Optimalizace svozových úloh

Bc. Dominik Závada

Diplomová práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Dominik Závada**
Osobní číslo:	**A21820**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Optimalizace svozových úloh**
Téma práce anglicky:	**Optimization of Collection Tasks**

Zásady pro vypracování

1. Seznamte se s problematikou spojenou s matematickou optimalizací, a to zejména se zaměřením na odpadové hospodářství a modely svozu odpadu.
2. Zpracujte rešerši existujících modelů svozu odpadu s ohledem na problematiku modelování (časových) zdržení v dopravních sítích.
3. Popište tuto problematiku pomocí teorie grafů a sestavte jednoduchý matematický model pro plánování svozu. Tento model implementujte ve zvoleném optimalizačním softwaru (např. GAMS či Julia) a otestujte na vybraném datovém souboru.
4. Proveďte důkladnou diskuzi dosažených výsledků. Popište omezení a limity zvoleného přístupu a vytyčte směry pro případný další výzkum.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. GENDREAU, M. a POTVIN, J.-Y. Handbook of Metaheuristics. Second edition. Springer (2010). ISBN: 978-1-4419-1663--1.
2. ROTHLAUF, F. Design of Modern Heuristics: Principles and Application. Springer (2011). ISBN: 978-3-540-72962-4.
3. GHIANI, G., LAPORTE, G. a MUSMANNO, R. *Introduction to logistics systems planning and control*. John Wiley & Sons (2004). ISBN: 0-470-84917-7.
4. TOTH., P. a VIGO, D. *Vehicle Routing: Problems, Methods, and Applications*. Second edition. SIAM (2014). ISBN: 978-1--611973-58-7.
5. BEN TICHA, H., ABSI, N., FEILLET, D., QUILLIOT. A. Vehicle routing problems with road-network information: State of the art. Networks, 72: 393-406 (2018). DOI: 10.1002/net.21808.
6. PIRES, A., MARTINHO, G., RODRIGUES, S. a GOMES, M.I. *Sustainable Solid Waste Collection and Management*. Springer (2019). ISBN: 978-3-319-93199-9.

Vedoucí diplomové práce: **Ing. Dušan Hrabec, Ph.D.**
Ústav matematiky

Datum zadání diplomové práce: **2. prosince 2022**
Termín odevzdání diplomové práce: **26. května 2023**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

Jméno, příjmení: Dominik Závada, Bc.
Název diplomové práce: Optimalizace svozových úloh
Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Dominik Závada, v.r.
podpis studenta

ABSTRAKT

Práce se zabývá optimalizací úloh spojených se svozem odpadu. Pro tento účel byl sestrojen matematický model spadající do kategorie Vehicle routing problems se zohledněním časových závislostí. Díky tomuto modelu je svozové vozidlo schopno přizpůsobit se aktuální dopravní situaci a adaptovat svou trasu v průběhu celého svozu. Tím se zajistí využití méně zatížených tras namísto hlavních cest, kde hrozí vyšší riziko časových zdržení. K dosažení optimálních výsledků bylo navrženo řešení ve formě svozových tras pro libovolnou denní dobu pomocí Clark & Wright Savings algoritmu a sady 25 optimalizačních operátorů, jako jsou například 2-opt, 3-opt nebo Or-opt.

Klíčová slova: Odpadové hospodářství, Svoz odpadu, Časové omezení, Matematická optimalizace, Teorie grafů, Vehicle routing problem, VRP, TDVRP, Clark & Wright algoritmus, Optimalizační operátory, NP-těžké

ABSTRACT

This thesis deals with optimizing waste collection tasks. For this purpose, a mathematical model belonging to the category of Vehicle Routing Problems was constructed, taking into account time dependencies. Thanks to this model, the waste collection vehicle is capable of adapting to the current traffic situation and adjusting its route throughout the entire collection process. This ensures the use of less congested routes instead of main roads, where would be a higher risk of time delays. To achieve optimal results, a solution in the form of waste collection routes for any given time of day was proposed, using the Clark & Wright Savings algorithm and a set of 25 optimization operators, such as 2-opt, 3-opt, and Or-opt.

Keywords: Waste management, Waste collection, Time constraints, Mathematical Optimization, Graph theory, Vehicle routing problem, VRP, TDVRP, Clark & Wright savings algoritm, Optimization operators, NP-hard

Úvodem bych rád poděkoval vedoucímu práce Ing. Dušanu Hrabcovi, Ph.D. za jeho odborné vedení a cenné konzultace po celou dobu vytváření práce. Rovněž bych chtěl poděkovat Alině G. Dragomir, Ph.D. z Vídeňské univerzity za její přínos při objasnění stěžejní problematiky a za konzultace při implementaci praktické části práce.

V neposlední řadě bych rád poděkoval rodině, přítelkyni a přátelům za velkou podporu po celou dobu studia a při vypracovávání práce.

Tato práce vznikla i díky podpoře z projektu Nové přístupy operačního výzkumu pro udržitelnost v odpadovém hospodářství reg. č. GA20-00091Y a projektu Optimalizační modely pro udržitelnost v logistice FSR FORD 5-6 /2021-23/FAI/002

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Produkce a následný sběr odpadu je neodmyslitelnou součástí každé společnosti a zásadní službou pro všechny občany. S narůstající populací se však zvyšuje objem vyprodukovaného odpadu, což vede k větší poptávce po jeho odvozu. Zvláště ve městech se tato situace stává stále náročnějším úkolem, který přináší řadu problémů a komplikací. Jedním z hlavních problémů je zhoršující se dopravní situace, způsobující dopravní zácpy a snižování průměrné rychlosti vozidel při sběru odpadu, což má za následek celkový nárůst nákladů. Avšak této situaci lze předejít s využitím nástrojů matematické optimalizace, která přináší výhody lepší organizace a plánování tras, čímž z velké části eliminuje ztráty času při sběru odpadu.

Řešením tohoto problému se zabývá diplomová práce prostřednictvím zohlednění časových závislostí, díky kterým je matematický model schopen vypočítat optimální trasu pro každý okamžik v průběhu dne. Vozidlo se díky tomu dokáže přizpůsobit aktuální dopravní situaci a využít méně frekventovaných cest v době, kdy hlavní tahy bývají často přetížené. To je docíleno za použití reálných cestovních časů získaných z Google API.

Teoretická část práce je strukturována do pěti kapitol, které pokrývají různé aspekty problematiky. V první kapitole je podrobně rozebráno odpadové hospodářství a zdůrazněny možné výhody, které přináší jeho správná optimalizace. Následující kapitola se zaměřuje na matematickou optimalizaci a zkoumá její výpočetní složitost a třídy složitosti. Třetí kapitola poskytuje základní popis teorie grafů, který je klíčový pro lepší porozumění hlavnímu obsahu následující kapitoly. Tato část představuje různé modely pro svoz odpadu, přičemž je zde kladen důraz na Time-Dependent Vehicle Routing Problem, Clark & Wright algoritmus a optimalizační operátory, které slouží jako základ pro implementovaný matematický model. Poslední pátá kapitola teoretické části práce se věnuje rešerši vědeckých publikací zabývajících se touto problematikou.

Praktická část práce je rozdělena do tří kapitol. Začátek tvoří popis matematického modelu, kde jsou uvedeny příslušné pseudokódy a ukázky vstupních matic. V další kapitole je detailně rozepsána implementace modelu v programovacím jazyce Python. Jsou zde prezentovány názorné ukázky kódu, které slouží k lepšímu pochopení celého procesu. V poslední části jsou prezentovány výsledky modelu při použití na zlínské síti elektrokontejnerů. Součástí jsou názorné ukázky map svozů, které ilustrují fungování modelu. V závěru jsou diskutovány limity a omezení vybraného přístupu a navrženy možné směry pro budoucí výzkum.

I. TEORETICKÁ ČÁST

1 ODPADOVÉ HOSPODÁŘSTVÍ

Odpadové hospodářství představuje obor zaměřený na celkový proces řízení odpadu, tzn. od jejich vzniku až po konečné zpracování. Zahrnuje nejen sběr a likvidaci, ale také prevenci vzniku odpadu, recyklaci a jejich opětovné využití. Hlavním cílem odpadového hospodářství je minimalizovat negativní dopady odpadů na životní prostředí, zdraví lidí a současně maximalizovat využití jejich potenciálu a zdrojů. [1]

Sběr odpadu je z technického, ekonomického, environmentálního a sociálního hlediska klíčovým prvkem v celém procesu nakládání s tuhými odpady, kdy jeho efektivní proces má zásadní význam pro udržitelný rozvoj měst. [2]

1.1 Optimalizace v odpadovém hospodářství

Odpadová krize je trvalým problémem moderní společnosti, neboť s narůstajícím počtem obyvatel a rozvíjejícími se ekonomikami se neustále zvyšuje objem produkovaného odpadu. Avšak neefektivní a mnohdy nedostačující způsob svozu, zejména v ekonomicky rozvojových zemích, kde téměř polovina obyvatel světa nemá přístup ke službám v oblasti nakládání s odpady, má za následek, že odpad často končí na otevřených a černých skládkách s omezenou možností recyklace. [3] Absence integrovaných systémů nakládání s pevným odpadem přispívá k nekontrolovanému nakládání, což ohrožuje životní prostředí a zdraví lidí. Je tedy nezbytným krokem zavést systémy svozu a nakládání s odpadem, které jsou plánovány s ohledem na ochranu životního prostředí a zdraví a které představují hnací sílu hospodářského růstu. [2]

Špatné a neefektivní nakládání s odpadem není jen problémem rozvojových zemí, ale postihuje i velké megalopole. V důsledku hospodářského růstu a moderního způsobu života produkují obyvatelé těchto oblastí velké množství odpadů a nedostatečné nakládání s nimi může způsobit vysoké náklady, kde navíc nastává problém dostupnosti prostoru pro skladování opadu.

Obory matematické optimalizace a logistiky se věnují problematice svozu odpadu, který je často plánován nedostatečně. Cílem je najít nejlepší způsob organizace přepravy odpadu a optimalizovat tok materiálu, aby se minimalizovaly negativní dopady na zdraví lidí a životního prostředí. [4]

1.1.1 Optimalizace z ekonomického hlediska

Nakládání s tuhým komunálním odpadem je považováno za náročnou záležitost současných měst z důvodů rychlého růstu množství produkovaného opadu a vysokých nákladů na následný sběr [5], který představuje až 85 % nákladů na systém nakládání s tuhým odpadem. [6] Z toho následně plyne smysluplnost optimalizovat řešení svozu odpadu v dané oblasti.

1.1.2 Optimalizace z ekologického hlediska

Kvůli neefektivním trasám je svoz odpadu nákladnější a zároveň zatěžuje životní prostředí vysokým množstvím vyprodukovaných emisí CO_2. Optimalizace svozu odpadu by mohla snížit zátěž na okolní prostředí a zlepšit efektivitu celého procesu.

1.2 Prevence vzniku odpadu

Prevence vzniku odpadu je nezbytná pro udržitelné a účinné nakládání s odpady. Jejím cílem je minimalizovat množství vytvořeného odpadu a zabránit negativním dopadům, které by mohly ohrozit zdraví lidí a životní prostředí. [2] Prevence zahrnuje mnoho aktivit, jako je například recyklace, kompostování a snižování spotřeby zboží.

Tyto postupy nejenže snižují množství odpadů, které se ukládají na skládky, ale také zmenšují objem odpadů, které je třeba zpracovávat, což vede k menšímu počtu potřebných svozových cest. [1] Proto je důležité, aby byly prevence odpadů prioritou v politických a společenských rozhodnutích a aby byly podporovány inovace a technologie, které pomáhají minimalizovat množství vytvářeného odpadu.

2 MATEMATICKÁ OPTIMALIZACE

Produkce odpadu se s narůstajícím počtem obyvatel dynamicky zvyšuje a často se stává významným problémem pro mnoho měst. Řešení může být v některých případech velmi složité, avšak optimalizace nabízí užitečnou techniku pro lepší řešení problémů spojených nejen s odpadovým hospodářstvím a svozem odpadu.

Matematická optimalizace je obor matematiky, který se zaměřuje na nalezení nejlepšího možného řešení daného problému, tedy nalezení minimální nebo maximální hodnoty funkce, která podléhá určitému počtu omezení. Optimalizační problémy se vyskytují v mnoha oblastech, jako je průmysl, doprava nebo logistika a optimalizace může mít výrazné dopady na efektivitu, úsporu náklady nebo větší výnosy díky nalezení nejlepšího možného řešení. [7]

2.1 Formulace úlohy

Pro správnou formulaci daného optimalizačního problému je důležité vždy definovat proměnné, účelovou funkci a náležité omezení.

Proměnné se obvykle označují písmenem nebo symbolem a mohou být ovlivněny omezeními a účelovou funkcí. Účelová funkce určuje cíl optimalizačního problému, tedy hodnotu, kterou je potřeba minimalizovat nebo maximalizovat. Obvykle se jedná o matematickou funkci, která závisí na proměnných. Omezení určují podmínky, který daný algoritmus musí splňovat pro dosažení optimálního řešení. Omezení mohou být lineární nebo nelineární a mohou se vztahovat na proměnné nebo na účelovou funkci.

Zápis rovnice 1 představuje obecný tvar úlohy matematické optimalizace, kde je cílem minimalizovat účelovou funkci $f_0(x):\ R^n \to R$ při splnění omezení $f_i(x) \leq b_i : R^n \to R$, pro $i = 1, \dots, m$ kde x představuje vektor proměnných $x = (x_1, \dots, x_n)$. [8]

Rovnice 1 Obecný tvar optimalizační úlohy

$$\begin{aligned} &minimize && f_0(x) \\ &subject\ to && f_i(x) \leq b_i, \quad i = 1, \dots, m. \end{aligned}$$

Obecně se rozlišují různé typy optimalizačních problémů, které mají specifické formy účelových a omezujících funkcí. Například problém z výše uvedeného příkladu se nazývá lineární program a spadá do třídy lineárního programování. To znamená, že se může řešit pomocí lineárních algebraických metod.

2.2 Řešení optimalizačních problémů

Počínaje koncem čtyřicátých let 20. století se začalo věnovat velké úsilí vývoji algoritmů pro řešení různých tříd optimalizačních problémů, analýze jejich vlastností a vylepšování kvality softwarových implementací. Algoritmus použitý pro řešení dané třídy optimalizačních problémů vypočítá řešení s určitou přesností na základě konkrétní formulace problému. Efektivita těchto algoritmů je odlišná v závislosti na konkrétních formách cílových a omezujících funkcí, počtu proměnných a dalších faktorech. Některé přístupy při řešení problémů se musí rozhodnout pro určitý kompromis mezi dlouhým výpočetním časem a dosažením přesného řešení. [8]

Optimalizační problémy mohou být řešeny pomocí různých metod, včetně lineárního programování, smíšeného celočíselného programování, nelineárního programování nebo pomocí heuristických metod.

Optimalizovat svozu odpadu typu Vehicle routing problem (kapitola 4.2) je všeobecně velmi náročné z hlediska času a paměti potřebné pro výpočet, jelikož tyto úlohy spadají do třídy výpočetní složitosti NP-těžké. Praktická část diplomové práce se proto věnuje využití heuristických metod (kapitola 4.3) s cílem nalézt optimální řešení této problematiky.

2.3 Výpočetní složitost

Výpočetní složitost je klíčovým pojmem v oblasti teoretické informatiky, která se zaměřuje na analýzu vnitřní složitosti výpočetních úloh, konkrétně na měření časové a prostorové náročnosti algoritmů. Jedním z důležitých nástrojů je Turingův stroj, který umožňuje matematicky analyzovat algoritmy, včetně jejich složitosti. I když je problém v principu rozhodnutelný, a tedy výpočetně řešitelný, nemusí být v praxi efektivní, pokud jeho řešení vyžaduje nepřiměřeně mnoho času nebo paměti. Výpočetní složitost je tedy obecným studiem, čeho lze dosáhnout v omezeném čase a požadavků na paměť. [9]

Turingův stroj je abstraktní matematický model počítače, který byl navržen britským matematikem Alanem Turingem v roce 1936. Dokáže simulovat práci jakéhokoliv počítače, který lze vytvořit, a umožňuje tak matematickou analýzu algoritmů a problémů, včetně jejich složitosti a je také základem pro definici tříd složitosti.

2.3.1 Časová náročnost

Popisuje, jak je měněna časová náročnost algoritmu s rostoucí velikostí vstupních dat. Konkrétně vyjadřuje počet kroků (operací) potřebných ke zpracování vstupu o dané velikosti.

2.3.2 Prostorová složitost

Určuje, kolik paměti je potřeba pro vykonání algoritmu. Obvykle se vyjadřuje jako počet paměťových buněk, které algoritmus využívá v závislosti na velikosti vstupu. Podobně jako u časové složitosti je cílem minimalizovat prostorovou složitost, aby byl algoritmus co nejefektivnější a dobře škálovatelný s rostoucí velikostí vstupu.

2.3.3 Asymptotická analýza

Protože přesná doba běhu algoritmu je často složitý výraz, obvykle se pouze odhaduje. Asymptotická analýza umožňuje předpovědět, jak se bude chovat algoritmus pro různé velikosti vstupu. Jedním z nejčastěji používaných způsobů je použití Landaurovy notace, označované také jako "big O", která umožňuje odhadnout nejhorší možný růst algoritmu v závislosti na velikosti vstupu. To umožňuje klasifikovat algoritmy do tříd podle toho, jak s rostoucí velikostí vstupu roste jejich čas běhu nebo prostorové nároky. [10]

Zároveň existuje velké množství problémů, které jsou triviálně řešitelné prohledáváním exponenciálního počtu instancí hrubou silou, ale v současné době pro ně není znám žádný efektivní algoritmus.

2.3.4 Třídy složitosti

V teorii výpočetní složitosti se rozlišují třídy algoritmů podle jejich časové a prostorové náročnosti, kdy nejznámější otázkou teorie výpočetní složitosti je P vs NP, u které nastává dilema, zda je hledání řešení těžší než kontrola správnosti nalezeného řešení. [9]

Důležitým pojmem u tříd složitosti je polynomiální čas. Ten označuje dobu, kterou algoritmus potřebuje k vyřešení daného problému v závislosti na vstupu.

2.3.4.1 Třída P

Třída P (Polynomiální) je základní třídou výpočetní složitosti. Obsahuje rozhodovací problémy, které lze vyřešit deterministickým Turingovým strojem pomocí polynomiálního množství výpočetního času. To znamená, že pro tyto problémy existují algoritmy, které je

dokážou řešit v rozumném čase, a to bez ohledu na velikost vstupních dat. Třída P tedy zahrnuje problémy, které jsou považovány za " řešitelné".

Jedním z příkladů je určování, zda je číslo prvočíslem. [11] V roce 2002 bylo dokázáno, že pro tento problém existuje algoritmus, který jej dokáže řešit v polynomiálním čase, a tedy že náleží do třídy P.

Celá třída P je obsažená ve třídě složitosti NP. Je to dáno tím, pokud je problém řešitelný v polynomiálním čase, pak je jeho řešení ověřitelné také v polynomiálním čase. Ovšem třída NP obsahuje mnohem více problémů, z nichž nejtěžší se řadí do třídy NP-úplné. Zmíněná otázka P = NP se ptá, zda existuje algoritmus, který dokáže vyřešit v polynomiálním čase problém ve třídě NP-úplné a z toho plynoucích všech problémů z NP. V současnosti se ovšem domnívá, že obě třídy si nejsou rovny, jelikož vědci investovali obrovské úsilí do nalezení algoritmů s polynomiálním časem pro NP úlohy, ovšem bez úspěchu. Tudíž nejlepší známá deterministická metoda pro rozhodování problémů v NP je exponenciální čas. [10]

2.3.4.2 Třída NP

Třída NP (Nedeterministicky polynomiální) je množina rozhodovacích problémů, které jsou ověřitelné v polynomiálním čase deterministickým Turingovým strojem, nebo alternativně množina problémů, které jsou řešitelné v polynomiálním čase nedeterministickým Turingovým strojem. [10] Ten na rozdíl od deterministického dokáže v každém kroku rozvětvit výpočet do n větví, ve kterých se hledá řešení současně.

2.3.4.3 Třída NP-těžké

Třída NP-těžké zahrnuje problémy, které jsou alespoň tak těžké jako nejtěžší problémy v třídě NP. Neboli, každý problém v NP lze redukovat na problém v třídě NP-těžké pomocí polynomiálního času. [10] Nicméně tyto problémy sami o sobě nemusí být ve třídě NP, což znamená, že nemusí být nutně nalezeno řešení těchto problémů v polynomiálním čase. [12]

2.3.4.4 Třída NP-úplné

Významným přínosem na otázku P vs NP byla práce Stephena Cooka a Leonida Levina na počátku 70. let minulého století, kteří identifikovali určité problémy v NP třídě, jejichž složitost souvisí se složitostí celé třídy. [12] Pokud by některý z těchto problémů mohl být

řešen algoritmem s polynomiálním časem, byly by všechny problémy v NP řešitelné v polynomiálním čase. Tyto problémy se nazývají NP-úplné. [10]

Teorie NP-úplnosti se zakládá na konceptu efektivní redukce, kdy jeden výpočetní problém v NP lze redukovat na jiný problém právě tehdy, pokud je k dispozici algoritmus pro efektivní řešení jiného problému. Z toho plyne, že není těžší řešit první problém než druhý. Jinými slovy, problém náleží třídě NP-úplné, pokud je redukovatelný jakýmkoli problémem z třídy NP a dokazovat, že problém náleží třídě NP-úplné znamená, že problém nemůže patřit třídě P, pokud třída NP není v třídě P. [9]

NP-úplné rozhodovací problémy jsou obecně považovány za velmi obtížné a z praktického hlediska může fenomén NP-úplnosti zabránit zbytečnému plýtvání času hledáním dost možná neexistujícího algoritmu pro řešení problémů v polynomiálním čase, jelikož čas a prostředky jsou omezené zdroje. Nicméně dokázat, že daný problém náleží do NP-úplných a není řešitelný v polynomiálním čase, je silným důkazem jeho nepolynomiálnosti a tedy že třída P není rovna třídě NP. [10]

3 TEORIE GRAFŮ

Teorie grafů je obor diskrétní matematiky, která se zabývá matematickou abstrakcí situací z reálného světa. Ty lze pohodlně popsat pomocí diagramu skládajícího se z množiny bodů a množiny úseček spojujících určité dvojice bodů. Například v odpadovém hospodářství body mohou představovat odpadní nádoby a úsečky charakterizují pak jednotlivé cesty. Tento typ matematické abstrakce se nazývá graf a lze ho použít k reprezentaci téměř jakékoliv fyzikální situaci zahrnující diskrétní objekty a vztahy mezi nimi. Teorie grafů má díky své přirozené jednoduchosti velmi rozmanitou škálu možností a nachází široké uplatnění v různých oblastech, jako je informatika, fyzika nebo matematika. [13, 14]

3.1 Struktura grafu

Graf se formálně skládá z uspořádané dvojice $G = (V, E)$, kde $V = \{v_1, v_2, \dots\}$ označuje množinu vrcholů a $E = \{e_1, e_2, \dots\}$ představuje množinu hran. Každá hrana je definována neuspořádanou dvojicí (v_i, v_j) vrcholů, které spojuje a jsou nazývány jako koncové vrcholy. Hrany mohou mít přiřazené ohodnocení (váhu), což reprezentuje patřičné náklady spojené s danou hranou. Nejběžnějším znázorněním grafu je diagram, v němž jsou vrcholy znázorněny jako body a každá hrana jako úsečka. [14]

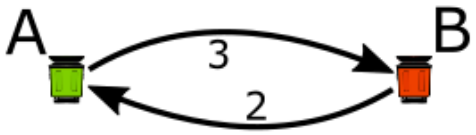
3.2 Rozdělení grafů

Grafy se základně dělí na orientovaný a neorientovaný, avšak rozdělení na základě orientace není jediným způsobem kategorizace grafů. Dalšími typy grafů jsou vážené a nevážené grafy, cykly, stromy a další. [15]

V praktické části práce je pro optimalizaci svozu odpadu využíván Vehicle routing problem (kapitola 4.2) pomocí matic časů, které představují orientované vážené grafy.

3.2.1 Orientovaný graf

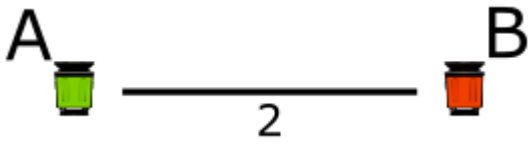
Orientovaný nebo také směrový graf (anglicky označován jako "digraph") je takový, kde každá hrana má přiřazený směr. Jinými slovy, náklady na cestu z vrcholu "A" do vrcholu "B" mohou dosahovat jiných hodnot než náklady na cestu z vrcholu "B" do vrcholu "A". To znamená, že hrany jsou jednoznačně určené jako vstupní nebo výstupní hrany pro každý vrchol a případě orientovaného grafu jsou hrany doplněny o šipky směřující k příslušnému vrcholu. [13] Příklad orientovaného grafu znázorňuje Obrázek 1.



Obrázek 1 Směrové hrany s ohodnocením

3.2.2 Neorientovaný graf

Neorientovaný nebo také nesměrový graf zobrazuje obrázek 2. Zde kde každá dvojice vrcholů je propojena jednou obousměrnou hranou. To znamená, že náklady na cestu z vrcholu "A" do vrcholu "B" jsou stejné jako náklady na cestu z vrcholu "B" do vrcholu "A". Na rozdíl od orientovaného grafu zde neexistuje pojem vstupní nebo výstupní hrana pro vrchol. V důsledku toho nemají hrany šipky, které by určovaly směr, a jsou jednoduše reprezentovány spojnicemi mezi vrcholy, které mohou být procházeny v obou směrech.



Obrázek 2 Nesměrová hrana s ohodnocením

3.2.3 Vážený graf

Vážený graf je takový, u kterého jsou hranám přiřazeny váhy (ohodnocení) značící jejich důležitost nebo význam v dané situaci. Váhy mají obvykle charakter nezáporného reálného čísla reprezentující různé vlastnosti jako je vzdálenost, čas, kapacita nebo cena. Vážené grafy mohou být neorientované či orientované. Často se používají při modelování dopravy, komunikace nebo železniční sítě, kdy umožňují mnohem přesněji popsat situace z reálného světa, kde váhy nejsou považovány za stejně důležité jako u neorientovaného grafu. [13]

3.2.4 Nevážený graf

Nevážený graf nemá definované váhy či ohodnocení hran. Jedná se o jednodušší formu grafu, která zohledňuje pouze informaci o tom, zda jsou dva vrcholy spojeny nebo ne. Tento typ grafu může být orientovaný nebo neorientovaný a všechny hrany mají stejnou "důležitost". Délka cesty v neváženém grafu je dána počtem hran, kterými se prochází, na rozdíl od váženého grafu, kde délka cesty je určena součtem vah hran.

4 MODELY SVOZU ODPADU

Svoz odpadu je obecně považován za kombinatorický problém, jelikož je potřeba najít nejlepší kombinaci (trasu) mezi všemi dostupnými kontejnery, aby bylo možné sbírat odpad z různých míst co nejefektivněji a nejúsporněji a doručit jej na místo k dalšímu zpracování. Při plánování a optimalizaci svozu opadu je důležité zohlednit různé typy faktorů, jako je množství odpadu v kontejnerech, počet a kapacita vozidel, a hlavně aktuální dopravní situaci.

Řešení úloh kombinatorické optimalizace nabízí dva přístupy spojené s modelováním a plánováním tras svozu odpadu – TSP (Travelling salesman problem) a VRP (Vehicle routing problem), označované také jako problémy se směrováním, anglicky "Routing problems". [16] Tyto metody jsou široce využívány k nalezení optimálního řešení reálných problémů v oblasti odpadového hospodářství. [5, 6]

I když oba typy problémů lze snadno definovat, jejich řešení spadá do třídy výpočetní složenosti NP ("Nedeterministicky Polynomiální"), z čehož vyplývá, že v současné době neexistuje algoritmus, který by dokázal spočítat přesné řešení v polynomiálním čase. [17]

4.1 Travelling Salesman Problem

V české literatuře nazýván jako Problém obchodního cestujícího byl poprvé formulován v roce 1930. Nicméně, matematici Sir William Rowan Hamilton a Thomas Penyngton Kirkman studovali problémy související s TSP již v 18. století, i když problém samotný nebyl tehdy definován. Zásluhy za publikování TSP do vědeckých kruhů má Merrill M. Flood, který se v roce 1956 zaměřil na hledání řešení směřování školních autobusů a publikoval tento problém na akademické půdě. Ovšem kdo přesně vymyslel pojmenování " TSP“ není dodnes zcela známo. [18, 19]

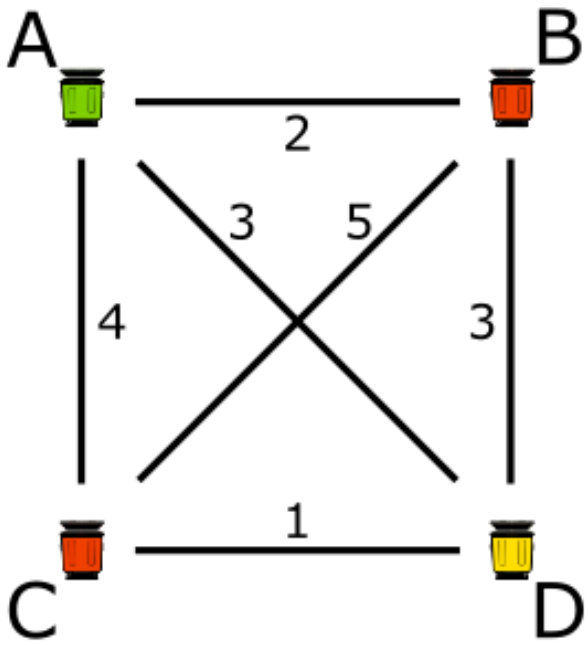
4.1.1 Definice

Je dána množina uzlů, pro které jsou předem definované náklady na cestu (vzdálenost nebo čas) mezi každou dvojicí uzlů. TSP lze popsat jako hledání nejkratší trasy, která prochází napříč všemi uzly v dané množině a vrátí se zpět do výchozího bodu, přičemž každý uzel musí být navštíven a opuštěn právě jednou.

TSP může být rozdělen do dvou různých typů – Symetrického a Asymetrického, které se liší zejména použitými vstupními daty. Pro lepší pochopení dané problematiky jsou oba podrobně rozepsány a doplněny o náležitý graf z teorie grafů.

4.1.2 Symetrický TSP

Symetrická verze TSP má vlastnost, kterou je symetrie mezi body. To znamená, že náklady na cestu (vzdálenost, čas atd.) mezi každou dvojici uzlů jsou stejné v obou směrech, což vytváří neorientovaný graf (viz Obrázek 3). Díky symetrii se také snižuje počet možných řešení na polovinu a není nutné ukládat druhou polovinu záznamů, jelikož cesta z bodu 1 do bodu 2 je stejná jak z bodu 2 do bodu 1. Nicméně i přesto se symetrický problém řadí do NP-těžkých a pro velké instance je takřka nemožné nalézt exaktní řešení v rozumném čase. Z toho důvodu se používají heuristické algoritmy, které se snaží najít alespoň vhodné aproximativní řešení.

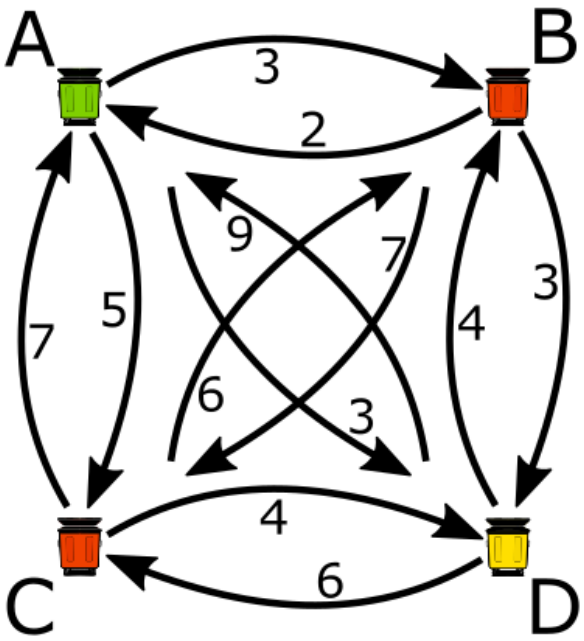


Obrázek 3 Symetrický TSP

4.1.3 Asymetrický TSP

Asymetrický TSP se na rozdíl od předchozí verze odlišuje tím, že náklady na cestu mezi uzly se mohou lišit v obou směrech, což vytváří orientovaný graf (viz Obrázek 4). Tento rozdíl hodnot může být způsoben faktory jako jsou například dopravní zácpy či zdržení způsobené zhoršenou dopravou. Protože Asymetrický TSP již nevykazuje symetrii, vzniká nutnost ukládat veškeré záznamy, což s sebou nese vyšší počet možných řešení a zvyšuje

časovou a paměťovou náročnost algoritmu. Proto je důležité používat vhodné heuristické algoritmy při řešení problému.



Obrázek 4 Asymetrický TSP

Studie [16] ukazuje, že pro řešení asymetrického TSP lze použít algoritmy s lokálním vyhledáváním typu 2-opt, Swap a Move (někdy označované jako "Insertion"). Tyto algoritmy jsou schopny řešit asymetrický problém s téměř stejnou flexibilitou jako problém symetrický.

4.1.4 Matematická formulace

Problém lze matematicky popsat pomocí Miller-Tucker-Zemlin formulace z knihy [20], která je zde označována za nejkompaktnější pro tento účel. Byla představena v roce 1960 a je považována za velmi důležitou, jelikož používá lineární programování pro hledání optimální trasy obchodnímu cestujícímu. Kompaktnost v tomto kontextu znamená použití minimálního počtu nadbytečných informací, což ji činí velmi efektivní pro matematické řešení daných problémů. Ukázka matematické formulace ukazuje tabulka 1.

Tabulka 1 Miller-Tucker-Zemlin formulace TSP [20]

Find variables x_{ij} and U_i ; i, j = 1, 2, …,n

Min:

$$z = \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}x_{ij} \qquad \text{(Ia)}$$

Subject to:

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, 2, \dots, n \qquad \text{(IIa)}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, 2, \dots, n \qquad \text{(IIIa)}$$

$$u_i - u_j + n \cdot x_{j_j} \leq n - 1 \qquad i, j = 2, \dots, n; i \neq j \qquad \text{(IVa)}$$

$$x_{ij} = 0{,}1 \qquad \forall\, i, j \qquad \text{(Va)}$$

Rovnice (Ia) se v matematické formulaci rovněž nazývá jako účelová funkce a její úkol je minimalizovat celkové náklady na cestu. Kromě účelové funkce jsou určeny také omezení daného modelu pomocí zbylých rovnic ukázky. Rovnice (IIa) a (IIIa) udávají, že vozidlo může vstoupit a vystoupit z každého uzlu pouze jedenkrát. Rovnice (IVa) zabraňuje vytváření dílčích tras, což znamená, že vozidlo musí pokračovat v cestě, dokud nejsou všechny uzly navštíveny. Poslední rovnice (Va) stanovuje, že proměnná x může nabývat pouze hodnot 0 nebo 1, přičemž hodnota 1 indikuje, že daný uzel byl navštíven, zatímco hodnota 0 znamená, že nebyl.

4.2 Vehicle Routing Problem

VRP, v české literatuře známý také jako Problém směrování vozidel, lze považovat za rozšíření TSP. Od svého zavedení v roce 1959 Dantzigem a Ramserem [21] je předmětem intenzivního výzkumu. Ve své práci představili první matematický model pro nalezení optimální trasy týkající se dodávek benzínu flotilou nákladních vozidel a navrhli první formulaci matematického programování a algoritmickou metodu, která umožňuje algoritmický přístup. V následujících letech se postupně začaly vyvíjet heuristiky, kdy mezi nejznámější se řadí heuristika úspor Clark & Wright (1964), která díky své rychlosti a přesnosti vylepšila Danzigův-Ramserův přístup. S nástupem 90. let začal vývoj moderních heuristik a metaheuristik, které umožňují řešit složitější varianty a v dnešní době se VRP uplatňuje v oblastech, jako je doprava, logistika nebo řízení zásob. [22]

VRP bylo úspěšně aplikováno v mnoha reálných situací, kde se ukázalo, že jeho použití v plánovací nebo operativní fázi přináší značné úspory v nákladech za dopravu. V porovnání s postupy, které nevyužívají optimalizační techniky, umožňuje VRP také lepší využití vozového parku. [23]

4.2.1 Definice

VRP je optimalizační úloha, která spočívá v nalezení nejlepší trasy pro vozidlo nebo více vozidel s cílem minimalizovat celkové náklady, přičemž musí splňovat stanovené požadavky a omezení, jako je návštěva každého uzlu a zároveň začínat a končit vždy ve výchozím bodu.

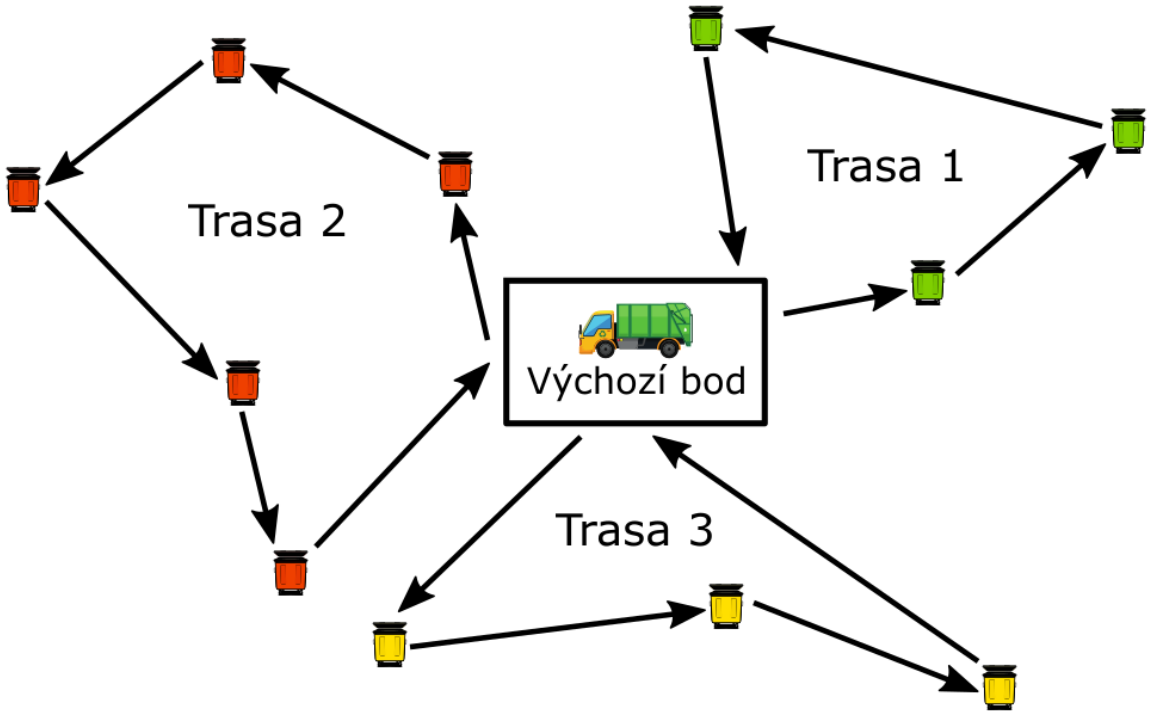
Od prvního zavedení do praxe existuje mnoho variant VRP, které se neustále vyvíjejí a přizpůsobují se různým typům problémů s odlišnými požadavky a omezeními. Tato práce se zaměřuje především na dvě z nich: Capacitated Vehicle Routing Problem, která stanovuje omezení kapacity vozidel a Time-Dependent Vehicle Routing Problem, která bere v úvahu časová omezení a zahrnuje změny rychlosti v průběhu dne.

I pro VRP platí, že problém může být dále rozdělen mezi symetrické a asymetrické náklady (vzdálenost, čas) mezi dvěma body. Nicméně princip rozdělní je zcela totožný, jak již bylo vysvětleno v předešlém TSP přístupu. Tedy, symetrický VRP předpokládá, že náklady na přepravu mezi dvěma místy jsou totožné bez ohledu na směr, zatímco asymetrický VRP bere v úvahu, že náklady se mohou lišit v závislosti na směru.

4.2.2 Capacitated Vehicle Routing Problem

Capacitated Vehicle Routing Problem, zkráceně CVRP je nejvíce studovanou a rozšířenou verzí VRP. [23] Oproti základní verze se odlišuje zejména přidáním omezení pro kapacitu vozidel, které musí být dodržováno v každém okamžiku trasy. To znamená, že vozidlo má stanovenou maximální kapacitu a po dovršení této kapacity musí být vráceno zpět do výchozího bodu (depa), vyprázdněno a poté může pokračovat v dokončení cesty.

Obrázek 5 zobrazuje jednu z možných realizací CVRP. Vozidlo začíná svozem zelených kontejnerů, ale po naplnění své kapacity se musí vrátit do výchozího bodu, aby mohlo pokračovat dalším svozem, tentokrát červených a žlutých kontejnerů. Celková trasa je rozdělena do tří úseků různých délek, kdy třikrát vozidlo muselo přerušit svou činnost a vyložit svůj náklad.



Obrázek 5 Capacitated Vehicle Routing Problem

4.2.2.1 Matematická formulace

Formulace [24] popisuje matematický model pro řešení CVRP se omezením kapacity vozidel. Model zobrazen v tabulce 2 využívá binární proměnnou $x_{i,j}$ s dvěma indexy pro určení, které indikují, zda hrana mezi dvěma uzly i a j je součástí optimální trasy. Pokud je hodnota této proměnné rovna 1, hrana je součástí trasy, v opačném případě je hodnota rovna 0 a hrana není zahrnuta do trasy. Výpočetní složitost modelu spadá do kategorie ($O(n^2)$), což

zmámená, že s narůstajícím počtem uzlů se zvyšuje čas potřebný k nalezení optimální trasy v poměru k druhé mocnině počtu uzlů.

Tabulka 2 Matematická formulace CVRP [24]

Min:		
$\sum_{i \in V}^{n} \sum_{j \in V}^{n} C_{ij} x_{ij}$		(Ib)
Subject to:		
$\sum_{i \in V} x_{ij} = 1$	$\forall j \in V \setminus \{0\}$	(IIb)
$\sum_{j \in V} x_{ij} = 1$	$\forall i \in V \setminus \{0\}$	(IIIb)
$\sum_{i \in V} x_{i0} = K$		(IVb)
$\sum_{j \in V} x_{0j} = K$	$\forall\, i, j$	(Vb)
$\sum_{i \notin S} \sum_{j \in S} x_{0j} \geq r(S)$	$\forall S \subseteq V\ \{0\}, S \neq \emptyset$	(VIb)
$x_{ij} \in \{0, 1\}$	$\forall\, i, j\ \in V$	(VIIb)

Rovnice (Ib) minimalizuje celkové náklady, kde $C_{i,j}$ vyjadřuje dobu jízdy a binární proměnná $x_{i,j}$ signalizuje, zda je daná hrana součástí trasy. Omezení (IIb) a (IIIb) garantují, že vozidlo může do každého uzlu (s výjimkou výchozího uzlu) vstoupit a vystoupit pouze jedenkrát. Omezení (IVb) a (Vb) definují, že každé vozidlo bude mít právě jeden výchozí uzel (depo), do které se bude vracet po dokončení trasy. Omezení (VIb) udává, že každá podmnožina S uzlů (kromě výchozího uzlu) musí obsahovat alespoň jedno vozidlo, které je bude obsluhovat a také obsahuje kapacitní omezení pro vozidla. Poslední omezení (VIIb) definuje binární proměnnou, která nabývá pouze hodnot 1 a 0, což určuje, zda jsou hrany součástí konečné trasy (hodnota 1) nebo ne (hodnota 0).

4.2.3 Time-Dependent Vehicle Routing Problem

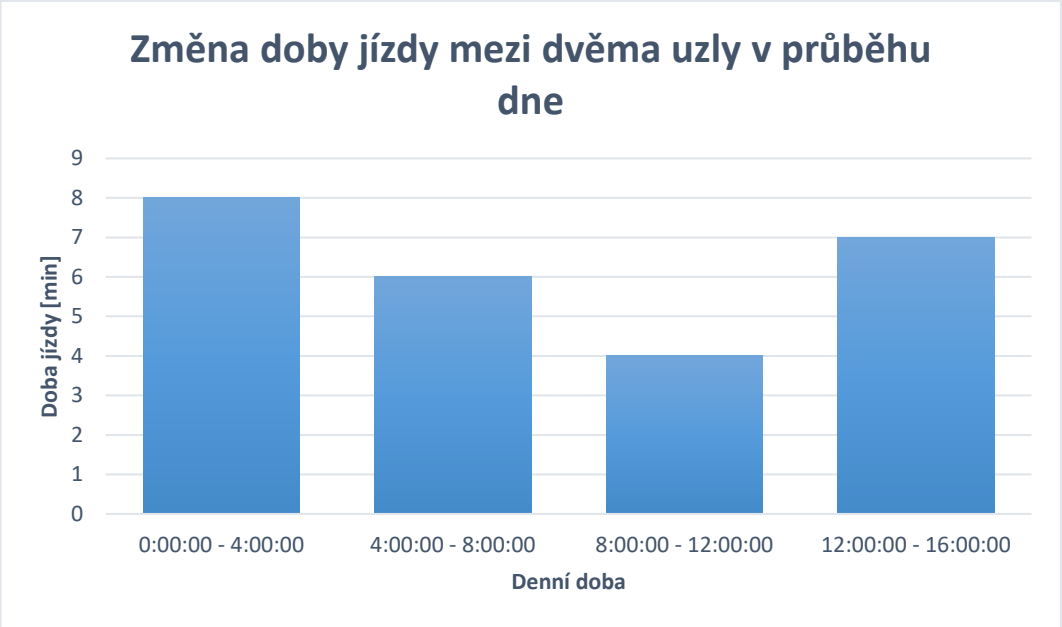
Většina výzkumných studií zabývající se VRP předpokládá, že atributy silniční sítě jsou jednoduše spočitatelné předem a zadány jako vstupní parametry. Například cestovní doba mezi dvěma uzly může být snadno vypočtena na základě průměrné rychlosti vozidla v daném úseku. Tento předpoklad usnadňuje určení optimální trasy. Nicméně, v reálných situacích se mohou objevit různé faktory, jako jsou dopravní zácpy, nehody nebo uzavírky silnic, které mohou ovlivnit plánovanou trasu a snížit průměrnou rychlost. [25]

V přetíženém městském provozu doba cesty mezi dvěma body nezávisí pouze na samotné ujeté vzdálenosti a rychlost vozidel se mohou v průběhu jízdy výrazně měnit. Aby bylo možné provést realistickou optimalizaci, TDVRP uvažuje skutečnost, že doba jízdy mezi dvěma uzly nebo mezi uzlem a výchozím bodem může být proměnlivá v závislosti na čase a na aktuální dopravní situaci. Pokud při řešení nejsou zohledněny časové závislosti na denní době, mohou vzniknout suboptimální výsledky s odlišnou strukturou trasy, než by bylo optimální. [26]

Díky rozdělení dne na časové intervaly, kdy cestovní rychlost je měněna v závislosti na daném intervalu, algoritmus dokáže lépe určit optimální trasu skrz silniční síť města a minimalizovat čas potřebný k dosažení cíle. Nejčastější uplatnění je v prostředí s vysokou dopravní zátěží, například v centru měst nebo na hlavních tazích, kde hrozí vyšší riziko zdržení.

Studie [27] se zabývá aplikací TDVRP na reálnou silniční síť velkoměsta. K dosažení řešení bylo využito 7 časových intervalů o délce 2 hodin. Použití časových omezení vedlo ke snížení celkových nákladů až o 37 %, v porovnání s klasickými VRP přístupy, které ignorují proměnlivé cestovní rychlosti. Avšak použití těchto metod může být komplikované kvůli nízké přesnosti dopravních informací a potřebě zahrnutí široké škály faktorů, které zvyšují výpočetní náročnost úlohy.

V grafu níže je znázorněna variace cestovního času mezi dvěma specifickými body v průběhu dne. Obrázek 6 zahrnuje čtyři časové intervaly, které se pravidelně střídají každé čtyři hodiny. V prvním intervalu náklady na cestu činí 8 minut, v druhém 6 minut, ve třetím pouze 4 minuty a v posledním 7 minut. Toto omezení umožňuje vozidlu optimálně plánovat návštěvu uzlů v pořadí, aby se nejlépe přizpůsobilo aktuální dopravní situaci.



Obrázek 6 Graf změny doby jízdy TDVRP

4.3 Heuristické metody pro efektivní řešení svozu odpadu

V případech, kdy se problém jeví zvláště složitý a není možné nalézt exaktní řešení problému, je přijato takové řešení, které nemusí být nutně nejlepší možné, ale dostatečně efektivní vzhledem k náročnosti problému a použitého výpočetního času. Tento přístup je označován jako heuristický.

4.3.1 Clark & Wright algoritmus

Jedná se o jednoduchý algoritmus pro řešení VRP úloh, který využívá hledání úspor při sloučení tras, čímž se minimalizují celkové náklady spojené s trasou. Algoritmus je vhodný pro problémy, kde počet vozidel je rozhodovací proměnná a lze jej využít jak pro orientované, tak pro neorientované grafy. [24] Díky své jednoduché implementaci a rychlosti výpočtu se řadí mezi nejznámější heuristiky pro řešení VRP. Ovšem nevýhodou použití je, že bez jakýkoliv vylepšení poskytuje řešení, které mají k optimálnímu řešení daleko. [28]

Existují dvě verze algoritmu – sekvenční a paralelní. V sekvenční verzi se postupně po jedné sestavují trasy. Naopak v paralelní verzi se zároveň sestavuje více tras najednou, což může vést ke zvýšení efektivity. Princip celého algoritmu pak funguje následovně.

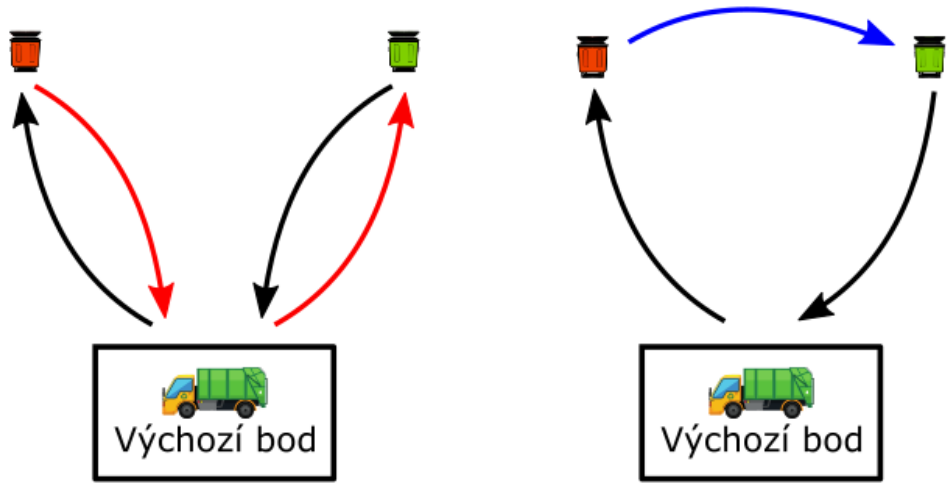
Nejprve jsou vypočítány úspory "savings" dle rovnice 2 pro každou dvojici vrcholů grafu (uzlů). Tyto úspory jsou definovány jako rozdíl mezi náklady na obsluhu každé dvojice, pokud by byl každý vrchol obsloužen samostatně a náklady, pokud by byly vrcholy obslouženy společně.

Rovnice 2 Výpočet úspor Clark & Wright algoritmu

$$s(i,j) = d(D,i) + d(D,j) - d(i,j)$$

Kde $s(i,j)$ označuje vypočítanou úsporu vyplývající ze spojení dvou vrcholů grafu s indexy i a j. Relativně vysoké hodnoty $s(i,j)$ naznačují, že s ohledem na náklady je výhodné navštívit uzel j bezprostředně za uzlem i. Ovšem uzly i a j nelze spojit do trasy, pokud by spojení porušovalo omezení VRP. Zároveň $d(D,i)$ označuje ohodnocení hrany mezi výchozím bodem a uzlem i, $d(D,j)$ naznačuje ohodnocení hrany mezi výchozím bodem a uzlem j a poslední $d(i,j)$ znázorňuje ohodnocení hrany mezi oběma uzly.

Následující obrázek 7 znázorňuje princip rovnice úspor, kde červené šipky představují hodnoty $d(D,i)$ a $d(D,j)$ a modrá šipka ukazuje na hodnotu $d(i,j)$. Celkově obrázek ilustruje úsporu nákladů získanou spojením dvou tras do jednoho celku.



Obrázek 7 Clark & Wright algoritmus

Po vypočtení všech úspor je seznam seřazen sestupně od nejvyšší hodnoty po nejnižší a vybere se dvojice vrcholů vždy s nejvyšší úsporou. Tyto dva vrcholy jsou následně přidány do trasy a pokračuje se v dalším výpočtu úspor ze seznamu s následujícími podmínkami:

a. Pokud jsou oba vrcholy již součástí trasy, úspora je ignorována.

b. Pokud jeden vrchol již je součástí trasy a druhý není, lze jej přidat pouze tehdy pokud přidání nijak neporušuje omezení plynoucí z VRP a zároveň nesníží efektivitu trasy.

c. Pokud jsou ova vrcholy dosud nezařazeny, lze je přidat pouze tehdy pokud přidání neporušuje omezení plynoucí z VRP a zároveň nesníží efektivitu trasy.

Postup se opakuje, dokud nelze přidat žádný další vrchol do trasy, který by splňoval dané podmínky.

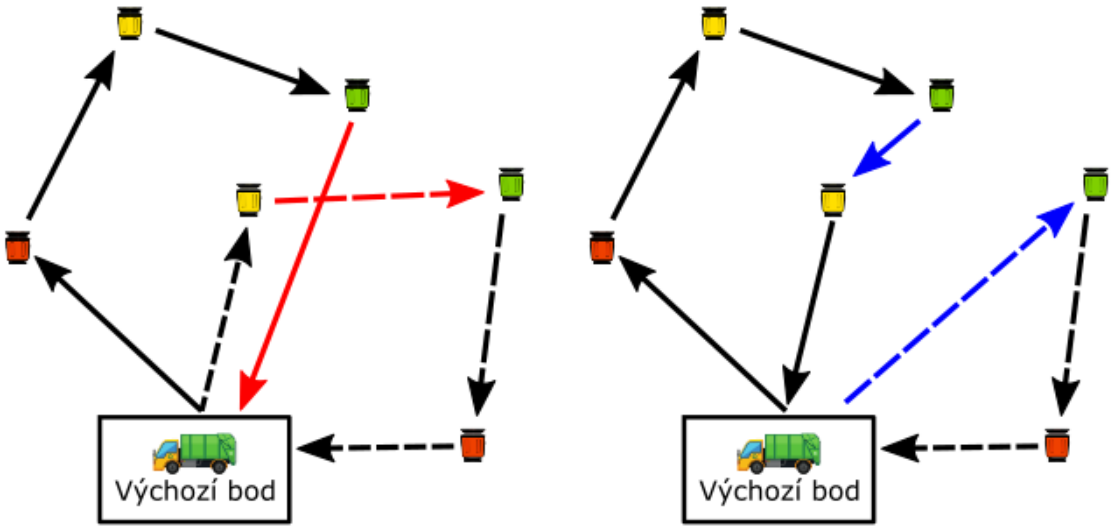
Jelikož diplomová práce se zabývá problematikou časových zdržení pomocí TDVRP, v porovnání s klasickým Clark & Wright algoritmem zde nastává nutnost kontrolovat aktuální časový interval vozidla. Pokud je interval překročen, musí se upravit odpovídající ohodnocení hran.

4.3.2 Local search

Jedná se o typ heuristické metody při řešení optimalizačních problémů. Metoda se soustředí na prohledávání sousedství aktuálního řešení s cílem nalézt lepší řešení, přičemž nezaručuje nalezení globálního optima, nicméně často vede k nalezení přijatelného řešení v krátkém čase. Základní myšlenkou je vycházet z řešení získaného konstrukční metodou a postupně jej lokálně vylepšovat. Proces se opakuje, dokud nedojde k dalšímu zlepšení. [29]

4.3.2.1 Move

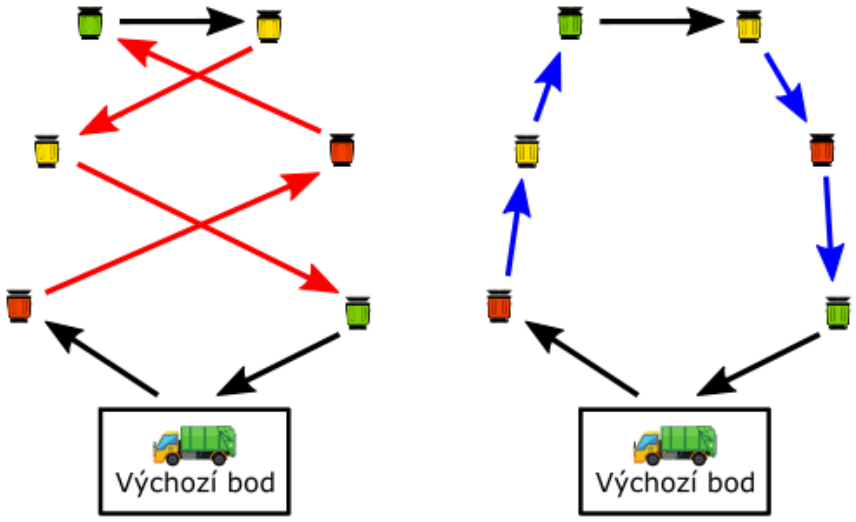
Jedná se o metodu, která přemísťuje existující vrcholy grafu (uzly) na nové pozice. [30] Počet přemístěných vrcholů grafu v každé iteraci záleží na dané implementaci. Cílem je prohledat celý prostor a najít lepší řešení než to stávající. Algoritmus se opakuje, dokud není dosaženo požadovaného cíle nebo je není splněna ukončující podmínka. Příklad metody move popisuje obrázek 8.



Obrázek 8 Metoda Move

4.3.2.2 Swap

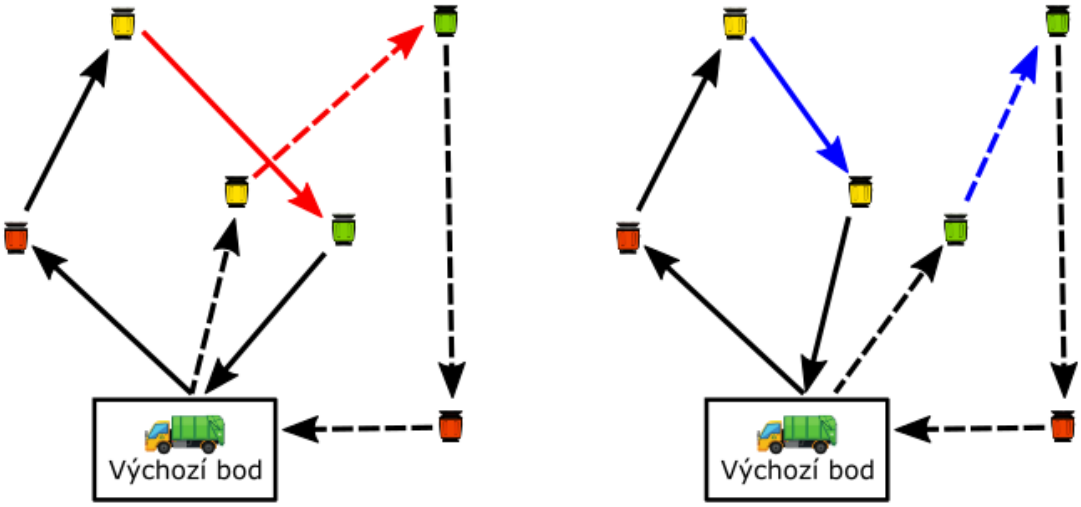
Obrázek 9 demonstruje metodu swap, která v každém kroku provádí výměnu dvou vrcholů (uzlů) grafu za účelem vylepšení aktuálního řešení optimalizačního problému [30]. Proces se opakuje, dokud není nalezeno lepší řešení nebo dokud není splněna podmínka ukončení.



Obrázek 9 Metoda Swap

4.3.2.3 2-opt

Obrázek 10 představuje metodu 2-opt, která v každém kroku odstraní dvě hrany z řešení a nahradí je dvěma novými tak, aby vznikla nová cesta s nižšími celkovými náklady. [30] Tento proces se opakuje, dokud není nalezeno lepší řešení nebo dokud není splněna podmínka ukončení.

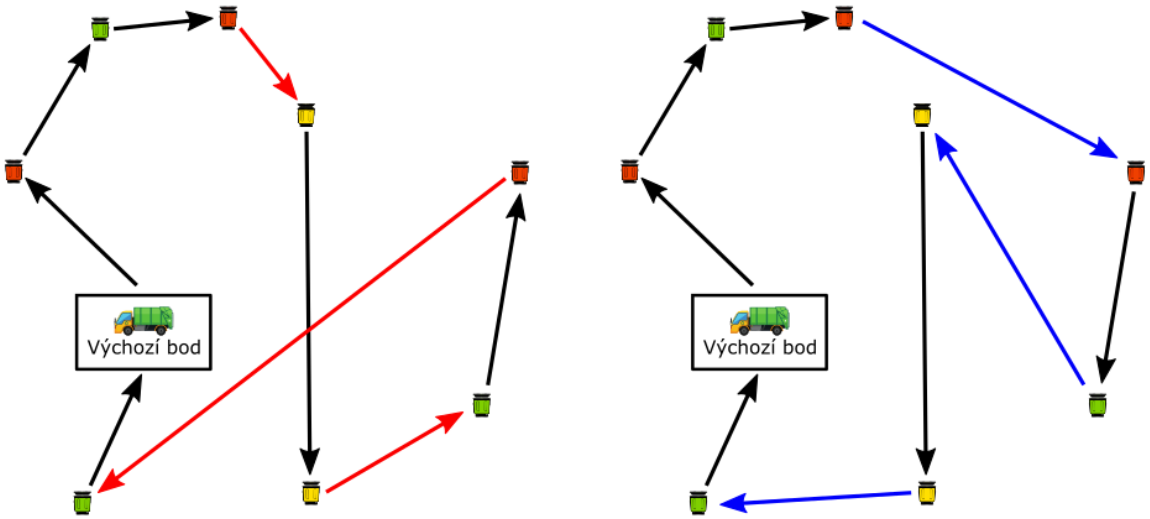


Obrázek 10 Metoda 2-opt

4.3.2.4 3-opt

Metoda 3-opt je účinná technika založena na práci se třemi hranami mezi vrcholy grafu (tzv. tří cest mezi uzly), které se postupně obměňují. Cílem algoritmu je nalézt lepší kombinaci tří hran, která by nahradila současnou kombinaci hran a vedla k vylepšení stávajícího řešení.

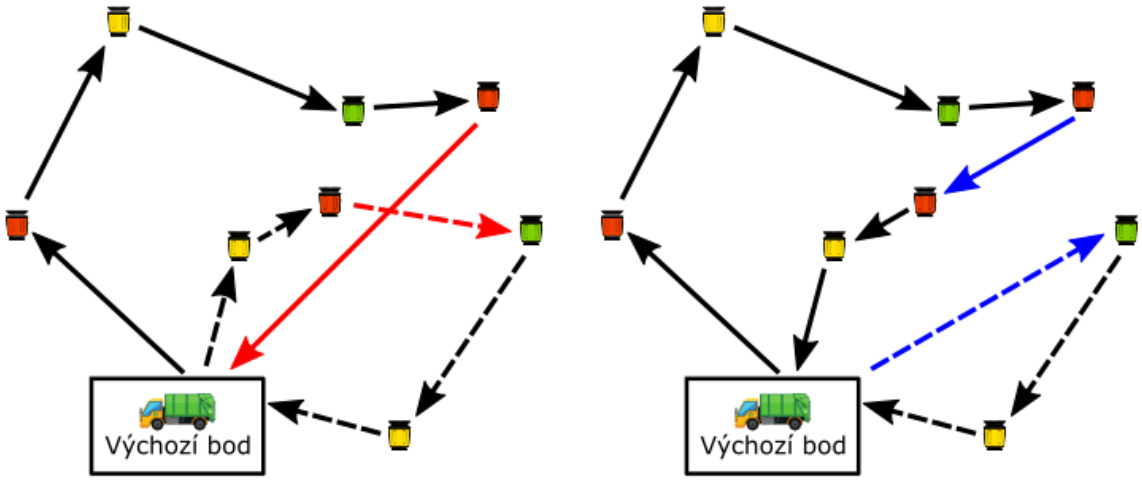
[30] Proces se opakuje, dokud již není možné najít kombinaci, která by vedla k vylepšení řešení nebo je dosaženo kritéria na ukončení algoritmu. Optimalizaci pomocí metody 3-opt je nastíněn v obrázku 11.



Obrázek 11 Metoda 3-opt

4.3.2.5 Or-opt

Or-opt (někdy označována jako N-opt) je metoda, která v každém kroku odstraní určitý řetězec hran (nikoliv vrcholů) grafu a následně vytvoří novou trasu vkládáním tohoto řetězce na jiné místo v grafu, jak ukazuje obrázek 12. Počet vybraných hran závisí na konkrétní implementaci. Cílem je najít takovou kombinaci hran, která minimalizuje celkové náklady. Postup se opakuje do doby, dokud není nalezeno lepší řešení nebo není splněna podmínka ukončení.



Obrázek 12 Metoda Or-opt

4.3.3 Metaheuristiky

Účinnou alternativou pro efektivní řešení problému svozových tras je využití metaheuristik. Oproti heuristikám nabízejí vyšší úroveň abstrakce a obecnosti, což umožňuje jejich použití pro různé typy problémů a nejsou tak omezeny na jeden konkrétní případ. Tyto metody využívají různé mechanismy a strategie, aby se vyhnuly uvíznutí v lokálním minimu a mohly hledat nejlepší možné řešení. Důležitým prvkem je použití sousedství, které umožňuje definovat přípustné tahy pro přechod z jednoho řešení do druhého. Mezi nejznámější zástupce se řadí: Optimalizace pomocí roje částic (PSO), Genetické algoritmy (GA), Simulované žíhání (SA), Mravenčí kolonie (ACO) a Tabu Search (TS).

V posledních letech se heuristiky a metaheuristiky staly preferovaným způsobem řešení mnoha složitých kombinatorických problémů, právě díky účinným nástrojům a mechanismům, které vznikly při jejich tvorbě. I když nedokážou potvrdit optimálnost nalezených řešení, v porovnání s exaktními postupy se často ukázaly jako účinnější, zvláště pro reálné problémy s vysokou úrovní složitostí. [31]

5 ZPRACOVÁNÍ REŠERŠE MODELŮ SVOZU ODPADU

V akademické činnosti je nezbytné věnovat se zpracování vědeckých prací před začátkem vytváření matematického modelu či implementace. Ty mají za cíl rozšířit poznání v dané oblasti, vymezit problematiku zkoumaného tématu a přispět k identifikaci možného rozvoje a navrhovaných postupů.

V tabulce 3 je shrnut výběr publikací, které se věnují zkoumané problematice. Publikace jsou seřazeny sestupně podle roku vydání a v případě, že byla v publikaci specificky zohledněna konkrétní oblast problému, je toto pole v tabulce označeno symbolem "✓". Tento způsob zobrazení pomáhá rychle identifikovat, které publikace se více soustředí na danou oblast zkoumaného tématu a umožňuje snadnou orientaci v zobrazených pracích.

Vysvětlení klíčových slov v tabulce je následující: "VRP" označuje Vehicle Routing Problem, což je problém optimalizace trasy vozidel, "TDVRP" značí Time-Dependent Vehicle Routing Problem, tedy časově závislý problém trasy vozidel, "CWA" představuje algoritmus Clark & Wright Savings Algorithm a poslední klíčové slovo "Jiné heuristiky" zahrnuje skupinu různých heuristik, které byly použity v rámci daného výzkumu. Mezi nejčastěji používané heuristiky patřily lokální prohledávání, optimalizace rojem částic (PSO), simulované žíhání (SA) nebo Tabu Search (TS). Tyto heuristiky představují metody hledání přibližného řešení problémů, což je v mnoha případech dostačující pro praktické využití výsledků v reálném světě.

Při návrhu časových intervalů se stala velkou inspirací práce Mugayskikh et al. [27], která se zabývala aplikací TDVRP na reálnou silniční síť v Petrohradě s vylepšením celkových nákladů až o 37 %. Práce Kritzinger et al. [32] přispěla k lepšímu pochopení logiky TDVRP a práce Groër et al. [30] byla klíčová při vytváření a následné implementaci optimalizačních operátorů 2-opt, 3-opt, Or-opt, Swap a Move. Všechny další práce byly použity k prohloubení problematiky popsané v teoretické části této práce.

Tabulka 3 Rešerše odborných prací

Autor	Rok vydání	VRP	TDVRP	Svoz odpadu	CWA	Jiné heuristiky
Gutuérrez-sánchez et al. [33]	2022	✓		✓		✓
Liang et al. [34]	2022	✓		✓		✓
Ma et al. [35]	2021	✓		✓	✓	✓
Gmira et al. [36]	2021		✓			✓
Liu et al. [37]	2021	✓		✓	✓	✓
Gruler et al. [38]	2020		✓	✓		✓
WU et al. [5]	2020	✓		✓		✓
Sarmah et al. [39]	2019	✓		✓	✓	
Wahyu Katon [40]	2019	✓		✓	✓	✓
Mat et al. [41]	2018		✓	✓		✓
Mat et al. [42]	2018	✓		✓		✓
Mugayskikh et al. [27]	2018		✓			
Pichpibul et al. [28]	2012	✓			✓	
Buhrkal et al. [43]	2012	✓		✓		✓
Kritzinger et al. [32]	2012		✓			✓
Groër et al. [30]	2010	✓			✓	✓

Z výsledků rešerše vyplývá, že zatím nebylo dostatečně prozkoumáno propojení Clark & Wright Savings algoritmu s časově závislým svozem odpadu (TDVRP). Tato diplomová práce se proto zaměřuje na řešení tohoto problému, které je podrobně popsáno v praktické části práce a podloženo výsledky měření.

II. PRAKTICKÁ ČÁST

6 NÁVRH OPTIMALIZAČNÍHO MODELU

V matematické optimalizaci je základní začít s pečlivým návrhem matematického modelu, sloužící jako základ pozdější implementace. Je důležité být důkladný a přesný, protože i sebemenší nepřesnost nebo nedbalost může mít za následek nepřesné výsledky. Každý detail je třeba pečlivě prověřit a zajistit, aby matematický model byl korektně definován a reprezentoval požadovaný optimalizační problém. Pouze je možno dosáhnout spolehlivých a kvalitních výsledků optimalizace.

6.1 Představení úlohy

Úkolem níže popsaného modelu bylo vymyslet optimální návrh svozu odpadkových nádob s ohledem na časové zdržení v dopravních sítích. Důvodem pro zkoumání tohoto problému je stále se zhoršující dopravní situace v centru a okolních částech města, která způsobuje zpoždění a zvyšuje celkové náklady na svoz odpadu.

Navrhovaný model přímo vychází z VRP a pro dosažení závislosti na danou dopravní situaci bylo zapotřebí vymyslet strukturu, jak toho docílit. Klíčová inovace nastala po sestrojení logiky časových intervalů rozdělující den na úseky stejné délky (viz Tabulka 4). Díky této funkcionalitě dokážeme měnit vstupní data v průběhu trasy, které přímo ovlivňují rozhodování vozidla. Namísto pouhých vzdáleností mezi dvěma body byly využity časy průjezdů mezi nimi získané z Google Map. Toto je zcela rozhodující, protože časy průjezdů se liší každou hodinou, na rozdíl od konstantních vzdáleností, a tím dokážeme redukovat dobu strávenou v dopravních zácpách či ucpaných silnicích způsobených dopravní špičkou.

Základním stavebním kamenem pro vytvoření VRP trasy je použit Clark & Wright savings algoritmus, který navrhuje čistě deterministické počáteční řešení. Postupně je vzniklý výsledek modifikován díky optimalizačním operátorům jako jsou: 3-opt, 2-opt, Or-opt, Swap a Move. Jejich úkolem je záměna pořadí jednotlivých uzlů ve výchozí trase se snahou zlepšit celkovou kvalitu trasy. Výhodou této heuristiky je rychlost a snadná implementovatelnost. Nicméně nevýhodou, která může nastat je riziko uvíznutí v lokálním optimu, tedy že metoda nenajde nejlepší globální řešení. Klíčovým prvkem algoritmu je aktualizace vstupních dat v závislosti na denní době a omezení kapacity vozidla, se snahou co nejvíce aproximovat reálnou situaci.

V ulicích města Vídně byl testován podobný způsob provedení, také s využíváním časových intervalů, který podle článku [32] při použití kvalitních dat vykazuje razantně lepší výsledky.

Matematický model je schopný vytvořit řešení trasy svozového vozidla pro jakoukoli denní dobu, tedy aby se minimalizoval čas strávený na cestách a vozidlo se pohybovalo efektivně v závislosti na dopravní situaci. Jelikož v současné době se nenachází algoritmus, který dokáže vypočítat optimální řešení v polynomiálním čase, tato práce se zabývá přiblížení se k optimu pomocí heuristik.

6.2 Vstupní data

Nejdůležitější vstupní data pro danou úlohu jsou jednotlivé matice časů uchovávající dobu průjezdů mezi jednotlivými body. Ty se v optimalizačních problémech nazývají uzly či vrcholy a představují vždy jeden prvek, který je spojen s dalšími prvky (uzly) pomocí hran. Například u problému hledání nejkratší trasy je hrana reprezentována jako místo, které je spojeno s ostatními místy cestami různých délek.

Uzel také může mít další parametry jako hmotnost, časová okna ve smyslu, kdy daný uzel může být obsloužen, čas potřebný k obsluze daného uzlu a další.

V této diplomové práci se jako uzel používá kontejner pro sběr odpadu a hrana je znázorněná jako čas v sekundách. Ten nabývá vždy různých hodnot a závisí na konkrétní denní době, tudíž dokáže navrhnout přesnější řešení než s použitím konstantní vzdálenosti.[32] Největší rozdíl je ve městech s rušnou dopravou, kdy se s každou přibývající hodinou mění dopravní situace a čas přejezdu městským centrem extrémně narůstá díky zdržením ve formě dopravních kolon.

Dalšími důležitými vstupními daty jsou: geografické souřadnice uzlů, čas potřebný k obsluze konkrétního uzlu a množství odpadu daného uzlu a kapacita svozového vozidla.

6.2.1 Časy průjezdů mezi uzly

Pro řešení bylo použito 8 časových intervalů (viz Tabulka 4). Každá časový interval uchovává matici dat a má pevně stanovenou délku 3 hodiny. Čím více intervalů je využito, tím přesnější je posléze výsledné řešení. V aplikaci Excel jsou matice pojmenovány podle jejich horních mezí intervalů, v němž se jsou data uloženy.

Tabulka 4 Rozložení denní doby pro časové intervaly

0:00 – 2:59:59 interval "0"
3:00 – 5:59:59 interval "1"
6:00 – 8:59:59 interval "2"
9:00 – 11:59:59 interval "3"
12:00 – 14:59:59 interval "4"
15:00 – 17:59:59 interval "5"
18:00 – 20:59:59 interval "6"
21:00 – 23:59:59 interval "7"

Časy byly vygenerovány pomocí Google programového rozhraní Directions API. Jedná se o webovou službu, která vždy pro konkrétní požadavek vrátí data ve formátu JSON nebo XML. API bylo zvoleno s cílem získání optimálních časů průjezdů vozidlem s ohledem na dopravní zdržení v určitém časovém okamžiku pomocí konkrétních geografických souřadnic. Rozhraní vždy vrací nejefektivnější trasy, kdy jako hlavní faktor je doba cesty, ale využívá i jiné faktory jako vzdálenost, počet odbočení a další. Google Directions API využívá nejnovější mapovací a navigační technologie, díky čemuž dokáže zajistit přesné a aktuální informace o dopravní situaci. [44]

Data z API byla generována pomocí funkce "TRAVELTIME", znázorněná v kódu 1. Funkce je napsaná v jazyce VBA (Visual Basic for Applications) používající se v Excelu s podporou maker. Účel funkce je vypočítat odhadovanou dobu cesty mezi dvěma zadanými uzly s přihlédnutím na aktuální dopravní situaci. Výstup z funkce je čas průjezdu mezi dvěma uzly v sekundách pomocí dotazu. Ten byl sestaven z geografických souřadnic míst odjezdu a příjezdu a klíče, který byl posléze zaslán na koncový bod. Pro správné fungování je nutné si předem vygenerovat API klíč na Google Cloud Platformě.

Kód 1 Funkce TRAVELTIME pro generování časů

```
Function TRAVELTIME(origin, destination, departure, apikey)
    Const API_ENDPOINT As String = "https://maps.googleapis.com/maps/api/directions/json"
    Const GET_METHOD As String = "GET"
    Dim url As String
    Dim response As String
    Dim parsed As Dictionary
    Dim seconds As Long
    Dim leg As Dictionary
    url = API_ENDPOINT & "?departure_time=" & departure & "&destination=" & destination &
          "&origin=" & origin & "&key=" & apikey
    Set httpReq = CreateObject("WinHttp.WinHttpRequest.5.1")
    With httpReq
        .Open GET_METHOD, url, False
        .Send
    End With
    response = httpReq.ResponseText
    Set parsed = JsonConverter.ParseJson(response)
    For Each leg In parsed("routes")(1)("legs")
       seconds = seconds + leg("duration_in_traffic")("value")
    Next leg
    TRAVELTIME = seconds
End Function
```

Jelikož vstupní matice dat je sestavena s ohledem na specifické požadavky sestrojeného modelu, je nutné při ukládání jednotlivých matic v aplikaci Excel dodržovat určitou strukturu pravidel, aby model dokázal jednoznačně určit, o který uzel se jedná.

Nezbytné je značení osy X a Y, kde se uzly označují čísly vzestupně od 0, která znamená výchozí bod. Každý další přidaný uzel je označen číslem o jedna vyšší a označuje kontejner.

Díky správnému dodržení datové struktury jsou matice snadno uživatelsky čitelné a umožňují lépe se orientovat v datech a případně je měnit. Datová struktura hraje důležitou roli při zvyšování efektivity algoritmu. [45]

Dalším důležitým pravidlem je dodržení nulové diagonály, která vyplývá z toho, že nelze cestovat mezi stejnými uzly.

Při řešení daného modelu je zásadní, aby všechny ostatní záznamy v matici měly kladné hodnoty, s výjimkou diagonálních záznamů. To je nezbytné z důvodu zajištění přístupnosti všech kombinací pro řešení daného kombinatorického problémů. V případě záznamů se zápornými hodnotami, by výpočet mohl být nepřesný, a tudíž by nemohla být nalezena optimální trasa.

Poslední pravidlo stanovuje, že matice časů musí být čtvercového typu, což znamená, že musí mít stejný počet záznamů na ose X i Y, nesplnění této podmínky by zapříčinilo neúplnost vstupních dat a chybějící kombinace v hledání trasy. Tabulka 5 uvádí příklad korektního zápisu záznamů do matice časů přejezdu mezi body, která obsahuje 6 uzlů.

Tabulka 5 Vstupní matice časů přejezdů mezi uzly

Node	0	1	2	3	4	5
0	0	302	328	199	293	462
1	310	0	108	348	438	601
2	325	84	0	364	450	615
3	199	321	344	0	204	390
4	314	429	454	218	0	369
5	501	620	643	411	382	0

6.2.2 Souřadnice uzlů

Geografické souřadnice v podobě zeměpisné šířky a zeměpisné délky, anglicky označované jako "Latitude" a "Longitude" jsou používány pro generování časů přejezdů mezi uzly z Google API pomocí funkce "TRAVELTIME". Později slouží pro tvorbu svozových map. Demonstrace vhodného zápisu vstupních dat pro depo a 3 další uzly ukazuje tabulka 6.

Tabulka 6 Vstupní matice "Coordinates"

Node	Latitude	Longitude
0	49.21656	17.61761
1	49.20741	17.59369
2	49.20742	17.59
3	49.2174	17.63261

V řešené úloze byly využity kontejnery, které jsou rozestavěny v různých lokalitách města Zlína a jeho přilehlých částech. Detailní přehled všech umístěných kontejnerů lze nalézt na oficiální webové stránce [46] nebo na přiloženém obrázku 13.

6.2.3 Čas potřebný k obsluze uzlů

Pro každý uzel je potřeba specifická doba na jeho obsluhu. Tento čas zahrnuje přesunutí odpadu z kontejneru do svozového vozidla a následné připravení odpadové nádoby ke znovu použití. Pro tento požadavek se využívá matice "Service time" znázorněna v tabulce 7.

Tabulka 7 Vstupní matice "Service time"

Node	Service time
0	360
1	360
2	360
3	360

Čas na obsluhu odpadových nádob je nastaven pro všechny uzly v úloze na konstantní hodnotu 360 sekund. Je důležité nastavit tyto hodnoty tak, aby byly reálné dosažitelné a řádně dodržovány. Díky tomu se stává celkový proces svozu odpadu efektivnějším.

6.2.4 Množství odpadu v uzlu

Míra zaplnění odpadové nádoby je zcela nezbytná informace pro návrh trasy. Vstupní matice vyobrazena v tabulce 8 zaznamenává předpokládaný objem odpadu každého uzlu v m^3. Nepřesný prvotní odhad o zaplnění může vést k nalezení trasy, která neodpovídá skutečnosti.

Tabulka 8 Vstupní matice "Waste"

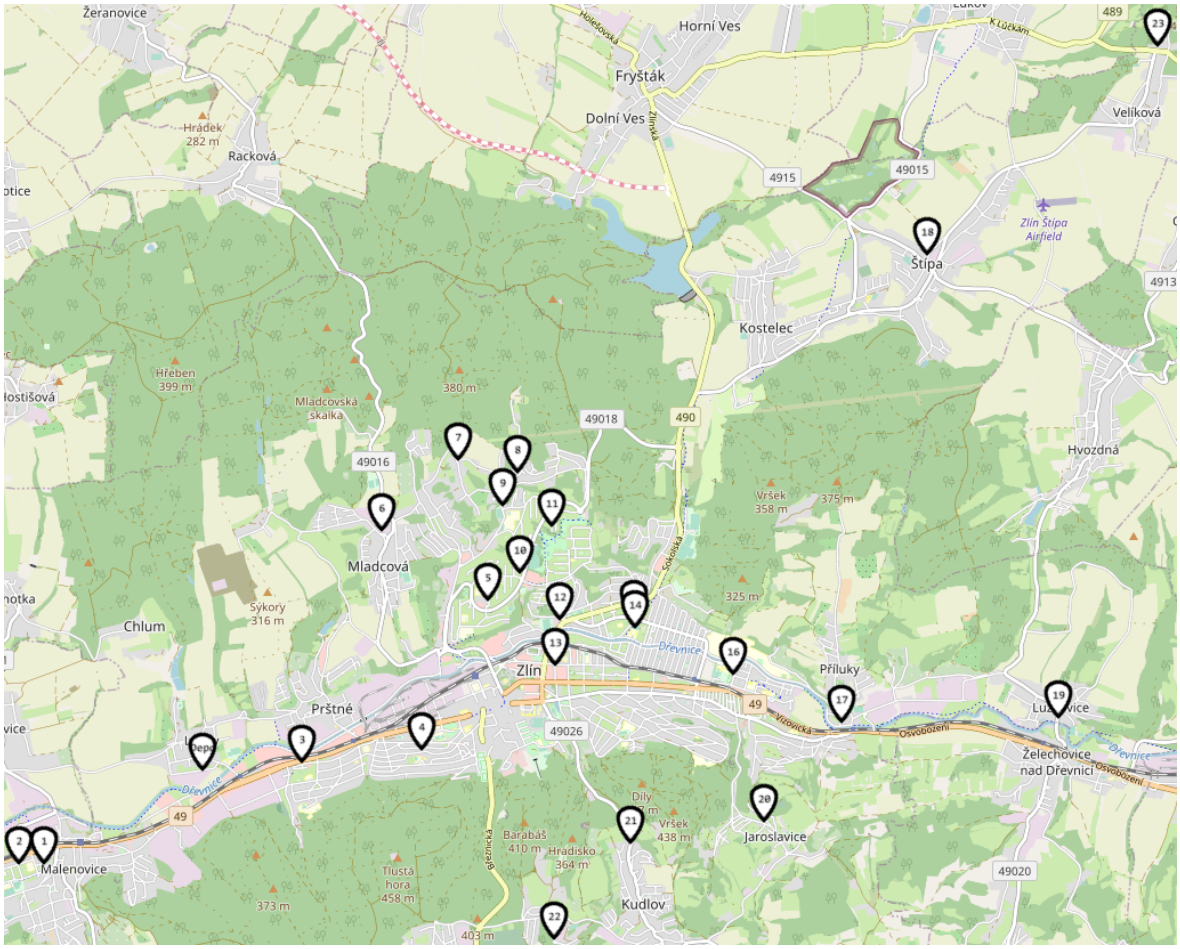
Node	Waste
0	0
1	2.01
2	1.98
3	1.7

Depo (uzel 0 v ukázce) má kapacitu 0, jelikož se zde vozidlo po zaplnění celkového nákladového prostoru jede vyvézt, aby bylo možné dokončit plánovaný svoz.

6.3 Přehled rozmístění kontejnerů

Pro svoz jsou použity odpadové nádoby na elektroodpad od firmy Asekol a.s., označované jako "Červené kontejnery". Rozměry nádob činí 1 300 x 1200 mm a výškou 1800 mm. S objemem 2,15 m^3 jsou vhodné pro sběr elektrozařízení o maximálních rozměrech 50 x 40 cm nebo pro vhoz baterií o celkových rozměrech 65 x 35 mm. Podmínky pro umístění nádoby stanovují, že musí připadat alespoň 1000 obyvatel na jeden stacionární kontejner. [47]

V obrázku 13 je patrné umístění nádob jak v centru města, tak v okrajových částech a přilehlých obcích: Štípa, Velíková, Želechovice nad Dřevnicí, Jaroslavice, Kudlov a Mladcová. Díky tomuto rozpoložení mají všichni obyvatelé možnost k přístupu, což přispívá k udržitelnosti města a zlepšení kvality životního prostředí.



Obrázek 13 Mapa umístění kontejnerů ve městě Zlín

Ovšem toto rozvržení může představovat výzvu, kdy centrum Zlína s horší dopravní situací v určitých hodinách způsobuje riziko zdržení v dopravních zácpách, a tedy neschopnost zajistit včasné odvážení odpadu. To také přispělo k vytvoření řešení pomocí časově závislého plánování, kterým se práce zabývá.

6.4 Popis funkcionality matematického modelu

Fungování modelu je detailně popsáno pseudokódy, což je zjednodušený zápis algoritmu obsahující logické operace a základní kroky vždy specifické pro danou úlohu. Kromě toho je také využita teorie grafů, která je často využívaná pro názornější reprezentaci vztahů mezi uzly a hranami modelu. Pro zápis pseudokódů je zvolena angličtina z důvodů používání terminologie, která je výhradně psaná v tomto jazyce.

6.4.1 Popis modelu pomocí teorie grafů

Tato diplomová práce využívá teorii grafů pro reprezentaci svozu kontejnerů na elektroodpad. Uzly v grafu odpovídají jednotlivým kontejnerům a jsou ohodnocený geografickými souřadnicemi, kapacitou (množství zaplnění) a časem potřebným na obsluhu v sekundách. Hrany mezi uzly představují cestovní čas mezi uzly a jsou ohodnoceny jako doba jízdy v sekundách. Jedná se o asymetrický graf (viz Obrázek 4), což znamená, že doba jízdy mezi dvěma uzly může nabývat různých hodnot pro cestu tam a zpět.

6.4.2 Clark & Wright savings algorithm

Model začíná Clark & wright savings algoritmem, popsaným v teoretické části práce. V iteračním procesu se vypočítávají hodnoty "savings", což označují jednotlivé úspory času.

Rozdíl oproti normálnímu algoritmu spočívá v jeho závislosti na čase, kdy v každém přidaném uzlu do svozové trasy se kontroluje, zda není překročena horní hranice časového intervalu. Pokud tak nastane, vstupní matice časů průjezdů se aktualizuje a výpočet pokračuje dále s nově vygenerovanými "savings".

Algoritmus bez časové a kapacitní úpravy je popsán také ve článku [48].

Pseudokód 1 Časově závislý Clark & Wright savings algoritmus

```
Step 1: Initialize an empty route list, full_route list and visited_nodes list
Step 2: Calculate all savings for all possible pairs of nodes and list them
Step 3: Sort the savings in decreasing order by their savings value
Step 4: Set actual_time_interval
        Set actual_vehicle_capacity = 0
        While visited_nodes < length(nodes):
            For each savings pair (i, j) in the sorted savings list:
                If collection_time < upper bound of actual_time_interval:
                    If actual_vehicle_capacity < max_vehicle_capacity:
```

```
                a) If i and j are in route list, skip this pair
                b) If i and j are not in route list
                    Add node i and j to route list and visited_nodes
                    Update collection_time and actual_vehicle_capacity
                c) If j is not in route list and i is equal with the last element in route list
                    Add node j to last position in route list and visited_nodes
                    Update collection time and actual_vehicle_capacity
                d) If i is not in route list and j is equal with the first element in route list
                    Add node i to first position in route list and visited_nodes
                    Update collection time and actual_vehicle_capacity
            else:
                Add depot to route list and Set actual_vehicle_capacity to 0
                Update collection time and Add route list to full_route list
                Clear route list
        else:
            actual_time_interval = actual_time_interval + 1
            Add route list to full_route list
            Clear route list
            Update sorted savings list based on the newly added nodes in visited_nodes
Step 5: Return the full_route list
```

Pseudokód 1 popisuje funkci algoritmu, rozdělen do 5 kroků. Krok 1 až 3 slouží pro inicializaci prázdných listů nutných pro výpočet, Spočítání kombinací každého páru Clark & Wright algoritmem značící jako "savings" a následné seřazení podle nejlepších hodnot po nejhorší. Následně se pracuje již jen s uzly, hodnoty sloužily jen k uspořádání kandidátů.

Čtvrtý krok obsahuje klíčovou logiku a podmínky. Nejprve se nastaví aktuální časový interval a kapacita vozidla na hodnotu 0. Poté následují 2 cykly. První, který končí za předpokladu, že jsou navštíveny všechny uzly a druhý, který postupně prochází každou dvojici uzlů (i, j) ze seznamu seřazených "savings". Pakliže jsou splněny následující podmínky: svozový čas je menší než horní mez daného časového intervalu a vozidlo nedosáhlo maximální kapacity, jsou k dispozici následující možnosti:

a) Pokud jsou uzly i, j již v seznamu trasy, přeskoč dvojici a pokračuj další iterací.

b) Pokud se uzly i, j nenachází v seznamu trasy, vlož je do seznamu trasy a seznamu navštívených uzlů. Zároveň aktualizuj svozový čas a kapacitu vozidla.

c) Pokud se uzel j nenachází v seznamu trasy a uzel i má stejnou hodnotu jako poslední prvek v seznamu trasy, vlož uzel j do seznamu trasy na poslední pozici a do seznamu navštívených uzlů. Zároveň aktualizuj svozový čas a kapacitu vozidla.

d) Pokud se uzel i nenachází v seznamu trasy a uzel j má stejnou hodnotu jako první prvek v seznamu trasy, vlož uzel i do seznamu trasy na první pozici a do seznamu navštívených uzlů. Zároveň aktualizuj svozový čas a kapacitu vozidla.

Jestliže není splněna druhá podmínka algoritmu a kapacita vozidla by po přidání uzlu přesáhla maximální povolenou kapacitu, do seznamu trasy se vloží depo, svozový čas se aktualizuje a kapacita vozidla se nastaví na 0. Poté je trasa vložena do úplné trasy a vymaže se ze seznamu trasy.

Když není splněna první podmínka a svozový čas přesahuje horní mez daného časového intervalu, interval se nastaví na hodnotu o jedna vyšší. Trasa se následně vloží do úplné trasy a poté je vymazána ze seznamu. Nakonec se vygeneruje nový seznam "savings" z důvodu změny časů přejezdů mezi uzly.

Algoritmus je dokončen, když jsou obslouženy všechny uzly a v pátém kroku vrátí úplnou trasu.

6.4.3 Optimalizační operátory

Po vytvoření prvotní trasy pomocí výše zmíněného procesu, následuje úprava pozic uzlů trasy pomocí předem definovaných optimalizačních operátorů: 2-opt, 3-opt, Or-opt, Swap a Move. Cílem je získat modifikovanou trasu s nižším konečným svozovým časem než před provedením úprav. Celkový proces modifikace popisuje pseudokód 2.

Pseudokód 2 Modifikace trasy užitím optimalizačních operátorů

```
Step 1: Input initial solution: original_route list
        Input collection_time
        Input initial parameters: max_iterations
Step 2: Initial empty best_route list
        Set best_collection_time = collection_time
        Initialize empty operator_list
        Set improvement = True
Step 3: For iteration in range(max_iterations):
            Set operator_list with random order of operators
```

```
        Set fitness_route = original_route.copy()
        Set fitness_collection_time = collection_time
        improvement = True
          While improvement:
            improvement = False
            For each operator in operator_list:
              Calculate modified_route and modified_collection_time using operator
              If modified_collection_time < fitness_collection_time:
                Set fitness_route = modified_route
                Set fitness_collection_time = modified_collection_time
                Set improvement = True
            If fitness_collection_time < best_collection_time:
              Set best_route = fitness_route.copy()
              Set best_collection_time = fitness_collection_time
Step 4: If best_route is empty:
        Set best_route = original_route.copy()
Step 5: Return the best_route list
```

Pseudokód 2 startuje krokem 1, kde se načítá "collection_time" a "original_route" z předešlé metody. Tyto hodnoty symbolizují konečný svozový čas a svozovou trasu. Dále se zadávají počáteční parametry jako je maximální počet prováděných iterací "max_iterations".

Krok 2 slouží na inicializaci prázdných seznamů "Best_route", který později uchovává nejlepší nalezenou trasu a "Operator_list", určující pořadí, v jakém se operátory budou aplikovat. V úloze se výhradně používá náhodný výběr operátorů, s výjimkou poslední iterace, kdy je použito předdefinované řazení. "Best_collection_time" je nastavena na hodnotu originálního svozového času z předešlé metody. "Improvement" je prvotně nastaven na hodnotu "True", určující začátek iteračního cyklu. V případě, kdy operátor najde lepší svozový čas, než je aktuální nejlepší, je hodnota zlepšení opět nastavena na "True" a celý iterační cyklus se opakuje ve snaze najít ještě lepší cestu. V opačném případě, kdy operátor nenalezne lepší řešení, zůstává parametr nastaven na "False" indikující konec iteračního procesu a přechod na jiný operátor.

V dalším kroku 3 se pomocí cyklu ze seznamu "operator_order" vybere aktuální operátor, který postupně modifikuje celou trasu. Pokud je nalezená trasa lepší, než je aktuálně nejlepší, "fitness_route" a "fitness_collection_time" se aktualizují na právě nalezené lepší řešení.

Tento proces se opakuje, dokud je boolean hodnota "improvement" nastavena na "True". V případě nenalezení lepší trasy v rámci iterace, parametr zůstává na "False" a cyklus končí. Dále jsou porovnány hodnoty "fitness" s aktuálně nejlepšími. Jestli jsou "fitness" hodnoty lepší, nahradí se. Rovněž se zvýší hodnota cyklu o jedna a celý proces se opakuje do chvíle, dokud není dosaženo maximálního počtu iterací stanovených v "max_iterations".

Předposlední krok obsahuje podmínku, která se provede pouze v případě, že je seznam nejlepší trasy "best_route" prázdný. Tento případ může nastat pouze v situaci, kdy metoda nedokáže najít žádné lepší řešení než to původní. Jestliže tak nastane, původní trasa ze vstupu se vloží do seznamu "best_route".

Poslední krok 5 vrátí nejlepší řešení "best_route" pro případné další zpracování a funkce ukončuje svůj proces.

6.5 Rozdíly s obecným VRP přístupem

Základ modelu je použití VRP logiky, avšak zvolený přístup se od obecného modelu v několika zásadních bodech odlišuje, neboť klade důraz na i další zohledněné faktory kromě minimalizace vzdáleností nebo času mezi uzly. [24]

První změnou se stala implementace časové závislosti. Její výhody jsou značné oproti obecnému přístupu, zejména v oblasti přesnějšího a efektivnějšího plánování v reálném čase. S přibývajícím časem vozidlo je schopné lépe vyhodnocovat a rozhodovat o nejlepším možném pokračování trasy s ohledem na aktuální dopravní situaci. Naopak obecný model s použitím konstantních vzdáleností nebo časů nedokáže detekovat případné změny v dopravě a výsledné řešení by mohlo vést k nepřesným výsledkům.

Další a zároveň poslední přidanou změnou, která byla brána v potaz je zohlednění času potřebného k obsluze jednotlivých uzlů. Defaultně nastavená konstantní hodnota 360 sekund má pouze orientační charakter a je možné ji upravovat dle konkrétních potřeb a požadavků.

7 IMPLEMENTACE MODELU

V této kapitole je nastíněn průběh implementace matematického modelu do programovacího jazyka Python pomocí popisu celé funkcionality. Práce obsahuje pouze stěžejní definice a implementace ve formě názorných ukázek kódu, které jsou určeny pro lepší pochopení. Celý spustitelný program je k dispozici v příloze.

Jako jazyk úryvků kódů byla zvolena angličtina, kvůli jejímu běžnému používání jako unifikovaného jazyka v programování a jeho terminologii. Většina dokumentace a knihoven jsou psány v angličtině. Tudíž používání názvů proměnných a funkcí v tomto jazyce usnadňuje komunikaci, umožňuje snadnější sdílení mezi platformami nebo programovacími jazyky či zrychluje proces pochopení kódu mezi programátory napříč celým světem.

7.1 Použitý jazyk projektu

Volba programovacího jazyka je zcela zásadní rozhodnutí na začátku každého projektu. Je důležité pečlivě zvážit všechny možné alternativy a vybrat tu nejvhodnější. Pro tuto práci byla zvolena volba v podobě Pythonu, který se řadí mezi interpretované, objektově orientované programovací jazyky. Konkrétně byla použita verze 3.8.

Mezi přínosy této verze je použití tzv. "f-strings", které umožňují snadné vkládání proměnných do textových řetězců. Užitečným může být také varování "SyntaxWarning" ohledně chybné syntaxe a dále byly provedeny drobné změny a vylepšení v optimalizaci, což přispělo k rychlejšímu a stabilnějšímu běhu jazyka. [49]

7.2 Důvody volby Pythonu

Python se podle "PYLP" indexu za rok 2022 se řadí jako nejpoužívanější programovací jazyk světa. [50] Důvodů je hned několik, jako například podpora stále zvětšující se komunity, velké množství knihoven a balíčků, čitelnost, jednoduchost nebo multiplatformnost.

Díky značnému množství knihoven a balíčku je snadnější práci například s daty, které jsou pro řešení VRP zásadní. Jejich využíváním se programátoři mohou pohodlně soustředit na daný problém bez zbytečné implementace základních funkcí potřebných pro běh.

Snadnější syntaxe a podpora široké komunity přispívají při produktivitě práce, a díky tomu, že Python je multiplatformní, kód může být spuštěný na různých operačních systémech bez potřeb dodatečných úprav.

7.3 Struktura repositáře

V repositáři jsou k dispozici následující soubory: "requirements.py", "main.py", "data_model.py", "data_validation.py", "clark_and_wright.py", "post_processing.py" a "map_generation.py". Navíc zahrnuje 2 složky souborů: "input_files" a "output_files".

Rozdělení kompletní funkcionality do více souborů zajišťuje lepší orientaci a přehlednost kódu, a zároveň minimalizuje riziko vzniku rekurze. Každá položka ze seznamu bude dále popsána a vysvětlena.

7.3.1 Soubor "requirements.py"

Uchovává veškeré informace o potřebných knihovnách, modulech a balíčků, které jsou vždy specifické pro daný projekt. Také obsahuje nezbytné verze, nutné pro spuštění a správné fungování programu. Často se používá s nástrojem pro správu balíčků "pip", umožňující snadné stažení a instalování všech nezbytných balíčků a knihoven.

Zároveň lze soubor automaticky generovat příkazem : "pip freeze > requirements.txt". Pakliže již existuje, přepíše jej novou verzí, což umožňuje provádět snadné aktualizace.

Projekt používá dvě knihovny ukázané v kódu 1. První "folium" verze 0.14.0 je použita pro generování výstupních map znázorňující svozovou trasu vozidla. Druhá "pandas" verze 1.5.3 načítá vstupní data z Excelu do programu a rovněž generuje výsledky do Excelu, umístěného ve složce "output_files".

Kód 1 Ukázka "requirements.py"

```
folium==0.14.0
pandas==1.5.3
```

7.3.2 Soubor "main.py"

Soubor definuje prvotní funkci uživatelského nastavení (viz Kód 2), kde je možné upravit parametry algoritmu dle svých potřeb. Těmi jsou: počet a rozložení časových intervalů ("time_intervals"), maximální kapacita vozidla ("vehicle_capacity"), začáteční ("start_time") a konečný ("end_measurement_time") čas pro účely opakovaného provádění měření s jinými startovacími časy. Například, pokud je "start_time" nastavena na 10:00 a "end_measurement_time" na 12:00, algoritmus se spustí třikrát v časech 10:00, 11:00 a 12:00 s přednastaveným hodinovým rozestupem. Výsledky algoritmu jsou posléze uloženy do seznamů a exportovány do excelového souboru (viz Obrázek 14).

Kromě toho přidává dodatečná omezení na "post-procesingové" výpočty s cílem zvýšení efektivnosti. Ty umožňují nastavení maximálního času provádění ("execution_time_limit") v sekundách. Po dosažení této hodnoty se výpočet jedné sady operátorů ukončí a pokračuje se další. Počet sad, tedy počet opakování, se určuje maximálním počtem kalkulací ("max_iterations").

Kód 2 Uživatelské nastavení algoritmu

```
def algorithm_settings():
    """Setting datetime format: Year, Month, Day, Hour, Minute, Second"""
    start_time = datetime(2022, 10, 28, 11, 0, 0)

    """Setting ending time of measurement format: Hour, Minute, Second"""
    end_measurement_time = time(15, 0, 0)

    """Setting the maximum vehicle capacity."""
    vehicle_capacity = 13

    """Setting the upper bounds of the time intervals."""
    time_intervals = [time(hour, 59, 59) for hour in range(2, 24, 3)]

    """Operators (post processing) settings."""

    """Setting the maximum execution time for operators."""
    execution_time_limit = 10000  # In seconds

    """Setting the maximum number of post processing iterations
       (calculations) for route modification."""
    max_iterations = 10

    return start_time, vehicle_capacity, time_intervals,
           end_measurement_time, execution_time_limit, max_iterations
```

Využívá také importované moduly obsahující funkce pro vytvoření datového modelu, validaci dat, post-processingové operace a generaci map. Jelikož spuštění programu je cyklické, je nutné ukládat výsledky každého jednotlivého běhu do lokálních seznamů. Právě tyto seznamy používá funkce "export_to_excel" na konci ukázky kódu 3.

Po úspěšné validaci dat začíná výpočet funkcí "clark_and_wright", která vrací seznam postupně navštívených uzlů a celkový čas. Tyto dvě položky formují "původní trasu", ze které je generována mapa svozu. Posléze je původní trasa upravována funkcí "post_processing", která vrací optimalizované trasy spolu s celkovým časem svozu. Z těchto aktualizovaných položek opět vzejde mapa. Díky oběma mapám je poté možné provést porovnání změn před a po modifikaci původní trasy a následně provést zpětnou analýzu. Výsledné změny původní trasy se zejména odvíjí na kvalitě post-processingových operací, které výhradně ovlivňují nastavená omezení. Vyšší hodnoty "max_iterations" a

"execution_time_limit" umožní heuristice více času a prostoru pro nalezení lepších výsledků, ale za cenu zvýšení výpočetní doby.

Kód 3 Funkční logika celého programu

```
def algorithm_calculation(start_time, vehicle_capacity, time_intervals,
                          end_measurement_time, execution_time_limit,
                          max_iterations):
    """Lists needed to create a Data frame for excel output."""
    all_start_times = []
    all_routing_times = []
    all_full_routes = []
    all_spent_times = []

    while start_time.time() <= end_measurement_time:
       data = create_data_model(start_time, vehicle_capacity, time_intervals)

       if data_validation(data):
            """Calculated collection time and route after
               Clark & Wright algorithm."""
            points, routing_time = clark_and_wright(data)
            map_generator(points, start_time, data['coordinates'], False)

            """Execution time limit for post processing operators."""
            max_execution_time = datetime.now() +
                                 timedelta(seconds=execution_time_limit)
                                 if execution_time_limit else None

            """Calculated modified collection time and route
               after nodes change using operators."""
            modified_points, modified_routing_time = post_processing(data,
                   points, vehicle_capacity, routing_time, max_iterations,
                   max_execution_time)
            map_generator(modified_points, start_time,
                          data['coordinates'], True)

            all_start_times.append(start_time.time())
            all_routing_times.append(routing_time.time())
            all_spent_times.append(f"{modified_routing_time - start_time}")
            all_full_routes.append(
                         '->'.join(str(node) for node in modified_points))

        start_time += timedelta(minutes=60)

    if all_start_times and all_routing_times and
       all_full_routes and all_spent_times:
        """Export algorithm results to excel."""
        export_to_excel(all_start_times, all_routing_times, all_spent_times,
                        vehicle_capacity, all_full_routes)
```

V neposlední řadě obsahuje také funkci main(), která je vstupním bodem pro spuštění programu (viz Kód 4). Uvnitř funkce se zavolá metoda "algorithm_calculations" se všemi argumenty nastavenými funkcí "algorithm_settings".

Kód 4 Hlavní funkce pro spuštění modelu

```
def main():
    algorithm_calculation(*algorithm_settings())
```

7.3.3 Soubor "data_model.py"

Definuje dvě hlavní funkce, které se starají o korektní načtení dat z Excelu do slovníku, anglicky "dictionary", které nadále používají heuristické algoritmy. Třetí se volá ke konci programu a stará se o export výsledných dat zpět do aplikace Excelu.

V přiložených ukázkách jsou pro přehlednost a snadnější pochopení vloženy komentáře označovány pomocí znaku "#" vyznačující se šedou barvou nebo uvnitř uvozovek vyznačující se barvou zelenou.

Základem je definovat slovník (viz Kód 5) pro uložení všech potřebných dat. Těmi jsou 3 parametry od uživatele: čas začátku svozu ("start_routing_time") ve formátu HH:MM:SS , kapacita vozidla ("vehicle_capacity") určující maximální povolené zaplnění vozidla, horní meze všech osmi časových intervalů ("upper_bounds_of_time_interval"). Parametr " number_of_intervals" určuje počet intervalů a s každým příbytkem či úbytkem počtu intervalů musí být modifikován. Ostatní záznamy jsou uloženy v Excelu, které se načítají pomocí funkce ukázané kódem 6. Jsou to: míry odpadu v každém uzlu "Waste", časové intervaly značící se od "0" po "7" a geografické souřadnice pro pozdější generování map "Coordinates".

Kód 5 Definování slovníku pro uložení dat

```
def create_data_model(start_time, vehicle_capacity, time_intervals):
    """Stores the data for the heuristic algorithm."""
    data = {
        # Time interval 0: 0:00 - 2:59
        0: [],
        # Time interval 1: 3:00 - 5:59
        1: [],
        # Time interval 2: 6:00 - 8:59
        2: [],
        # Time interval 3: 9:00 - 11:59
        3: [],
        # Time interval 4: 12:00 - 14:59
        4: [],
        # Time interval 5: 15:00 - 17:59
        5: [],
        # Time interval 6: 18:00 - 20:59
        6: [],
        # Time interval 7: 21:00 - 23:59
        7: [],
        'number_of_intervals': 8,  # Number of time intervals.
        'waste': [],  # Waste amount of each node.
```

```
        'vehicle_capacity': vehicle_capacity,  # Vehicle capacity.
        'service_time': [],  # Time required to service each node.
        'start_routing_time': start_time,  # Waste collection start time.
        'upper_bounds_of_time_interval': time_intervals,  # Upper limit
        'coordinates': [],  # Coordinates of each node.
    }

    """Inserting data from an excel sheets into lists."""
    import_from_excel(data)

    return data
```

Slovníky jsou implementací datové struktury v Pythonu ukládající data ve formě "klíč : hodnota", známé též jako asociativní pole. Definuje se pomocí složených závorek a každý použitý klíč ve slovníku musí být unikátní neboli není povoleno užití duplikátů. Tím pádem se nemůže se stát, že slovník obsahuje 2 záznamy se stejným klíčem. Naopak hodnoty za dvojtečkou se mohou opakovat a skládat se z jakýkoliv datových typů.

Namísto slovníku mohou být použity alternativy typu seznamů ("lists") nebo "DataFrame". Všechny alternativy jsou si velice podobné a záleží na preferenci daného uživatele. Jediný rozdíl je alternativa "Tuples", které jsou po vytvoření neměnné a uplatnění v tomto případě není bráno v potaz, kvůli potřebě modifikovat data v průběhu programu.

Po vytvoření slovníku, který je určen jako cíl pro uložení dat je potřeba jeho následné načtení vstupními daty. Pro tento účel byla vybrána knihovna "pandas", poskytující užitečné funkce pro import z různých zdrojů. Knihovna nabízí uživatelsky přívětivé využití a poskytuje několik variant nastavení.

Použití zahrnuje pouze specifikace každého jména listu v Excelu, který chceme vybrat a je třeba přesně definovat umístění souboru Excel v adresářové struktuře počítače. Nakonec je nutné zvolit místo importu, v našem případě do slovníku a funkce může pracovat. Dále je dobré nastavit různé znaky pro doplnění chybějících dat, jako prázdné řetězce, nuly nebo anglické zkratky "NaN" (Not a number). či "NA" (Not available) symbolizují nedostupnost čísel.

Nesmí se zapomenout na import knihovny před jejím následným použitím. Naštěstí většina programových prostředí dokáže upozornit uživatele na chybějící import ještě před samotným spuštěním kódu. Ukázka (viz Kód 6) znázorňuje nahraní všech dat do slovníku.

Kód 6 Import dat pomocí knihovny Pandas

```
import pandas as pd
import os.path
def import_from_excel(data):
    """Exporting data from Excel and adding them into lists."""

    input_file = 'route_data.xlsx'
    input_dir = './files/input_files/'
    full_path = os.path.join(input_dir, input_file)
    sheets = ['3.00','6.00','9.00','12.00','15.00','18.00','21.00','24.00']
    for i, sheet in enumerate(sheets):
        tmp = pd.read_excel(full_path, sheet_name=sheet, na_values='NA',
                            index_col=0)
        data[i] = tmp.values.tolist()

    tmp = pd.read_excel(full_path, sheet_name='Service time', na_values='NA',
                        index_col=0)
    data['service_time'] = tmp['Service time'].values.tolist()

    tmp = pd.read_excel(full_path, sheet_name='Waste', na_values='NA',
                        index_col=0)
    data['waste'] = tmp['Waste'].values.tolist()

    tmp = pd.read_excel(full_path, sheet_name='Coordinates', na_values='NA',
                        index_col=0)
    data['coordinates'] = tmp.values.tolist()
```

Celý proces je napsán dvěma způsoby. Prvním je postupné vypisování listů, což je vidět v dolní části kódu 6. Druhý, elegantnější způsob, je vložení vstupních listů do seznamu s postupnou změnou jednotlivých názvů pomocí cyklu. Nicméně, druhý způsob je aplikovatelný pouze za předpokladu, že klíčové hodnoty ve slovníku jsou čísla a zároveň mají shodné hodnoty jako iterující cyklus. Proto například názvy klíčových názvů "service_time", "waste" nebo "coordinates" musí být vypsány prvním způsobem, tedy vypisováním jednotlivých listů bez možnosti cyklu.

Po dokončení výpočtu všech zadaných tras je důležité výsledky uchovat pro případnou analýzu. Data k exportu musí být vložena v případě knihovny "pandas" do datové struktury "DataFrame". Ten umožňuje nahrát pouze ty údaje, které pro účel analýzy potřebujeme, včetně času začátku svozu ("Start time"), času konce svozu ("End time"), času stráveného svozem ("Spent time"), maximální povolené kapacity vozidla ("Max vehicle capacity") a řetězce uzlů v pořadí jejich postupného svozu ("Full route (0 = Depot)"). Kromě toho se také specifikuje adresář pro uložení souboru v lokálním uložišti a jeho název. Ukázka je zobrazena v kódu 7.

Kód 7 Export výsledných dat do Excelu

```
def export_to_excel(all_start_times, all_routing_times, all_spent_times,
                    vehicle_capacity, all_full_routes):
    """Creation a Data frame and an excel file from the algorithm results."""

    data = {
        'Start time': all_start_times,
        'End time': all_routing_times,
        'Spent time': all_spent_times,
        'Max vehicle capacity': vehicle_capacity,
        'Full route (0 = Depot)': all_full_routes
    }

    df = pd.DataFrame(data)

    output_file = 'route_results.xlsx'
    output_dir = './files/output_files/'
    full_path = os.path.join(output_dir, output_file)

    """Export to excel and set dynamic column width and center align."""
    with ExcelWriter(path=full_path) as writer:
        df.to_excel(writer,  sheet_name='results', index=False, na_rep='NaN')
        for column in df.columns:
            column_width = max(df[column].astype(str).map(len).max(),
                               len(column))
            col_idx = df.columns.get_loc(column)
            workbook = writer.book
            new_format = workbook.add_format()
            new_format.set_align('center')
            writer.sheets['results'].set_column(col_idx, col_idx,
                                                column_width, new_format)
```

Základní nastavení zvolené funkce "ExcelWriter" (viz Kód 7) využívá pevnou šířku pro všechny sloupce. To však v případě práce s uzly není praktické, jelikož po přidání nebo odebrání uzlu by pevná šířka opět neodpovídala aktuálnímu obsahu buňky a text by byl "useknutý". Za tímto účelem byla v implementaci funkce "export_to_excel" zvolena dynamická šířka sloupců, která se přizpůsobuje obsahu buňky a zajišťuje tak, aby všechna data byla čitelná a srozumitelná.

Výsledný soubor v obrázku 14 je defaultně ukládán do projektové složky s názvem "output_files".

	A	B	C	D	E
1	Start time	End time	Spent time	Max vehicle capacity	Full route (0 = Depot)
2	10:00:00	15:16:11	4:51:18	13	0->12->18->23->19->17->13->3->0->5->10->11->7->8->9->6->0->14->15->16->20->21->22->4->0->2->1->0
3	11:00:00	16:20:49	4:51:23	13	0->12->18->23->19->17->13->3->0->14->15->16->20->21->22->4->0->2->1->0->5->10->11->7->8->9->6->0
4	12:00:00	17:17:21	4:55:35	13	0->22->21->20->17->19->23->18->0->12->16->14->15->13->4->0->6->5->10->11->7->8->9->3->0->2->1->0
5	13:00:00	18:20:24	4:54:56	13	0->3->4->22->21->20->14->15->0->2->1->5->10->11->7->8->9->0->12->18->23->19->17->16->13->6->0
6	14:00:00	19:32:41	4:53:26	13	0->12->16->17->19->23->18->11->6->0->13->14->15->20->21->22->4->0->2->1->3->9->7->8->10->5->0
7	15:00:00	20:09:19	4:44:35	13	0->3->4->22->21->7->8->9->5->0->6->10->11->18->23->19->17->20->0->12->16->14->15->13->2->1->0

Obrázek 14 Výsledky výpočtů heuristik exportovaných do Excelu

7.3.4 Soubor "data_validation.py"

Po úspěšném importu a nahrání dat přichází jejich validace. Zároveň se tento proces stává nezbytným, při řešení jakéhokoliv typu úlohy pracující se vstupními daty. Neboť je důležité zajistit správnost dat a také že odpovídají požadovanému očekávání. Poněvadž nesprávná data mohou vést k chybám a k jiným výsledkům, které se na první pohled nemusí jevit jako nesprávné a mnohdy je těžké objevit důvod vzniku. Validace je tedy nepostradatelným krokem, který napomáhá zajistit správnost výsledků a minimalizovat chyby v procesu tvorby datového modelu.

Při validaci (viz Kód 8) se ověřuje nulová diagonála, nezápornost a nenulovost všech hodnot mimo diagonálu, stejný počet záznamů na ose x a y, neprázdnost matice a také zda odpad každého uzlu nepřevyšuje maximální povolenou kapacitu vozidla. Pokud jsou data neplatná, musejí být opravena nebo nahrazena platnými daty pro zajištění správné funkce programu.

Kód 8 Validace vstupních dat

```
def data_validation(route_data):
    """Matrix validation"""
    number_of_intervals = route_data['number_of_intervals']
    for time_interval in range(number_of_intervals):
        data = route_data[time_interval]

        """Check if any matrix is empty."""
        if not data:
            print(f"Matrix {time_interval} is empty!")
            return False

        """Check if the matrix has zero diagonal."""
        diagonal = [data[i][i] for i in range(len(data))]
        if any(element != 0 for element in diagonal):
            i = diagonal.index(next((x for x in diagonal if x != 0), None))
            print(f"Matrix {time_interval}, List {data[i]} has no zero value
                   at position {i}. Change the value: {diagonal[i]} to 0 to
                   maintain the zero diagonal of the matrix!")
            return False

        """Check if the matrix is square."""
        if any(len(row) != len(data) for row in data):
            print(f"Matrix {time_interval} has different length!")
            return False

        """Check for more than one zero value in a row."""
        for row in data:
            if row.count(0) > 1:
            print(f"Matrix {time_interval}, List {row} has more
                  than one zero value!")
                return False

        """Check for negative values in a row."""
```

```
        for i, row in enumerate(data):
            if any(x < 0 for x in row):
            print(f"Matrix {time_interval}, List {row} has
                  a negative value!")
                return False

         print(f"Input data from time interval {time_interval}
               has been read properly!")

    """Check that the node's waste does not exceed the vehicle's capacity."""
    if any([x >route_data['vehicle_capacity'] for x in route_data['waste']]):
         print("Error: Node has more waste amount than
           the maximum capacity of the vehicle!")
         return False

    return True
```

V případě detekce chyby je běh programu okamžitě zastaven a uživateli je na obrazovku v konzoli vypsána chybová hláška s daným popisem problému, jak ukazuje obrázek 15. Čím detailnější a informativnější tato hláška je, tím snazší je následné nalezení a oprava chybného záznamu. Ovšem důležité je také informovat uživatele o úspěšném dokončení validace, aby mohl bezstarostně pokračovat v dalších krocích s jistotou, že vstupní data jsou v pořádku naformátována a připravena k užití.

```
Input data from time interval 0 has been read properly!
-------------------------------------------------------------------------------------
Input data from time interval 1 has been read properly!
-------------------------------------------------------------------------------------
Input data from time interval 2 has been read properly!
-------------------------------------------------------------------------------------
Matrix 3, List [508, 725, 770, 458, 472, 338, 12, 430, 382, 350, 363, 463, 403, 443, 607, 629,
 742, 840, 907, 1034, 806, 556, 713, 1136] has no zero value at position 6. Change the value: 12
  to 0 to maintain the zero diagonal of the matrix!
```

Obrázek 15 Příklad nalezení chyby ve validačním procesu

7.3.5 Soubor "clark_and_wright.py"

Na začátku je potřeba zjistit aktuální časový interval, z důvodu vybrání správných vstupních dat. Proto se při inicializaci seznamů a proměnných volá funkce "interval_check" (viz Kód 9), která na základě každého času začátku svozu určí odpovídající interval a pokračuje se výpočtem seznamu "savings".

Kód 9 Prvotní určení časového intervalu

```
def interval_check(routing_time, upper_bounds_of_time_intervals):
    """Detection in which time interval the vehicle is currently in."""
    for x in range(0, len(upper_bounds_of_time_intervals)):
        if routing_time.time() < upper_bounds_of_time_intervals[x]:
            return x
```

Každá položka obsahuje dvojici uzlů spolu s číslem znázorňující úsporu času. Díky hodnotám je tento seznam seřazen od nejvyšší hodnoty po nejnižší, a poté je číslo smazáno a dále se pracuje pouze s dvojicemi uzlů. Jelikož se při změně časového intervalu mění i vstupní data, je vždy nutné vypočítat nový seznam, jinak by řešení vedlo k nepřesným výsledkům. Celý postup výpočtu je ukázán níže (viz Kód 10).

Kód 10 Počítání "savings" seznamu pro algoritmus

```
def savings_calculation(route_data, actual_time_interval):
    """Calculating saving value for each pair of nodes."""
    saving_list = []
    for j in range(1, len(route_data[actual_time_interval])):
        for i in range(1, len(route_data[actual_time_interval])):
            if j != i:
              """Equation: s(j,i) = d(D,j) + d(D,j) - d(j,i);
                 where s = savings, d = distance, D = depot"""
              saving_list.append([j, i,route_data[actual_time_interval][j][0]
                                  + route_data[actual_time_interval][0][i]
                                  - route_data[actual_time_interval][j][i]])

    """Descending order of nodes according to their values."""
    saving_list = sorted(saving_list, key=itemgetter(2), reverse=True)
    saving_list = [item[:-1] for item in saving_list]

    return saving_list
```

Hlavní logika algoritmu je založena na 2 hlavních cyklech. První iteruje celý seznam "savings" a hledá potřebného kandidáta splňující určité podmínky, aby mohl být vložen do trasy "route". Druhý cyklus představuje navštívené uzly a ukončuje se v momentě, když všechny uzly náleží trase.

Nutno poznamenat, že algoritmus v každé iteraci pracuje pouze s první položkou seznamu "savings". Každá položka obsahuje dvojici uzlů a jsou k dispozici 4 možnosti zpracování, vždy s náležitou ukázkou pro lepší představu použité techniky.

1) Pokud je trasa prázdná, vlož oba uzly do trasy.

   saving list: [[18, 23], …] ; route : [-] → route : [18, 23]

2) Pokud se oba uzly z páru nacházejí ve trase, přeskoč / smaž položku ze seznamu.

   saving list: [[23, 18], …] ; route : [18, 23] → route : [18, 23]

3) Pokud se první uzel páru rovná poslednímu uzlu v trase, vlož druhý uzel z páru.

   saving list: [[23, 19], …] ; route : [18, 23] → route : [18, 23, 19]

4) Pokud se druhý uzel páru rovná prvnímu uzlu v trase, vlož druhý uzel z páru.

   saving list: [[16, 18], …] ; route : [18, 23, 19] → route : [16,18, 23, 19]

Uzel je začleněn do trasy pouze v případě, že splňuje jednu z podmínek 1, 3 nebo 4. Jestli že spadá do druhé podmínky, pokračuje se následujícím párem do doby, než se nalezne potřebný kandidát. Postup se opakuje, dokud vozidlo nepřekročí aktuální časový interval nebo maximální kapacitu. Pokud je časový interval převýšen, seznam trasy ("route") je vložen do úplné trasy ("full_route") spolu s hodnotou intervalu, poté je smazán a celý proces začíná od znova, dokud všechny uzly nenáleží úplné trase. Pokud je překročena maximální kapacita vozidla, postup je podobný, ale s tím rozdílem, že na konci seznamu "route" je přidáno číslo 0, které indikuje nutnost cestovat do depa a vyprázdnit nákladový prostor.

Změna časového intervalu z 3 na 4:

route : [0, 18, 23, 19] ; full route : [] → route: [] ; full route: [[3, [0, 18, 23, 19]]]

Překročení maximální kapacity vozidla:

route : [0, 18, 23, 17] ; full route : [] → route: [] ; full route: [[3, [0, 18, 23, 17, 0]]]

V prvním / posledním seznamu "route", je na začátek / konec vložen uzel 0 symbolizující výjezd z / návrat do depa.

Podobně jako v souboru "data_model", i zde se výpočty části trasy ("route") ukládají do slovníku "solution"(viz Kód 11), takže mimo jiné dokáže lehce uchovat aktuální informace o časovém intervalu, času svozu a kapacitě zaplnění vozidla.

Kód 11 Použití slovníku pro ukládání dat u Clark & Wright algoritmu

```
...
solution = {
    "Time interval": actual_time_interval,
    "Routing time": route_data['start_routing_time'],  # Current route time.
    "Vehicle capacity": 0,  # Current capacity of the vehicle
    "Route": []  # Part of the full route
}
...
```

Příklad úplné trasy ("full_route") znázorňuje Obrázek 16 skládající se ze 2 seznamů "route". Mezi těmito seznamy došlo ke změně časového intervalu, kde první seznam obsahuje číslo 3, což odpovídá třetímu intervalu, zatímco druhý seznam již obsahuje číslo 4. Dále je uvedena kapacita vozidla, která je pro aktuální řešení "solution" nastavena na hodnotu 8.88.

```
Method: 12
Actual time interval: 4
Saving list: [[11, 18], [11, 23], [23, 11], [18, 11], [8, 18], [8, 23], [10, 18], [10, 23]
Solution: [13, 12, 20, 21, 22]
Actual capacity: 8.88
Full route: [[3, [0, 18, 23, 19, 17]], [4, [14, 15, 16, 0]]]
Visited nodes: [18, 23, 19, 17, 14, 15, 16, 12, 20, 21, 13, 22]
Actual time: 13:45:59
Time spent on the route: 2:45:59
```

Obrázek 16 Konzolový výpis části výpočtu Clark & Wright algoritmu

Důležitou částí je rozpoznání změny intervalu, což je realizováno pomocí slovníku "solution" obsahující aktuální čas svozu, který je porovnáván s horní mezí následujícího intervalu za použití if, elif a else podmínek. Tímto způsobem můžeme po přidání každého uzlu do trasy zkontrolovat časový interval a případně jej změnit.

Je důležité upozornit, že v jazyce Python není k dispozici příkaz "switch" podobně jako v některých jiných jazycích typu C++ nebo C#. Avšak od verze Pythonu 3.10 je dostupná podobná funkcionalita nazvaná "match" s příslušnými "case" větvemi. Přesto je v Pythonu více užívaným postupem využití konstrukce "if-elif-else", jak ukazuje následující kód 11.

Kód 11 Metodika změny intervalů

```
def time_interval_selector(routing_time, upper_bounds_of_time_intervals,
                           actual_time_interval):
    """Selection of time intervals according to the current time."""
    if time(0, 0) <= routing_time.time() < upper_bounds_of_time_intervals[0]:
        actual_time_interval = 0

    elif upper_bounds_of_time_intervals[0] <= routing_time.time() <
        upper_bounds_of_time_intervals[1]:
        actual_time_interval = 1

    ...

    return actual_time_interval
```

Po dokončení se z úplné trasy, kterou vrací "clark_and_wright" funkce generuje mapa a spolu s konečným časem svozu přichází jako vstupní argumenty do post-processingových operátorů umístěných v souboru "post_processing.py".

7.3.6 Soubor "post_processing.py"

Post-processing je navržen způsobem pro postupné zlepšování trasy trasu. Ovšem ne každá modifikace trasy vede k lepším výsledkům ba naopak se může se stát, že originální trasa je optimální již před použitím operátorů anebo některé případy vůbec nejsou proveditelné. Z těchto důvodů je nutno provést kontrolu každé modifikace. Proces je realizován

jednoduchou metodou, která se skládá z jednoho cyklu. Ten postupně prochází seznam "collection_route", v předchozím souboru známý jako "full_route". Díky jeho složení čítající několik podseznamů, z nichž každý obsahuje seznam po sobě jdoucích uzlů vždy s příslušným časovým intervalem je velmi jednoduché ověřit platnost modifikovaného řešení.

O ukládání dat potřebných pro výpočet platnosti trasy se stará třída "Vehicle". Jejími atributy jsou: čas svozu ("routing_time"), aktuální kapacitu vozidla ("capacity") a maximální kapacitu vozidla ("max_capacity"). Použití je podobné jako při použití slovníku, ale nabízí také další funkce typické pro objektově orientované programování. Můžeme také definovat vlastní metody, které mohou provádět různé operace s atributy třídy nebo s ní samotnou za účelem usnadnění práce. V případě kódu 12 byla třída rozšířena o metodu "get_routing_time" sloužící pro vrácení aktuálního času jízdy.

Kód 12 Definice třídy "Vehicle"

```
class Vehicle:
    def __init__(self, routing_time, capacity, max_capacity):
        self.routing_time = routing_time
        self.capacity = capacity
        self.max_capacity = max_capacity

    def get_routing_time(self):
        return self.routing_time
```

I když je demonstrace této metody spíše ilustrativní, jelikož vrací pouze hodnotu atributu a nemá zvlášť zásadní vliv na efektivitu, přesto nám metody pomáhají přizpůsobit kód našim konkrétním potřebám, zejména za účelem zjednodušení. Poněvadž veškerá logika a operace související s danou třídou jsou soustředěny uvnitř. To umožňuje psát čitelnější a lépe organizovaný objektivně orientovaný kód což přispívá ke snadnější údržbě a případnému rozšiřování programu.

Hlavním prvkem post-processingu jsou tzv. operátory, které mají za úkol optimalizovat originální svozové trasy sestavené Clark & Wright algoritmem. V současné době je v projektu k dispozici celkem 25 operátorů různých druhů.

Mezi ty jednodušší se řadí například "Move", který posouvá jeden nebo více uzlů na předem stanovenou nebo náhodnou pozici, nebo "Swap" měnící pozici dvou nebo více uzlů v určitém pořadí. Kromě toho jsou k dispozici i sofistikovanější operátory, jako je "2-opt", který uskutečňuje výměnu dvou hran v trase, dále "3-opt", který realizuje změny mezi třemi hranami v trase, a nakonec "Or-opt", který vykonává výměny mezi jednou hranou v trase a jednou nebo více novými hranami. Všechny tyto operátory jsou oblíbené díky své rychlosti

a schopnosti navrhovat podstatně lepší řešení než ta originální. Je však třeba zmínit, že žádný operátor sám o sobě nemusí vždy najít nejlepší řešení. Proto jsou navrženy jako součást komplexnějších sad obsahujících větší počet operátorů.

Výběr je implementován obdobně jako v případě časových intervalů. Využívá se konstrukce po sobě jdoucích "if-elif-else" podmínek (viz Kód 12). Hodnota "operator_number" vybírá operátor, který právě provádí změny v posloupnosti uzlů dané trasy. V každém okamžiku je možné pracovat s výhradně jedním operátorem současně.

Kód 12 Implementace optimalizačních operátorů

```
def operators(collection_route, position_first, position_second,
              position_third, operator_number):
    """Selecting operators for post modification of collection route."""

    if operator_number == 0:
       """Operator for swapping two nodes: First, Second -> Second, First."""
        collection_route[position_first], collection_route[position_second] =
        collection_route[position_second], collection_route[position_first]

    elif operator_number == 1:
        """2-opt algorithm"""
        collection_route[position_first:position_second] =
        collection_route[position_second - 1:position_first - 1:-1]

    elif operator_number == 2:
        """Or-opt algorithm"""
        segment = collection_route[position_first:position_second + 1]
        collection_route = collection_route[:position_first] +
                           collection_route[position_second + 1:]
        collection_route = collection_route[:position_third] + segment +
                           collection_route[position_third:]
    ...

return collection_route
```

Funkce nabízí dva způsoby pro výběr pořadí operátorů. První způsob je pomocí cyklu náhodně vytvořit seznam 25 po sobě jdoucích čísel, z nichž každé reprezentuje daný operátor ("operator_number"), pokud by operátorů bylo víc, bylo by potřeba vytvořit delší seznam zahrnující i nově vytvořené operátory. Podle této hodnoty je následně vybrána příslušná sekce (viz Kód 12), která modifikuje originální trasu a následně ji předá do metody ověřující její platnost či případné zlepšení oproti původní trase. Druhý způsob se realizuje pouze v případě poslední iterace post-processingu, která je definovaná omezením "max_iterations" v souboru "main.py". Jedná se o předem navrhnutou posloupnost sady operátorů, kdy prvním je jednoduchý "Swap" 2 uzlů, následuje "2-opt", "Or-opt", "3-opt", "Move" a nakonec zbylé "Swap" operátory.

Náhodné pořadí operátorů vnáší do modelu prvky stochastismu a není vždy zaručeno nalezení optima. Z tohoto důvodu je implementován druhý přístup s přesně navrženým pořadím zajišťující ,že i v nejhorším případě budou nalezeny alespoň stejně dobré výsledky, jako v předešlé iteraci nebo běhu programu. Kombinace obou přístupů eliminuje nejistotu spojenou s náhodným výběrem a poskytuje určitou garanci úrovně kvality získaných výsledků.

Pro dosáhnutí ještě větší efektivity při hledání optimálního řešení je celý proces hledání navržen jako smyčka s podmínkou kontrolující zlepšení trasy ("improved"). Operátory jsou vždy spouštěny v cyklech, které zajišťují otestování všech možných kombinací uzlů a hran. V případě, že operátor najde lepší řešení, hodnota "improved" je změněna na "True" a celý cyklus prohledávání trasy se následně opakuje. Smyčka je ukončena, pokud po celou dobu cyklu nebylo nalezeno lepší řešení než to aktuální.

7.3.7 Soubor "map_generation.py"

Funkce, jak již plyne z názvu se zaměřuje na vytváření dvou verzí map pro každý začátek svozu definovaný pomocí proměnné "start_time" umístěné v souboru "main.py". První mapa s názvem "route_map_{hodina}h_{minuta}m_{sekunda}s.html" znázorňuje trasu bez modifikací provedených operátory, tedy ihned po aplikaci Clark & Wright algoritmu. Zatímco druhá verze mapy se vytváří až po skončení modifikačního procesu operátory a je označena jako "route_map_{hodina}h_{minuta}m_{sekunda}s_modified.html". Parametry "hodina", "minuta" a "sekunda" v názvu obou map jsou převzaty právě z parametru "start_time" určující dobu začátku svozu.

Celou funkcionalitu zastřešuje knihovna "folium", která je založena na knihovně "Leaflet" pro JavaScript, poskytující širokou škálu funkcionalit pro tvorbu interaktivních map v jazyce Python. Ve výchozím nastavení generuje mapy v samostatném souboru HTML a poskytuje jednoduché a efektivní rozhraní pro konstrukci různých typů map nejen s bodovými značkami, jak je použito v projektu, ale také teplotní, liniové nebo polygonní mapy. [51]

Pro zvýšení přehlednosti map jsou implementovány dvě vylepšení. Každý výjezd vozidla z depa je na mapě zobrazován svou specifickou barvou. Za účelem zachování odlišnosti použitých barev byl vytvořen seznam obsahující 9 vzájemně odlišitelných a pestrobarevných odstínů. Po vyčerpání seznamu barev jsou další generovány náhodně z celé škály RGB barev. Kromě toho je každému uzlu na mapě přidána značka ve tvaru obrácené kapky ("arrow-down") s číslem označující pořadí návštěvy. Číslo začíná nulou, na mapě ale

označováno slovem "depo" a každý další navštívený uzel (kromě depa) dostane číslo o jedna vyšší.

K samotnému procesu vytvoření mapy je potřeba mít k dispozici několik informací. Za prvé, zeměpisné souřadnice každého uzlu, které jsou obsaženy ve vstupních datech a následně jsou uchovávány ve slovníku. A zadruhé, přesné pořadí uzlů, které vrací funkce pro vytvoření nebo modifikaci tras. Postupně pomocí cyklu je každý uzel vykreslován do mapy vždy s odpovídajícím číslem ve značce a barvou. Po dokončení jsou vygenerované mapy uloženy do složky "output_files" umístěné v kořenovém adresáři projektu.

7.4 Rozdíly v implementaci s předešlou prací

V předešlé bakalářské práci [52] bylo snahou dosáhnout optimálního řešení pomocí CPLEX solveru, což je běžná praxe při užití binárních proměnných v kombinaci s celými čísly. Avšak při použití 24 uzlů výpočetní složitost extrémně narostla a nebylo možné docílit optima, i přestože ukončující podmínka byla nastavena na 15 000 sekund, po které byl výpočet každé trasy přerušen.

Bylo zapotřebí si uvědomit, že řešení složitých kombinatorických úloh, jako je VRP, vyžaduje změnu přístupu. Vhodným řešením se stalo zvolení specializovaných algoritmů a heuristik, které dosahují relativně přesných výsledků v reálném čase, díky kterým se podařilo elegantně dosáhnout efektivních řešení. Použití Clark & Wright savings algoritmu ve spojení se sadou optimalizačních operátorů vedlo k mnohem lepším výsledkům než v předešlé práci s jiným typem přístupu. Srovnání obou typů řešení bude ukázáno a popsáno v následující kapitole.

Další změnou je umožnění svozovému vozidlu, které dosáhne maximální kapacity, vrátit se během jedné trasy zpět do depa, poté vyprázdnit svůj náklad a pokračovat ve svozu zbývajících uzlů. Tímto novým přístupem je eliminována potřeba použití více vozidel, pokud celková kapacita odpadu v uzlech přesahuje maximální kapacitu jednoho vozidla, což bylo řešení v minulé práci. Dá se říct, že tato změna více odpovídá reálným požadavkům, neboť zvolení dvou a více vozidel je nákladově dražší možnost, a díky tomu můžeme docílit vyšší celkové úspory nákladů a zdrojů na svoz odpadu.

Nakonec došlo ke změně v nastavení časových intervalů, kdy původní konfigurace čítala 7 nepravidelně rozložených intervalů. Ty byly rozděleny po dvou , čtyřech nebo šesti hodinových bloků. Nový přístup čítá 8 rovnoměrně rozdělených tří hodinových intervalů ve

snaze dosáhnout lepších výsledků v řešení dané problematiky. Další zvýšení počtu časových bloků by mohlo přispět k ještě přesnějším výsledkům než v dosavadních experimentech.

Bohužel tato práce nemá možnost použít více svozových vozidel, jelikož by to vyžadovalo úplné přepracování zvoleného přístupu. Toto omezení bude dále popsáno v závěru práce v části "Možnosti zlepšení budoucího výzkumu". Nicméně celkově lze konstatovat, že nová verze úlohy přináší vylepšení a ve srovnání s předchozí verzí práce je mnohonásobně rychlejší.

8 VÝSLEDKY MODELU

Výsledky provedených měření jednoho vozidla jsou rozděleny do dvou skupin, které představují dva různé scénáře. První skupina znázorňuje simulaci VRP, kde množství odpadu ve všech navštívených kontejnerech nepřesáhlo maximální povolenou kapacitu vozidla. V tomto případě nebylo nutné vyprazdňovat odpad během trasy. Druhá skupina zobrazuje CVRP scénář, kde celkové množství odpadu přesáhlo maximální povolenou kapacitu vozidla. To vyžadovalo opakované návštěvy depa za účelem vyprázdnění nákladového prostoru. Výsledky obou variant jsou prezentovány prostřednictvím přiložených tabulek, grafů a map.

Pro každý případ bylo vyhotoveno 24 měření, vždy s hodinovým rozestupem. První případ je navíc porovnáván s předešlou bakalářskou prací, což představuje 2 a 3 sloupec v tabulce 9, přičemž poslední sloupec poukazuje na úsporu času dosaženou v aktuální práci ve srovnání s předchozí prací. Vzhledem k tomu, že předchozí práce [52] vykonávala měření s použitím zaokrouhlených časů průjezdů mezi uzly v minutách, zatímco aktuální práce obsahuje přesné a nezaokrouhlené časy v sekundách, byla provedena nová sada měření s použitím aktuálních dat, aby bylo možné realizovat porovnání obou typů řešení.

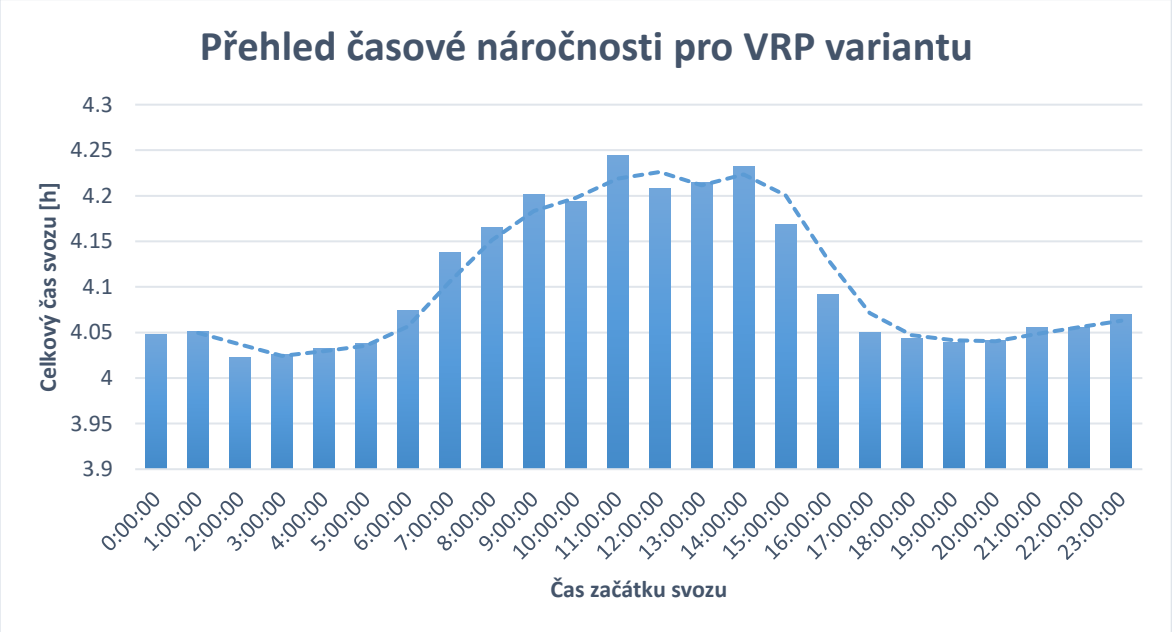
8.1 Měření VRP varianty

Tabulka 9 Výsledné časy měření VRP varianty

Čas začátku svozu	Celkový čas svozu v aktuální práci	Celkový čas svozu v předešlé práci	Ušetřený čas
0:00:00	4:01:11	4:12:44	0:11:33
1:00:00	4:03:05	4:13:37	0:10:32
2:00:00	4:01:20	4:15:14	0:13:54
3:00:00	4:01:35	4:15:20	0:13:45
4:00:00	4:01:58	4:16:22	0:14:24
5:00:00	4:02:18	4:20:36	0:18:18
6:00:00	4:04:28	4:21:56	0:17:28
7:00:00	4:08:15	4:24:43	0:16:28
8:00:00	4:09:55	4:26:42	0:16:47
9:00:00	4:12:06	4:30:25	0:18:19
10:00:00	4:11:38	4:30:36	0:18:58
11:00:00	4:14:38	4:34:19	0:19:41
12:00:00	4:12:30	4:27:14	0:14:44

13:00:00	4:12:52	4:25:46	0:12:54
14:00:00	4:13:57	4:20:13	0:06:16
15:00:00	4:10:08	4:16:37	0:06:29
16:00:00	4:05:32	4:10:22	0:04:50
17:00:00	4:03:01	4:10:30	0:07:29
18:00:00	4:02:37	4:11:05	0:08:28
19:00:00	4:02:22	4:11:10	0:08:48
20:00:00	4:02:28	4:11:41	0:09:13
21:00:00	4:03:21	4:12:30	0:09:09
22:00:00	4:03:21	4:12:31	0:09:10
23:00:00	4:04:12	4:12:38	0:08:26

Přiložený graf v obrázku 17 zobrazuje průběh časové náročnosti, kdy vozidlo nemusí v průběhu cesty navštěvovat depo. Osa x udává časový průběh dne s hodinovým rozdílem. Naopak osa y označuje celkový čas svozu. Z důvodu vytváření grafu v aplikaci Excel jsou oproti tabulkám všechny časy převedeny na hodiny. Rozsah hodnot je od nejrychlejšího času, tj. 3 hodiny 54 minut (v grafu hodnota 3.9) po nejpomalejší čas, tj. 4 hodiny 18 minut (v grafu hodnota 4.3). Plynulý průběh také ukazuje přerušovaná čára neboli spojnici trendu.



Obrázek 17 Graf časové náročnosti svozů pro VRP variantu

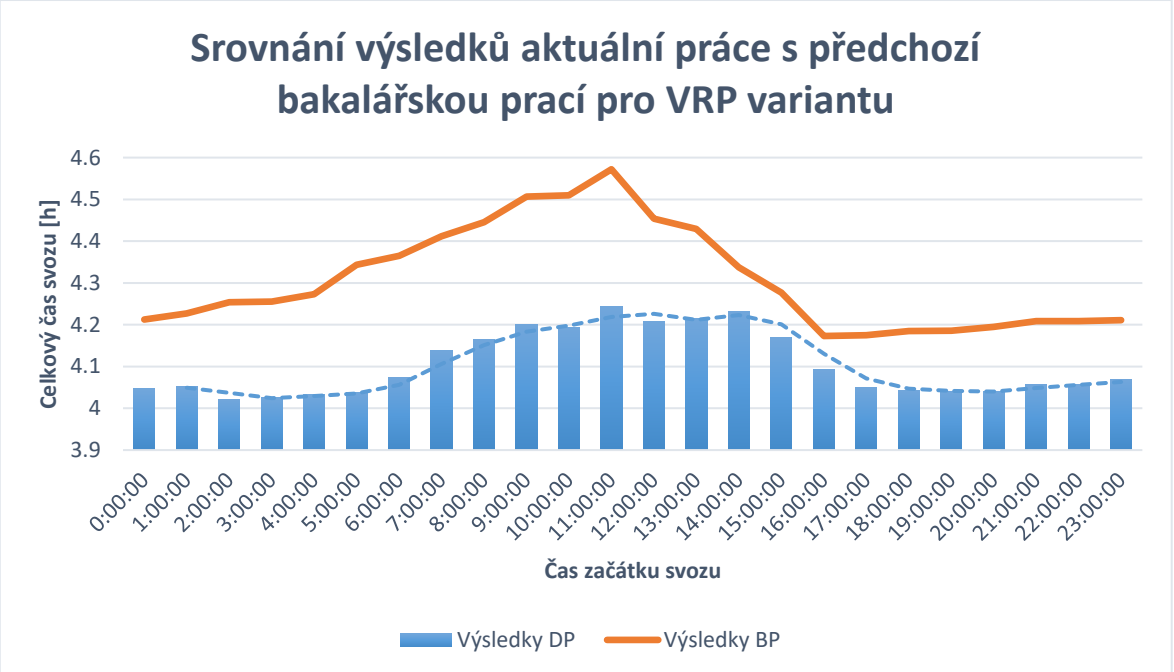
V grafu lze pozorovat plynulý průběh hodnot Celkového času svozu (viz Tabulka 9) pro brzké ranní hodiny, kdy je dopravní situace klidná. Avšak od šesté hodiny ranní je patrný nárůst času svozu způsobený postupným zhoršováním dopravní situace, zapříčiněnou vyšší

koncentrací lidí směřující do práce nebo děti přijíždějící do škol. Situace se začíná uklidňovat kolem čtvrté hodiny odpoledne, kdy lidé postupně opouštějí svá pracoviště a vracejí se zpět do svých domovů. U večerních hodin je situace na silnicích normální s nepatrným mírným nárůstem času svozu, který se velmi podobá hodnotám z ranních hodin.

Rozdíl mezi nejnižší a nejvyšší hodinou v grafu činí 13 minut 27 sekund. Tato hodnota znázorňuje zdržení způsobené zhoršenou dopravou, která vede ke snížení rychlosti či stání v dopravních kolonách. Avšak díky optimálnímu plánování začátků svozu je možno tento čas redukovat a tím snížit celkové náklady spojené se svozem odpadu a produkci zplodin.

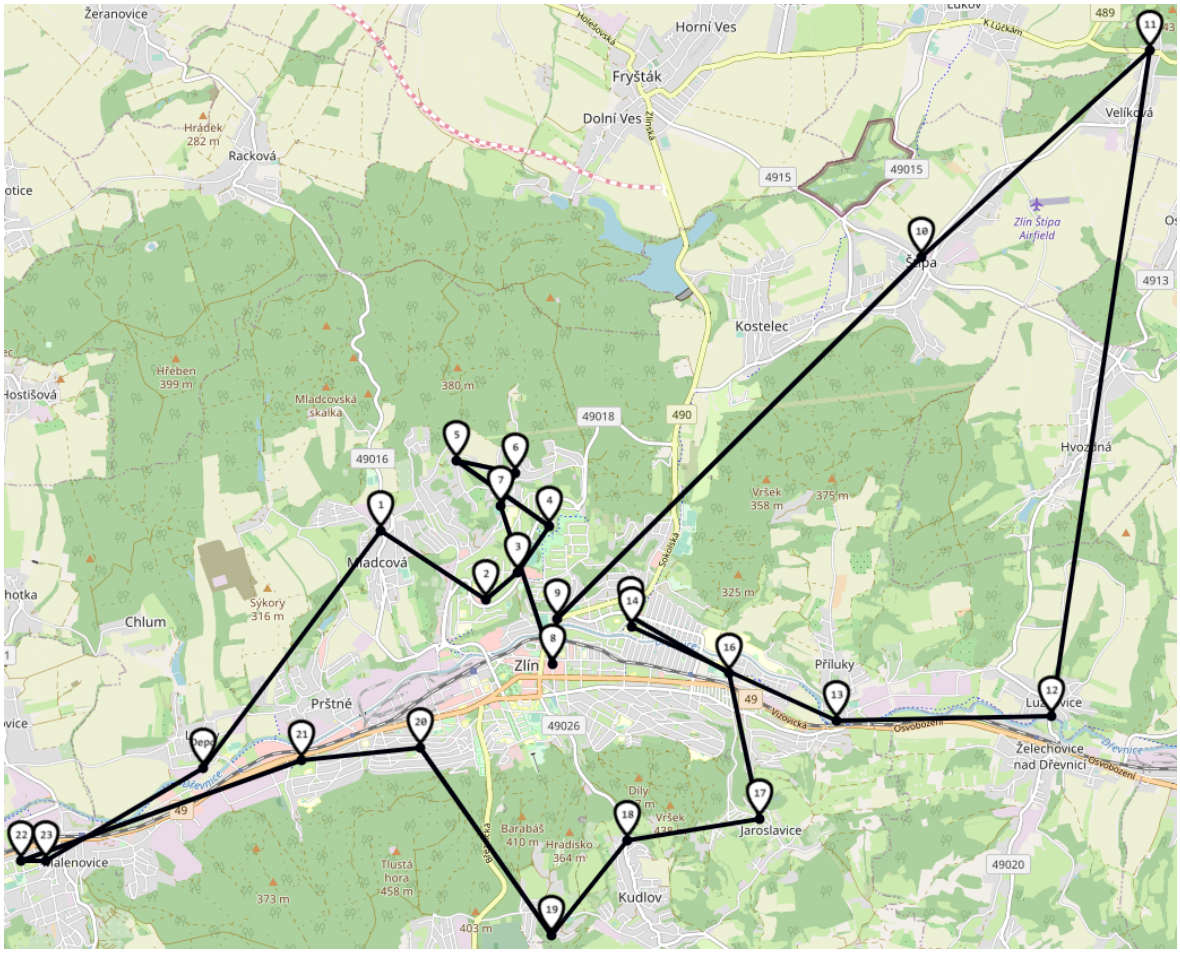
Je ale důležité mít na paměti, že konstruovaný model optimalizuje trasu v závislosti na danou dopravní situaci během celého dne. Proto použití modelu, který tento faktor nezohledňuje, může způsobit výrazné zvýšení svozového času a tím i celkové náklady spojené se svozem.

Graf v obrázku 18 porovnává výsledky aktuální diplomové práce (modrá barva) s předchozí bakalářskou prací (oranžová barva). Je zřejmé, že aktuální přístup dosahuje mnohem lepších výsledků, kdy nejvýraznější rozdíl je patrný zejména v ranních a poledních hodinách. Hlavně v těchto hodinách dochází ke častým dopravním zdržením a vzniku dopravních kolon, která snižuje průměrnou rychlost vozidla a časové zdržení výrazně narůstá. Na rozdíl od předchozího přístupu, který kvůli extrémnímu časovému nároku výpočtů nedokázal spočítat optimální výsledky pro daná vstupní data, se aktuální přístup opírá o heuristiky s minimálním potřebným časem výpočtů, což umožnilo úspěšné vypracování této práce.



Obrázek 18 Graf srovnání výsledů aktuální práce s předchozí prací

8.1.1 Začátek svozu v ranních hodinách

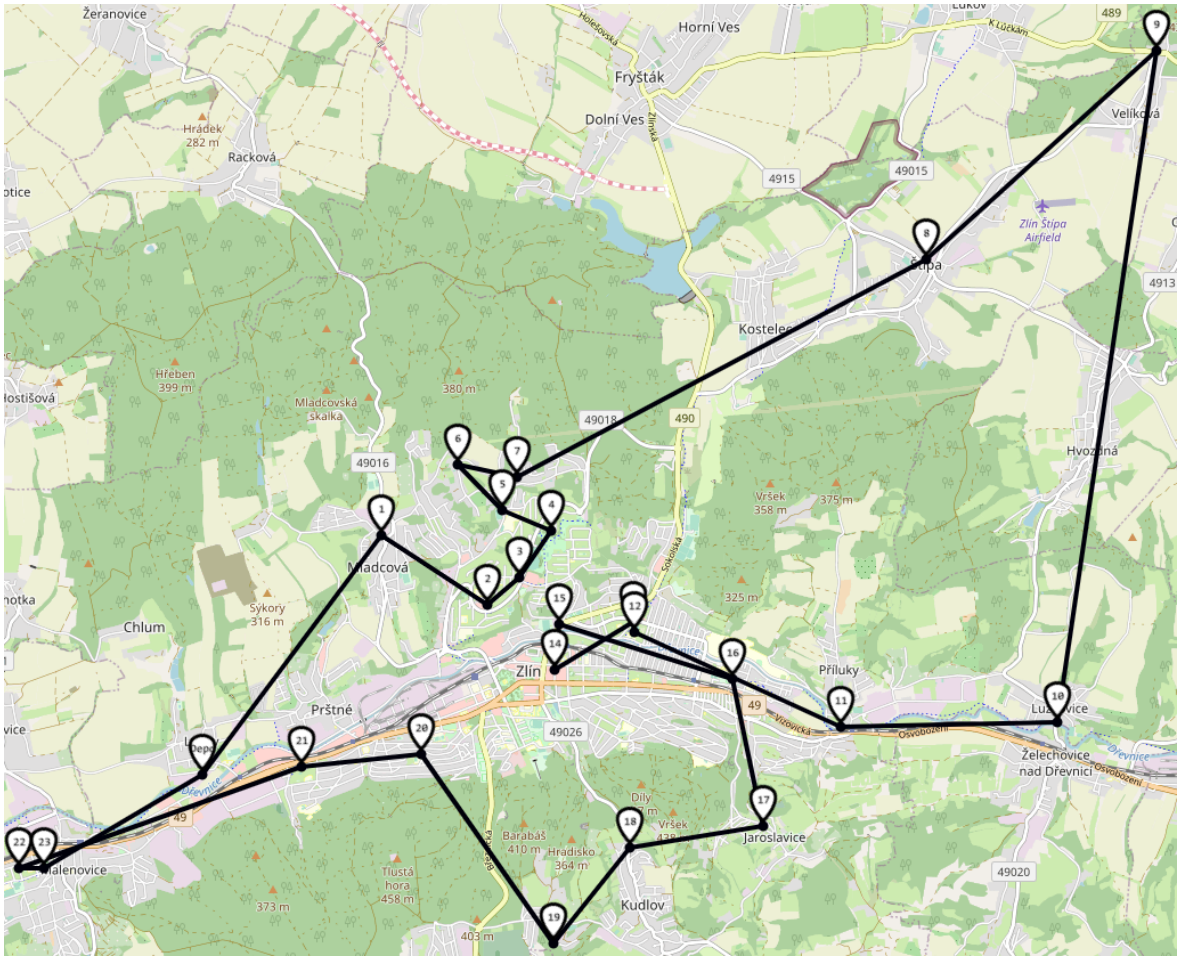


Obrázek 19 Mapa začátku svozu v 5:00 VRP varianty

Na mapě v obrázku 19 je znázorněna trasa vozidla vyjíždějící z depa v 5:00 hodin ráno. Podle předešlého grafu čas se pro pozdější hodiny začíná zvyšovat, tudíž může představovat potenciální trasu pro budoucí svoz. Při podrobnějším zkoumání je očividné, že vozidlo se zcela záměrně vyhýbá hlavním tahům vedoucím do Zlína a centra města, kde tou dobou začíná doprava houstnout a vytvářejí se kolony aut. Na začátku tedy vozidlo směřuje do okrajových částí města a pobývá v centru jen po dobu nezbytně nutnou pro svoz kontejnerů.

Vyhýbání se hlavním přeplněným tahům a volba méně frekventovaných silnicích je strategickým tahem modelu, který mu umožňuje efektivně minimalizovat čas strávený na cestách.

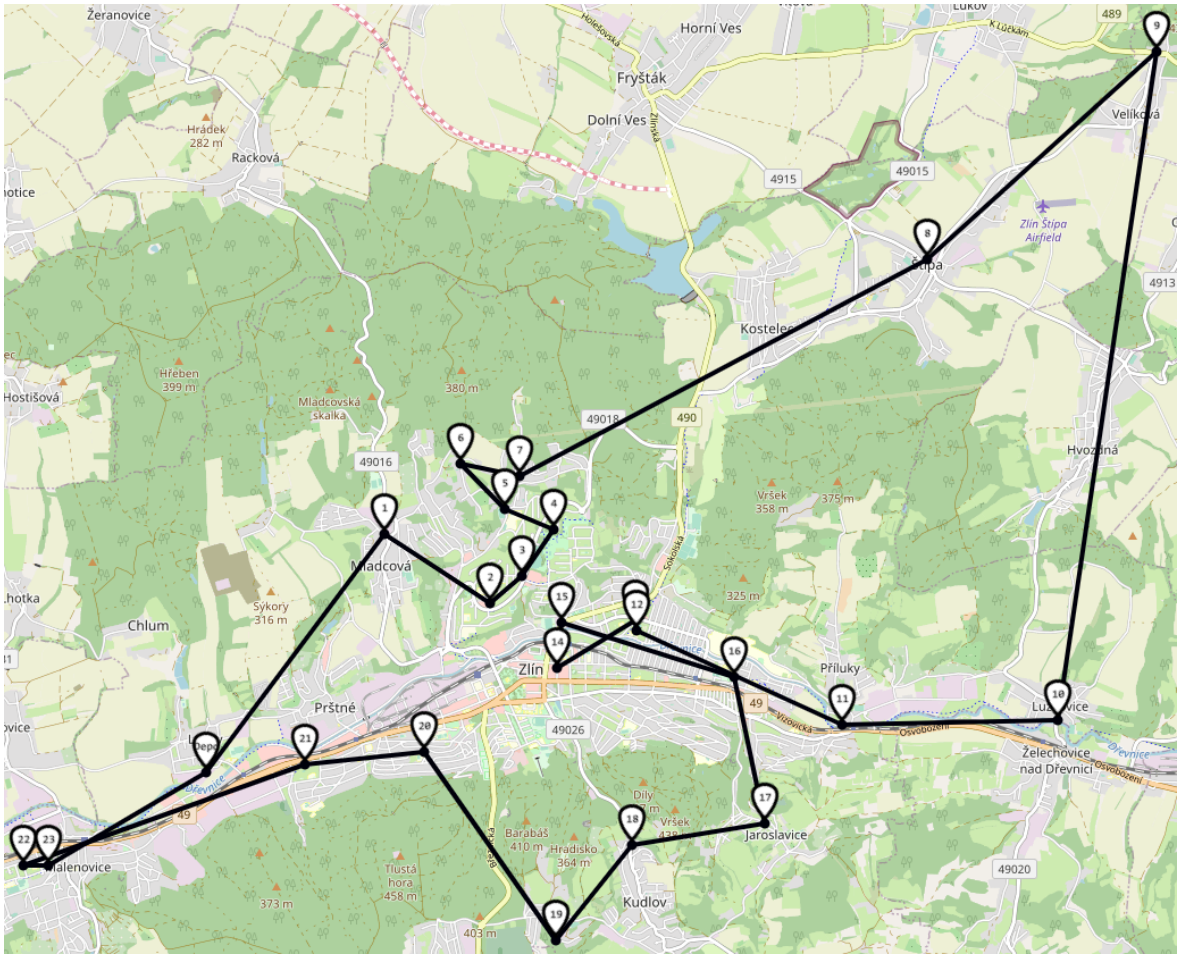
8.1.2 Začátek svozu v dopoledních hodinách



Obrázek 20 Mapa začátku svozu v 11:00 VRP varianty

Naopak trasa v obrázku 20, kdy vozidlo opouští depo přesně v 11 hodin, znázorňuje dobu největší časové prodlevy (viz Tabulka 9). Ačkoliv tento případ dosahuje nejvyšších hodnot v grafu, stále je patrný charakteristický "kruhovitý" útvar celé trasy. Což znamená, že i při zhoršených dopravních podmínkách model dokáže optimálně navrhnout možnou trasu. Nejvýraznější rozdíl oproti předchozí mapě je právě v samotném centru města, kde vozidlo v minulém případě sváželo dva kontejnery v ulici Dlouhá při opouštění sídliště Jižní Svahy a dokončilo sběr v okolí centra až ve druhé půlce cesty. Naopak nyní vozidlo svezlo všechny kontejnery v centru až na zpáteční cestě do Zlína, kdy se dopravní situace začala zlepšovat. Zbývající část trasy vypadá obdobně jako ta předešlá.

8.1.3 Začátek svozu v odpoledních hodinách



Obrázek 21 Mapa začátku svozu v 17:00 VRP varianty

Poslední ukázka VRP mapy v obrázku 21 znázorňuje svoz odpadu v odpoledních hodinách, kdy se dopravní situace na silnicích stabilizuje. Její vzhled je identický s předchozí (viz Obrázek 20). Shodnost obou tras je dána díky nalezení optimální cesty napříč všemi kontejnery bez ohledu na počáteční svozový čas.

Reprezentanti v podobě jednotlivých map pro ranní, polední a odpolední hodiny ukazují, že trasy napříč dnem se dokážou přizpůsobit dle aktuální dopravní situace a dosáhnout tak optimálního řešení v každém okamžiku.

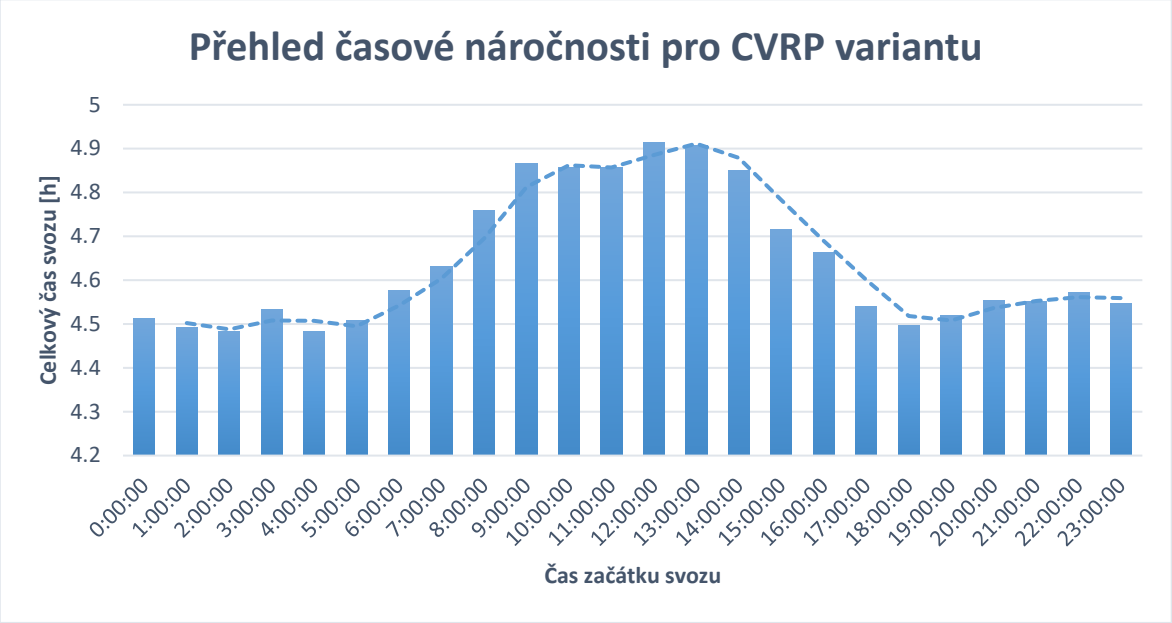
8.2 Měření CVRP varianty

Tabulka 10 Výsledné časy měření CVRP varianty

Čas začátku svozu	Celkový čas svozu v aktuální práci
0:00:00	4:30:43
1:00:00	4:29:32
2:00:00	4:29:01
3:00:00	4:33:59
4:00:00	4:28:57
5:00:00	4:30:27
6:00:00	4:34:36
7:00:00	4:37:31
8:00:00	4:45:33
9:00:00	4:51:59
10:00:00	4:51:26
11:00:00	4:51:23
12:00:00	4:54:53
13:00:00	4:54:29
14:00:00	4:51:00
15:00:00	4:42:55
16:00:00	4:39:45
17:00:00	4:32:20
18:00:00	4:29:51
19:00:00	4:31:09
20:00:00	4:33:15
21:00:00	4:33:06
22:00:00	4:34:17
23:00:00	4:32:51

Následující graf v obrázku 22 zobrazuje časové náročnosti měřených tras, kde vozidlo musí na rozdíl od předchozího případu v průběhu cesty navštěvovat depo za účelem vyvezení svého odpadu. Stejně jako v předchozím případě jsou na ose x ukázány časy začátků všech svozů. Osa y také označuje celkový čas svozu v hodinách. Ve srovnání s předchozím grafem jsou zde rozdíly mezi nejrychlejším časem, tj. 4 hodiny 29 minut a 1 sekunda (v grafu hodnota 4.4836) a nejpomalejším časem, tj. 4 hodiny 54 minut a 53 sekund (v grafu hodnota

4.9147) celkem 25 minut a 52 sekund. Vyšší rozdíl mezi časy je zapříčiněn opakovaným navštěvováním depa vozidlem kvůli daným omezením, které musí dodržovat.

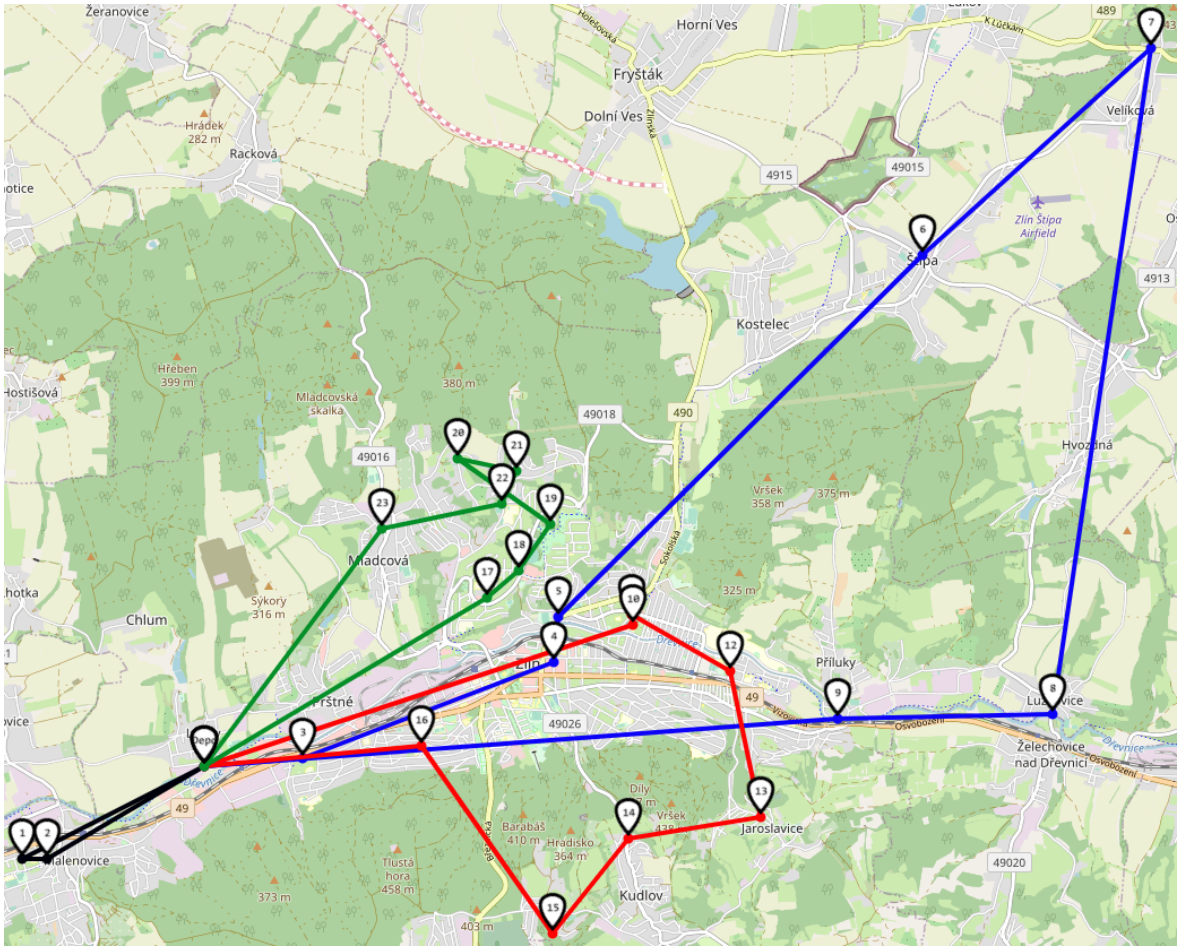


Obrázek 22 Graf časové náročnosti svozů pro CVRP variantu

Graf výše ukazuje podobný průběh jak předchozí (viz Obrázek 17). Dopravní situace zůstává klidná a doba svozu odpadu se relativně neměnní až do 6:00 hodin pouze s malými, avšak zanedbatelnými výkyvy. Avšak posléze časy začínají rapidně stoupat, kdy nejvyšší hodnota času je dosažena v pravé poledne. Po poledni se časové zdržení postupně snižuje až do 17:00 hodin, kdy se situace uklidňuje a ustaluje na normální úroveň. Rozdíl mezi nejnižší a nejvyšší hodnotou je oproti předchozí části vyšší o 12 minut a 25 sekund, zapříčiněným faktem, že vracející se vozidlo musí dorazit do depa, což prodlužuje jeho trasu.

Vzhledem k opakovanému navštěvování depa vozidlem, je zde velmi důležité, aby mapy byly pro lepší uživatelskou přehlednost upraveny tak, aby zohledňovaly tento fakt a odlišovaly tak od sebe jednotlivé úseky svozu specifickými a pestrými barvami. Ty umožňují snadnější identifikaci a vzájemné rozlišení všech segmentů trasy. Tím se snižuje pravděpodobnost vzniku nedorozumění, zmatků a chyb způsobené čtením a interpretací map.

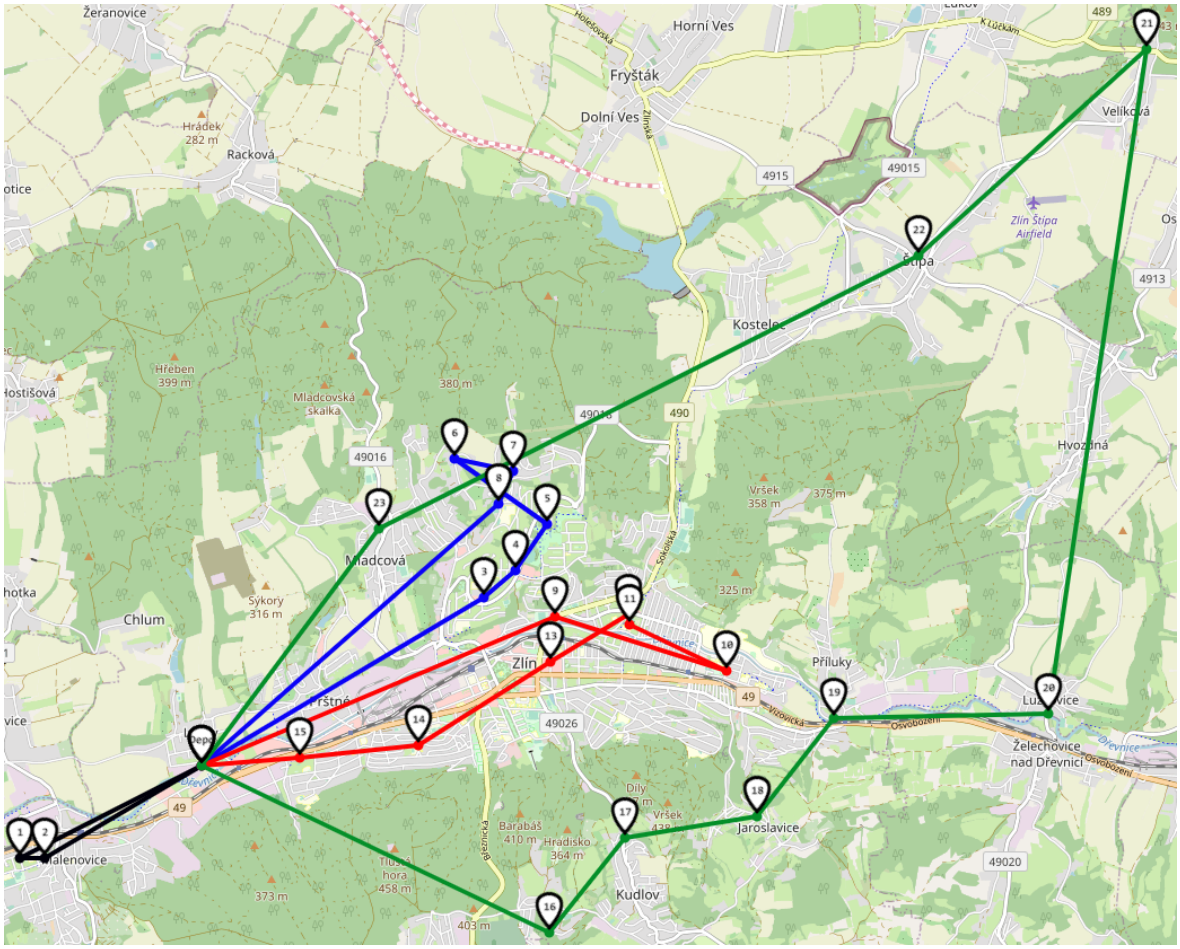
8.2.1 Začátek svozu v ranních hodinách



Obrázek 23 Mapa začátku svozu v 5:00 CVRP varianty

První úsek mapy v obrázku 23 vyznačující se černou barvou je poměrně krátký a obsahuje ve výčtu jen dva kontejnery nacházející se v Otrokovicích, tudíž na opačné straně od ostatních. Následují tři úseky, značené modrou, červenou a zelenou barvou jsou delší a čítají vetší počet zastávek. Zelená barva svou trasu začíná v Mladcové a pokračuje napříč celým sídlištěm, modrá barva obsluhuje kontejnery nacházející se na nejvzdálenějších lokacích od depa a červená barva dokončuje svoz zbylých kontejnerů v okrajových částech města. Celkově jsou úseky rovnoměrně rozděleny a zajišťují úplné pokrytí celé trasy.

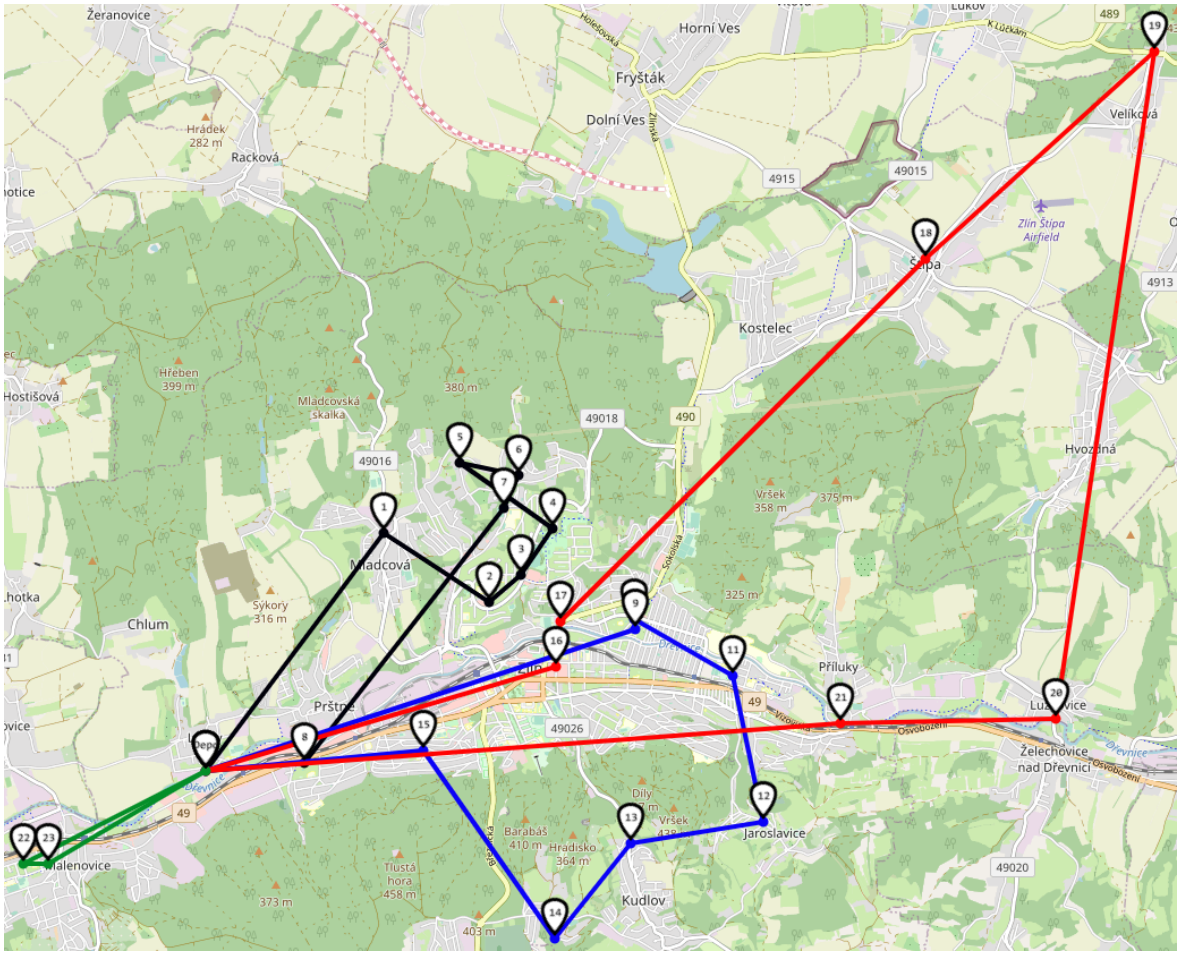
8.2.2 Začátek svozu v poledních hodinách



Obrázek 24 Mapa začátku svozu ve 12:00 CVRP varianty

Stejně jak v předchozích popisech map, i zde obrázek 24 zobrazuje začátek svozu, ovšem nyní pro polední hodiny. Trasa začíná svozem dvou kontejnerů, které jsou značeny černou barvou. Ovšem změna přichází s nadcházející modrou barvou, kde vozidlo oproti předchozímu případu nejede na Mladcovou, ale namísto toho se soustředí čistě na sběr sídliště. Zelená barva pak vytváří nepřekřížený "obvodový" tvar, který zahrnuje nejen nejvzdálenější kontejnery, ale i všechny okrajové části města. Celý svoz dokončuje červený segment zahrnující úsek s největším provozem v centru města. V době času svozu centra se dopravní situace postupně začíná uklidňovat, a proto je tento úsek plánován až na samém konci, což umožňuje úsporu celkového času oproti situaci, kdy by vozidlo nejprve projelo celé centrum města na začátku svozu.

8.2.3 Začátek svozu v odpoledních hodinách



Obrázek 25 Mapa začátku svozu v 17:00 CVRP varianty

Poslední mapa v obrázku 25 se odlišuje především v samotném pořadí jednotlivých segmentu, kdy první etapa obsluhuje celé sídliště namísto Otrokovic. Druhý modrý úsek rozděluje svoz na dvě poloviny mezi centrem města a jeho přilehlých částí, předposlední červený segment směřuje do zbylých okrajových částí města a k nejvzdálenějším kontejnerům v okolních obcích a poslední zelený segment dokončuje svoz Otrokovicemi. Protože se jedná o svoz v odpoledních hodinách, kdy je příznivá dopravní situace, vozidlo může obsluhovat centrum již v útlé fázi svozu bez rizika zpoždění a nárůstu celkové doby svozu.

Vybraní zástupci reprezentující danou část jsou vzájemně více odlišní než v předchozí VRP části, což je způsobeno rozdílností jednotlivých segmentů a také nutností řešit odvoz odpadu poté co je zaplněna maximální kapacita vozidla, což se více podobá omezením v reálných situacích.

8.3 Limity a omezení zvoleného přístupu

Hlavním omezením současného přístupu je nemožnost zvolit větší počet svozových vozidel. Současný model pracuje s použitím pouze jednoho vozidla, které postupně obsluhuje všechny uzly vstupní matice. Nicméně, pro větší množství uzlů a větší města s rozsáhlým vozovým parkem by bylo klíčové implementovat použití více vozidel, které by rovnoměrně rozložily celkový svoz mezi předem definovaný počet. To by mohlo zlepšit celkovou efektivitu a snížit náklady.

Dalším omezením algoritmu je neschopnost akceptovat horší řešení od aktuálního nejlepšího řešení. To může způsobit uvíznutí v lokálním minimu, kdy algoritmus nedokáže najít lepší globální minimum. Princip fungování současných optimalizačních operátorů je na způsob "Hill climbing", kdy postupně akceptují každou modifikaci, která nabízí lepší řešení a snižuje svozový čas. Ovšem s použitím speciálních heuristik typu Simulovaného žíhání by vedlo k možnosti přijmout i horší řešení za účelem vymanění se z lokálního minima a následného nalezení globálního minima.

8.4 Směry případného budoucího výzkumu

Budoucí výzkum se zaměří na obě zmíněné omezení současného přístupu. Následujícím krokem bude implementace heuristiky, například zmíněného Simulovaného žíhání s cílem vymanění se z lokálního minima a dosažení globálního. Dalším krokem bude zaměření se na přetvoření současného modelu a vytvoření nové funkcionality, která umožní použití více svozových vozidel současně. Následně dojde k porovnání obou přístupů, díky kterému bude možné určit optimální počet vozidel k získání nejlepší efektivity a snížení potřebných nákladů.

ZÁVĚR

Po implementaci do jazyce Python byl vytvořený matematický model, za použití Clark & Wright algoritmu a optimalizačních operátorů 3-opt, 2-opt, Or-opt, Swap a Move testován na úloze čítající 23 kontejnerů rozmístěných ve městě Zlín s cílem minimalizace celkového času svozu.

Výpočty byly provedeny pro každou započatou hodinu za účelem simulace dvou možných situací. První představovala případ Vehicle Routing Problem (VRP), kdy vozidlo nemuselo průběžně navštěvovat depo k vyprázdnění nákladu. Druhá situace znázorňovala případ Capacitated Vehicle Routing Problem (CVRP), kdy bylo vozidlo omezeno kapacitou a muselo pravidelně navštěvovat depo.

Z výsledků prvního měření (viz obrázek 17) vyplývá, že optimální začátek svozu pro ranní hodiny je možné provést v 5:00 a pro odpolední hodiny v 17:00. V těchto časech dosahují hodnoty dolních mezí v grafu, což minimalizuje zdržení způsobené zhoršenou dopravní situací. Navíc zde bylo provedeno srovnání výsledků s přístupem použitým v předchozí bakalářské práci [52]. Aktuální řešení přináší zlepšení až 7,23 % (19 minut a 41 sekund) v čase 11:00, kdy dochází k nejvyšším hodnotám způsobené dopravním zdržením.

Podle následující druhé sady měření (viz obrázek 22) je optimální začátek svozu stanoven opět v čase 5:00 pro ranní hodiny a v 17:00 pro odpolední hodiny. To potvrzuje konzistentní výsledky, které naznačují, že tyto časy dosahují nejvyšší efektivity a eliminují nežádoucí čas strávený v dopravních zdržení.

Vzhledem k podobným trendům v obou grafech lze konstatovat, že implementovaný matematický model úspěšně zvládl oba typy úloh (VRP i CVRP) a dosáhl optimálního řešení pro oba scénáře. Tento výsledek přispívá k důvěře v budoucím použití modelu při řešení podobných problémů.

Implementovaný matematický model nabízí potenciál pro zlepšení efektivity a optimalizace sběru odpadu. Jeho použití by mohlo vést ke snížení časových zdržení, nižším nákladům a zlepšení celkového výkonu systému odpadového hospodářství. Další výzkum a zdokonalování modelu mohou přinést ještě lepší výsledky a přispět k efektivnějšímu a udržitelnějšímu nakládání s odpadem ve městském prostředí.

SEZNAM POUŽITÉ LITERATURY

[1] SELVAM, Ammaiyappan a Jonathan WONG. Waste Management and Sustainability: An Introduction. In: WONG, Jonathan W. C., Rao Y. SURAMPALLI, Tian C. ZHANG, Rajeshwar D. TYAGI a Ammaiyappan SELVAM, ed. Sustainable Solid Waste Management [online]. Reston, VA: American Society of Civil Engineers, 2016, 2016-08-29, s. 1-6 [cit. 2023-05-04]. ISBN 9780784414101. Dostupné z: doi:10.1061/9780784414101.ch01

[2] PIRES, Ana, Graça Martinho, Susana RODRIGUES a Maria Isabel GOMES. Sustainable solid waste collection and management. Springer, 2019. ISBN: 978-3-319-93199-9. Dostupné z doi:10.1007/978-3-319-93200-2

[3] FEGAN-WYLES, Sally a Achim STEINER, 2013. Guidelines for National Waste Management Strategies. 1. UNEP. ISBN 978-92-807-3333-4.

[4] GHIANI, Gianpaolo, Gilbert LAPORTE a Roberto MUSMANNO. Introduction to logistics systems planning and control. Hoboken, NJ, USA: J. Wiley, c2004. ISBN: 0- 470-84917-7

[5] WU, Hailin, Fengming TAO a Bo YANG. Optimization of Vehicle Routing for Waste Collection and Transportation. International Journal of Environmental Research and Public Health [online]. 2020, 17(14) [cit. 2023-04-21]. ISSN 1660-4601. Dostupné z: doi:10.3390/ijerph17144963

[6] AWAD, A.R., I. VON POSER a M.T. ABOUL-ELA. The Solution of the Traveling Salesman Problem of Solid Waste Routing in Cities Using Real Genetic Algorithms [online]. In: . - [cit. 2023-04-21]. Dostupné z: doi:10.4203/ccp.84.57

[7] GUENIN, B., J. KÖNEMANN a L. TUNÇEL. A Gentle Introduction to Optimization [online]. Cambridge University Press, 2018 [cit. 2023-05-02]. ISBN 9781107053441. Dostupné z: doi:10.1017/CBO9781107282094

[8] BOYD, Stephen P. a Lieven VANDENBERGHE. Convex optimization. Cambridge: Cambridge University Press, 2004. ISBN 978-0-521-83378-3

[9] GOLDREICH, Oded. P, NP, and NP-Completeness [online]. New York: Cambridge University Press, 2012 [cit. 2023-05-02]. ISBN 9780521192484. Dostupné z: doi:10.1017/CBO9780511761355

[10] SIPSER, Michael. Introduction to the theory of computation. Third edition. Australia: Cengage Learning, [2013]. ISBN 978-1-133-18779-0.

[11] AGRAWAL, Manindra, Neeraj KAYAL a Nitin SAXENA. PRIMES is in P. Annals of Mathematics [online]. 2004, 160(2), 781-793 [cit. 2023-05-03]. ISSN 0003-486X. Dostupné z: doi:10.4007/annals.2004.160.781

[12] ARORA, Sanjeev a Boaz BARAK. Computational complexity: a modern approach. Cambridge: Cambridge University Press, c2009. ISBN 978-0-521-42426-4.

[13] BONDY, J. A., MURTY U. S. R. Graph Theory With Applications. 5th Edition. New York : Macmillan Press, 1982. 264s. ISBN 0-444-19451-7.

[14] DEO, Narsingh. Graph theory: with applications to engineering and computer science. Dehli: PHI Learning Private Limited ; [1974]. Eastern Economy Edition. ISBN 978-81-203-0145-0.

[15] WILSON, Robin J. Introduction to graph theory. Fourth edition. Harlow: Prentice Hall, 1996. ISBN 0-582-24993-7.

[16] RAHMAN, Md. Azizur a Jinwen MA. Solving Symmetric and Asymmetric Traveling Salesman Problems Through Probe Machine with Local Search. In: HUANG, De-Shuang, Vitoantonio BEVILACQUA a Prashan PREMARATNE, ed. Intelligent Computing Theories and Application [online]. Cham: Springer International Publishing, 2019, 2019-07-24, s. 1-13 [cit. 2023-04-22]. Lecture Notes in Computer Science. ISBN 978-3-030-26762-9. Dostupné z: doi:10.1007/978-3-030-26763-6_1

[17] ZELINKA, Ivan, Roman SENKERIK, Magdalena BIALIC-DAVENDRA a Donald DAVENDR. Chaos Driven Evolutionary Algorithm for the Traveling Salesman Problem. In: DAVENDRA, Donald, ed. Traveling Salesman Problem, Theory and Applications [online]. InTech, 2010, 2010-11-30 [cit. 2023-04-21]. ISBN 978-953-307-426-9. Dostupné z: doi:10.5772/13107

[18] DAVENDRA, Donald, ed. Traveling Salesman Problem, Theory and Applications [online]. InTech, 2010 [cit. 2023-04-21]. ISBN 978-953-307-426-9. Dostupné z: doi:10.5772/547

[19] The travelling salesman problem (A guided tour of combinatorial optimisation), edited by E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys. Pp 465. 1985. ISBN 0-471-90413-9

[20] Gavish, B., & Graves, S. C. (1978). The travelling salesman problem and related problems. Working paper GR-078-78. Cambridge, MA: Operation Research Center, Massachusetts Institute of Technology.

[21] George B Dantzig and John H Ramser. 1959. The truck dispatching problem. Management science 6, 1 (1959), 80—91

[22] LAPORTE, Gilbert, Paolo TOTH a Daniele VIGO. Vehicle routing: historical perspective and recent contributions. EURO Journal on Transportation and Logistics [online]. 2013, 2(1-2), 1-4 [cit. 2023-04-23]. ISSN 21924376. Dostupné z: doi:10.1007/s13676-013-0020-6

[23] TOTH, Paolo a Daniele VIGO. Vehicle routing: problems, methods, and applications. Second edition. Philadelphia: Mathematical Optimization Society, [2014]. ISBN 978-1-611973-58-7

[24] TOTH, Paolo a Daniele VIGO. The Vehicle Routing Problem [online]. Philadelphia: Society for Industrial and Applied Mathematics, 2002 [cit. 2021-4-26]. ISBN 978-0-89871-498-2. Dostupné z: doi:10.1137/1.9780898718515

[25] BEN TICHA, Hamza, Nabil ABSI, Dominique FEILLET a Alain QUILLIOT. Vehicle routing problems with road-network information: State of the art. Networks [online]. 2018, 72(3), 393-406 [cit. 2023-04-24]. ISSN 00283045. Dostupné z: doi:10.1002/net.21808

[26] Chryssi Malandraki, Mark S. Daskin Time Dependent Vehicle Routing Problems: Formulations, Properties and Heuristic Algorithms. Transportation Science 26 (3) 185-200

[27] MUGAYSKIKH, Alexander V., Victor V. ZAKHAROV a Tero TUOVINEN. TimeDependent Multiple Depot Vehicle Routing Problem on Megapolis Network under Wardrop's Traffic Flow Assignment. In: 2018 22nd Conference of Open Innovations Association (FRUCT) [online]. IEEE, 2018, 2018, s. 173-178 [cit. 2021-4-24]. ISBN 978-952-68653-4-8. Dostupné z: doi:10.23919/FRUCT.2018.8468273

[28] PICHPIBUL, Tantikorn a Ruengsak KAWTUMMACHAI. An improved Clarke and Wright savings algorithm for the capacitated vehicle routing problem. ScienceAsia [online]. 2012, 38(3) [cit. 2023-04-30]. ISSN 1513-1874. Dostupné z: doi:10.2306/scienceasia1513-1874.2012.38.307

[29] TAILLARD, Éric D. Design of Heuristic Algorithms for Hard Optimization [online]. Cham: Springer International Publishing, 2023 [cit. 2023-04-28]. Graduate Texts in Operations Research. ISBN 978-3-031-13713-6. Dostupné z: doi:10.1007/978-3-031-13714-3

[30] GROËR, Chris, Bruce GOLDEN a Edward WASIL. A library of local search heuristics for the vehicle routing problem. Mathematical Programming Computation [online]. 2010, 2(2), 79-101 [cit. 2023-04-28]. ISSN 1867-2949. Dostupné z: doi:10.1007/s12532-010-0013-5

[31] GENDREAU, Michel a Jean-Yves POTVIN, ed. Handbook of Metaheuristics [online]. Boston, MA: Springer US, 2010 [cit. 2023-04-27]. International Series in Operations Research & Management Science. ISBN 978-1-4419-1663-1. Dostupné z: doi:10.1007/978-1-4419-1665-5

[32] KRITZINGER, Stefanie, Karl F. DOERNER, Richard F. HARTL, G.Ÿnter KIECHLE, Horst STADLER a Senthanal Sirpi MANOHAR. Using Traffic Information for Time-Dependent Vehicle Routing. Procedia - Social and Behavioral Sciences [online]. 2012, 39, 217-229 [cit. 2023-04-04]. ISSN 18770428. Dostupné z: doi:10.1016/j.sbspro.2012.03.103

[33] GUTIÉRREZ-SÁNCHEZ, Alexander a Linda Bibiana ROCHA-MEDINA. VRP variants applicable to collecting donations and similar problems: A taxonomic review. Computers & Industrial Engineering [online]. 2022, 164 [cit. 2023-05-05]. ISSN 03608352. Dostupné z: doi:10.1016/j.cie.2021.107887

[34] LIANG, Yun-Chia, Vanny MINANDA a Aldy GUNAWAN. Waste collection routing problem: A mini-review of recent heuristic approaches and applications. Waste Management & Research: The Journal for a Sustainable Circular Economy [online]. 2022, 40(5), 519-537 [cit. 2023-05-05]. ISSN 0734-242X. Dostupné z: doi:10.1177/0734242X211003975

[35] MA, Yanfang, Wen ZHANG, Cuiying FENG, Benjamin LEV a Zongmin LI. A bi-level multi-objective location-routing model for municipal waste management with obnoxious effects. Waste Management [online]. 2021, 135, 109-121 [cit. 2023-05-05]. ISSN 0956053X. Dostupné z: doi:10.1016/j.wasman.2021.08.034

[36] GMIRA, Maha, Michel GENDREAU, Andrea LODI a Jean-Yves POTVIN. Tabu search for the time-dependent vehicle routing problem with time windows on a road

network. European Journal of Operational Research [online]. 2021, 288(1), 129-140 [cit. 2023-05-05]. ISSN 03772217. Dostupné z: doi:10.1016/j.ejor.2020.05.041

[37] LIU, Lin a Wenzhu LIAO. Optimization and profit distribution in a two-echelon collaborative waste collection routing problem from economic and environmental perspective. Waste Management [online]. 2021, 120, 400-414 [cit. 2023-05-05]. ISSN 0956053X. Dostupné z: doi:10.1016/j.wasman.2020.09.045

[38] Gruler, Aljoscha; Pérez-Navarro, Antoni; Calvet, Laura; Juan, Angel A. "A simheuristic algorithm for time-dependent waste collection management with stochastic travel times". SORT-Statistics and Operations Research Transactions, 2020, Vol. 44, Num. 2, pp. 285-310, https://doi.org/10.2436/20.8080.02.103.

[39] SARMAH, S.P., R. YADAV a P. RATHORE. Development of Vehicle Routing model in urban Solid Waste Management system under periodic variation: A case study. IFAC-PapersOnLine [online]. 2019, 52(13), 1961-1965 [cit. 2023-05-05]. ISSN 24058963. Dostupné z: doi:10.1016/j.ifacol.2019.11.490

[40] Wahyu Katon, DS (2019) Shortest Route for Waste Transportation in Northern Bandung using Vehicle Routing Problem - Clarke and Wright - Saving Method. International Journal of Innovative Techmology & Exploring Enngineering, 8. pp. 679-686. ISSN 2278-3075

[41] MAT, Nur Azriati, Aida Mauziah BENJAMIN a Syariza ABDUL-RAHMAN. ENHANCED HEURISTIC ALGORITHMS WITH A VEHICLE TRAVEL SPEED MODEL FOR TIME-DEPENDENT VEHICLE ROUTING: A WASTE COLLECTION PROBLEM. Journal of Information and Communication Technology [online]. 2017, 17(1) [cit. 2023-05-05]. ISSN 2180-3862. Dostupné z: doi:10.32890/jict2018.17.1.4

[42] AZRIATI MAT, Nur, Aida Mauziah BENJAMIN a Syariza ABDUL-RAHMAN. Efficiency of Heuristic Algorithms in Solving Waste Collection Vehicle Routing Problem: A Case Study. The Journal of Social Sciences Research [online]. 2018, (SPI6), 695-700 [cit. 2023-05-05]. ISSN 24136670. Dostupné z: doi:10.32861/jssr.spi6.695.700

[43] BUHRKAL, Katja, Allan LARSEN a Stefan ROPKE. The Waste Collection Vehicle Routing Problem with Time Windows in a City Logistics Context. Procedia - Social and Behavioral Sciences [online]. 2012, 39, 241-254 [cit. 2023-05-05]. ISSN 18770428. Dostupné z: doi:10.1016/j.sbspro.2012.03.105

[44] Google Maps Platform Documentation | Directions API | Google Developers. Google Developers [online]. Dostupné z: https://developers.google.com/maps/documentation/directions

[45] DU, Ding-Zhu, Panos M. PARDALOS, Xiaodong HU a Weili WU. Introduction to Combinatorial Optimization [online]. Cham: Springer International Publishing, 2022 [cit. 2023-04-05]. Springer Optimization and Its Applications. ISBN 978-3-031-10594-4. Dostupné z: doi:10.1007/978-3-031-10596-8

[46] Třiďme smysluplně - Zlín.eu. Oficiální stránky města Zlína [online]. Copyright © 2023 Magistrát města Zlína [cit. 05.04.2023]. Dostupné z: https://www.zlin.eu/tridme-smysluplne-0

[47] ASEKOL » Červené kontejnery. ASEKOL [online]. Copyright © by ASEKOL a.s. 2023. All rights reserved. [cit. 05.04.2023]. Dostupné z: https://www.asekol.cz/cervene-kontejnery/

[48] Origins of life | MIT - Massachusetts Institute of Technology [online]. Dostupné z: https://web.mit.edu/urban_or_book/www/book/chapter6/6.4.12.html

[49] Python 3.8: Cool New Features for You to Try – Real Python. Python Tutorials – Real Python [online]. Copyright © 2012 [cit. 09.04.2023]. Dostupné z: https://realpython.com/python38-new-features/

[50] Most popular programming languages 2022 [Ranking]. Infrastructure and Private Cloud experts | Stackscale [online]. Dostupné z: https://www.stackscale.com/blog/most-popular-programming-languages/

[51] What is Folium? | Domino Data Science Dictionary. Domino Data Lab | Unleash Data Science at Scale [online]. Copyright © 2023 Domino Data Lab, Inc. Made in San Francisco. [cit. 15.04.2023]. Dostupné z: https://www.dominodatalab.com/data-science-dictionary/folium

[52] ZÁVADA, Dominik. Modelování svozových úloh s dopravními omezeními [online]. Zlín, 2021 [cit. 2023-04-08]. Dostupné z: https://theses.cz/id/edxsql/. Bakalářská práce. Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky. Vedoucí práce Ing. Dušan Hrabec, PhD.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

TSP Travelling Salesman Problem

VRP Vehicle Routing Problem

CVRP Capacitated Vehicle Routing Problem

TDVRP Time-Dependent Vehicle Routing Problem

CWA Clark & Wright savings Algorithm

PSO Particle Swarm Optimization

GA Genetic Algorithm

SA Simulated Annealing

ACO Ant Colony Optimization

TS Tabu Search

API Application Programming Interface

JSON JavaScript Object Notation

VBA Visual Basic for Applications

PYPL Poluparity of Programming Language

NaN Not a Number

NA Not Available

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH CD

Obsah CD:

- Diplomová práce ve formátu .pdf
- Složka "Project":
    - Zdrojový kód pro Clark & Wright algoritmus – "clark_and_wright.py"
    - Zdrojový kód pro zpracování dat – "data_model.py"
    - Zdrojový kód pro validaci dat – "data_validation.py"
    - Zdrojový kód pro spuštění projektu – "main.py"
    - Zdrojový kód pro generování map – "map_generator.py"
    - Zdrojový kód pro optimalizační operátory – "post_processing.py"
    - Soubor s potřebnými balíčky - "requirements.txt"
    - Složka "files":
        - Obsahuje Excel soubor se vstupními daty – "route_data.xlsx"
        - Obsahuje výstupní generované mapy a Excel soubor s výsledky