecutive algorithms for clustering of address points, collection design for several types of waste with a heterogeneous vehicle fleet and for creating collection proposal for a several-week long collection plan were developed. The presented approach was tested on real data from an existing union of municipalities in the South Moravian Region.

KEYWORDS

Waste transportation, operational research, heuristic algorithms, clustering, vehicle routing problem, scheduling.

KUČERA, Jiří. *Modelování logistiky meziobecní přepravy odpadu*. Brno, 2022, 62 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta Strojního inženýrství, Ústav matematiky. Vedoucí práce: Ing. Vlastimír Nevrlý, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Modelování logistiky meziobecní přepravy odpadu“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval mému vedoucímu diplomové práce Ing. Vlastmíru Nevrlému Ph.D. za odborné vedení, konzultace a podnětné návrhy k práci. Dále bych rád poděkoval Ing. Veronice Smejkalové a Ing. Radovanu Šomplákovi, Ph.D. za pomoc, součinnost a trpělivost, kterou měli při častých konzultacích.

Obsah

1 Úvod

1.1 Motivace

Sběr a svoz odpadu hraje zásadní roli z pohledu celkových nákladů v systému nakládání s odpady. V případě komunálních odpadů (KO) je to očekávatelné, protože jsou jednotlivé frakce KO shromažďovány obyvateli obvykle poblíž jejich domovů. Tato sběrná místa je pak nutné obsloužit svozovými firmami, případně technickými službami dané obce. Tlak na vyšší míru recyklace (popřípadě energetického využití) KO je ukotven v balíčku oběhového hospodářství EU (Circular economy package – CEP [7]). Dílčí milníky jsou dále promítány do legislativy jednotlivých členských států. Česká republika některé body CEP již implementovala do nedávno vydaného Zákona o odpadech (ZoO, 2021 [46]). Cíle spojené s vyšší recyklací KO mají za následek stále hustší sběrnou síť, což celý systém sběru a svozu prodražuje. Náklady dále rostou se zvyšujícím se počtem separovaných frakcí KO, které je nutné obvykle svážet samostatně. Příkladem z nedávné doby je zavedení sběru olejů a jedlých tuků. V jejich případě mají obce povinnost umožnit svým občanům separovaný sběr od roku 2021 (ZoO). Obdobně bude následovat textilní odpad s termínem od roku 2025. Získat konkrétní data o reálných nákladech obcí na sběr a svoz jednotlivých frakcí KO je obtížné, protože se jedná o diskrétní smluvní informace se svozovými firmami. Vodítkem mohou být agregovaná data vydaná Institutem pro udržitelný rozvoj měst a obcí, o.p.s. Na jejich webových stránkách jsou uvedena data z roku 2015 a starší, viz (IURMO [20]). S ohledem na jejich relativní aktuálnost tyto informace poskytují dostatečný vhled do této problematiky. Konkrétně průměrné náklady v roce 2015 spojené se separací papíru činily asi 4 200 Kč/t, u plastu přibližně 7 100 Kč/t a u skla asi 1 950 Kč/t. U směsného komunálního odpadu (SKO) to bylo v průměru okolo 1 500 Kč/t (pouze sběr a svoz). Aby bylo možné data o nákladech uvedených výše dostat do kontextu, je třeba uvést průměrnou produkci na obyvatele pro tyto frakce KO vyprodukované obcemi v roce 2015. V případě papíru se jedná o cca 30,4 kg/obyv., pro plast cca 11,7 kg/obyv., pro sklo 11,7 kg/obyv. a pro SKO cca 198,8 kg/obyv. (ISOH [21]). Do roku 2019 se průměrné produkce na osobu zvýšily o cca 7 % u papíru, o cca 32 % u plastu, o cca 19 % u skla a v případě SKO klesla produkce asi o 1,5 %. Poslední dostupná data jsou z roku 2020, ale s ohledem na pandemii covid-19 jsou data v tomto roce nevypovídající. I když se celkové množství SKO s vyšší separací odpadu snižuje, celkově se náklady na sběr a svoz KO rapidně zvyšují. Je třeba doplnit, že jsou uvedeny pouze některé hlavní frakce KO. Např. v případě bioodpadu, u kterého narostla produkce o 59 % za posledních 5 let na 63 kg/obyv., je nárůst nákladů extrémní. Jedná se navíc o typ odpadu se sezón-

ním charakterem, přičemž hodnoty významně kolísají v průběhu roku. Pro tento typ odpadu jsou v poslední době zaváděny samostatné sběrné nádoby a v budoucnu bude tento trend pokračovat. Trend zvyšujících se nákladů spojených s nakládáním s odpady motivuje obce hledat nové směry a koncepty ve zpracovatelských řetězcích. Jednou z možností je tvorba svazků obcí, u nichž je možné využít větší vyjednávací síly při komunikaci se svozovými firmami. V případě větších svazků obcí je možné pořídit vlastní svozová vozidla a obstarávat si svoz odpadů samostatně. Příkladem takového svazku obcí jsou Technické služby Malá Haná s.r.o. (TSMH), které byly založeny v roce 2017. V roce 2022 mají TSMH přes 50 členských obcí. Svazky obcí, které si sami zajišťují svoz odpadu, musí vytvořit svozové plány. I v případě nízkých jednotek svozových aut (TSMH disponuje šesti vozy) se jedná o poměrně obsáhlé svozové území s desítkami obcí. Při kombinovaném svozu více frakcí KO je již sestavování plánu složité a je vhodné využít sofistikovaných optimalizačních přístupů. Další komplikací je případná změna členů svazku, obvykle je značný zájem nových obcí se do svazku zapojit. V takovém případě je třeba celý svozový plán upravit. Navíc se může stát, že při navýšení počtu členů svazku již nebude postačovat aktuální vozový park. Tento typ plánování, obzvláště pro větší svazky, již není možné efektivně vykonávat bez příslušných výpočtových nástrojů.

1.2 Definice problému

Cílem je vyvinout optimalizační přístup pro tvorbu svozových plánů, který bude zahrnovat následující body:

- Vícekomoditní svoz odpadu, jmenovitě: SKO, papír, plast, sklo, bioodpad, kovy, textil, objemný odpad a jedlé oleje a tuky.
- Více typů svozových vozů z pohledu objemové a hmotnostní kapacity.
- Kapacitní a časové omezení svozu.
- Různá koncová zařízení pro každou frakci odpadu a možnost existence více zařízení pro jednu frakci.
- Různá frekvence svozu pro jednotlivé frakce KO.
- Možnost vyhodnocovat svoz s ohledem na očekávané změny produkce KO.

Obce je třeba reprezentovat vhodným počtem uzlů s ohledem na kapacity svozových vozidel. S ohledem na nutné opakované plánování v akceptovatelném čase je potřeba uzpůsobit rozsah a způsob řešení úlohy. Je možné poskytovat pouze lokální optima řešení s ohledem na charakter problému.

1.3 Vybrané existující publikace v této oblasti

Tvorba svozových plánu se běžně definuje jako tzv. Vehicle Routing Problem (VRP). Tento typ algoritmu je běžně používaný v celé řadě aplikací a je možné ho ještě zobecnit (Baldacci a kol. [1]). Konkrétní modely se zaměřují na různé problematické části – více komodit (Cattaruzza a kol. [5]), více typů vozidel (Gendreau a kol. [16]), omezení času (Bräysy a Gendreau [4]). V případě komplexnějších problémů nebo v rozsáhlejších úlohách (Nucamendi-Guillén a kol. [35]) není obvykle exaktní řešení možné a je třeba využít heuristického přístupu. Jak exaktních, tak aproximačních přístupů pro řešení těchto úloh existuje celá řada (Laporte [31]). Rozsáhlejší rešeršní publikace nabízí komplexnější pohled do této problematiky (Laporte [32]), z pohledu časových závislostí (Gendreau a kol. [17]) nebo z pohledu heterogenního vozového parku a vhodných heuristikách (Koç a kol. [27]). Obvykle, když je řešen konkrétní problém s reálnými daty, je třeba sestavit algoritmus tzv. „na míru“, což je situace řešená v této práci. Nicméně hlavní myšlenky budou čerpány z výše uvedených článků. Pokud by se jednotlivé svozové trasy řešily zvlášť a až následně se z nich tvořil svozový plán, taková situace vede na problém zvaný scheduling. Přístup založený na schedulingu byl na základě existující literatury vybrán pro aplikaci v této práci. I pro tuto oblast jsou prezentovány ukázky kvalitně zpracované rešerše na využívané algoritmy a řešené problémy (Cebi a kol. [6]). Objevují se i modely a algoritmy řešící oba problémy (svozové trasy a svozový plán) současně (Nuortio a kol. [36]), nicméně při zohlednění všech poznatků z reálného provozu se jeví jako vhodnější úlohu rozdělit (Tung a Pinnoi [41]). U rozsáhlých úloh je navíc obvykle nutná dekompozice, což je většinou případ obcí s velkým počtem obyvatel. Protože v případě meziobecní přepravy odpadu nelze rozsáhlou obec, nebo spíše město, obvykle rozumně reprezentovat jedním uzlem tak je třeba obec rozdělit do více uzlů s ohledem na produkci odpadu a kapacitu svozových vozidel. Takové rozdělení může být navrženo např. pomocí shlukovacích algoritmů (clustering). Pro clustering existuje celá řada přístupů, nejčastěji se zakládají na tzv. metodě k-means. Takový typ clusteringu v oblasti odpadového hospodářství je použit např. v (Eghtesadifard a kol. [13]). Pro lepší vhled lze opět uvést rešeršní článek na tuto oblast (Jambudi a Gandhi [22]). Významnou roli však hrají omezení na velikost clusterů (Davidson a Ravi [9]), a je tedy nutné aplikovat modifikované přístupy (Ganganath a kol. [15]).

1.4 Cíle diplomové práce

Práce navazuje na výsledky diplomové práce (Zamazal [45]), kde byla data poskytnutá z monitoringu svozu od TSMH podrobena statistické analýze. Dílčí výsledky – statistické modely – slouží pro tvorbu vstupních dat v rámci implementace do

VRP algoritmů, čímž je zajištěna reálnost výstupu pro finální konstrukci svozových plánů pro svazek obcí TSMH. Cíle diplomové práce mohou být rozděleny do dvou skupin. Hlavní cíle, motivované především body uvedenými v zadání práce a definicí problému (viz výše):

- Rešerše, výběr a aplikace existujících nástrojů a funkcionalit k řešení optimalizačního modelu dle definice problému.
- Vytvoření přístupu pro pravidla clusteringu měst a obcí.
- Sestavení optimalizačního modelu dle definice problému.
- Tvorba heuristiky (v případě potřeby vícestupňové) pro zajištění řešitelnosti rozsáhlých úloh.
- Implementace navržených algoritmů do vhodně zvoleného programovacího jazyka.
- Analýza výsledků a ověření funkčnosti vzniklé softwarové aplikace.

Vedlejší cíle, které souvisí s výzkumnými aktivitami Ústavu procesního inženýrství (ÚPI) – domovský ústav vedoucího této závěrečné práce):

- Implementace vstupních dat a statistických modelů na základě dostupných informací od ÚPI.
- Navázání algoritmů na webovou aplikaci a komunikace pomocí formátu JSON[1].
- Využití dílčích výstupů/svozových plánů z výpočtu pro další testování a aktualizaci komplexního nástroje vznikajícího na ÚPI.

Text této diplomové práce je dále členěn následovně. Kapitola 2 je se zabývá metodami pro tvorbu clusterů. Kapitola 3 řeší teoretický pohled na VRP. Následuje kapitola 4, ve které je teoreticky popsáno sestavení svozového plánu pomocí Scheduling úlohy. V 5. kapitole je popis vlastní implementace úloh a algoritmů z kapitol 2, 3 a 4. Poslední 6. kapitola uvádí případovou studii pro svazek obcí TSMH a možný další vývoj.

[1] JavaScript Object Notation

2 Clustering

Clustering nebo také česky shluková analýza či clusterová analýza je statistická metoda, která slouží ke klasifikaci objektů. V rámci této diplomové práce budou využívány názvy clustering pro samotnou analýzu a cluster pro jednotlivý shluk bodů.

Everrit [14] uvádí, že schopnost klasifikace objektů do skupin/shluků/clusterů je důležitou součástí civilizace od jejího počátku, kdy si už první lidé museli uvědomovat podobnosti mezi jedlými rostlinami či rozdíly mezi agresivními a neagresivními zvířaty, takových příkladů bylo, a i nadále je mnoho. V dnešní době je klasifikace objektů využívána na denní bázi bez toho, aniž si to uvědomujeme, jelikož rozlišování je jedním ze základních kamenů každého jazyka, v rámci něhož nám jednotlivá slova pomáhají vyjadřovat naše myšlenky. Podstatná jména slouží k rozdělení objektů do tříd s podobnými vlastnostmi, například slovo stromy pod sebou sdružuje rostliny s dřevěným stonkem (kmenem), jako dub, lípa atp. Na tomto příkladu lze tedy vidět, že pojmenování je ve své podstatě synonymum pro klasifikování/klasifikace, a tedy pro clustering.

Klasifikace objektů je tedy nedílnou součástí civilizace a je důležitou součástí mnohých vědeckých odvětví. Velké uplatnění má například v biologii, kde se skrývá pod názvem taxonomie, což je věda zabývající se klasifikací organismů. Jako další příklad klasifikace může posloužit periodická soustava prvků, která měla velký dopad na lidské porozumění atomu.

Je tedy zřejmé, že clustering je, byl a bude důležitou součástí lidstva, a proto budou v následující části popsány dva základní algoritmy K-means a DBSCAN, které se ke clusterování používají, budou shrnuty jejich aplikace, výhody a nevýhody. Tyto algoritmy byly vybrány, jelikož se staly inspirací pro finální algoritmus, který je použit v kapitole 5 v části 5.1.

Z důvodu rychlosti výpočtu je pro clustering na použitých datech nutné transformovat zeměpisné souřadnice, způsobu, kterým to lze udělat, se věnuje následující část.

2.1 Křovákovo zobrazení

Problematika

Vzhledem k podobě problému, kterým se tato diplomová práce zabývá, je zřejmé, že je potřeba znát vzdálenosti mezi vybranými body na Zemi. Data, se kterými se pracuje, mají určenou zeměpisnou šířku i délku, a lze tedy mezi nimi zjistit vzdálenost. Ideální situace by byla znát vzdálenosti mezi body na silnicích a ty poté rozdělit do

menších clusterů, čemuž se podrobněji věnuje právě tato kapitola, a na nich řešit úlohu VRP (3). Pro nevelké množství bodů není problém silniční vzdálenost zjistit. Pro dva body se například dvakrát jednoduše klikne v libovolné mapové aplikaci, a ta zobrazí vzdálenost mezi nimi. Ovšem při problému, kterým se zabývá tato práce, kdy se pracuje s tisícovkami adresních bodů, ruční zadávání souřadnic nedává smysl. Existují samozřejmě nástroje, které umí vyhodnotit vzdálenosti po silnicích mezi větším množstvím bodů, jedním z takových je například API[2] od mapy.cz. Ovšem ani tyto aplikace nejsou vhodné pro získání plné matice vzdálenosti mezi všemi body, proč tomu tak je, je popsáno v kapitole 5, v části 5.1. Této API je využito v části VRP, jelikož se zde pracuje s mnohonásobně menším počtem bodů než v části clusteringu, kde je potřeba rozdělit velké množství bodů do menších clusterů podle vzájemných vzdáleností. Vzhledem k tomu, že jednotlivé clustery nebudou nikterak velké a snahou bude mít v jednotlivých clusterech body co nejblíže od sebe, tak se může počítat se vzdušnou vzdáleností na Zemi a vzniklá chyba výrazně neovlivní řešení.

Otázkou nyní je, jak nejlépe získat nejkratší vzdálenost mezi dvěma body na Zeměkouli, takzvanou ortodromu. K tomu se využívá například haversine formule:

$$\mathrm{hav}\left(\frac{d}{r}\right) = \mathrm{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\mathrm{hav}(\lambda_2 - \lambda_1) \tag{2.1}$$

kde

- d – vzdálenost mezi dvěma body na Zemi
- r – poloměr Země
- φ – zeměpisná šířka
- λ – zeměpisná délka

Pro funkci haversine platí:

$$\mathrm{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos(\theta)}{2}$$

Vzdálenost d lze poté vyjádřit následovně:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_1 \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{2.2}$$

Tímto se tedy dostane vzdálenost 2 bodů na zeměkouli. Z výpočetního hlediska při využití dnešních počítačů se nejedná o žádný velký problém ani pro větší množství bodů. Existuje však ještě rychlejší řešení, kterého se dosáhne pomocí Křovákova zobrazení.

[2] Application Programming Interface

Toto zobrazení vzniklo již v roce 1922 a jedná se o konformní kuželové zobrazení v obecné poloze a vzniklo jako součást referenčního systému jednotné trigonometrické sítě katastrální (S-JTSK). Předností tohoto zobrazení je, že dokáže pro Českou republiku převést polohu ze sféry do roviny, přičemž zachovává vzdálenosti ([29]). Na takto převedené souřadnice již lze aplikovat známá Euklidova vzdálenost definovaná jako:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2} \tag{2.3}$$

Je zřejmé, že Euklidova vzdálenost je na výpočet jednodušší než haversine formule, celý výpočet se tedy touto úpravou urychlí. Transformovaná data se nyní mohou rozdělit do clusterů. Tato transformace slouží pouze ke zjednodušení a k zrychlení výpočtu a nejedná se o nutnou akci, která by musela být provedena, aby další výpočty mohly pokračovat.

2.2 Základní poznatky

S pojmem cluster se často pracuje na intuitivní úrovni, Šarmanová [39] popisuje cluster jako množinu objektů $\boldsymbol{O} = \{O_1, O_2, ..., O_m\}$, které jsou zadané reálnými atributy. Clustering poté zkoumá, zda existují podmnožiny množiny objektů $\boldsymbol{O}$, ve kterých jsou jednotlivé objekty podobné ostatním prvkům z podmnožiny a rozdílné oproti objektům z ostatních podmnožin. Clustering tedy není jedním všemocným algoritmem, ale jedná se o mnoho různorodých metod. Výběr metody závisí na požadovaném typu výsledků, na typu dat, nebo třeba na definici clusteru.

Před popsáním konkrétních metod je na místě zavést některé pojmy a ujasnit si problematiku clusteringu.

Základní problémy při řešení clusteringu podle Šarmanové [39] jsou:

- Výběr atributů pro popis podobnosti objektů a tomu odpovídající výběr způsobu, jak tyto atributy změřit. Nejčastěji se používají například koeficienty korelace nebo metriky.
- Definice clusteru.
- Počet clusterů rozkladu a počáteční rozklad.
- Vzdálenost clusterů.

Častý atribut, který se využívá, je vzdálenost objektů. Pro měření vzdálenosti se využívají metriky a volba metriky závisí na daném typu dat. V této diplomové práce bude využívána euklidovská metrika, jelikož pro studovaný případ nejvíce vyhovuje potřebě zjistit vzdálenost mezi body, bližší popis se nachází v kapitole 5, v části 5.1. Jak už bylo zmíněno clusterovacích algoritmů je mnoho a jdou rozdělit do několika základních skupin skupin. Mezi nejpoužívanější skupiny patří:

Algoritmy založené na rozdělení oblastí

Do této skupiny patří již zmíněný K-means. Základní myšlenkou těchto algoritmů je považování středu vybraných dat za střed clusteru a poté přiřazování nejbližších bodů k němu.

Algoritmy založené na hierarchii

Tyto metody jsou postavené na konstrukci hierarchie v datech. Myšlenka je následující: každý jednotlivý bod lze na začátku považovat za samostatný cluster, poté se vždy spojí dva nejbližší clustery a toto se opakuje do doby, než se vytvoří jeden velký cluster. Tento postup lze aplikovat i z opačného konce, tedy od jednoho velkého clusteru, který se bude dělit na menší.

Algoritmy založené na hustotě oblasti

Myšlenka algoritmů z této skupiny spočívá v seskupení bodů z oblastí s vysokou hustotou do stejného clusteru. Výhodou je efektivnost, a vhodnost pro jakýkoliv tvar, který se v datech může objevit. Nevýhodou je nutnost mít data s rovnoměrně rozloženou hustotou prvků, a také vysoká závislost na vstupních parametrech.

2.3 K-means

K-means clustering, v češtině také k-průměrů, je ve své podstatě jednoduchý, ale zároveň hojně využívaný, algoritmus používaný ke klasifikaci objektů do clusterů.

Jedná se o iterativní metodu nehierarchického clusterování, která rozděluje množinu s n objekty do $k \geq 2$ clusterů tak, že objekty v jednotlivých clusterech jsou sobě podobné a zároveň rozdílné oproti prvkům z ostatních clusterů.

2.3.1 Formulace úlohy

Cílem K-means je najít pro množinu n bodů $\{\boldsymbol{x}_i\}_{i=1}^n \in \mathbb{R}^d$ k centroidů $\{\boldsymbol{c}_j\}_{j=1}^k$ takových, že celková suma vzdáleností mezi body a jím odpovídajících centroidů je minimální. Když se množina bodů náležící centroidu j označí jako $\{\Gamma_j\}_{j=1}^k$, tato množina bodů se nazývá cluster, tak celková suma vzdáleností $f(\boldsymbol{c}_j, \Gamma_j)$ je:

$$f(\boldsymbol{c}_j, \Gamma_j) := \sum_{j=1}^{k} \sum_{i \in \Gamma_j} d(\boldsymbol{x}_i, \boldsymbol{c}_j) \tag{2.4}$$

Rovnice (2.4) je účelová funkce, která je závislá na centroidech $\{\boldsymbol{c}_j\}_{j=1}^k$ a na clusterech $\{\Gamma_j\}_{j=1}^k$. Funkce $d(\boldsymbol{x}, \boldsymbol{y})$ značí nějakou vybranou metriku, měřící podobnost (vzdálenost) mezi $\boldsymbol{x}$ a $\boldsymbol{y}$. Cílem je poté funkci 2.4 minimalizovat.

2.3.2 Algoritmus

K-means lze jednoduše popsat následujícím algoritmem. Vzhledem k faktu, že výsledné řešení závisí na počáteční volbě centroidů, která je provedena náhodným výběrem, tak se jedná o nedeterministický algoritmus.

Algoritmus 2.1 K-means

1: Vybrat počet clusterů k.
2: Náhodně umístit centroidy $c_1, c_2, ..., c_k$
3: Pro každý prvek najít nejbližší centroid
4: Přiřadit každý prvek jeho nejbližšímu centroidu
5: Pro každý cluster $1, ..., k$ spočítat nový centroid (např. průměr všech prvků)
6: Opakovat kroky 3-5 dokud nebude splněno kritérium optimality, nebo do přesně stanoveného počtu iterací

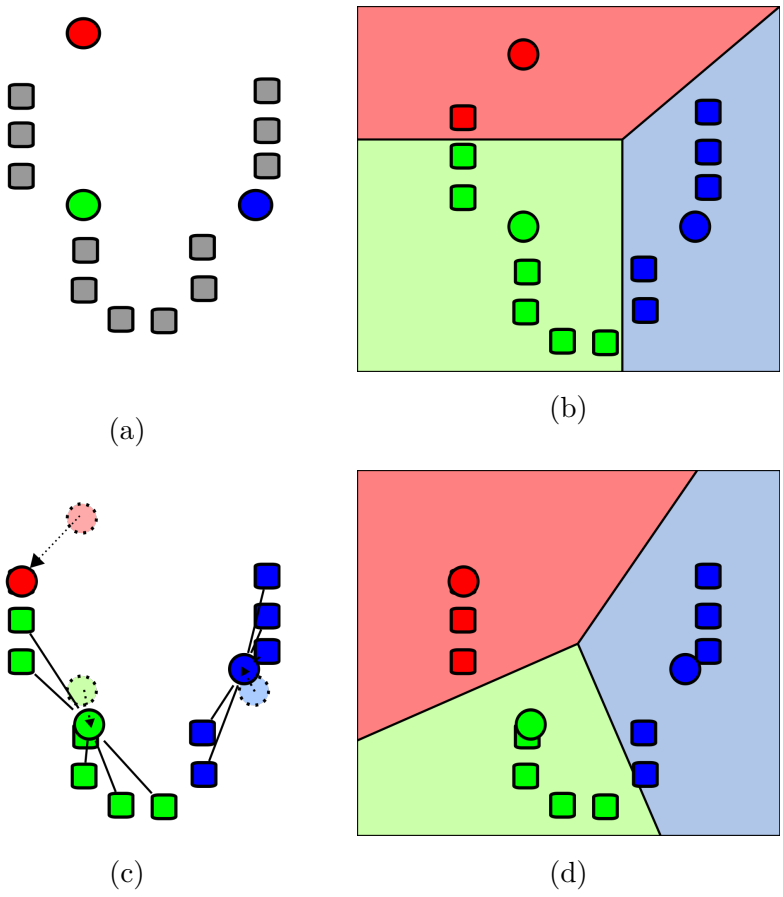
Aby algoritmus mohl skončit, musí být definovány podmínky, za kterých algoritmus ukončí svůj výpočet.

Kritéria ukončení algoritmu

K-means algoritmus má tři základní podmínky pro své ukončení

1. Nově vypočítané centroidy se neliší od předchozích (např. rozdíl v jejich poloze je menší než zadaná tolerance).
2. Jednotlivé objekty zůstávají ve stejných clusterech během iterování.
3. Je dosaženo maximálního počtu iterací.

Na obrázku 2.1 je znázorněna jednoduchá grafická interpretace základní myšlenky K-means algoritmu.



Obr. 2.1: K-means algoritmus [25]

- 2.1a: k prvních centroidů, které byly náhodně vybrány z dostupných bodů
- 2.1b: Vytvoří se k clusterů tak, že každý bod se přiřadí nejbližšímu centroidu
- 2.1c: Vypočítání nových centroidů
- 2.1d: Kroky z obrázků 2.1b a 2.1c jsou opakovány, než nastane některé z kritérií ukončení algoritmu.

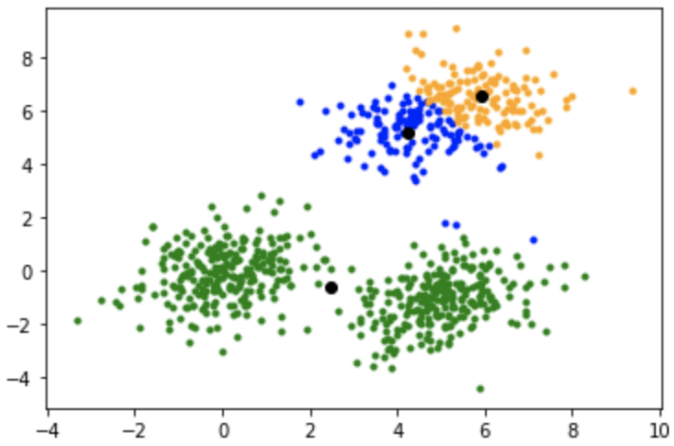
2.3.3 Centroidy

V předchozí části byl několikrát zmíněn pojem centroid, je tedy vhodné se nyní zaměřit vymezení pojmu. Pojem centroidu může být intuitivně jasný. Centroid, nebo také geometrický střed, případně těžiště, je v clusteringu nějaký významný bod, kolem kterého se shlukují ostatní body. Není ovšem jednoznačná definice toho, jak by měly centroidy vypadat, jelikož jejich podoba se odvíjí od typu dat, které se chtějí clusterovat. Stručně by šel centroid popsat jako imaginární nebo reálný bod v datech, reprezentující střed clusteru. Pro data, která jsou použita v této práci, tedy pro adresní body s přesně danými zeměpisnými souřadnicemi, je nejjednodušší

volbou centroidu aritmetický průměr zadaných zeměpisných souřadnic všech bodů v daném clusteru. V případě, že o datech jsou známy další informace, jako například počet obyvatel ke každému bodu, tak vhodnější volbu může být vážený aritmetický průměr. Díky použití průměru je pravděpodobnost trefení se do bude, který není obsažený v datech, takřka stoprocentní. To v závislosti na řešeném problému není žádoucí, lze to vyřešit použitím varianty K-means clusteringu zvané K-medoids. K-medoids algoritmus používá skutečný bod z dat jako reprezentanta centra clusteru, medoid je tedy bod nejvíce ve středu clusteru s minimálním součtem vzdáleností k ostatním bodům v daném clusteru (Jin [23])

Inicializace centroidů

V předchozí části je napsáno, že centroidy se vyberou náhodně z dostupných dat, to však může vést k problému zobrazeném na obrázku 2.2, kde jsou centroidy zobrazeny černými body.



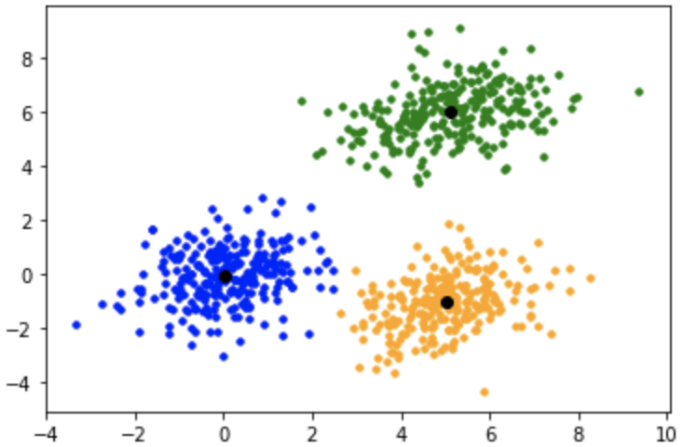
Obr. 2.2: Špatná inicializace centroidů [24]

Na obrázku výše lze vidět tři kompaktní shluky bodů, proto je přirozené data clusterovat na tři clustery. Nastala zde však situace, kdy se při náhodném výběru počátečních centroidů vybraly dva body blízko sebe (z modrého a ze žlutého clusteru) a poté jeden vzdálený bod někde ze zeleného clusteru. Na takto vybraných centroidech se dostane právě řešení zobrazené na obrázku 2.2 a od pohledu je jasné, že takové řešení není zrovna ideální. Tomuto problému se snaží předejít K-means++ algoritmus. Jedná se o stejný algoritmus jako klasický K-means, až na počáteční výběr centroidů, který probíhá podle následujícího algoritmu 2.2

Při aplikaci této inicializace počátečních centroidů na stejná data jako na obrázku 2.2 řešení vyjde, tak jak je zobrazené na obrázku 2.3.

Algoritmus 2.2 K-means++

1: Náhodně vybrat první centroid
2: Spočítat vzdálenosti vybraného centroidu ke všem ostatním bodům a bod, který leží nejdále, tak vybrat jako další centroid
3: Vytvořit clustery tak, že ostatní body se přiřadí nejbližšímu centroidu
4: Bod, který je nejdál od svého centroidu, zvolit jako další centroid
5: Opakovat kroky 3 a 4 do doby, než se najde požadované množství centroidů.



Obr. 2.3: Dobrá inicializace centroidů [24]

Na obrázku 2.3 lze tedy vidět, jak dopadne clustering při inicializaci počátečních centroidů pomocí K-means++ algoritmu, a je snadno viditelné, že clustering tentokrát dopadl podle očekávání, tedy tři výrazné shluky bodů byly rozděleny do tří clusterů. Vlastní implementace K-means++ algoritmu v jazyce python je ukázána v kapitole 5, v části 5.1.

2.4 DBSCAN

Při fázi vývoje clusterovacího algoritmu bylo uvažováno nad vícero algoritmy než jen K-means, jedním z nich je i tzv. DBSCAN (akronym z anglického „Density-based spatial clustering of applications with noise“). Jedná se o clusterovací algoritmus používaný ke zpracování obrazu, strojovému učení, data miningu a v mnoha dalších odvětvích. Tento algoritmus je založený na hustotě prvků v prostoru. Everrit [14] popisuje DBSCAN jako algoritmus sloužící ke klasifikování objektů do clusterů podle hustoty oblastí. Stručně řečeno shlukuje k sobě prvky, které jsou blízko sebe, a vyznačuje outliery, které leží v oblastech s nízkou hustotou.

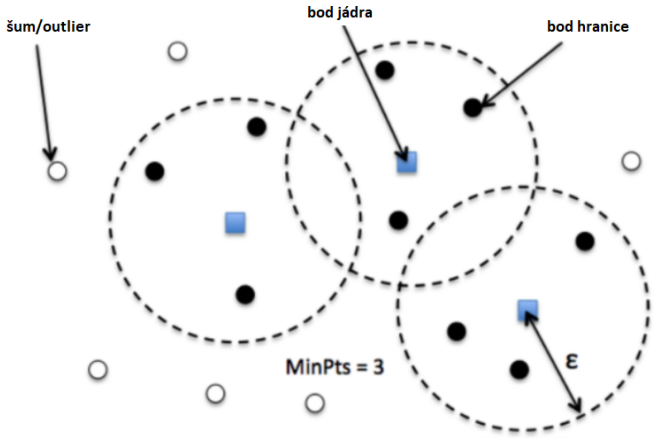
2.4.1 Algoritmus

Algoritmus má dva vstupní parametry [11]:

1. ε-okolí, které definuje oblast s poloměrem ε od objektu.
2. m – minimum počtu bodů, aby oblast byla považována za hustou.

Po ukončení algoritmu je každý bod možno popsat jako jádro, hranice, nebo šum.

- bod je jádro, jestliže v jeho ε-okolí je alespoň m bodů.
- bod je hranice, jestliže v jeho ε-okolí je alespoň jedno jádro.
- bod je šumem, jestliže není ani jádro ani hranice a v jeho ε-okolí je méně než m bodů.



Obr. 2.4: Klasifikace DBSCAN bodů [11]

Kroky algoritmu: Nechť $X = \{x_1, x_2, ..., x_n\}$ je množina bodů z datového souboru. DBSCAN poté potřebuje dva vstupní parametry, a to: ε pro vytvoření ε-okolí a minimální počet bodů m aby byl vytvořený cluster.

Algoritmus 2.3 DBSCAN

1: Vybrat libovolný bod x_i, kde $i = 1, 2, ..., n$, který ještě nebyl vybrán.
2: Jestliže v ε-okolí vybraného bodu je alespoň m dalších bodů, tak všechny tyto body jsou uvažovány jako součást stejného clusteru, v opačném případě, když je v ε-okolí méně než m bodů, tak je bod označen jako outlier.
3: Opakovat kroky 1-2 dokud nebudou navštíveny všechny body.

Základní DBSCAN algoritmus je jednoduchý k naprogramování a zvládá pracovat s clustery různých tvarů, jak už bylo zmíněno, tak nevýhodou je špatná aplikovatelnost na data, ve kterých jsou oblasti s rozdílnou hustotou bodů.

2.5 Shrnutí

Výše popsané algoritmy jsou základním úvodem do clusterovací problematiky. I přes to, že se jedná ve své podstatě o velice jednoduché metody, tak obě mají hojné využití. Během vývoje konečného algoritmu popsaného v kapitole 5 se oba staly ať už větší, v případě K-means, nebo menší, v případě DBSCAN, inspirací. Nakonec se vzhledem k podobě problému, kterým se tato práce zabývá, ukázalo, že pro potřebu aplikace by mohl stačit K-means++ clustering, důvody jsou shrnuty v kapitole 5.

3 Vehicle routing problem

VRP je zobecněním klasické optimalizační úlohy obchodního cestujícího - Travelling Salesman Problem (TSP). TSP má ve své podstatě velice jednoduché zadání, ve kterém je cílem najít nejkratší možnou cestu mezi jednotlivými body, které má obchodník obsloužit s tím, že každou cestu mezi jednotlivými body může absolvovat pouze jednou. TSP se zabývá cestou jednoho obchodního cestujícího. Rozšíření s názvem Vehicle Routing Problem (VRP) tuto úlohy rozšiřuje o situaci, kdy je k dispozici více obchodních cestujících. Jak z názvu vyplývá, jedná se o úlohy, ve kterých hlavní roli hrají vozidla, která zastávají funkci obchodního cestujícího z TSP. Cílem VRP není najít optimální trasu pro jedno vozidlo, ale najít ideální kombinaci tras trasy pro specifikovaný počet vozidel, která všechna začínají v počátečním uzlu a jejich trasy nemají, kromě počátečního bodu, žádný společný bod.

Historie VRP sahá do padesátých let dvacátého století, kdy byla zveřejněna varianta „Truck Dispatching Problem“ pro homogenní vozový park, který z centrálního depa obsluhoval čerpací stanice s co nejmenší najetou vzdáleností. Později byla tato úloha zobecněna na úlohu lineární optimalizace, která se využívá v logistice.

Původně navržené modely jsou samozřejmě jednodušší než současně využívané VRP modely, které se snaží pojmout komplexity reálného života jako, dopravní situaci, pracovní dobu, dynamickou poptávku atp. (Braekers a kol. [3]).

V této diplomové práci je VRP uveden z toho důvodu, že lze aplikovat na svoz odpadu, ve kterém z centrálního depa vyjíždí popelářská auta, aby svezla odpad od lidí, a z provozních důvodů je žádoucí, aby trasy těchto aut byly co nejkratší a obecně, aby provozní náklady byly co nejmenší. Význam VRP pro tuto práci je blíže uveden v kapitole 5 v části 5.2.

3.1 Základní poznatky

Jak už bylo zmíněno, tak VRP je rozšířením úlohy TSP, pro úplnost bude tedy uvedena nejdříve formulace TSP, kterou Williams v [43] uvádí následovně:

Nechť města k navštívení jsou očíslovaná $0, 1, 2, ..., n$, pak binární proměnná δ_{ij} je zavedena takto:

$$\delta_{ij} = \begin{cases} 1 & \text{když existuje přímá cesta mezi městy } i \text{ a } j \\ 0 & \text{jinak} \end{cases}$$

a c_{ij} určuje vzdálenost (nebo cenu) mezi městy i a j.
Cíl je pak jasný, a to minimalizovat účelovou funkci. Úloha je tedy formulována

následovně:

$$\min \sum_{i} \sum_{j} c_{ij} \delta_{ij} \tag{3.1}$$

$$\sum_{\substack{j=0 \\ i \neq j}}^{n} \delta_{ij} = 1, \qquad i = 0, 1, ..., n \tag{3.2}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{n} \delta_{ij} = 1, \qquad j = 0, 1, ..., n \tag{3.3}$$

Podmínka (3.2) říká, že právě jedno město musí být navštíveno po městě i a podmínka (3.3) určuje, že právě jedno město musí být navštíveno před městem j.

Výše uvedená formulace je opravdu jen základní verze TSP. Často se uvažuje více podmínek pro zpřesnění úlohy reprezentující skutečnost. Ovšem i tato jednoduchá formulace stačí k zavedení úlohy VRP.

3.2 Rozšíření na VRP

Cílem té nejzákladnější úlohy VRP je naplánovat trasu pro nějaké množství vozidel, tak aby byli obslouženi všichni zákazníci. Oproti TSP, kde stačilo najít pořadí zákazníků, ve kterém se všichni navštíví, tak cílem ve VRP je naleznout dopředu neznámý počet tras pro známý počet vozidel (ne všechna vozidla musí být využita). I ve VRP platí, že ke každému zákazníkovi je známa poptávka a každý zákazník je navštíven právě jednou. Základní model ovšem nereflektuje reálnou situaci, jelikož se v něm neuvažuje omezená kapacita vozidel a stejně tak se neuvažují časová omezení, jako například pracovní doba řidičů, proto v následující části bude uveden model, ve kterém je zahrnuta omezená kapacita vozidel. Tento rozšířený model je následně v kapitole 5, v části 5.2 rozšířen ještě o časové omezení, právě z důvodu zahrnutí pracovní doby.

Goel a Maini v [18] představují VRP jako orientovaný graf $G(V, E)$, kde $V = \{v_0, v_1, ..., v_n\}$ je množina uzlů (zákazníků), které se mají navštívit z depa v_0 a $E = [\{v_i, v_j\}, (i, j) = 0, 1, 2, ..., n, i \neq j]$ je množina hran, které spojují uzly i a j. Za předpokladu homogenní flotily vozidel, tj. vozidel se stejnou kapacitou, se dostává následující účelová funkce:

$$\min \sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{v=0}^{V} d_{ij} x_{ijv} \tag{3.4}$$

a tyto podmínky:

$$\sum_{v=1}^{V}\sum_{j=1}^{N} x_{ijv} \leq V \quad \text{pro } i = 0, \tag{3.5}$$

$$\sum_{v=1}^{V} x_{ijv} = \sum_{j=1}^{N} x_{jiv} \leq 1 \quad \text{pro } i = 0 \text{ a } v \in \{1, ..., V\}, \tag{3.6}$$

$$\sum_{v=1}^{V}\sum_{j=0}^{N} x_{ijv} = 1 \quad \text{pro } i \in \{1, ..., N\}, \tag{3.7}$$

$$\sum_{v=1}^{V}\sum_{i=0}^{N} x_{ijv} = 1 \quad \text{pro } j \in \{1, ..., N\}, \tag{3.8}$$

$$\sum_{i=0}^{N} c_i \sum_{j=0}^{N} x_{ijv} \leq q_v \quad \text{pro } v \in \{1, ..., V\}. \tag{3.9}$$

$$x_{ij} = \begin{cases} 1 & \text{když vozidlo po obsloužení zákazníka } i \text{ jede přímo k zákazníkovi } j \\ 0 & \text{jinak} \end{cases}$$

V – celková velikost vozového parku

N – počet zákazníků k navštívení

c_i – poptávka zákazníka i (i=$1, 2, ..., N$

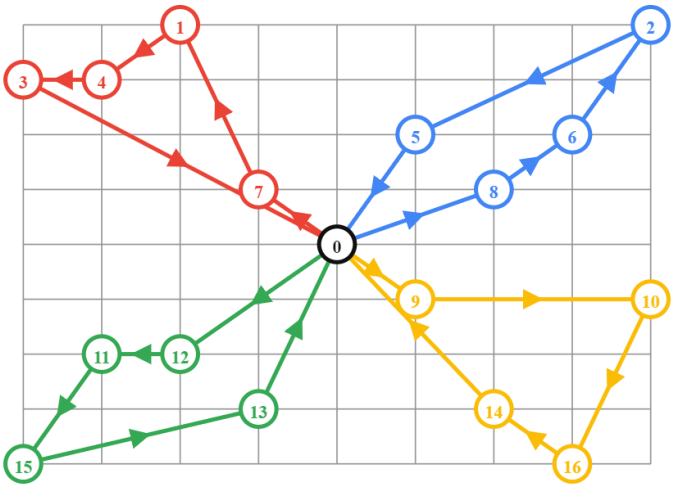
c_0 – depo

d_{ij} – vzdálenost mezi zákazníky i a j

q_v – maximální kapacita vozidla

Cílem je nyní minimalizovat účelovou funkci danou vztahem (3.4) při splnění podmínek (3.5) až (3.9). V účelové funkci se minimalizuje celková vzdálenost, která by ovšem šla nahradit např. časem nebo cenou za přepravu. Podmínka (3.5) zajišťuje, že na všechny trasy bude použito maximálně V vozidel. Díky omezení (3.6) se zajistí, že startovací a koncové místo je stejné pro každé vozidlo. Podmínky (3.7) a (3.8) zaručují, že každý zákazník se navštíví právě jednou. A nakonec podmínka (3.9) zajišťuje, že suma poptávek zákazníků na trase nebude vyšší než kapacita vozidla vybraného pro danou trasu.

Na následujícím obrázku 3.1 si je možné prohlédnout jednoduchou ilustraci řešení VRP modelu pro 15 zákazníků a 4 vozidla na orientovaném grafu.



Obr. 3.1: VRP [19]

Tento model je jednou ze základních variant VRP, na které ovšem staví mnohé další a stejně tomu tak bude i v části 5.2 kde budou do modelu zahrnuta časová okna, ve kterých bude možno provádět sběr odpadu.

3.3 Varianty VRP

Jak už bylo zmíněno, tak výše uvedený model není tou nejzákladnější verzí úlohy VRP, jedná se totiž o takzvaný Capacitated Vehicle Routing Problem. Kromě této varianty existuje ovšem řada dalších, které v sobě zahrnují různé aspekty potřeb reálného světa, Goel a Maini [18] uvádí např. následující varianty.

VRP s časovými okny

Jedním z rozšířeních je VRP s časovými okny tzv. Vehicle Routing Problem with Time Windows. V tomto případě má každý zákazník přesně stanovenou dobu, ve které může být navštíven. Tato doba může být stanovena pevně, tak že za žádných okolností nemůže vozidlo obsloužit zákazníka mimo tuto dobu. Druhá varianta je ta, že se při navštívení zákazníka mimo stanovenou dobu, zavede nějaký typ penalizace, nejpřirozeněji to je peněžní penalizace, se kterou se nadále počítá.

VRP s vyzvednutím a doručením

V dalším rozšíření VRP se pracuje s vyzvednutím poptávky u zákazníka a následným doručením jinému zákazníkovi tzv. Vehicle Routing Problem with Pick-up and

Delivery s cílem minimalizovat najetou vzdálenost všech vozidel. V tomto rozšíření vstupuje do hry další podmínka, která je z lidského pohledu samozřejmá, ale musí být uvedena, a to, že když vozidlo naloží např. balíček u jednoho zákazníka, tak to samé vozidlo musí tento balíček opět vyložit. Tato podmínka zaručuje dvě věci. Zaprvé nemůže nastat situace, že by vozidlo naložený náklad nikde nevyložilo. A zadruhé se nemůže stát, aby náklad naložený jedním vozidlem, byl doručen jiným vozidlem.

Ostatní

Existuje samozřejmě více variant, jak lze rozšířit základní VRP model, může se jednat třeba o případ, kde je více než jedno depo nebo se může přidat možnost, že se vozidla nemusí navrátit do depa na konci své cesty. Mezi studované varianty se často řadí různé kombinace výše zmíněných modelů, které se vznikají z důvodu, lepší reflexe skutečných podmínek, které mohou nastat. Všemožné varianty a kombinace jsou pro zajímavost uvedeny v příloze A na obrázku 1.

3.4 Metody řešení

Způsobů řešení VRP úlohy lze v literatuře najít mnoho. VRP je často modelován pomocí grafů nebo jako úloha smíšeného celočíselného programování. Obecně lze optimalizační úlohy řešit pomocí exaktních algoritmů nebo pomocí heuristik.

3.4.1 Exaktní řešení

Metody k získání přesného řešení VRP jsou založeny na metodách smíšeného celočíselného programování, které (alespoň teoreticky) zaručují nalezení optimálního řešení, pokud tedy existuje. Nejlepšími metodami k řešení VRP jsou takzvané „branch-and-cut-and-price“ metody. Nicméně tyto metody často vyžadují velké množství času a počítačové paměti. Navíc pro stejně veliké úlohy mohou být zcela odlišné časy potřebné k vyřešení (Drexl [12]), chybí zde tedy konzistence, se kterou by tyto metody přinášely řešení.

Navíc kvůli garanci optimálního řešení, musí algoritmy určené k přesnému řešení najít toto optimální řešení a zároveň musí dokázat, že se opravdu jedná o optimální řešení. Musí tedy prohledat všechny možné varianty řešení nebo explicitně určit, že lepší řešení neexistuje (Sörensen [40]).

Přesná řešení, která jsou často výpočetně náročná se ovšem dají nahradit takzvanými heuristickými metodami. Vzhledem k podobě úlohy řešené v této diplomové práci, zde přesná řešení nebudou dále rozváděna.

3.4.2 Heuristické přístupy

Oproti metodám určeným k nalezení přesného řešení, heuristické metody nemají zaručenou optimalitu řešení, na druhou stranu nejsou svázány omezeními přesných metod a zároveň jsou tyto metody schopné najít téměř přesné řešení nebo alespoň dostatečně dobré řešení i pro větší problémy.

Salhi v [38] shrnuje charakteristiky heuristik následovně:

1. Jednoduché a jasné, heuristiky by měly následovat přesně definované kroky.
2. Účinné a robustní, měly by spolehlivě nacházet dostatečně dobrá řešení.
3. Efektivní, čas výpočtu musí být „rozumný“.
4. Flexibilní, heuristiky by mělo být snadné adaptovat a modifikovat.

Z heuristik se postupem času vyvinuly takzvané metaheuristiky (nebo také moderní heuristiky), které se rychle uchytili a řadí se mezi často používané metody řešení. Metaheuristiky jsou metody schopné přibližně řešit těžké optimalizační úlohy bez toho, aby se výrazně upravovaly podle daného problému. Metaheuristiky jsou navrženy tak, aby vedly a upravovaly jiné heuristiky a vyhnuly se tak riziku setrvání v lokálním optimu ovšem ani tak nemají zaručené nalezení globálního optima. Metaheuristiky samotné jsou označovány jako „higher level heuristiky“ a mají mít kontrolu nad „low level heristikami“ (Salhi [38]).

3.5 OR-Tools

Pro řešení VRP úlohy byl pro tuto práci zvolen open source software OR-Tools od společnosti Google. Celý software je popsán v dokumentaci [19] a slouží pro řešení optimalizačních úloh jako je právě VRP nebo TSP. Také lze použít k řešení scheduling problému, k lineární optimalizaci atd. Pro potřeby této práce se hodí zejména díky svým rozsáhlým možnostem řešení VRP, kde lze nastavit velké množství podmínek jako například kapacitní podmínky pro vozidla i pro jednotlivé uzly. Dále lze zahrnout časové podmínky, různé penalizace za nedoručení nebo pozdní doručení nebo také lze omezit celkově najetou vzdálenost. Další nespornou výhodou je, že lze tento nástroj využít hned v několika programovacích jazycích, konkrétně, v době psaní této práce, je dostupný v jazycích C++, Python, Java a C# s názornými příklady v každém jazyce. V této diplomové práci je využita verze pro Python.

3.5.1 Počáteční řešení

K řešení VRP úlohy OR-Tools využívají metaheuristiky (3.4.2). A postup algoritmu se skládá z několika kroků. Nejdříve je potřeba určit tzv. počáteční řešení. Způsobů k nalezení tohoto počátečního řešení je více a OR-Tools ve své verzi 9.3.10497 jich nabízí celkem 13. Mezi těmito třinácti jsou například:

- Path cheapest arc – iterativně rozšiřuje aktuální trasu, vybráním nejbližšího uzlu a přidáním tohoto uzlu do aktuální trasy.
- Path most constrained arc – jedná se o podobnou metodu jako Path cheapest arc, zde jsou navíc hrany mezi uzly porovnávány na základě porovnávacího kritéria, které upřednostňuje nejvíce omezené hrany. Omezení hrany závisí například na tom, zda koncový uzel dané hrany je i konečným uzlem celé orientované cesty nebo na tom, jestli je nutné koncový uzel hrany navštívit.
- Global cheapest arc – iterativně spojuje dva dosud nespojené uzly tak, že se vyberou uzly, které mezi sebou mají nejmenší vzdálenost z celé množiny nespojených uzlů.
- Local cheapest arc – vybere jeden uzel a spojí ho s nejbližším.

Celý výčet algoritmů k nalezení počátečního řešení, které OR-Tools poskytují je k dispozici v [19]. Mimo explicitní výběr, tento nástroj poskytuje i automatickou volbu, kdy software sám zvolí na základě zadaného modelu, jaký algoritmus využít. Při vývoji se nejlepších výsledků dosahovalo za použití Path cheapeast arc algoritmu. Ten k řešení VRP úlohy preferují i OR-Tools, které při automatickém výběru algoritmu vždy vybraly právě Path cheapest arc.

Path cheapest arc

Tento algoritmus sestrojuje jednotlivé trasy jednu po druhé. V prvním kroku se vybere uzel nejblíže depu, který není součástí žádné trasy, tím se trasa inicializuje. V druhém kroku se vybere nejbližší uzel k poslednímu přidanému uzlu a do trasy se přidá za předpokladu, že trasa po přidání tohoto uzlu zůstane přípustná tj. budou splněny zadané podmínky například na délku nebo čas. Druhý krok se opakuje tak dlouho dokud žádné další uzly nejdou přidat do aktuální trasy. V případě, že existují uzly, které nejsou zahrnuty v žádné trase, tak se inicializuje nová trasa a sestaví se stejně jako předchozí (Maatta [33]). Počáteční řešení se dostane tehdy, když jsou všechny uzly součástí některé z vytvořených tras. Určené počáteční řešení se dále iterativně vylepšuje například pomocí local search algoritmů.

3.5.2 Vylepšení počátečního řešení

Local search

Stejně jako u hledání počátečního řešení, tak i u local search algoritmů OR-Tools nabízí několik variant výpočtu. K dispozici je celkem pět algoritmů a opět automatický výběr, při kterém software sám určí jaký algoritmus se využije. K dispozici jsou jmenovitě následující algoritmy:

- Greedy descent.

- Guided local search (GLS).
- Simulated annealing.
- Tabu search.
- Objective tabu search.

Voudouris a kol. v [42] popisují local search algoritmy jako základ mnoha heuristik pro optimalizační úlohy, jedná se o jednoduché iterativní metody, které pracují na principu pokus-omyl. Jak metody fungují lze uvést na následujícím příkladu:

Nechť kombinatorická optimalizační úloha je definovaná dvojicí (S, g), kde S je množina všech přípustných řešení a g je účelová funkce, která každý prvek $s \in S$ zobrazí na reálnou hodnotu. Cílem je najít takové řešení s které minimalizuje účelovou funkci g, tedy:

$$\min g(s), s \in S$$

Okolí N úlohy (S, g) je dáno zobrazením S do své potenční množiny:

$$N : S \to 2^S$$

$N(s)$ je okolím s a obsahuje všechna řešení blízká k s. Blízkostí řešení je myšleno takové řešení, které vzniklo transformací původního řešení pouze malými modifikacemi. Řešení x se nazve lokálním minimem g vzhledem k okolí N když:

$$g(x) \leq g(y), \forall y \in N(x)$$

Local search je metoda minimalizace funkce g pomocí jednotlivých kroků, kdy se v každém kroku nahradí řešení x řešením y tak, že:

$$g(y) < g(x), y \in N(x)$$

Local search často začíná v libovolném řešení a a je ukončen v lokálním minimu. V obecném případě výpočetní náročnost závisí na velikosti okolí a na čase potřebného k nalezení blízkého řešení. Čím větší bude okolí, tím více času bude potřeba, ale tím lepší řešení se dostane (Voudouris a kol. [42]).

V dokumentaci [19] se zmiňuje, že GLS je obecně nejefektivnější heuristikou pro VRP úlohu. Tohoto tvrzení se drží i automatický výběr local search algoritmu, který je k dispozici, když při vývoji programu k řešení VRP, byl v každém případě automaticky vybrán právě GLS algoritmus.

Guided local search

Kilby a kol. v [26] popisují GLS algoritmus jako metaheuristiku založenou na penalizacích. Principem této metody je přidávání penalizací k účelové funkci, které jsou

založené na zkušenostech, které tento algoritmus získal během předešlého hledání optimální trasy.

Jak už bylo zmíněno, tak GLS slouží k dostání se z lokálního minima, toho dosahuje penalizací určitých vlastností řešení, o kterých si „myslí“, že nejsou součástí optimálního řešení. GLS Definuje účelovou funkci rozšířenou o množinu penalizací za nežádoucí vlastnosti. Poté se použijí local search metody, které vylepšují rozšířenou účelovou funkci. Cyklus local search algoritmů a aktualizace penalizací se může opakovat tak dlouho jak bude potřeba do nalezení přípustného řešení, tedy pokaždé když použitý local search algoritmus nalezne lokální optimum rozšířené účelové funkce, tak se účelová funkce aktualizuje a díky tomu dokáže „utéct“ z lokálního optima a pokračovat v hledání lepšího řešení. GLS potřebuje následující vstupní parametry:

- Množina vlastností F, pro kterou platí: $f_i \in F$, $f_i(S) = 1$, když vlastnost i je součástí řešení S a $f_i(S) = 0$ jinak
- Množinu cen C, $c_i \in C$, kde c_i je „cena“ vlastnosti i
- Penalizační faktor λ

Počet penalizací dané vlastnosti během chodu algoritmu se zaznamenává pomocí penalizačního vektoru $\mathbf{p}$, kde p_i je celé číslo určující kolikrát vlastnost i byla penalizována.

Rozšířená účelová funkce $O'(S)$ je definovaná pomocí původní účelové funkce $O(S)$ a sumou použitých penalizací následovně:

$$O'(S) = O(S) + \lambda \sum_{i}^{N} f_i(S) p_i c_i \tag{3.10}$$

Souhrn kroků GLS je v algoritmu 3.1. Funkce InitialSolution() spočítá počáteční řešení, tak jak to bylo popsáno v této kapitole, to se se dále v algoritmu vylepšuje pomocí funkce LocalSearch(), určené k nalezení lepšího řešení. StoppingCondition() funkce definuje podmínky k ukončení výpočtu. Výpočet může být ukončen například, když u řešení během iterací dochází jen k malým nebo žádným změnám, případně po dosažení maximálního počtu iterací.

Algoritmus 3.1 GLS

1: $\mathbf{p} := \vec{0}$
2: $S :=$ InitialSolution()
3: $S^* :=$ LocalSearch(S,$\mathbf{p}$)
4: $S^* := S$
5: **while not** StoppingCondition() **do**
6: $f :=$ ChoosePenalty(S,$\mathbf{p}$)
7: **for** $\forall i$ in f **do**
8: $p_i := p_i + 1$
9: **end for**
10: $S :=$ LocalSeach(S,$\mathbf{p}$)
11: **if** $O(S) < O(S^*)$ **then**
12: $S^* := S$
13: **end if**
14: **end while**
15: $S^* :=$ LocalSearch(S^*,$\vec{0}$)
16: **return** S^*

3.6 Shrnutí

V této kapitole byl popsán teoretický základ VRP úlohy, její základní myšlenky, způsoby řešení a různá rozšíření, která jsou v praxi využívána. Souhrn VRP úloh, představil Drexl v [12]. Ve své publikaci rozebírá možnosti, které VRP úlohy nabízejí a shrnuje je do přehledného digramu, který je k dispozici v příloze A na obrázku 1. Ve své publikaci rozebírá možnosti, které VRP úlohy nabízejí. Dále byla kapitola věnována softwaru OR-tools, vybraného k řešení VRP úlohy a byly rozebrány principy výpočtů, které software využívá.

Samotná implementace algoritmů k řešení VRP je popsána v kapitole 5 v části 5.2, kde je úloha blíže rozebrána z pohledu svozu odpadu a je představen způsob použitý k řešení VRP úlohy.

4 Scheduling

Scheduling úlohy se dají popsat jako rozhodovací proces pro výrobní či jiné odvětví. Jedná se o úlohy, které mají za cíl přiřadit dostupné zdroje k úkolům/úkonům/činnostem, které se mají vykonat během nějakého časového úseku.

Zdroje a úkony k vykonání mohou mít obecně mnoho rozdílných podob. Zdroje mohou být například stroje v továrnách, ranveje na letištích, nákladní auta, může se jednat také o lidské zdroje například to mohou být dělníci na stavbě atd. Úkony poté mohou být například vzlety a přistání na ranveji, fáze výstavby atd., přičemž každý takový úkon může mít jinou prioritu, různou dobu trvání nebo třeba rozdílný čas, ve kterém je během dne proveditelný. Scheduling je tedy významnou součástí výrobních procesů, transportu zboží, distribuce a mnoha dalších odvětvích (Pinedo [37]).

Jak takový scheduling vypadá se dá ilustrovat na následujícím příkladě, který zhruba popisuje případ, kterým se zabývá tato práce. Uvažujme společnost na svoz odpadu, která má k dispozici nějaké konečné množství vozidel. Společnost má nasmlouvaný určitý počet obcí, ve kterých musí svážet všechen odpad. Trasy, po kterých v jednotlivých obcích bude odpad svezen, si společnost může zjistit vypočítáním VRP úlohy 3, jako řešení dostane seznam tras, které musí projet. Vzhledem k možným rozdílným frekvencím svozu odpadů (mezi obcemi se frekvence může lišit i u stejného typu odpadu) je třeba sestavit dlouhodobý plán (např. měsíční) a do tohoto plánu poskládat všechny dostupné trasy tak, aby byla zachována frekvence svozu, dostupnost vozidel a zároveň je nutné zahrnout všechny typy odpadu. Cílem je minimalizovat celkový počet použitých vozidel tak, aby byly splněny časové podmínky nastavené délkou pracovní doby. Tento konkrétní příklad je důležitou součástí této diplomové práce a je blíže rozebrán a popsán v kapitole 5 v části 5.3. Ve zbytku této kapitoly je popsáno teoretické pozadí schedulingu.

4.1 Formulace úlohy

Základní scheduling úloha popsaná v [28, 34] vypadá následovně:
Mějme množinu úkolů J, které musí být vykonány a množinu zdrojů (stroje nebo lidi) W. Každý úkol $j \in J$ musí být vykonán právě jedním (unikátním) zdrojem, přičemž úkol nemůže být započat na jednom zdroji a být dokončen na jiném. Každý úkol $j \in J$ je definován intervalem $[s_j, e_j]$, kde s_j označuje startovací čas úkonu a e_j určuje konečný čas. Přirozeným požadavkem je poté, aby zdroj přiřazený úkolu j nemohl vykonávat jiné úkoly během doby trvání intervalu $[s_j, e_j]$. Dalším přirozeným požadavkem, běžně se vyskytujícím, je fakt, že ne každý úkol může být vykonán na každém zdroji (např. trasa při svozu odpadu, může být naplánovaná pro jinou

velikost auta). Tato situace je popsána podmnožinu $J_w \subseteq J$, kde J_w značí úkoly, které může vykonávat zdroj w. Analogicky podmnožina $W_j \subseteq W$ určuje zdroje, které jsou schopné vykonávat úkol j. Tyto parametry tedy definují scheduling úlohu.

$J :=$ Množina úkolů k přiřazení$\{1, 2, ..., n\}$.

$W :=$ Množina zdrojů, které mohou vykonávat úkoly $\{1, 2, ..., m\}$.

$[s_j, e_j] :=$ Časový interval potřebný k vykonání úkolu $j \in J$.

$J_w \subseteq J :=$ Podmnožina úkolů, které mohou být přiřazeny zdroji $w \in W$.

$W_j \subseteq W :=$ Podmnožina zdrojů, schopná vykonávat úkol $j \in J$.

Cílem je poté vykonat všech n úkonů za použití co nejmenšího počtu strojů.

Kolen a kol. v [28] dále popisují několik následujících variant tzv. interval scheduling úloh:

- Scheduling úlohy s povinnými úkoly. Jedná se o variantu, ve které každý úkol musí být vykonán. Cílem je naleznout plán, ve kterém budou všechny činnosti a zároveň bude co nejmenší cena.
- Scheduling úlohy s fixním počtem strojů. Zde je dán počet úkonů a počet strojů, kde nastane zisk, když úkon j je proveden na stroji i. Cílem je pak naleznout plán, který maximalizuje zisk (v tomto případě se nemusí provést všechny úkony).
- Diskrétní scheduling úlohy. V těchto úlohách je místo pevného času začátku každého úkonu k dispozici diskrétní množina možných časů startu, ze kterých je možné vybrat.
- Online scheduling úlohy. Jedná se o variantu, ve které musí plánovač (osoba či stroj) rozhodnout, zda daný úkol zařadit do procesu před tím, než je známá informace o dalším úkonu. Cílem je maximalizovat celkovou délku časových intervalů jednotlivých úkonů s tím, že žádné dva intervaly se nesmí překrývat.

4.2 Hladový algoritmus

Hladové algoritmy (v angličtině Greedy algorithm) jsou jednoduchou a často používanou skupinou algoritmů k řešení optimalizačních úloh. Hladový algoritmus bývá popisován jako posloupnost rozhodnutí, kde každé rozhodnutí je to nejlepší, které v daný moment lze udělat, na základě dostupných informací, přičemž se algoritmus nevrací k dřívějším rozhodnutím. Jedná se o univerzální algoritmy, vhodné k mnoha problémům jako například komprese dat, sekvencování DNA nebo k hledání nejkratší cesty v grafu pomocí Dijkstrova algoritmu. Hladové algoritmy mohou být pro některé problémy použity jako heuristiky k nalezení přibližného řešení, pro další

mohou posloužit k nalezení dokonce přesného řešení. Nicméně neexistuje garance, že algoritmus dosáhne přesného řešení, proto je na to při tvorbě hladových algoritmů nutné myslet (Curtis [8]). Algoritmy lze obecně shrnout následovně:

Algoritmus 4.1 Hladový algoritmus

1: Načtení dat.
2: Určení podoby výstupu.
3: Definování kritérií pro pořadí, v jakém budou data do algoritmu vstupovat.
4: Postupně projít všechna data a podle definovaných kritérií rozhodnout, zda budou zahrnuta do výstupu. Jakmile je jednou rozhodnuto, tak už nelze toto rozhodnutí zvrátit.

4.3 Vehicle scheduling

V této kapitole byl zatím zmíněn obecný pohled na scheduling, nicméně je vhodné se podívat na konkrétnější podobu, která se přímo zabývá sestavováním plánu pro vozidla tzv. Vehicle Scheduling Problem (VSP). Tuto variantu schedulingu popisují Békési a spol v [2]. Do VSP jsou vstupními parametry množina vozidel V a množina vytvořených tras T, kde každá trasa je určena svou délkou, k níž se váže startovací a konečné místo a čas trasy. Ke každé trase je také přiřazena množina vozidel, které danou trasu mohou vykonat.

Vehicle scheduling přiřazuje vozidlům trasy, které se mají vykonat a zároveň určuje v jaké dny se trasy pojedou. Každé vozidlo je vázáno na centrální depo, ve kterém parkují, dochází v něm k údržbě, případně zde mohou vozidla i tankovat. Depo jednoznačně určuje začátek a konec každé trasy, jelikož na startu trasy auto vyjíždí z depa a na konci trasy se do depa musí opět vrátit. Důležité je zmínit, že flotila vozidel nemusí být homogenní, tedy se vozidla se od sebe mohou lišit a ne každému vozidlu může být přiřazena každá trasa. To může nastat z důvodu nedostatečné kapacity vozidla nebo třeba z důvodu, že vozidlo není pro daný typ trasy určeno (např. vozidlo může svážet jen bio odpad, ale trasa je naplánovaná pro svoz papíru).

Vehicle scheduling úlohy jsou obecně uváděny s časovou nebo vzdálenostní podmínkou a stejně tak tomu je i v této diplomové práci. Jedná se o přirozené podmínky, jelikož v reálném životě není možné pracovat neustále bez přestávky nebo se musí brát ohledy na maximální možnou vzdálenost, kterou vozidlo může najet. Všechna specifika a problémy, které se vztahují na studovaný problém jsou uvedena v následující kapitole 5 v části 5.3.

5 Implementace algoritmů

Tato kapitola se soustředí na řešení problémů a implementaci algoritmů teoreticky popsaných v kapitolách 2, 3 a 4. V první části kapitoly je popsán vývoj clusterovacího algortimu společně s daty, na které je použit. V další části je popsána konkrétní VRP úloha založená na reálných datech ze svazku obcí TSMH. Je zde také popsána komunikace mezi aplikací a výpočetním jádrem a následně je ukázána a popsána implementace některých funkcí z kódu pro výpočet VRP. V poslední části kapitoly se poté píše o způsobu, jakým bylo přistoupeno k problému schedulingu a formátu výstupu v jakém se data předávají zpět do aplikace.

5.1 Clustering

Tato část se věnuje implementaci a vývoji clusterovacího algoritmu. Vyvíjený algoritmus stojí na teoretických základech popsaných v kapitole 2. Nejdříve je uveden popis dat, na kterých byl samotný algoritmus ke clusteringu vyvíjen. A následně je uveden popis vývoje algoritmu a jsou ukázány některé funkce použité v konečné podobě algoritmu.

5.1.1 Data

Před samotným clusterovacím algoritmem je dobré si představit data, která byla v této práci použita k jeho vývoji. Jedná se o data z České republiky, konkrétně ze Zlínského kraje. Podoba dat je ukázána v následujících tabulkách 5.1, 5.2, 5.3 (pro zobrazení bylo nutné rozdělit jednu tabulku na tři části).

Adresa	Průměrný počet obyvatel
Vidče 598, Vidče, okres Vsetín, Zlínský kraj, Česko	5.313755796
Vigantice 331, Vigantice, okres Vsetín, Zlínský kraj, Česko	2.380841122
Vigantice 66, Vigantice, okres Vsetín, Zlínský kraj, Česko	2.380841122
Vigantice 327, Vigantice, okres Vsetín, Zlínský kraj, Česko	2.380841122
Vigantice 332, Vigantice, okres Vsetín, Zlínský kraj, Česko	2.380841122
Vigantice 326, Vigantice, okres Vsetín, Zlínský kraj, Česko	2.380841122
Vigantice 329, Vigantice, okres Vsetín, Zlínský kraj, Česko	2.380841122
Vigantice 328, Vigantice, okres Vsetín, Zlínský kraj, Česko	2.380841122
Vidče 601, Vidče, okres Vsetín, Zlínský kraj, Česko	2.656877898

Tab. 5.1: 1. část

Zeměpisná šířka	Zeměpisná délka	ORP	Obec
49.44126732	18.09210713	Rožnov pod Radhoštěm	Vidče
49.43516286	18.19909081	Rožnov pod Radhoštěm	Vigantice
49.44403027	18.18490966	Rožnov pod Radhoštěm	Vigantice
49.43778543	18.19985763	Rožnov pod Radhoštěm	Vigantice
49.43512722	18.20053046	Rožnov pod Radhoštěm	Vigantice
49.43842023	18.20103806	Rožnov pod Radhoštěm	Vigantice
49.43812172	18.19928894	Rožnov pod Radhoštěm	Vigantice
49.44007469	18.19581306	Rožnov pod Radhoštěm	Vigantice
49.44251304	18.08619091	Rožnov pod Radhoštěm	Vidče

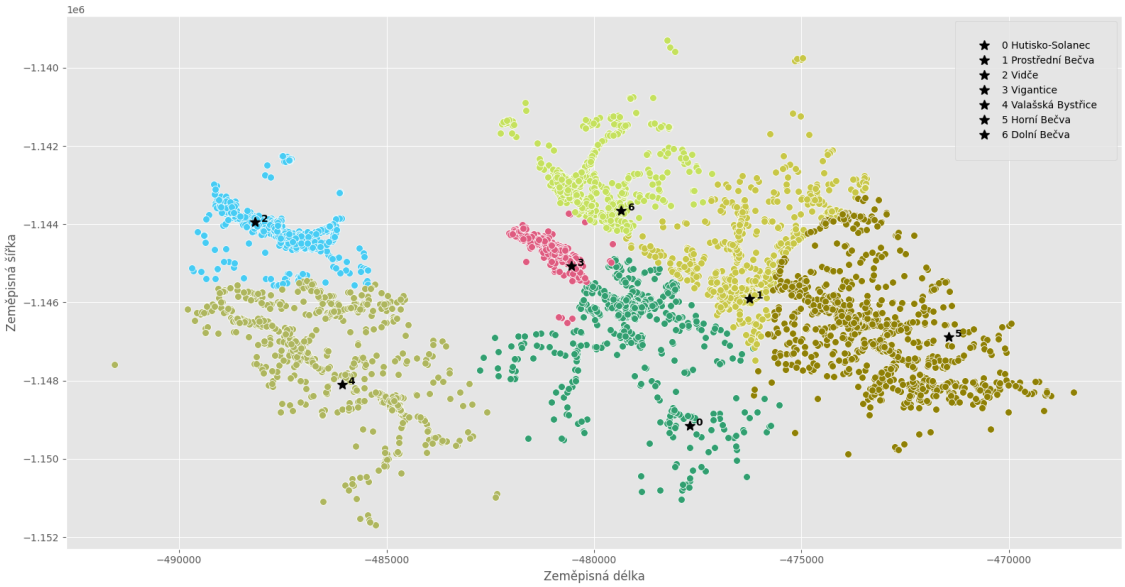
Tab. 5.2: 2. část

Část	X	Y
Vidče	-488112.5185481712	-1144056.4759078324
Vigantice	-480442.5779264894	-1145410.074815381
Vigantice	-481381.6585427539	-1144338.47454402
Vigantice	-480361.9179602309	-1145124.3251203285
Vigantice	-480338.89823193336	-1145423.0641721361
Vigantice	-480270.51823853486	-1145061.4045617005
Vigantice	-480399.76839906577	-1145083.494997159
Vigantice	-480632.08789707546	-1144845.2844007513
Vidče	-488527.6885809246	-1143880.7055619217

Tab. 5.3: 3. část

Vzhledem k účelu, za jakým vzniká tato práce, tedy vytvoření nástroje pro plánování svozových cest pro popelářská auta pro svazky obcí, tak není potřeba dělat clustering na všech bodech, ale stačí si vybrat nějakou podmnožinu, která dává smysl, tedy například body, které leží v sousedních obcích. Pro ukázku výsledků byla zvolena data vesnic z obce s rozšířenou působností Rožnov pod Radhoštěm. Jmenovitě se jedná o obce Hutisko Solanec, Prostřední Bečva, Vidče, Vigantice, Valašská Bystřice, Horní Bečva a Dolní Bečva.

V tabulkách výše lze vidět, že ke každému bodu je známá adresa, průměrný počet obyvatel (určeno na základě bytů na adresním bodě tj. jestli se jedná o rodinný domek, panelový dům atd. a na celkovém počtu obyvatel v dané obci), dále jsou známy zeměpisná šířka a délka dané v geodetickém standardu WGS84, obec s rozšířenou působností, ke které bod spadá, název obce, místní část obce a souřadnice X a Y dané transformací zeměpisné délky a šířky pomocí Křovákova zobrazení 2.1. Tyto body jsou vykresleny na obrázku 5.1.



Obr. 5.1: Vybrané obce

Nyní když jsou k dispozici data určená ke clusteringu, tak nastávají otázky, jakým způsobem data clusterovat. První otázkou je, zda data z vybraných obcí clusterovat jako celek, nebo si úlohu rozdělit a clustering provést na každé obci zvlášť. Oba způsoby jsou určitě obhajitelné a měly by své využití, pro potřeby této práce však byl zvolen druhý způsob, tedy provést clustering na každé obci zvlášť. Tento způsob byl zvolen převážně z důvodu lepší přehlednosti o tom, do jakých obcí budou jaká auta pro odpad jezdit a případně také tato varianta umožňuje, získané výsledky clusteringu uložit do databáze, což by urychlilo budoucí výpočty, jelikož by přidání nebo odebrání nějaké jiné obce nezměnilo výsledky clusteringu v ostatních obcích. Clustering by tedy stačilo jednou spustit na všech obcích a v budoucích výpočtech by se pouze načetly výsledky z databáze, čímž by se celý proces urychlil. Další otázkou je na kolik clusterů a na jak veliké clustery danou obec rozdělit. Ideální situace by byla, aby se data clusterovat vůbec nemusela a optimální cesta se hledala mezi všemi body ve všech obcích, tato možnost by ovšem vytvořila obrovský výpočetní problém, proto je vhodné obce rozdělit na menší části, ke kterým se pomocí regrese, provedené na straně webové aplikace, získají odhady stráveného času a najeté vzdálenosti. Zbývá tedy vyřešit otázku, na kolik obce rozdělit.

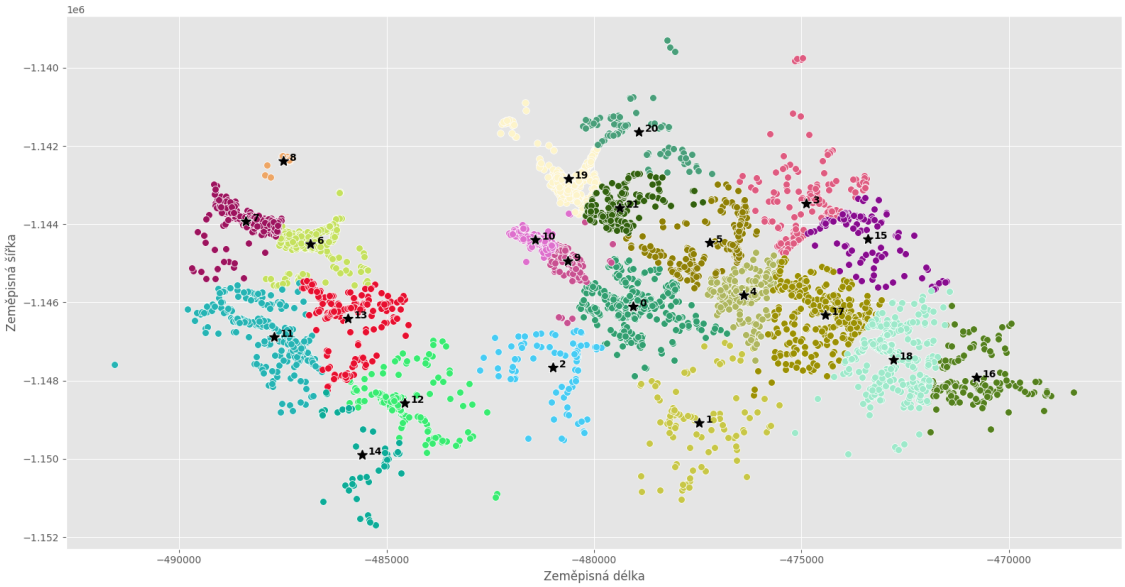
První odpověď na tuto otázka by mohla být, že by se obce rozdělily podle velikosti aut, přičemž produkce v jednom clusteru by odpovídala velikosti auta. S touto myšlenkou byla také vyvíjena první verze clusteringu nastalo ovšem několik problémů, kvůli kterým se tato myšlenka opustila.

První problém, který se během vývoje vyskytl, byl ten, jakým způsobem data clusterovat na přibližně stejně veliké clustery. Klasický K-means velikosti clusterů neurčuje, ten určí akorát počet clusterů, přičemž se clustery svou velikostí mohou výrazně lišit. Existují i varianty K-means, které omezují velikosti, žádná ovšem nepřinesla požadované výsledky. Další pokus byl vyzkoušení algoritmu na principu DBSCAN, jehož výsledky nebyly ideální. Jak již bylo zmíněno v kapitole 2, tak se jedná o algoritmus založený na hustotě bodů. Hustota zabydlení v obcích však může být velice rozdílná a tak výstupem nebyla konzistentní řešení.

Problém s velikostmi clusterů byl vyřešen až při práci na VRP modelu. Do VRP části vstupují regresní odhady času stráveného svozem odpadu pro vybraná místa. Z těchto odhadů vyšlo najevo najevo, že při svozu spíše skončí pracovní doba, než se naplní celé auto. Když by se tedy vytvářely clustery se stejnou velikostí, která by odpovídala např. 90% kapacitě vozidla, tak by pro danou úloho neexistovalo přípustné řešení, protože by čas svozu byl větší než pracovní doba. Velikost clusterů tedy musí taková, aby v nich bylo svoz možné vykonat. Clustery je tedy žádoucí vytvářet spíše menší než větší, není však jasné na čem by měla být velikost založena. Jelikož clustering není součástí VRP nebo VSP části, ale probíhá samostatně, tak během něho, není známo nic o kapacitě vozidel, které se použijí ke svozu. Musel by se tedy najít jiný parametr než velikost vozidel k určení podoby clusteru.

Takovým parametrem by mohla být pevně určená hodnota. To znamená, že produkce ve všech clusterech by přibližně byla rovna této zvolené hodnotě. Pevné zvolení velikosti clusterů by vytvořilo problém pro vozidla, která by měla menší kapacitu než je zvolená hodnota, protože by nemohly odpad svážet. Řeším by také nebylo, vytvořit příliš malé clustery, jelikož by do VRP vstupovalo velké množství bodů, čímž by se výrazně zpomalilo nalezení řešení. Definovat univerzální velikost clusterů, která by vyhovovala různorodým zadáním, ve kterých je možné mít i nehomogenní flotilu vozidel, je tedy velice obtížné. Bylo tedy rozhodnuto, že clustering se provede pomocí K-means++ algoritmu, který každou obec rozdělí do n clusterů, kde n je závislé na počtu obyvatel v dané obci. Je jasné, že v nově vzniklých clusterech může opět nastat situace, kdy by produkce v clusteru byla větší než kapacita vozidel, tento problém však bude řešen až v části VRP.

Modelový příklad použití K-means++ algoritmu je zobrazen na obrázku 5.2. Při zvolených hodnotách kapacity vozidla sedm tun a roční produkce daného odpadu na obyvatele 537 Kg. Z výsledku clusteringu je zřejmé, že K-means algoritmus neudělá stejně veliké clustery, což jak už bylo zmíněno není problém. Cílem clusteringu bylo rozdělit obce na menší části, ke kterým se pomocí regresních odhadů získají odhady produkce a času, což K-means zvládá velice obstojně.



Obr. 5.2: K-means++ použitý na data z obrázku 5.1

5.1.2 Implementace

Nyní budou popsány a ukázány . V následujícím textu jsou uvedeny vybrané části clusterovacího algoritmu, implementovaném v jazyce python, konkrétně se jedná o funkce k inicializaci počátečních centroidů pomocí K-means++ algoritmu, přiřazení bodů do clusterů, vypočítání nových centroidů a nalezení nejbližších existujících bodů k vypočítaným centroidům.

Inicializace centroidů pomocí K-means++

Funkce k vytvoření prvotních centroidů je založena na teoretickém popisu z části 2.1 kapitoly 2. Samotná implementace je ukázána na funkci 5.1. Vstupními hodnotami do funkce jsou body, ze kterých počáteční centroidy mají být vybrány. Dalším vstupním parametrem je číslo $k \in \mathbb{N}$ určující počet centroidů a tedy i počet clusterů. Výstupem funkce je seznam k bodů, které se uloží jako centroidy, ke kterým se ve funkci 5.2 přiřazují ostatní body.

Funkce 5.1: Získání počátečních centroidů

```
def getPPCentroids(townData, numberOfClusters):
    # Get random point from data set
    centroids = [townData[np.random.randint(townData.shape[0]), :]]
    for counter in range(numberOfClusters - 1):
        distances = []
        for i in range(townData.shape[0]):
            point = townData[i, :]
            smallestDistance = sys.maxsize
            # compute distance of point from each
             centroid and store the minimum distance
            for j in range(len(centroids)):
                # Get Euclidean distance between 2 points
                smallestDistance = min(smallestDistance, distance(point, centroids[j]))
            distances.append(smallestDistance)
        distances = np.array(distances)
        # select point with maximum distance to its
        centroid as a next centroid
        nextCentroid = townData[np.argmax(distances), :]
        centroids.append(nextCentroid)
    return centroids
```

Funkce 5.2: Přiřazení bodů do clusterů

```
def assignLocationsToClusters(self):
    for town in self.listOfUniqueTowns:
        townData = self.uniqueTownsAllData[town]
        # Get subset of town data
        extractedTownData = townData[["X", "Y"]].values
        # compute distance matrix from each point to each centroid
        distanceMatrixToCentroids =
        scipy.spatial.distance.cdist(extractedTownData, self.newCentroids.get(town))
        # Initialization of list of empty lists for each cluster
        self.townClusters[town] = [[] for _ in range(len(self.newCentroids[town]))]

        for locIndex in range(len(extractedTownData)):
            # Get index of the closest centroid
            sortedDistancesToCentroids =
            sorted(range(len(distanceMatrixToCentroids[locIndex])),
             key=lambda k: distanceMatrixToCentroids[locIndex][k])
            dataPoint =
            townData.iloc[[locIndex ]][[ "X", "Y", "Průměrný počet obyvatel"]].values
            # Assign point to cluster
            self.townClusters[town][sortedDistancesToCentroids[0]].append(dataPoint)
```

Tato funkce pro každou obec spočítá vzdálenosti všech jejích bodů k jejím centroidů a každý bod přiřadí do clusteru k jeho nejbližšímu centroidu.

Funkce 5.3: Vypočítání nových centroidů

```
def calculateNewCentroids(self):
    for town in self.listOfUniqueTowns:
        for i in range(len(self.newCentroids[town])):
            townData = np.array(self.townClusters[town][i])
            self.newCentroids[town][i] = np.average(townData[:, 0, 0:2], axis=0, weights=townData[:, 0, 2])
```

Nové centroidy, počítané funkcí 5.3, jsou získány pomocí váženého průměru bodů přiřazených danému centroidu, kde váhy jednotlivých bodů odpovídají průměrnému počtu obyvatel těchto bodů.

Protože během clusteringu jsou centroidy počítané jako průměry bodů, tak konečné centroidy se zcela jistě nebudou rovnat bodům z databáze. Po dokončení výpočtu je tedy nutné k těmto centroidům naleznout nejbližší body z databáze a ty označit za finální. O toto přiřazení se stará funkce 5.4

Funkce 5.4: Přiřazení existujících bodů

```
def assignClosestPointsToAverageAsCentorid(self):
    for town in self.listOfUniqueTowns:
        for i in range(len(self.centroids[town])):
            townData = np.array(self.townClusters[town][i])
            distanceMatrixToCentroid =
            scipy.spatial.distance.cdist(townData[:, 0, 0:2], np.array([self.centroids[town][i]]))
            sortedDistancesToCentroids =
             sorted(range(len(distanceMatrixToCentroid)), key=lambda k: distanceMatrixToCentroid[k])
             self.centroids[town][i] = townData[sortedDistancesToCentroids[0]][0, 0:2]
```

V případě využití clusteringu v aplikaci, by bylo možné výsledky uložit do databáze a z nich odhadnout hodnoty produkce odpadu a čas, který by se svozem odpadu v příslušných clusterech strávil. Tyto hodnoty by poté posloužily jako vstupní parametry do VRP úlohy. V první verzi vytvářené aplikace však clustering zahrnut nebude. Z výpočetní strany je však clustering připraven k využití v aplikace.

5.1.3 Shrnutí

Problémem vytváření clusterů výše zmíněným způsobem může být fakt, že algoritmus nedokáže rozpoznat smysluplnost vytvořených clusterů, jelikož nemá informace např. geografických údajích clusterované oblasti. Bylo by tedy vhodné, kdyby výsledek byl zkontrolován někým, kdo posoudí, zda je přijatelný, nebo by se měl výpočet spustit znovu třeba s upravenými parametry.

Jedním z možných vylepšení navrženého clusterování by bylo místo vzdušných vzdáleností uvažovat skutečnou vzdálenost po silnicích. To ale sebou přináší otázku, jak tyto vzdálenosti mezi všemi body získat. Jedním způsobem by mohlo být využití API od mapy.cz[3]. Tato API ovšem není ideálním řešením pro hledání vzdáleností mezi velkým množstvím bodů v reálném čase. Při práci s touto API bylo dosaženo

[3]https://api.mapy.cz/

(při množství asi tisíce dvojic bodů) přibližně minuty běhu programu. To by pro obec o tří stech adresních bodech (tedy by se musela počítat vzdálenost pro 90000 dvojic bodů) znamenalo, že by program běžel nejméně hodinu a půl.

Možnosti, jak co nejefektivněji získat vzdálenosti po silnicích mezi velkým množstvím bodů, tedy stojí za další výzkum. Pro potřeby této práce však zatím stačí uvažovat vzdušné vzdálenosti a různě veliké clustery.

5.2 VRP

Tato část popisuje implementaci řešení VRP pomocí OR-Tools [19]. Na rozdíl od clusteringu, se kterým se do první verze vytvářené aplikace nepočítá, tak řešení VRP je stěžejním kamenem, který je pro správný chod celé aplikace nezbytný. Aby bylo možné provést výpočet VRP úlohy, je nutné si určit podobu vstupních dat. Ta se získají tím, že si uživatel aplikace vybere obce, pro které chce svážet odpad. Zadají se typy odpadu určeného ke svozu a frekvence svozu pro daný odpad a obec. Také je potřeba označit depo, ze kterého budou auta vyjíždět a pro každý druh odpadu určit místa, kam se daný odpad bude svážet. Dalším vstupním parametrem je počet vozidel ke svozu a jejich kapacity. Na základě dat z databáze o produkci dané frakce odpadu a počtu obyvatel ve vybraných obcí, se na straně webové aplikace pomocí regresního modelu spočítají odhady času obsluhy sběrných míst v obci a odhad najetých kilometrů v daných obcích (tyto odhady nejsou součástí této práce). K těmto datům se z databáze přidají ještě vzdálenosti mezi jednotlivými obcemi, ty vznikly použitím, již zmíněné API od mapy.cz. Všechny tyto informace poslouží jako vstup do výpočetního jádra ve formě JSON souboru s následující strukturou:

```
{
  "task_id": "id",
  "depot_id": "depot",
  "areas": {a list of areas}
  "matrix_distances": {distance matrix between points from list of areas}
  "matrix_times": {time matrix between points from list of areas}
  "waste_fractions": {JSONs for each waste fraction}
}
```

Jak lze vidět, tak vstupní JSON má 6 parametrů. Jedná se o unikátní id úlohy, určené pro interní účely aplikace, dále je obsaženo id obce, ve které se nachází depo, seznam obcí, matice vzdáleností a časů a jako poslední parametr je dán seznam frakcí odpadu, který je v podobě vnořeného JSON objektu, ve kterém se nachází informace o každé vybrané frakci odpadu opět ve formě JSON objektu s následující strukturou.

```
{
  "fraction_id": "id",
  "frequency": {list of frequencies for each area"}
  "days": {binary values for each day}
  "hours": "working hours"
  "disposal": {binary value for each area}
  "cars": {a list of cars with their capacity}
  "area_times": {a list of times spent in each area}
  "area_distances" {a list of distances travelled inside of each area}
  "area_production" {a list of waste production for each area and current
    fraction}
}
```

To, o jakou frakci odpadu se jedná (SKO, papír, plast atp.), je jednoznačně určeno parametrem „fraction_id“, parametr „frequency“ určuje s jakou frekvencí se daná frakce v dané obci sváží (nejčastější případy jsou jednou týdně, dvakrát za měsíc a jednou za měsíc, může nastat ale i případ svozu jednou za tři týdny). Parametr „days“ udává, ve které dny týdne se příslušná frakce sváží (jedná se o binární proměnnou, kde 1 znamená, že se v daný den sváží a 0, že svoz neprobíhá. „cars“ určuje kolik aut je k dispozici, každé auto má své vlastní id a omezenou kapacit. Další parametr „area_times“ popisuje kolik času se v každé oblasti svozem odpadu stráví a stejně tak „area_distances“ udává kolik kilometrů se v každé oblasti najede. Poslední parametr „area_production“ určuje pro každou obec odhadovanou produkci odpadu.

5.2.1 Úprava vstupních dat

Před spuštěním samotného výpočtu je na datech nutné udělat preprocessing, tak aby do řešiče šla kompatibilní data. V prvním kroku je nutné si uvědomit, že ke každé frakci odpadu se při výpočtu tras přistupuje samostatně z důvodu, že auta, která by najednou svážela více druhů odpadů, tj. auta, která mají několik samostatných nádob, nejsou (nejméně v České republice) rozšířená. Stejně tak je nutné k jednotlivým frekvencím svozu přistupovat jednotlivě. Může nastat situace, kdy pro jeden druh odpadu je více frekvencí svozu (to může být případ, kdy ve svazku obcí bude město kde je třeba svážet odpad jednou týdně, ale také v něm bude malá vesnice, kde stačí svážet například jednou za dva týdny). Je tedy nutné ke každé frakci a frekvenci přistupovat individuálně.

Ze zadání je známo, že k dispozici je n vozidel pro každý den. Plán svozu se ovšem nedělá na jeden den, ale nejméně na týden v případě týdenní frekvence svozu. Za předpokladu pětidenního pracovního týdne, je teoreticky k dispozici až $5n$ vozidel. Je tedy nutné „virtuálně“ rozšířit vozový park, toho je dosáhnuto pomocí následující funkce *getAvailableVehicle()*.

Funkce 5.5: Získání celkového počtu vozidel

```
def getAvailableVehicles(listOfVehicles, numberOfDays):
    vehicleCapacities = []
    for _ in range(numberOfDays):
        vehicleCapacities.extend(listOfVehicles)

    return vehicleCapacities
```

V dalším kroku je nutné zkontrolovat hodnoty časů a produkce v jednotlivých oblastech, zda nepřekračují povolené hodnoty. U produkce odpadu je zapotřebí, aby nebyla větší než kapacita auta, jelikož poté by úloha automaticky neměla řešení. U času (stráveného v dané oblasti) je zase nutné, aby nebyl větší než pracovní doba, pak by opět řešení neexistovalo. U času ovšem nestačí jen tato podmínka, protože se musí myslet i na časy přejezdů mezi depem, obcí a zpracovatelským zařízením. Tato hodnota ovšem nelze nějak jednoduše odhadnout, jelikož pro každou obec se liší, proto byla při pokusech zvolena hodnota $\frac{3}{10}$ pracovní doby. Když tedy daná obec nesplňuje tyto podmínky, tak je nutné obec rozdělit na umělé části, které již podmínky budou splňovat. Tento postup lze shrnout následujícím algoritmem 5.1.

Algoritmus 5.1

numberOfDummyNodes = $\lceil$productionInArea/vehicleCapacity$\rceil$
timePerArea = timeInArea/numberOfDummyNodes
while timePerArea $> \frac{3}{10} \cdot workingHours$ **do**
    numberOfDummyNodes + = 1
    timePerArea = timeInArea/numberOfDummyNodes
end while
for i = 1 to numberOfDummyNodes **do**
    newProduction = productionInArea/numberOfDummyNodes
    newTime = timePerArea
    save newTime, newProduction
end for

Dále je nutné myslet na definici VRP úlohy, kde se každé místo může navštívit pouze jednou. Problém, který by nastal kvůli definici se vyskytuje pouze u svozových míst, proto je nutné data rozšířit o umělé svozovny, počet umělých svozoven je nastavený podle celkového počtu obcí, které do výpočtu vstupují. Tímto byly shrnuty hlavní úpravy, které je potřeba udělat na datech, aby se mohlo přistoupit k samotnému výpočtu.

5.2.2 Využití Or-tools

Jak už bylo mnohokrát zmíněno, tak k výpočtu řešení VRP úlohy je využíván software OR-tools. Pro řešení routingu využívá objektů zvaných dimenze, ty slouží pro sledování změn, které nastávají na trase. V případě svozu odpadu se jedná o změny naložení auta, vzdálenost, kterou vozidla ujedou a čas, který jim to trvalo. Je nutné tedy tyto dimenze definovat.

Funkce 5.6: Přidání kapacitní dimenze

```
1  def addCapacityConstraints(routing, manager, data, demandEvaluatorIndex):
2      """Add capacity constraint"""
3      routing.AddDimensionWithVehicleCapacity(
4          demandEvaluatorIndex,
5          data['vehicleCapacity'],  # capacity slack
6          data["vehicleCapacities"],  # vehicle maximum capacities
7          True,  # start cumul to zero
8          'Capacity')
9      capacityDimension = routing.GetDimensionOrDie('Capacity')
10
11     # Set zero slack variables for all nodes but disposals
12     for node in range(0, data['numberOfDividedNodes']):
13         nodeIndex = manager.NodeToIndex(node)
14         if data["demands"][nodeIndex] != 0:
15             capacityDimension.SlackVar(nodeIndex).SetMax(0)
16         else:
17             # Allow to drop areas with zero demand e.g. one fraction is not collected in 1 town
18             routing.AddDisjunction([nodeIndex], 0)
19
20     # Allow to drop dummy disposals nodes with zero cost.
21     for node in range(data['numberOfDividedNodes'], data['num_locations']):
22         nodeIndex = manager.NodeToIndex(node)
23         routing.AddDisjunction([nodeIndex], 0)
24
25     # Force to unload vehicle before returning to depot (if possible)
26     for v in range(manager.GetNumberOfVehicles()):
27         end = routing.End(v)
28         capacityDimension.SetCumulVarSoftUpperBound(end, 0, 100_000)
```

Kapacitní dimenze ovlivňuje vícero věcí. Řádky 3-9 do výpočtu přidávají hodnoty produkce a informace o použitých vozidlech. Cyklus na řádcích 12-18, nastavuje každému místu buď nulovou přidatnou proměnnou (slack variable), čímž se zaručí, že z každého místa bude vyzvednuta veškerá produkce odpadu, nebo v případě, že místo má nastavenou nulovou produkci, tak se označí a řešiči se povolí povolí nenavštívení tohoto místa. Stejně tak se na řádcích 21-23 povolí nenavštívení uměle vytvořených svozoven. Poslední cyklus přidává penalizaci pro případ, kdy by auto nebylo na konci své cesty v depu vyprázdněné. Prakticky to znamená, že vozidla jsou před koncem cesty nucena navštívit zpracovatelské zařízení, kde vyloží svůj náklad a poté se až vrátí do depa.

Funkce 5.7: Přidání časové dimenze

```
1  def addTimeWindowConstraints(routing, manager, data, time_evaluator):
2      """Add Time windows constraint"""
3      # Preparation for case when different vehicles could have different working hours
4      vehicleTimePool = []
5      for _ in range(len(data["vehicleCapacities"])):
6          vehicleTimePool.append(data["numberOfWorkingHours"])
7      routing.AddDimensionWithVehicleCapacity(
8          time_evaluator,
9          data["vehicleMaxTime"], # allow waiting time
10         vehicleTimePool, # maximum time per vehicle
11         False, # Decides whether vehicles have to start at exaxt time
12         'Time')
13     timeDimension = routing.GetDimensionOrDie('Time')
14
15     # Add time window constraints for each location, this effectively constraints duration of routes
16     # to set value (8 hours)
17     for locationId, timeWindow in enumerate(data['timeWindows']):
18         index = manager.NodeToIndex(locationId)
19         timeDimension.CumulVar(index).SetRange(timeWindow[0], timeWindow[1])
20         routing.AddToAssignment(timeDimension.SlackVar(index))
```

Další dimenze, která je nutná přidat je časová dimenze zobrazená na funkci 5.7. Časová dimenze slouží ke sledování času v průběhu jednotlivých tras, ale také k omezení celkové doby trvání trasy, toho je docíleno na řádcích 7-12 tím, že se každému autu omezí provozní doba. Omezení celkového času je také v cyklu na řádcích 17-20, kdy se každému místu nastaví časový interval, během kterého je možné toto místo navštívit. Celý svoz odpadu od vyjetí z depa, až po opětovný návrat do depa, musí být proveden během nastavené pracovní doby.

5.2.3 Výstup

Při následném spuštění řešiče (za předpokladu nalezení řešení) se pro každé auto dostane posloupnost bodů reprezentující obce, ve kterých má být svoz proveden. Mimo to je k dispozici, také celkový čas a najetá vzdálenost. To ovšem není vše, co je potřebné pro další část vypočtu. Vzhledem ke známosti podoby další části, tedy schedulingu, je nutné k trasám přidat další informace. Díky tomu, že mohou vznikat i trasy, které netrvají celý pracovní den, tak může nastat situace, že by bylo výhodné složit více tras z více frakcí do jedné za splnění podmínek na délku výsledné trasy. Situaci lze ilustrovat následovně:

Pro plast například vyjde deset tras, přičemž osm z nich trvá přibližně jeden pracovní den, další půl den a poslední jen tři hodiny. Stejná situace nastane i u svozu papíru. Oba druhy odpadu sváží stejný typ auta a oba odpady se sváží se stejnou frekvencí, proto je vhodné se zamyslet, zda by nešlo některé z kratších tras spojit. K tomu, aby šly spojit ovšem nestačí jen vypočítané řešení VRP úlohy, jelikož to udává pouze posloupnosti bodů, které začínají i končí v depu a tomu odpovídající

vzdálenosti a časy tras. Při spojení tras však nemusí být žádoucí, aby se auto po dokončení jedné trasy vracelo zpět do depa, když by mohlo vyrazit hned na další trasu z místa vyložení odpadu posbíraného na předešlé trase a ušetřit zbytečnou cestu do depa. Mohla by tedy nastat situace, kdy po krátkém sběru plastu, se plast vyloží na odpovídajícím svozovém místě a z tohoto místa se vyrazí na další trasu, tentokrát určenou ke sběru papíru. Návrat do depa ovšem může být i vyžadován, v případě kdy by auto bylo zašpiněné a muselo by se nechat umýt před tím, než by mohlo svážet další frakci odpadu. Tento příklad ukazuje, že je vhodné znát další informace o jednotlivých trasách, a to konkrétně délku a čas každé trasy, když by její začátek byl v depu a konec byl ve svozovém místě. K tomu je potřeba dopočítat délky a časy tras, které mají své počátky ve všech svozových místech všech frakcí odpadu a konce v depu nebo v odpovídajícím zpracovatelském zařízení. Všechny výše uvedené informace se přidají k vypočítaným hodnotám a předají se do části schedulingu v podobě do sebe vnořených pythonovských slovníků, které svou stromovou strukturou kopírují JSON soubor.

5.3 Scheduling

Tato část práce popisuje způsob řešení VSP úlohy. Scheduling zde plynule navazuje na výsledky z předchozí části 5.2, které je nyní nutné vložit do plánu svozu. Plánem svozu se myslí plán pro jednotlivá vozidla, kterým jsou přiřazeny získané trasy z VRP úlohy, přičemž se určí, v jaký den a týden se daná trasa pojede. Délka plánu se odvíjí od frekvencí svozu a vypočítá se jako nejmenší společný násobek těchto frekvencí.

Před startem scheduling algoritmu si je nutné rozhodnout zda pro skládání tras do plánu je důležitější frekvence svozu nebo druh odpadu. Je jasné, že se musí vykonat všechny naplánované trasy, kdyby tomu tak nebylo, tak by vznikaly nespokojenosti mezi zákazníky (obyvatelstvem), protože by jim nebyl odpad odvážen. Každá frakce odpadu může mít definované preferované dny, ve kterých se má svážet. Dny svozu ovšem nelze obecně určit, protože při každém výpočtu se mohou nastavit rozdílné dny. S nastavováním preferovaných dnů svozu by v případě sestavování plánu podle pevně daného pořadí frakcí mohly vzniknout komplikace ilustrované následující situací. Ve výpočtu je pevně určené pořadí frakcí odpadu, v jakém se mají trasy do plánu skládat. První v pořadí je frakce, kterou není potřeba svážet tak často, může to být tedy frakce s frekvencí svozu jednou za měsíc. Další frakce v pořadí je svážena s týdenní frekvencí. Obě frakce přitom mají stejně nastavené preferované dny svozu. Mohlo by se tedy stát, že trasa s nižší frekvencí by v jeden den plánu zabírala místo trase s vyšší frekvencí. U svozu odpadu je důležité, aby se stejná trasa jezdila vždy ve stejný den tedy např. týdenní trasa naplánovaná na

pondělí musí být vždy vykonána v pondělí a nemůže nastat situace, že tři týdny by byla v pondělí, a čtvrtý týden byla v úterý, protože na pondělí by byla naplánovaná trasa s nižší frekvencí. Této situace se lze vyhnout tím, že se plán bude sestavovat podle frekvence svozu, a ne podle frakce odpadu.

5.3.1 Sestavení plánu

Pro přehlednost a jednodušší kontrolu toho, jak skládání probíhá, je samotný algoritmus, určený ke skládání tras do plánu, rozdělen na tři části. První část slouží ke kontrole, zda trasy jedné frakce (a frekvence) nelze udělat delšími. V druhé části se algoritmus snaží spojit trasy z rozdílných frakcí a poslední třetí část je určena k samotnému skládání tras do plánu. Jednotlivé části budou nyní podrobněji rozebrány.

První část

Jak už bylo zmíněno, tato část pro každou frakci a frekvenci kontroluje, jestli vypočítané trasy nelze mezi sebou spojit. Pro ilustraci situace je nejlépe uvést analogický příklad k příkladu z části 5.2.3. V této situaci je tedy cílem složit trasy stejné frakce. Pro frakci odpadu s danou frekvencí vyjde např. deset tras. Osm z nich trvá přibližně jeden osmihodinový pracovní den a je tedy zřejmé, že ty spojit nelze. Zbylé dvě ovšem trvají okolo čtyř hodin, jedna např. 4 h a druhá 4,2 h nabízí se tedy možnost zjistit, zda by při jejich spojení nevznikla trasa, která by trvala méně než osm hodin.

Na první pohled tyto dvě trasy spojit nelze, jelikož $4 + 4{,}2 = 8,2 > 8$. Tyto časy ovšem určují, jak dlouho trasy trvají, když začínají a končí v depu. Když by se trasy spojily, tak po ukončení první nemusí dojít k návratu do depa čímž se čas první trasy zkrátí o cestu ze svozovny do depa. Čas druhé trasy bude také rozdílný, protože tato trasa by už nezačínala v depu, ale ve svozovně, ve které skončila první trasa, proto čas druhé trasy se může i zvýšit, jelikož tato svozovna může být dále od prvního uzlu druhé trasy, než je depo.

Výsledný čas může vyjít kratší než osm hodin, ale také může vyjít i delší než v případě jednoduchého sečtení původních časů, které odpovídá situaci, kdy po dokončení první trasy se vozidlo navrátí zpět do depa, odkud pokračuje druhou trasou. Toto spojování tras je shrnuto v následujícím pseudokódu 5.2, jehož výsledkem je seznam tras, který vstupuje do druhé části.

Algoritmus 5.2 Spárování tras

1: Vybrat všechny trasy se stejnou frakcí odpadu a frekvencí svozu.
2: Seřadit vybrané trasy od nejdelší po nejkratší.
3: $b :=$ časová hranice: pracovní doba - čas nejkratší trasy.
4: $r :=$ nejdelší nepoužitá trasa
5: $t_r :=$ čas trasy r
6: **if** $t_r \leq b$ **then**
7: $\quad s :=$ nejkratší nepoužitá trasa
8: $\quad t_s :=$ čas trasy s
9: $\quad$ **while** $t_r + t_s <$ pracovní doba **do**
10: $\quad\quad r^{disp} :=$ trasa r končící v depu
11: $\quad\quad t_r^{disp} :=$ čas trasy r z depa do svozovny
12: $\quad\quad s^{dep} :=$ trasa s začínající ve svozovně a končící v depu
13: $\quad\quad t_s^{dep} :=$ čas trasy s ze svozovny do depa.
14: $\quad\quad$ **if** $t_r^{disp} + t_s^{dep} <$ pracovní doba **then**
15: $\quad\quad\quad$ Spojit trasy r^{disp} a s^{dep} do nové trasy rs^{disp}
16: $\quad\quad\quad r = rs^{disp}$
17: $\quad\quad\quad t_r = t_r^{disp} + t_s^{dep}$
18: $\quad\quad$ **else**
19: $\quad\quad\quad$ Spojit trasy r a s do nové trasy rs
20: $\quad\quad\quad r = rs$
21: $\quad\quad\quad t_r = t_r + t_s$
22: $\quad\quad$ **end if**
23: $\quad\quad s :=$ nejkratší nepoužitá trasa
24: $\quad\quad t_s :=$ čas trasy s
25: $\quad$ **end while**
26: **end if**
27: Uložit výslednou trasu r
28: Opakovat kroky 4-27 do doby, než budou využity všechny trasy

Bylo zmíněno, že tato část slouží jen pro kontrolu. Je to z toho důvodu, že k výše popsané situaci by mělo docházet jen velmi zřídka. Když by tato situace nastala, znamenalo by to, že výpočet VRP neproběhl ideálně, protože naplánoval více tras, než je nezbytné. Je to cena za použití heuristik místo exaktního řešiče, která je však kompenzována rychlostí výpočtu.

Druhá část

Druhá část je ve své podstatě velice podobná části první. Nespojují se zde však trasy jedné frakce, ale trasy z odlišných frakcí, ale se stejnou frekvencí svozu. Situace, kdy by šly spojit dvě trasy nebo více tras do jedné, je tedy pravděpodobnější. Algoritmus spojující trasy více frakcí je takřka totožný s algoritmem 5.2. Rozdíl je na prvním řádku, kdy se nyní pracuje se všemi frakcemi odpadu, které mají stejnou frekvenci svozu. Dalším rozdílem, který přibude, je nutná kontrola, zda vybrané trasy lze svážet ve stejné dny.

Výstupem algoritmu je seznam tras (pro všechny frakce), které už nelze nijak spojit. A je tedy možné z nich nyní sestavit konečný plán svozu.

Třetí část

Třetí část slouží k sestavení plánu svozu. Při tvorbě svozového plánu je nutné se držet předem definovaných pravidel. To asi nejdůležitější už bylo v této práci zmíněno a jedná se o podmínku, že stejná trasa se pokaždé musí vykonat ve stejný den. Tato podmínka zabraňuje například situaci, kdy by trasa s týdenní frekvencí byla jeden týden naplánovaná například na neděli a další týden hned na pondělí. V takovém plánu by byla zachována frekvence svozu jednou za týden, z logického hlediska by takový plán však nedával smysl. Dalším pravidlem je, že sestavování plánu probíhá od tras s nejvyšší frekvencí svozu po nejnižší frekvenci, důvody jsou shrnuty v úvodu této části kapitoly. Základní myšlenky algoritmu určeného k sestavení plánu jsou ukázány v pseudokódu 5.3.

Algoritmus 5.3 Sestavení plánu

```
1: Trasy := množina všech tras pro počítanou frekvenci
2: V := množina dostupných vozidel
3: N := množina nezařazených tras
4: f := délka tvořeného svozového plánu
5: for each r ∈ Trasy do
6:     D := preferované dny svozu pro frakci trasy r
7:     t_r := čas trasy r
8:     for each v ∈ V do
9:         v_r := vozidlo pro trasu r
10:        if v_r == v then
11:            t_d := volný čas ve dni d v plánu pro vozidlo v, d ∈ D
12:            if t_r + t_d < pracovní doba then
13:                Uložit trasu t do plánu pro vozidlo v a den d
14:            else
15:                C := množina f vozidel, se stejným preferovaným dnem svozu
16:                t_dn := volný čas ve dni dn množiny C, dn ∈ D
17:                if t_r + t_dn < pracovní doba then
18:                    Uložit trasu t do plánu pro vozidla c ∈ C a den d
19:                else
20:                    Uložit d do N
21:                end if
22:            end if
23:        end if
24:    end for
25: end for
```

Algoritmus 5.3 stručně popisuje postup sestavování svozového plánu. Pro každou trasu se algoritmus snaží najít vozidlo, které může trasu v její preferované dny vykonat. Primární snaha je přiřazení jednoho vozidla ke stejné trase po celou délku svozového plánu. V případě, že trasa nelze přiřadit jen jednomu vozidlu, tak se zkontroluje, jestli nelze přiřadit do plánu vícero vozidlům. Když by ani to nebylo možné, tak se trasa uloží do samostatného seznamu, který se zpracuje později.

Tímto způsobem se algoritmus pokusí zařadit všechny trasy do svozového plánu. Výsledkem je plán svozu požadované délky pro každé auto, každý týden a každý den plánu. Pro některé trasy může ovšem nastat situace, že nemohou být naplánovány k jejich preferovaným dnům. V takovém případě vzniká otázka, jak s takovou situací naložit. První způsob je celý scheduling ukončit se zprávou pro uživatele, že pro zadanou úlohou nelze nalézt řešení. Druhý způsob je označit úlohu, že nemá řešení

splňující zadané parametry, ale pokusit se nezařazené trasy naplánovat na jiné dny a uložit jaké trasy nejsou optimálně zařazené a ty uživateli zvýraznit. V případě, že by ani to nebylo možné, tak nezbývá nic jiného než výpočet ukončit.

V případě, že se výpočet neukončí, tak je nutné předat získaný plán zpět do aplikace a zobrazit výsledky uživateli. Podoba výstupu byla domluvena na JavaScript Object Notation (JSON) soubor s následující strukturou:

```
{
  "task_id": "id",
  "feasibility": [True/False],
  "statistics": [basic statistics],
  "weeks": {
        0: {  # week index
           0: { # vehicle ID
               0: [route] # Mon
               1: [route] # Tue
               2: [route] # Wed
               3: [route] # Thu
               4: [route] # Fri
               5: [route] # Sat
               6: [route] # Sun
           }
           1:{}
           .
           .
           .
          N-1:{}
        }
        1:{}
         .
         .
         .
        f-1:{}
  }
}
```

Parametr „task_id“ slouží jako identifikátor úlohy do databáze. „feasibility“ parametr určuje, zda má úloha přípustné řešení. Parametr „statistics“ v sobě zahrnuje základní údaje o počtu najetých kilometrů a času. Poslední parametr „weeks“ je samotný plán svozu, který je ve formě dalšího vnořeného JSON objektu, který má v sobě vnořeno f dalších JSON objektů, kde f je délka plánu. V případě čtyřtýdenního plánu by se tedy jednalo o čtyři vnořené objekty, kde by každý reprezentoval jeden týden. V každém z těchto objektů je vnořeno dalších N objektů, které odpovídají jednotlivým vozidlům, pro které je plán sestavován. Objekty reprezentující vozidla mají poté ke dnům v týdnu přiřazenou trasu, kterou mají v daný den vykonat. Takto

vytvořený JSON soubor se předá zpět do aplikace, aby mohl být zobrazen uživateli.

5.3.2 Shrnutí

V této části diplomové práce byl shrnut způsob tvoření svozového plánu na základě výsledků získaných z části VRP. Svozový plán je vytvořen ve formě JSON souboru, který bude předán zpět do webové aplikace, ze které vzniklo zadání celé úlohy. Výsledky poté budou pro uživatele zpracovány ve webové aplikaci do přehledné podoby. Zpracované výsledky jsou ukázány v následující kapitole 6, která se věnuje konkrétnímu ukázkovému příkladu.

6 Případová studie

6.1 TSMH

Vstupní data pro ukázkový příklad byla vytvořena v rámci práce [30]. Zadání vzniklo pro 51 obcí ze svazku obcí TSMH. Tento svazek disponuje šesti vozidly s depem v Letovicích. Plán svozu se vytváří pro 5 frakcí odpadu jmenovitě pro SKO, plast, papír, sklo a bio odpad. Každá frakce odpadu má přiřazené jedno či více zpracovatelských zařízení, která mohou, ale nemusí, být součástí svazku. Odpady a jejich zpracovatelská zařízení jsou zobrazeny v tabulce 6.1. Dostupná vozidla s jejich kapacitou jsou zobrazeny v tabulce 6.2

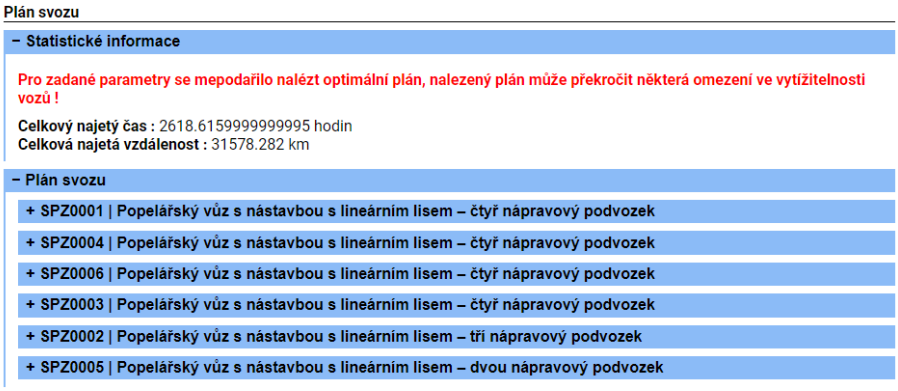
Obec se zpracovatelským místem	Odpad
Horní Smržov	SKO
Blansko	BIO
Letovice	Papír, SKO, plast
Kyjov	sklo

Tab. 6.1: Zpracovatelská zařízení

Označení vozidla	Objem [l]	Nosnost [t]
1	30000	10
2	24000	8
3	30000	10
4	30000	10
5	20000	8
6	30000	10

Tab. 6.2: Dostupná vozidla

Po získání výsledku se v aplikaci zobrazí navržený svozový plán, zobrazený na obrázku 6.2. Jako první jsou ukázány základní statistiky týkající se najeté vzdálenosti a času v rámci celého plánu. Když zadaná úloha nelze podle požadovaných parametrů splnit, ale stále existuje řešení, tak se ve statistických informací zobrazí hláška, že nebyly dodrženy všechny nastavené podmínky, ale i tak se zobrazí vypočítaný plán. Tato situace může nastat například v případě, že by v zadání pro všechny frakce byl nastaven stejný den svozu a nebylo by k dispozici dostatečné množství vozidel. V případě, že by celé zadání bylo navržené nevhodně, tak by výpočet VRP mohl proběhnout neúspěšně a byl by zobrazen prázdný plán.



Obr. 6.1: Zobrazení vozidel v aplikaci

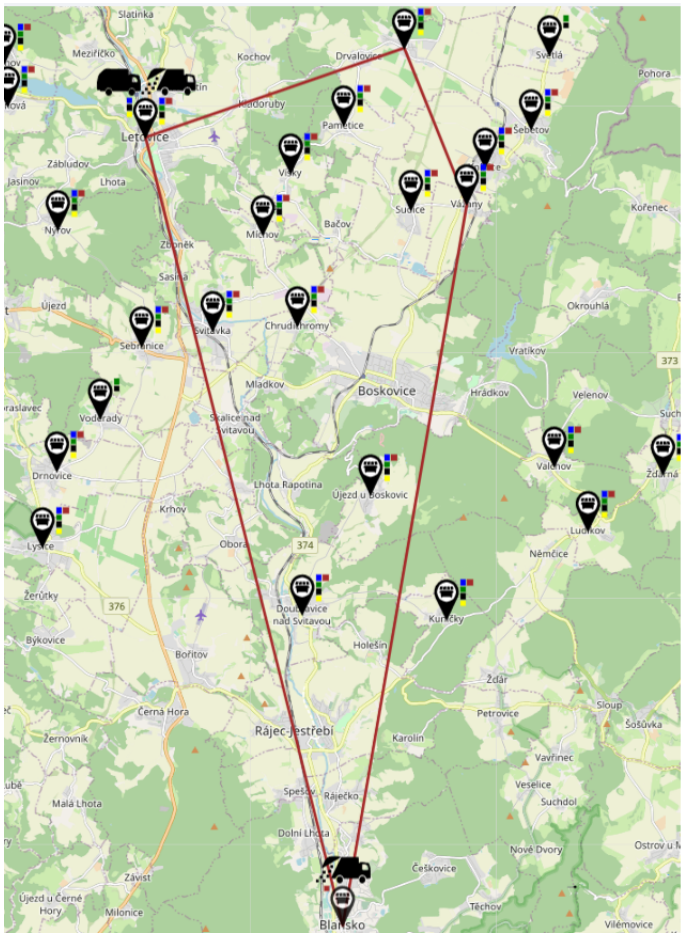
Plán je navržen pro jednotlivá auta a pro každé je vytvořen podrobný kalendář. V něm si lze prohlédnout detail s informacemi o trasách, které se mají vykonat. Na obrázku 6.2 je zobrazena část kalendáře. V každém dni jsou informace o svážené frakci a obcích. Když je v kalendáři několik stejných obcí za sebou, odpovídá to rozdělení obce do více clusterů. Například na obrázku 6.2 lze vidět, že v pondělí první týden má vozidlo naplánovanou trasu pro bio odpad, vyjíždí z depa v Letovicích, kde sesbírá jeden navržený cluster, odtud se přesune do Vanovic, kde se zdrží ve dvou clusterech, následně pojede do Vážan odkud vyrazí do jediného zpracovatelského zařízení pro bio odpad, které se nachází v Blansku. Na konci se poté vrátí zpět do depa v Letovicích.

– plán tras

	1. týden	2. týden	3. týden	4. týden
Po	BIO	SKO	BIO	SKO
	Letovice, Letovice, Vanovice, Vanovice, Vážany, Blansko, **Letovice**	**Letovice**, Prostřední Poříčí, Studlov, Chrastavec, **Letovice**	**Letovice**, Letovice, Vanovice, Vanovice, Vážany, Blansko, **Letovice**	**Letovice**, Prostřední Poříčí, Studlov, Chrastavec, **Letovice**
		Letovice, Voděrady, Voděrady, Letovice, **Letovice**		**Letovice**, Voděrady, Voděrady, Letovice, **Letovice**
Ut	BIO	PAPIR	BIO	SKO
	Letovice, Letovice, Letovice, Doubravice nad Svitavou, Blansko, **Letovice**	**Letovice**, Nýrov, Nýrov, **Letovice**	**Letovice**, Letovice, Letovice, Doubravice nad Svitavou, Blansko, **Letovice**	**Letovice**, Benešov, Benešov, ***Horní Smržov***
		Letovice, Skrchov, Stvolová, Rozhraní, Vanovice, Letovice, **Letovice**		***Horní Smržov***, Bělá u Jevíčka, Bělá u Jevíčka, ***Horní Smržov***
				Horní Smržov, Bělá u Jevíčka, Bělá u Jevíčka, Letovice, **Letovice**

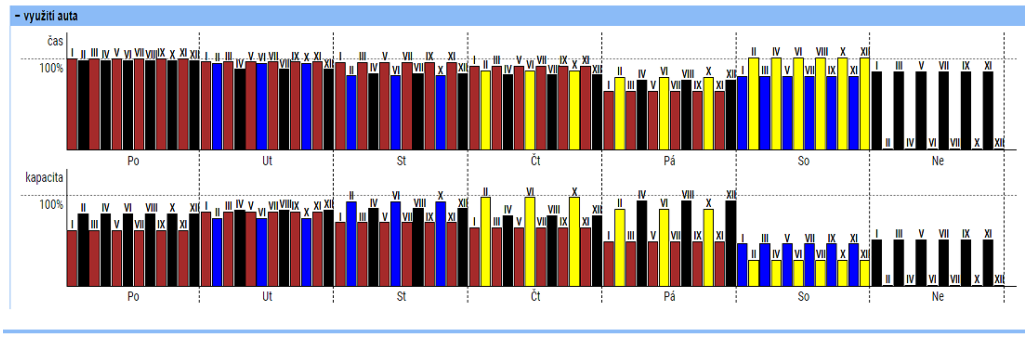
Obr. 6.2: Kalendář

Po kliknutí do kalendáře, se v mapovém podkladu zobrazí trasa pro vybraný den. Každá frakce odpadu má k sobě přiřazenou barvu, kterou je vykreslována. Na obrázku 6.3 je zobrazena trasa pro popelářské auto určené na bio odpad s depem v Letovicích a zpracovatelským zařízením v Blansku.



Obr. 6.3: Mapový podklad

Na obrázku 6.4 je zobrazeno vypočítané využití vybraného auta během celého svozového plánu. K dispozici je pro každý den časová a kapacitní náročnost přiřazených tras. Z obrázku je taky patrné, že naplánované trasy pro vybrané vozidlo, se se svou délkou blíží nastavené časové hranici, tedy jejich délky přibližně odpovídají délce pracovní doby.



Obr. 6.4: Využití vozidla

Vzhledem k rané fázi vývoje výpočtového jádra i webové aplikace, nemusí vypočítané hodnoty přesně reflektovat skutečnost. Jedná se však o dobrý základ pro další vývoj.

6.2 Další vývoj v oblasti algoritmizace

Výsledný algoritmus, který byl integrován do webové aplikace pro uživatelsky přívětivou obsluhu, má několik limitujících faktorů. Očekávají se tedy jeho další vylepšení a úpravy, které povedou k lepšímu nasazení celého softwaru do praxe.

První důležitou funkcionalitu představuje možnost manuální editace výsledného plánu uživatelem. Očekává se, že v navrženém plánu bude možné ručně měnit složení jednotlivých tras, přesouvat zařazení tras v rámci dnů nebo dokonce vozidel. Takto modifikovaný plán v grafické podobě je nutné opět vyhodnotit z pohledu času svozu, celkově najeté vzdálenosti a hlavně proveditelnosti. Pro tuto kontrolu a přepočítání musí vzniknout nový algoritmus nebo modifikace představených postupů, která se postará o jednotlivé aspekty vyhodnocení.

Dále se očekává přechod na nižší detail obecního dělení pro územní celky, u kterých je to možné, tj. obecní/městské části. V případě obce, která se skládá z hlavní části a okolních přidružených vesnic (např. Letovice mají 17 místních částí), je potřeba přijít se způsobem napojení těchto bodů na existující matice vzdáleností a času. S implementací silniční infrastruktury je úzce spojena také integrace vyvinutého shlukovacího algoritmu pro obce s produkcí odpadu větší, než je kapacita nejmenšího vozidla. Pro různé kapacity vozidel bude omezení pro maximální velikost clusteru dáno největším společným dělitelem, aby byla zachována možnost vkládat clustery do libovolného vozidla. S ohledem na různé typy odpadů a jejich vlastnosti (sypná hmotnost) musí být clustering aplikován pro všechny odpady samostatně.

Z praktických důvodů může být požadováno fixní přiřazení některých odpadů ke konkrétním vozidlům (nebo jejich typům). Tento požadavek by měl být zahrnut

v algoritmu řešící VRP, a to v omezení dostupnosti některých kapacit v průběhu iterování mezi všemi typy odpadu a frekvencemi. Svazky obcí se v současné době těší velkému rozmachu, a tak je běžné, že meziročně v každém existujícím svazku přibude nějaká obec. Rozšiřování svazku však významným způsobem ovlivňuje svozový plán. Techničtí pracovníci a obyvatelé obcí ze svazku nechtějí při každé expanzi svazku kompletně měnit svozové plány a dny, ve kterých jsou jejich kontejnery obsluhovány. Důsledkem je tedy požadavek na úpravu svozového plánu s co nejmenšími změnami tak, aby byl výsledný svoz s novými obcemi ekonomický a hlavně proveditelný.

Výše uvedené body představují požadavky z praxe na úpravu algoritmů. Očekává se, že budou řešeny v rámci další činnosti na ÚPI a v navazujících závěrečných pracích ve spolupráci s Ústavem matematiky.

Závěr

Cílem práce bylo představení problematiky logistiky přepravy odpadu na meziobecní úrovni, vývoj a implementace algoritmů pro agregaci adresních míst a tvorbu dopravní sítě a následné provedení případové studie na reálných datech.

Práce je strukturována tak, aby byl čtenář seznámen s celou problematikou svozu odpadu. Na úvodní kapitolu, ve které jsou zmíněny základní informace týkající se odpadového hospodářství, navazuje kapitola věnována tzv. clusteringu. V té je uveden matematický popis úlohy, její různé podoby a způsoby řešení.

Třetí kapitola popisuje tzv. Vehicle routing problem (VRP). Je v ní představen základní model úlohy TSP, který je následně rozšířen na model VRP. Dále jsou ve třetí kapitole rozebrány některé v praxi používané varianty VRP a také způsoby, jimiž lze přistupovat k řešení úlohy. Závěrem této kapitole je rozebrán software OR-tools využitý při tvorbě konečné aplikace.

Následuje čtvrtá kapitola, věnována problematice schedulingu, v ní je představena základní úloha, způsob řešení a rozšíření na tzv. Vehicle scheduling problem, který nejlépe odpovídá problému, kterému se tato práce věnuje. Tedy plánování tras pro popelářská auta.

Vývojem algoritmů popsaných v kapitolách 2-4 se zabývá pátá kapitola. V první části je popsán clusterovací algoritmus, na který navazuje popis implementace softwaru OR-tools. Závěr kapitoly se zabývá úlohou schedulingu a její aplikace na výsledky z VRP. Ke každé z dílčích úloh, v této kapitole jsou ukázky přímo pythonovského kódu nebo je uveden pseudokód.

V poslední šesté kapitole je uveden ukázkový příklad pro obce ze svazku obcí TSMH. Výsledky jsou graficky interpretovány pomocí webové aplikace k tomu určené. Jako poslední jsou zmíněny možnosti dalšího vývoje algoritmizace clusteringu a VRP.

Výstupem práce je soubor kódů v pythonu, které se integrují do webové aplikace, určené k vytváření plánu svozu odpadu. Budoucí vylepšení, celého systému plánování odpadového hospodářství, mohou zahrnovat lepší vytváření clusterů nebo například zahrnutí více podmínek do VRP úlohy a schedulingu.

Literatura

[1] Baldacci, R., Bartolini, E., Laporte, G. Some applications of the generalized vehicle routing problem (2010) Journal of the Operational Research Society, 61 (7), pp. 1072-1077. DOI: 10.1057/jors.2009.51

[2] BÉKÉSI, József, Balázs DÁVID a Miklós KRÉSZ. Integrated Vehicle Scheduling and Vehicle Assignment. Acta Cybernetica [online]. 2018, 23(3), 783-800 [cit. 2022-05-15]. ISSN 0324-721X. Dostupné z: doi:10.14232/actacyb.23.3.2018.4

[3] BRAEKERS, Kris, Katrien RAMAEKERS a Inneke VAN NIEUWENHUYSE. The vehicle routing problem: State of the art classification and review. Computers & Industrial Engineering [online]. 2016, 99, 300-313 [cit. 2022-03-14]. ISSN 03608352. Dostupné z: doi:10.1016/j.cie.2015.12.007

[4] Bräysy, O., Gendreau, M. Vehicle routing problem with time windows, Part I: Route construction and local search algorithms (2005) Transportation Science, 39 (1), pp. 104-118. DOI: 10.1287/trsc.1030.0056

[5] Cattaruzza, D., Absi, N., Feillet, D., Vigo, D. An iterated local search for the multi-commodity multi-trip vehicle routing problem with time windows (2014) Computers and Operations Research, 51, pp. 257-267. Dostupné z doi:10.1016/j.cor.2014.06.006

[6] Cebi, C., Atac, E., Sahingoz, O.K. Job Shop Scheduling Problem and Solution Algorithms: A Review (2020) 2020 11th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2020, art. no. 9225581, DOI: 10.1109/ICCCNT49239.2020.9225581

[7] Circular Economy Strategy - Environment - European Commission. ec.europa.eu [online]. [cit. 2022-04-12]. Dostupné online.

[8] CURTIS, S.A. The classification of greedy algorithms. Science of Computer Programming [online]. 2003, 49(1-3), 125-157 [cit. 2022-05-14]. ISSN 01676423. Dostupné z: doi:10.1016/j.scico.2003.09.001

[9] Davidson, I., Ravi, S.S. Clustering with constraints: Feasibility issues and the k-Means algorithm (2005) Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005, pp. 138-149. DOI: 10.1137/1.9781611972757.13

[10] DAVIS, Sashka a Russell IMPAGLIAZZO. Models of Greedy Algorithms for Graph Problems. Algorithmica [online]. New York: Springer-Verlag, 2007, 54(3), 269-317 [cit. 2022-05-08]. ISSN 0178-4617. Dostupné z: doi:10.1007/s00453-007-9124-4

[11] DBSCAN Clustering Algorithm in Machine Learning [online]. [cit. 2022-03-05]. Dostupné z: https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html

[12] DREXL, Michael. Rich vehicle routing in theory and practice. Logistics Research [online]. 2012, 5(1-2), 47-63 [cit. 2022-03-28]. ISSN 1865-035X. Dostupné z: doi:10.1007/s12159-012-0080-2

[13] Eghtesadifard, M., Afkhami, P., Bazyar, A. An integrated approach to the selection of municipal solid waste landfills through GIS, K-Means and multi-criteria decision analysis (2020) Environmental Research, 185, art. no. 109348, DOI: 10.1016/j.envres.2020.109348

[14] EVERITT, Brian. Cluster analysis. 5th ed. Chichester: Wiley, 2011, xii, 330 s. : il. ISBN 978-0-470-74991-3.

[15] Ganganath, N., Cheng, C.-T., Tse, C.K. Data clustering with cluster size constraints using a modified k-means algorithm (2014) Proceedings - 2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2014, art. no. 6984299, pp. 158-161. DOI: 10.1109/CyberC.2014.36

[16] Gendreau, M., Laporte, G., Musaraganyi, C., Taillard, E.D. A tabu search heuristic for the heterogeneous fleet vehicle routing problem (1999) Computers and Operations Research, 26 (12), pp. 1153-1173. DOI: 10.1016/S0305-0548(98)00100-2

[17] Gendreau, M., Ghiani, G., Guerriero, E. Time-dependent routing problems: A review (2015) Computers and Operations Research, 64, art. no. 3801, pp. 189-197. DOI: 10.1016/j.cor.2015.06.001

[18] Goel, R. and Maini, R. (2017) Vehicle routing problem and its solution methodologies: a survey, Int. J. Logistics Systems and Management, Vol. 28, No. 4, pp.419–435.

[19] Google OR-Tools [online]. [cit. 2022-03-15]. Dostupné z: https://developers.google.com/optimization/routing/vrp

[20] http://www.institut-urmo.cz/images/Hodnoceni_nakladu_na_hospodaren_s_KO_2015.pdf

[21] ISOH, Informační systém odpadového hospodářství. Dostupné z: https://www.cenia.cz/odpadove-a-obehove-hospodarstvi/isoh/

[22] Jambudi, T., Gandhi, S. Analytical review of K-means based algorithms and evaluation methods (2021) 12th International Conference on Advances in Computing, Control, and Telecommunication Technologies, ACT 2021, 2021-August, pp. 479-486

[23] Jin X., Han J. (2011) K-Medoids Clustering. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_426

[24] K-means algorithm [online]. In: . [cit. 2022-05-17]. Dostupné z: https://www.geeksforgeeks.org/ml-k-means-algorithm/

[25] K-means clustering. Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2001-. [cit. 2022-03-05] Dostupné také z: https://en.wikipedia.org/w/index.php?title=K-means_clustering&oldid=1071172431

[26] KILBY, Philip, Patrick PROSSER a Paul SHAW. Guided Local Search for the Vehicle Routing Problem with Time Windows. VOSS, Stefan, Silvano MARTELLO, Ibrahim H. OSMAN a Catherine ROUCAIROL, ed. Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization [online]. Boston, MA: Springer US, 1998, 1999, s. 473-486 [cit. 2022-04-05]. ISBN 978-0-7923-8369-7. Dostupné z: doi:10.1007/978-1-4615-5775-3_32

[27] Koç, C., Bektaş, T., Jabali, O., Laporte, G. Thirty years of heterogeneous vehicle routing (2016) European Journal of Operational Research, 249 (1), pp. 1-21. DOI: 10.1016/j.ejor.2015.07.020

[28] KOLEN, Antoon W.J, Jan Karel LENSTRA, Christos H PAPADIMITRIOU a Frits C.R SPIEKSMA. Interval scheduling: A survey. Naval research logistics [online]. Hoboken: Wiley Subscription Services, Inc., A Wiley Company, 2007, 54(5), 530-543 [cit. 2022-04-30]. ISSN 0894-069X. Dostupné z: doi:10.1002/nav.20231

[29] Křovákovo zobrazení [online]. [cit. 2022-05-15]. Dostupné z: https://ucebnice.geogr.muni.cz/kartografie/obsah.php?show=85&&jazyk=cz

[30] KUBOWSKÝ, Jiří. Webová aplikace pro plánování svozu frakcí odpadu. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2022, Diplomová práce. Vedoucí práce: Ing. Ladislav Dobrovský

[31] Laporte, G. The vehicle routing problem: An overview of exact and approximate algorithms (1992) European Journal of Operational Research, 59 (3), pp. 345-358. DOI: 10.1016/0377-2217(92)90192-C

[32] Laporte, G. Fifty years of vehicle routing (2009) Transportation Science, 43 (4), pp. 408-416. DOI: 10.1287/trsc.1090.0301

[33] Määttä, P. Comparison of solution methods for the dynamic pickup and delivery problem with time windows. (Aalto University. School of Science,2021), http://urn.fi/URN:NBN:fi:aalto-2021121910841

[34] NIRAJ RAMESH, D., Mohan KRISHNAMOORTHY a Andreas T. ERNST. Efficient Models, Formulations and Algorithms for Some Variants of Fixed Interval Scheduling Problems. SARKER, Ruhul, Hussein A. ABBASS, Simon DUNSTALL, Philip KILBY, Richard DAVIS a Leon YOUNG, ed. Data and Decision Sciences in Action [online]. Cham: Springer International Publishing, 2018, 2018-07-28, s. 43-69 [cit. 2022-05-14]. Lecture Notes in Management and Industrial Engineering. ISBN 978-3-319-55913-1. Dostupné z: doi:10.1007/978-3-319-55914-8_4

[35] Nucamendi-Guillén, S., Gómez Padilla, A., Olivares-Benitez, E., Moreno-Vega, J.M. The multi-depot open location routing problem with a heterogeneous fixed fleet (2021) Expert Systems with Applications, 165, art. no. 113846, . DOI: 10.1016/j.eswa.2020.113846

[36] Nuortio, T., Kytöjoki, J., Niska, H., Bräysy, O. Improved route planning and scheduling of waste collection and transport (2006) Expert Systems with Applications, 30 (2), pp. 223-232. DOI: 10.1016/j.eswa.2005.07.009

[37] PINEDO, Michael L. Scheduling: Theory, Algorithms, and Systems [online]. 5th ed. Springer, Cham, 2016 [cit. 2022-04-30]. ISBN 978-3-319-26580-3. Dostupné z: https://link.springer.com/book/10.1007/978-3-319-26580-3

[38] SALHI, Saïd. Heuristic Search [online]. Cham: Springer International Publishing, 2017 [cit. 2022-04-03]. ISBN 978-3-319-49354-1. Dostupné z: doi:10.1007/978-3-319-49355-8

[39] ŠARMANOVÁ, Jana. Metody analýzy dat: učební text. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2012. ISBN 978-80-248-2565-6.

[40] SÖRENSEN, Kenneth. Metaheuristics-the metaphor exposed. International Transactions in Operational Research [online]. 2015, 22(1), 3-18 [cit. 2022-04-03]. ISSN 09696016. Dostupné z: doi:10.1111/itor.12001

[41] Tung, D.V., Pinnoi, A. Vehicle routing-scheduling for waste collection in Hanoi (2000) European Journal of Operational Research, 125 (3), pp. 449-468. DOI: 10.1016/S0377-2217(99)00408-7

[42] VOUDOURIS, Christos, Edward P.K. TSANG a Abdullah ALSHEDDY. Guided Local Search. GENDREAU, Michel a Jean-Yves POTVIN, ed. Handbook of Metaheuristics [online]. Boston, MA: Springer US, 2010, 2010-8-12, s. 321-361 [cit. 2022-04-06]. International Series in Operations Research & Management Science. ISBN 978-1-4419-1663-1. Dostupné z: doi:10.1007/978-1-4419-1665-5_11

[43] WILLIAMS, H. P. Model Building in Mathematical Programming. Chichester: John Wiley, 1978.

[44] XU, Dongkuan a Yingjie TIAN. A Comprehensive Survey of Clustering Algorithms. Annals of Data Science [online]. 2015, 2(2), 165-193 [cit. 2022-03-06]. ISSN 2198-5804. Dostupné z: doi:10.1007/s40745-015-0040-1)

[45] ZAMAZAL, P.Statistická analýza rozsáhlých dat z průmyslu. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2021. 51 s. Vedoucí diplomové práce doc. Ing. Radovan Šomplák, Ph.D.

[46] Zákon č. 541/2020 Sb., Zákon o odpadech.

Seznam použitých zkratek

API Application Programming Interface.

CEP Circular economy package.
CVRP Capacitated Vehicle Routing Problem.

DBSCAN Density-based spatial clustering of applications with noise.

EU Evropská unie.

GLS Guided local search.

JSON JavaScript Object Notation.

KO Komunální odpad.

SKO Směsný komunální odpad.

TSMH Technické služby Malá Haná s.r.o..
TSP Travelling Salesman Problem.

ÚPI Ústav procesního inženýrství.

VRP Vehicle Routing Problem.
VRPDP Vehicle Routing Problemwith pick-up and delivery.
VRPTW Vehicle Routing Problem with time windows.
VSP Vehicle Scheduling Problem.

ZoO Zákon o odpadech.

Příloha A

Obr. 1: VRP [12]