VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ OPTIMALIZACE NÁKLADNÍ PŘEPRAVY

THESIS TITLE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **MICHAL BERÁNEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MICHAL BIDLO, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Beránek Michal, Bc.**
Program: Informační technologie
Obor: Matematické metody
Název: **Evoluční optimalizace nákladní přepravy**
Evolutionary Optimization of Freight Transportation
Kategorie: Umělá inteligence
Zadání:

1. Seznamte se s problémem optimalizace plánování přepravy a možnostmi jeho řešení.
2. Nastudujte problematiku evolučních algoritmů, přičemž se zaměřte na varianty vhodné pro optimalizaci přepravních plánů.
3. Po dohodě s vedoucím práce zvolte vhodný typ úlohy z oblasti plánování přepravy a pro tento navrhněte postupy optimalizace pomocí evolučních algoritmů.
4. Implementujte navržené techniky a proveďte sady experimentů řešících zvolené úlohy.
5. Získaná řešení srovnejte s výsledky obdrženými pomocí existujících evolučních či konvenčních metod.
6. Zhodnoťte úspěšnost navržených postupů a diskutujte možnosti pokračování projektu.

Literatura:

- Podle pokynů vedoucího projektu.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Bidlo Michal, Ing., Ph.D.**
Konzultant: Rudová Hana, doc. Mgr., Ph.D., FI MUNI
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 30. října 2020

Abstrakt

Práce se zabývá problémem optimalizace nákladní přepravy. Cílem je minimalizace nákladů spojených s přepravou, které vyplývají z ujeté vzdálenosti. Při správném naplánování tras lze tyto náklady výrazně snížit, obzvlášť když se jedná o velký počet zákazníků, které je potřeba obsloužit. Tato práce se soustředí na řešení pomocí evolučních algoritmů, což jsou metody optimalizace založené na principech evoluce. Hlavní zaměření je na problém směrování vozidel s omezenou heterogenní flotilou vozidel. V práci je představeno několik evolučních algoritmů a jejich výsledky jsou porovnány. Nejlepší z nich, evoluční strategie používající lokální prohledávání blízkého okolí, dosahuje podobných, pro některé konkrétní úlohy i lepších výsledků, než jiné existující evoluční algoritmy, vytvořené pro řešení stanoveného problému.

Abstract

The following thesis deals with optimization of freight transport planning. The goal is to minimize expenses connected to transportation, which emerge from travelled distance. The expenses can be heavily reduced, if the routes are correctly planned, especially when there is a large number of customers to be served. This thesis focuses on solving the problem by using the evolutional algorithms, that are optimization methods based on principles of evolution. Thesis concentrates on Heterogeneous Fixed Fleet Vehicle Routing Problem. Thesis introduces multiple evolutional algorithms and their results are compared. The best algorithm, evolutional strategy with local neighbourhood search, achieves similar, for certain tasks even better results, than other existing evolutional algorithms, created to solve given problem.

Klíčová slova

Evoluční algoritmy, optimalizace, problém směrování vozidel, heterogenní flotila, omezená flotila, evoluční strategie, genetický algoritmus, lokální prohledávání, C++, HFVRP

Keywords

Evolutional algorithms, optimization, vehicle routing problem, heterogeneous fleet, fixed fleet, evolutional strategy, genetic algorithm, local search, C++, HFVRP

Citace

BERÁNEK, Michal. *Evoluční optimalizace nákladní přepravy.* Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Bidlo, Ph.D.

Evoluční optimalizace nákladní přepravy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Bidla Ph.D. Další informace mi poskytla paní konzultantka doc. Mgr. Hana Rudová Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Michal Beránek
26. května 2021

Poděkování

Děkuji panu Ing. Michalovi Bidlovi Ph.D. za vedení této práce, poskytování podkladů a motivace potřebné k úspěšnému dokončení této práce. Dále bych rád poděkoval paní doc. Mgr. Haně Rudové Ph.D. za doporučení literatury a postupů týkajících se VRP problému.

Obsah

Kapitola 1

Úvod

Poptávka po přepravě různého zboží v dnešní době roste, obzvlášť dovoz zboží přímo ke koncovému zákazníkovi domů. Důležitým faktorem, který ovlivňuje konečnou cenu zboží, jsou náklady na dopravu. Náklady lze snížit zefektivněním přepravy, tedy zkrácením tras, které musí vozidla ujet, případně i snížením počtu aut potřebných k přepravě stejného množství zboží. Společnosti, které jsou schopné efektivnější přepravy výrobků, tak získávají výhodu oproti ostatním. Pro řešení takových úloh lze využít modelový problém směrování vozidel (angl. *Vehicle routing problem*) a jeho rozšíření. Cílem je nalézt řešení, které má nejnižší cenu přepravy. Algoritmy pro nalezení optimálního řešení pro některé typy problémů nemusí vůbec existovat, nebo jsou příliš pomalé pro praktické použití. Proto se pro řešení těchto problémů v praxi využívají různé heuristiky a metaheuristiky, které sice nutně nenajdou optimální řešení, ale pokud jsou dobře navržené, tak objeví dobré řešení v rozumném čase. Mezi takové metaheuristiky patří například evoluční algoritmy, které poskytují vzor pro řešení optimalizačních problémů a lze je snadno upravit pro řešení různých typů úloh. Evoluční algoritmy vychází z biologických konceptů přirozeného výběru a kódování dědičných vlastností do DNA, která se může různě měnit pomocí genetických operátorů, jako jsou například mutace či křížení. Ukázalo se, že pomocí evolučních algoritmů lze s využitím rozumného výpočetního času nalézt řešení, kterých pomocí konvenčních metod nelze dosáhnout [22]. Aplikace evolučních algoritmů na problém optimalizace nákladní přepravy bude hlavní náplní této práce.

Cílem této práce je průzkum konkrétních evolučních algoritmů, které byly úspěšné při řešení problému směrování vozidel a návrh úprav a rozšíření algoritmů za účelem získat lepší řešení nebo snížit množství potřebného výpočetního času. Primárně se bude jednat o genetické algoritmy a evoluční strategie, zaměřené na jedno z rozšíření problému směrování vozidel a to konkrétně na problém směrování vozidel s omezenou heterogenní flotilou vozidel (angl. *Heterogenous fixed fleet vehicle routing problem* – HFVRP).

Práce je dále rozdělena na několik částí. V kapitole 2 je definice a popis řešeného problému, jeho rozšíření a podrobnější pohled na problém s heterogenní flotilou vozidel. Kapitola 3 obsahuje charakteristiku principů evolučních algoritmů a jejich členění. V kapitole 4 jsou popsána konkrétní existující řešení problému a základní principy algoritmů, se kterými budou navržené metody porovnávány. Kapitola 5 se věnuje návrhu struktury evolučních algoritmů, metod pro efektivní optimalizaci a implementaci. V kapitole 6 jsou popsané a vyhodnocené experimenty. Závěr 7 se zabývá shrnutím a zhodnocením dosažených výsledků a dále diskuzí nad možnostmi dalších rozšíření.

Kapitola 2

Problém směrování vozidel

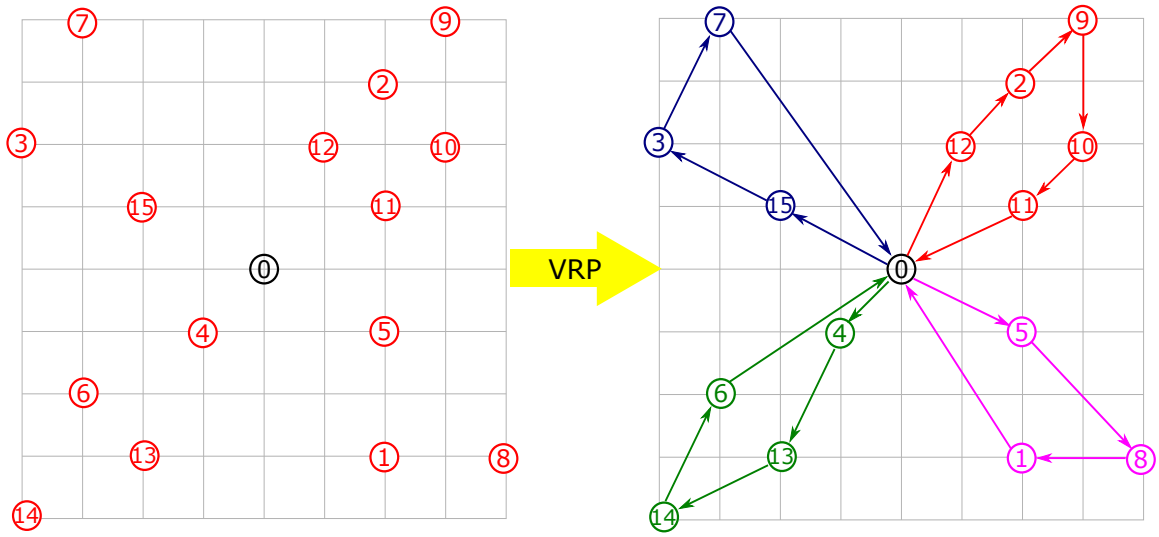
Problém směrování vozidel (angl. *Vehicle routing problem* - dále VRP) byl definován v roce 1959 v práci G.B.Dantziga a J.H.Ramsera [2] (pod názvem *Truck Dispatching Problem*). Praktickou motivací bylo nalezení optimálních tras pro nákladní auta, která rozvážela palivo ze skladiště na velké množství čerpacích stanic. Od té doby byla definice problému několikrát upravena a upřesněna a bylo navrženo mnoho exaktních algoritmů i heuristik pro řešení tohoto problému. Zatím ale nedosáhly dostatečné rychlosti a přesnosti, aby nemělo význam se problémem dále zabývat, a proto je VRP stále důležitým problémem v oblastech dopravy a logistiky. V této kapitole je popsána definice problému, nástin různých variant VRP s podrobnějším popisem VRP s heterogenní flotilou vozidel a nakonec popis konkrétního zaměření této práce.

2.1 Definice problému

VRP je kombinatorický optimalizační problém, který pokládá otázku „Jaká je optimální množina tras, kterou musí flotila vozidel projet, aby obsloužila danou množinu zákazníků?“. Jedná se o zobecnění problému obchodního cestujícího (angl. *Travelling salesman problem* - TSP), což je speciální případ VRP s jedním vozidlem. VRP je stejně jako TSP NP-úplný problém, tedy s rostoucím počtem zákazníků exponenciálně roste počet možných tras, což odpovídá počtu potenciálních řešení. Stejně roste i čas potřebný k nalezení řešení [23].

Vstupem obecného VRP problému je množina míst (požadavků), které je potřeba navštívit a množina (flotila) vozidel, které jsou k dispozici pro rozvoz z jednoho nebo více dep. Počet vozidel může být teoreticky neomezený, ale snadno lze vypozorovat, že nikdy nemůže být potřeba více vozidel, než je míst k navštívení. Cílem je najít množinu tras tak, aby celková cena, závislá na počtu vozidel a ujetých vzdálenostech, byla minimální a zároveň splňovala všechna vstupní omezení daná variantou VRP problému. Jinak řečeno, úkolem je přiřadit jednotlivým vozidlům množinu míst a pořadí, ve kterém je mají navštívit tak, aby cena byla optimální a aby byly splněny zadané podmínky (pokud daná úloha nějaké má). Grafické znázornění principu VRP je na obrázku 2.1.

Klasický VRP problém je definovaný následovně [9]: Nechť $G = (V, A)$ je graf, kde $V = \{1, ..., n\}$ je množina vrcholů, které reprezentují místa s požadavkem přepravy (zákazníkem) s depem umístěným ve vrcholu 1 a A je množina hran. Každé hraně (i, j) $i \neq j$ je přiřazena nezáporná cena c_{ij}, která je nejčastěji interpretována jako vzdálenost mezi vrcholy i, j, ale v některých případech se může jednat například o čas či náklady na trasu. Je-li matice $C = (c_{ij})$ symetrická ($c_{ij} = c_{ji}\ \forall i, j \in V$), pak lze z praktických důvodů množinu A nahradit



Obrázek 2.1: Grafické znázornění problému směrování vozidel. Vlevo je vstup VRP úlohy - požadavky na přepravu s depem v bodě 0 a vpravo optimální řešení problému, kde každá barva značí trasu jednoho vozidla.

množinou neorientovaných hran E. Dále předpokládejme, že se v depu nachází m vozidel, která jsou k dispozici, a platí $m_L \leq m \leq m_U$. Pokud $m_L = m_U$, pak se m nazývá *fixní*, pokud $m_L = 1, m_U = n - 1$, m se nazývá *volné*. Řešení VRP spočívá v nalezení množiny nejlevnějších tras vozidel takových, že:

1. Každé místo v $V \setminus \{1\}$ je navštíveno právě jednou a právě jedním vozidlem.
2. Všechna vozidla začínají a končí svoji trasu v depu.
3. Jsou splněny další vedlejší podmínky.

Vedlejším podmínkám a nejčastějším rozšířením VRP problému se věnuje podkapitola 2.2.

Jednoduchou a hojně využívanou možností, jak zadat úlohu VRP, je seznamem souřadnic míst, které je potřeba navštívit, a depa. Cena přepravy mezi dvěma místy je pak dána Euklidovskou vzdáleností těchto dvou míst. Tuto variantu lze snadno transformovat na graf z výše zmíněné definice, jelikož se principielně jedná o úplný ohodnocený graf s tím, že ceny hran nejsou explicitně vyjádřeny, ale lze je dopočítat. Hlavní výhodou této varianty je snadné automatické generování zadání, ale i převedení požadavku z reálného světa na model. Na druhou stranu vzdálenosti mezi místy v reálu často neodpovídají vzdušným vzdálenostem, které vyjadřuje Euklidovská vzdálenost.

2.2 Rozšíření VRP

Základní VRP může být nedostatečné pro modelování úloh z reálného světa. Počítá totiž s platností určitých předpokladů, jako například neomezenou kapacitu všech vozidel ve flotile, identická vozidla, existenci pouze jednoho depa atd. Proto vznikla rozšíření, která přidávají nebo odstraňují některá pravidla, aby lépe modelovala zadanou úlohu. Problémy reálného světa jsou často kombinací několika rozšíření, případně dalších omezujících podmínek daných konkrétní úlohou.

VRP s omezenou kapacitou vozidel

V rozšíření VRP s omezenou kapacitou vozidel (angl. *Capacitated Vehicle Routing Problem* – CVRP) má každé vozidlo k z flotily o velikosti m konečnou kapacitu C_k, kterou dokáže uvézt. Každý zákazník i má přiřazené nezáporné požadované množství výrobku d_i (nejčastěji se jedná o hmotnost nebo objem). Platí, že součet požadovaného množství všech zákazníků na trase R_k vozidla k nesmí překročit kapacitu vozidla C_k, tedy $\sum_{i \in R_k} d_i \leq C_k$. Rozšíření CVRP se většinou považuje za standard a je zahrnuto v téměř všech úlohách směrování vozidel bez explicitního označení. Pokud není řečeno jinak, tak všechna vozidla mají stejnou kapacitu C a předpokládá se, že je problém řešitelný, tedy je k dispozici dostatek vozidel s dostatečnou kapacitou k obsloužení všech zákazníků [23].

VRP s maximální délkou

Další možnou podmínkou je omezení vzdálenosti, kterou může každé vozidlo urazit (angl. *Distance-constrained Vehicle Routing Problem* – DVRP). Každému vozidlu k je přiřazena nezáporná vzdálenost T_k a součet vzdáleností hran t_{ij} na trase vozidla k nesmí překročit hodnotu T_k. Pokud hodnoty hran reprezentují čas cesty, může být každému zákazníkovi i přiřazen čas potřebný k obsloužení s_i, který označuje dobu, na kterou se vozidlo v daném místě musí zastavit [23].

VRP s časovými okny

U rozšíření VRP s časovými okny (angl. *Vehicle Routing Problem with Time Windows* – VRPTW) má každý zákazník vymezený časový interval $[a_i, b_i]$, který se nazývá časové okno. Obsluha zákazníka musí začít v daném časovém okně. Čas příjezdu vozidla k zákazníkovi je daný součtem cen t_{ij} všech hran, které vozidlo urazilo a součtem časů obsluhy s_i všech dříve obsloužených zákazníků. Vozidlo navíc může v případě brzkého příjezdu u libovolného zákazníka čekat, dokud nenastane příslušné časové okno. Z pravidla je čas výjezdu vozidel nastavený na 0 a cílem je opět minimalizace celkové ceny obsloužení všech zákazníků [22].

VRP s vyzvednutím zboží

V rozšíření VRP s vyzvednutím zboží (angl. *Vehicle Routing Problem with Backhauls* – VRPB) jsou zákazníci rozděleni na dvě části:

1. zákazníci, kteří požadují dodání daného množství zboží z depa
2. zákazníci, u kterých je potřeba vyzvednout dané množství zboží a dovézt ho do depa.

Hlavním omezením u této varianty je, že každé vozidlo musí na své trase obsloužit všechny zákazníky z první části dříve, než obslouží zákazníka z druhé části. Prakticky se jedná o rozvoz zboží z depa a jeho následné doplnění. Platí, že suma množství požadovaného zboží první skupiny zákazníků nesmí překročit kapacitu vozidla, stejně tak součet požadavků druhé skupiny [23].

VRP s vyzvednutím a doručením

V základní verzi VRP s vyzvednutím a doručením (angl. *Vehicle Routing Problem with Pick-Up and Delivery* – VRPPD) je požadavek zákazníka rozdělen na dvojici míst, vyzvednutí zásilky a doručení. Pro každý požadavek platí omezení, že vyzvednutí i doručení musí být

provedeno stejným vozidlem (na stejné trase) a vyzvednutí zásilky musí nastat před jejím doručením. Navíc v žádném okamžiku nesmí aktuální náklad vozidla překročit jeho kapacitu [23].

VRP s heterogenní flotilou vozidel

Rozšíření VRP s heterogenní flotilou vozidel (angl. *Heterogeneous vehicle routing problem* – HVRP) zavádí buď omezenou nebo neomezenou flotilu různých typů vozidel. Vozidla se liší v maximální kapacitě a každý typ vozidla může mít jinou *fixní cenu (angl. fixed cost)* a *variabilní cenu (angl. variable cost)*. Fixní cena k_f vozidla k je konstantní hodnota, která se přičte k celkové ceně, pokud vozidlo k obslouží alespoň jednoho zákazníka. Variabilní cena k_v vozidla k je koeficient, kterým se vynásobí celková vzdálenost, kterou vozidlo k při obsluhování zákazníků urazí. Varianta s neomezeným počtem vozidel se označuje jako *Fleet size and mix vehicle routing problem* (FSMVRP), někdy také *Vehicle fleet mix problem* (VFMP)[16]. Varianta s omezeným počtem vozidel se nazývá *Heterogeneous fixed fleet vehicle routing problem* (HFVRP) [8].

2.3 Zaměření této práce

Tato práce se dále zaměřuje na optimalizaci problému HFVRP, tedy problému VRP s omezenou heterogenní flotilou vozidel. Všechna vozidla začínají a končí svoji trasu v depu. Vstupem úlohy je množina požadavků zákazníků, kde každý požadavek je charakterizován místem, které je potřeba navštívit, a množstvím požadovaného výrobku. Kromě množiny požadavků je vstupem flotila heterogenních vozidel. Každý typ vozidla je charakterizován počtem vozidel, která jsou k dispozici, jejich kapacitou, fixní cenou a variabilní cenou. Cílem práce je vytvořit program, který pomocí evolučních algoritmů nalezne co nejlepší řešení pro zadanou úlohu typu HFVRP. Řešením se myslí nalezení množiny platných tras vozidel takové, že každý požadavek ze vstupu je obsloužen a není porušena žádná ze stanovených podmínek.

Příklad vstupu

Souřadnice depa: X=35.00, Y=35.00
Flotila vozidel v tabulce 2.1
Seznam zákazníků v tabulce 2.2

Počet	Kapacita	Fixní cena	Variabilní cena
1	41.00	49.00	10.00
2	35.00	17.00	7.00
3	55.00	45.00	13.00
4	55.00	20.00	19.00

Tabulka 2.1: Flotila vozidel

Č.zákazníka	X-ová souřadnice	Y-ová souřadnice	Požadované množství
1	41.00	49.00	10.00
2	35.00	17.00	7.00
3	55.00	45.00	13.00
4	55.00	20.00	19.00
5	15.00	30.00	26.00
6	25.00	30.00	3.00
7	20.00	50.00	5.00
8	10.00	43.00	9.00
9	55.00	60.00	16.00
10	30.00	60.00	16.00
⋮	⋮	⋮	⋮

Tabulka 2.2: Tabulka zákazníků

Kapitola 3

Evoluční algoritmy

Evoluční algoritmy vychází z myšlenek Charlese Darwina o přirozeném výběru a přenášení dědičných vlastností z rodičů na potomky. S rozvojem molekulární biologie byly odhalovány další poznatky, jako existence DNA její role v kódování fyzických vlastností do genetického kódu. Těmito myšlenkami se v druhé polovině 20. století inspirovali vědci v oblasti počítačových věd na institucích v různých částech světa a nezávisle na sobě začaly vznikat různé typy evolučních algoritmů. Mezi nejznámější z nich patří evoluční strategie, evoluční programování, genetické algoritmy a genetické programování. Každý typ algoritmů funguje trochu jinak, ale všechny vychází z původní myšlenky evoluce.

Princip optimalizace pomocí evolučních algoritmů je založený na udržování a iterativním aktualizování populace jedinců, kteří reprezentují konkrétní kandidátní řešení daného problému. Inicializace počáteční populace může být náhodná nebo pro ni lze využít vzorek z množiny známých řešení. Aby bylo možné určit, které řešení je z pohledu evolučního algoritmu lepší, musí být definovaná tzv. *fitness* funkce, která každému řešení přiřazuje jeho fitness hodnotu. Celý proces evoluce směřuje k nalezení řešení s nejlepší fitness hodnotou [20]. Fitness funkce může být totožná s *účelovou* funkcí, která vyhodnocuje kvalitu kandidátního řešení ve vztahu ke stanovenému problému.

V každé iteraci (generaci) jsou vybráni jedinci, kteří se stanou rodiči další generace. Jedinci s lepší fitness hodnotou mají větší pravděpodobnost výběru. Poté probíhá reprodukce pomocí variačních (genetických) operátorů (např. křížení nebo mutace), čímž vznikají noví potomci, což odpovídá objevování nových potenciálních řešení. Následně proběhne nahrazování jedinců v populaci podle stanoveného pravidla, které rozhodne o tom, kteří jedinci z množiny rodičů a potomků přežijí do další generace (např. přežije 50 jedinců s nejlepší fitness hodnotou). V budoucích generacích tak špatná řešení postupně vymizí a jsou nahrazena lepšími řešeními. Nové populace vznikají až do doby, kdy je dosaženo podmínky ukončení evolučního procesu. Nejčastěji se jedná o dosažení daného počtu generací nebo nalezení dostatečně vyhovujícího řešení (na základě fitness hodnoty). Základní struktura evolučního algoritmu je popsána v algoritmu 1.

Pro optimalizaci problému směrování vozidel byly z rodiny evolučních algoritmů použity genetické algoritmy [22], evoluční strategie [13] a genetické programování [7]. V dalších podkapitolách jsou tyto přístupy podrobněji rozebrány. Informace, které jsou shrnuty v této kapitole, jsou čerpány převážně z knihy *Natural Computing Algorithms*[1], která se evolučními algoritmy zabývá do hloubky.

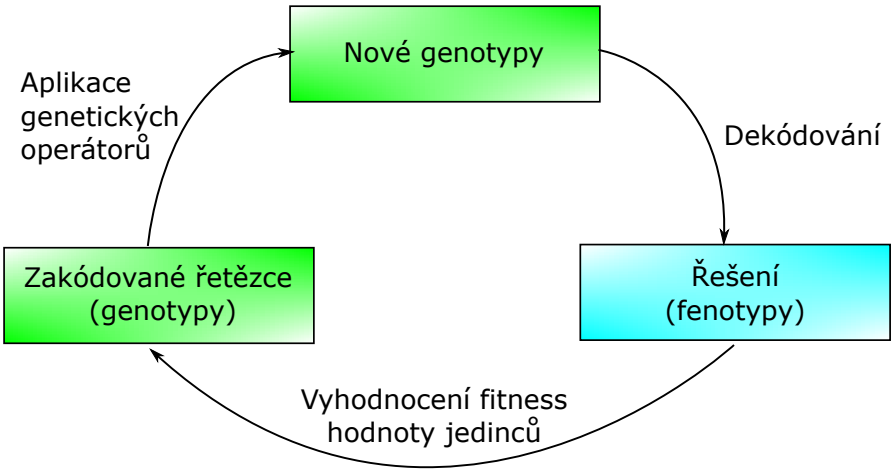
Algoritmus 1: Struktura evolučního algoritmu

Inicializuj populaci kandidátních řešení;
repeat
    Vyber jedince z aktuální populace, kteří se stanou rodiči;
    Z těchto rodičů vygeneruj nové jedince (potomky);
    Nahraď část nebo celou populaci nově vzniklými jedinci;
until *nastane ukončující podmínka*;

3.1 Genetické algoritmy

Vývoj genetických algoritmů (GA) začal už v 60. letech, ale populárními se staly až v 70. letech díky publikaci profesora Hollanda [6], který se zabýval principy adaptivních systémů, což byla jedna ze dvou hlavních oblastí výzkumu GA. Druhou oblastí byla optimalizace, kde GA reprezentuje populační optimalizační algoritmus.

V GA je podobně jako v genetice kladen důraz na rozlišování *genotypu* a *fenotypu*. Genotyp v biologickém pojetí obsahuje genetickou informaci, zatímco fenotyp odpovídá fyzickému projevu této informace. Podobně v GA, genotyp reprezentuje zakódované řešení a fenotyp odpovídá reálnému objektu, který představuje řešení daného problému. Pro získání fenotypu stačí dekódovat genotyp [3]. Genetické operátory jsou aplikovány na genotyp, kdežto vyhodnocení kvality řešení a zjištění fitness hodnoty jedinců je prováděno nad fenotypem. Tento princip je graficky znázorněn na obrázku 3.1.



Obrázek 3.1: Průběh GA z hlediska genotypu a fenotypu.

Pro správné fungování GA je důležité vhodné zakódování jedinců do genotypu. Výběr kódování výrazně ovlivňuje způsob prohledávání stavového prostoru a také funkcionalitu genetických operátorů. Původní GA k reprezentaci řešení využívaly pouze *binární chromozomy*, tedy binární řetězce, ale později se velmi rozšířilo celočíselné kódování. Jednotlivé základní hodnoty v chromozomu se nazývají geny. V ideálním případě je kódování takové, že při velké změně v genotypu dojde k velké změně ve fenotypu (podobně pro malé změny).

Průběh celého GA vychází z principů evolučních algoritmů. Na začátku je inicializována populace většinou desítek až stovek jedinců, kde každý jedinec reprezentuje kandidátní řešení. V každé generaci pak proběhne ohodnocení jedinců pomocí fitness funkce, selekce

rodičů na základě fitness hodnot, aplikace variačních operátorů křížení a mutace pro vznik nových potomků a výběr jedinců, kteří přežijí do další generace. Cyklus se opakuje, dokud není dosažena ukončující podmínka. Genetické operátory mají pravděpodobnostní charakter, celý proces je tudíž stochastický.

3.1.1 Genetické operátory

GA využívají tři genetické operátory: selekci rodičů, rekombinaci (křížení) a mutaci. Hlavním variačním operátorem v GA je křížení. Myšlenkou je, že pokud se povede zkřížit dva jedince tak, že nově vzniklý jedinec získá dobré vlastnosti prvního rodiče a dobré vlastnosti druhého, tak potomek odpovídá lepšímu řešení problému, než jeho rodiče. Protože je počet jedinců v počáteční populaci konečný, nelze některých vlastností dosáhnout pouze pomocí křížení a algoritmus by pravděpodobně skončil v lokálním optimu. Tomu má zabránit mutace, která uměle mění hodnotu náhodně vybraného genu jedince.

Selekce

Způsob výběru rodičů určuje tzv. *selekční tlak* (angl. *selection pressure*), který odpovídá míře preferování jedinců s lepší fitness hodnotou nad ostatními v populaci. Je-li selekční tlak příliš nízký, vlastnosti dobrých rodičů se v populaci budou šířit pomalu, což povede k nízké efektivitě prohledávání možností. Pokud je naopak příliš vysoký, rychle vymizí rozmanitost řešení v populaci a populace se pravděpodobně zasekne v lokálním optimu. Kvalitní strategie výběru vhodně balancují výběr jedinců s vysokou fitness hodnotou a zachování rozmanitosti.

Pro výběr rodičů existují dva základní přístupy, a to selekce úměrná k fitness (angl. *fitness proportionate selection*) a selekce podle pořadí (angl. *ordinal selection*). V případě selekce úměrné k fitness je každému jedinci přiřazena pravděpodobnost na základě jeho absolutní fitness hodnoty. U selekce podle pořadí záleží na jeho pořadí při seřazení všech jedinců podle jejich fitness hodnot. Výběr jedinců je pak proveden pomocí vhodné techniky, mezi nejčastěji používané patří vážená ruleta a turnaj. U **vážené rulety** je každému jedinci přiřazena pravděpodobnost úměrná jeho fitness funkci a s takovou pravděpodobností je vybrán jako rodič. Tento proces se opakuje, dokud není vybrán dostatek rodičů. Při **turnaji** se náhodně vybere daný počet jedinců, kteří se utkají v turnaji a jedinec s nejlepší fitness hodnotou je vybrán jako rodič [5]. Stejně jako u rulety se proces opakuje, dokud není vybrán dostatek rodičů. Techniky selekce rodičů bývají často doplněny o tzv. **elitismus**, při kterém určitý počet nejlepších jedinců přežije do další generace úplně beze změn, což zvyšuje selekční tlak [20].

Křížení

Křížení je jedním ze základních způsobů generování nových řešení. Konkrétní implementace je úzce spjata s reprezentací chromozomů. U binárních a celočíselných chromozomů se nejčastěji jedná o vzájemnou výměnu části chromozomů mezi dvěma rodiči. Pravděpodobnost, že křížení nastane, je dána parametrem $P_{cross} \in \langle 0; 1 \rangle$. Pokud křížení neproběhne, jsou potomci stejní jako rodiče. Mezi základní typy křížení patří:

- **Jednobodové křížení:** Náhodně se vybere místo křížení z $\{1, 2, ..., n-1\}$ pro chromozom délky n a rodiče si část před místem křížení vymění. Tím vzniknou dva noví potomci, jak je ukázáno na obrázku 3.2a.

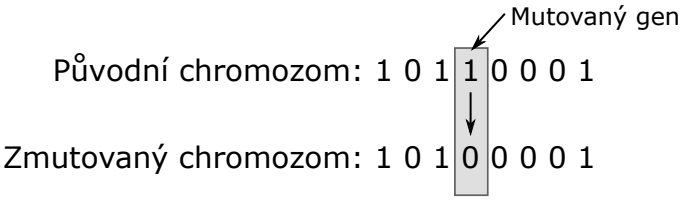
- **Dvoubodové křížení:** Náhodně se vyberou dvě místa křížení z $\{1, 2, ..., n-1\}$ pro chromozom délky n a rodiče si vymění část před prvním místem a část za druhým místem křížení (obrázek 3.2b).
- **Uniformní křížení:** Pro chromozom o délce n se náhodně vygeneruje bitová maska délky n, která určuje, ze kterého rodiče se přenese gen na potomka. Druhý potomek vznikne z nevyužitých hodnot (obrázek 3.2c), nebo opakováním procesu. Pravděpodobnost výběru z prvního rodiče bývá obvykle nastavena na 0.5.

Obrázek 3.2: Ukázka typů křížení: (a) jednobodové, (b) dvoubodové, (c) uniformní.

Pro jiné reprezentace chromozomů můžou být výše zmíněné typy křížení nepoužitelné. V tom případě je potřeba definovat nové operátory křížení, které pro danou úlohu dávají smysl. Například pro chromozomy reálných čísel se může jednat o průměr hodnot jednotlivých genů.

Mutace

Při mutaci dochází s určitou pravděpodobností k umělé změně hodnoty jednoho nebo více genů jedince. Mutace v GA zajišťuje, že proces prohledávání teoreticky nikdy neskončí, protože může v každé iteraci potenciálně odhalit novou užitečnou vlastnost. Pravděpodobnost mutace je potřeba zvolit dostatečně velkou, aby populace neuvázla v lokálním optimu (souvisí i se selekčním tlakem). Pokud ale bude pravděpodobnost příliš velká, budou operátory selekce a křížení potlačeny a z GA se efektivně stane náhodné prohledávání. Nejjednodušší mutací pro bitové chromozomy je změna hodnoty bitu z '0' na '1', nebo z '1' na '0'. U celočíselných chromozomů s omezeným rozsahem se může jednat o změnu hodnoty na jinou náhodnou hodnotu z daného rozsahu [1]. Ukázka bitové mutace je na obrázku 3.3.



Obrázek 3.3: Jednoduchá mutace obrácením bitu.

3.2 Evoluční strategie

Evoluční strategie (ES) je rodina algoritmů, která vznikla v 60. letech v Německu, když je Rechenberg a Schwefel využili pro optimalizaci aerodynamiky mechanických objektů [18].

Stejně jako všechny ostatní evoluční algoritmy vychází z původních myšlenek přirozeného výběru. Hlavním využitím ES bylo řešení optimalizačních problémů s reálnými hodnotami, ale dosáhly úspěchů i při použití pro řešení kombinatorických optimalizací a dalších typů problémů. Hlavními dvěma charakteristikami ES jsou použití reprezentace s reálnými čísly (ale existují i binární a celočíselné ES) a to, že primárně spoléhají na operátory selekce a mutace. Křížení se tedy v ES buď nevyužívá nebo se jedná pouze o doplňkový operátor, jehož cílem je zvýšit schopnosti explorace algoritmu.

Strategie reprodukce a přežití

Existuje několik variant ES, které se od sebe liší počtem rodičů μ a počtem potomků λ. Počáteční výzkum se soustředil na ES s jedním rodičem a jedním potomkem, která se označuje $(1+1)$. V tomto případě se celá populace skládá z jednoho jedince - rodiče, který generuje jednoho potomka. Rodič a potomek spolu následně soupeří a lepší z nich přežije do další generace. Problém tohoto přístupu je pomalý vývoj směrem k oblastem stavového prostoru s lepšími řešeními, navíc úplně nevyužívá potenciál populačního přístupu.

Pokročilejšími verzemi ES jsou strategie s více rodiči a více potomky. Takové strategie lze obecně rozdělit na dvě kategorie: $(\mu+\lambda)$ a (μ, λ). V obou případech je velikost udržované populace μ, nebo-li je v každé generaci vybráno μ jedinců, kteří přežijí do další generace a stanou se rodiči. Rodiče pomocí mutací vygenerují λ potomků ($\lambda > \mu$) a nastává selekce, která určí, kdo z rodičů a potomků přežije do další generace. Čím větší je λ, tím větší je selekční tlak ES [1].

V případě $(\mu+\lambda)$ je do výběru pro přežití do další generace zahrnuta celá populace rodičů a potomků, přičemž uspěje μ nejlepších. U varianty (μ, λ) jsou do výběru zahrnuti pouze potomci, takže každý jedinec přežije právě jednu generaci. $(\mu+\lambda)$ odpovídá elitismu, protože dobré řešení v původní populaci může být nahrazeno pouze lepším potomkem, zatímco (μ, λ) nezaručuje přežití dobrého rodiče do další generace. Strategie (μ, λ) je využívána častěji, protože se populace snadno odpoutá od lokálního optima. Průběh $(\mu + \lambda)$-ES je popsán v algoritmu 2.

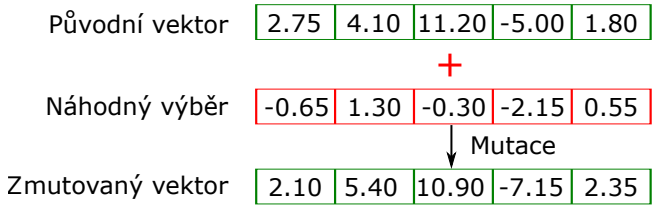
Algoritmus 2: Průběh evoluční strategie $(\mu + \lambda)$

```
Náhodně inicializuj populaci vektorů {x^1, ..., x^n};
Ohodnoť každého jedince v populaci;
repeat
    repeat
        Náhodně vyber rodiče z populace vektorů;
        Vytvoř potomka aplikováním mutačních operátorů na rodiče;
    until je vygenerováno λ potomků;
    Seřaď všechny (μ + λ) rodiče a děti od nejlepšího po nejhorší;
    Vyber z nich nejlepších μ jedinců, kteří přežijí do další generace;
until nastane ukončující podmínka;
```

Mutace v ES

Každý jedinec v ES je typicky reprezentovaný vektorem reálných čísel $x = (x_1, ..., x_n)$ a každý takový vektor má přiřazenou množinu jednoho nebo více parametrů strategie.

Tyto parametry řídí proces mutace a mohou se za běhu adaptovat. V nejjednodušším případě je uživatelem zvolen jeden parametr σ. Mutace pak probíhá tak, že pro každý prvek vektoru $x(t)$ je vybráno náhodné číslo z Gaussova rozdělení $N(0, \sigma)$, a je přičteno k hodnotě v původním vektoru. Princip mutace je znázorněn na obrázku 3.4. Vznik potomka v ES tak lze popsat rovnicí $x(t+1) = x(t) + N(0, \sigma)$. Parametr σ se označuje jako velikost mutačního kroku a ovlivňuje míru odchýlení potomka od rodiče.



Obrázek 3.4: Ukázka mutace v ES.

Výhodou tohoto mechanismu je snadná implementace, ale má dvě výrazné nevýhody:

- Nebere v úvahu různé škálování jednotlivých dimenzí, tedy velikost hodnot, ve kterých se jednotlivé prvky vektoru pohybují.
- Velikost mutačního kroku nijak nesouvisí s tím, v jaké fázi se nachází proces prohledávání.

První problém souvisí s tím, že každý prvek vektoru může mít odlišný rozsah hodnot. Druhý s fixní hodnotou parametru σ. Tyto problémy se řeší zavedením samoadaptivního parametru pro každý prvek vektoru. Každý jedinec v populaci je pak reprezentován vektorem $v = ((x_1, ..., x_n), \sigma_1, ..., \sigma_n)$, kde σ_i je parametr strategie prvku x_i. Míru odchýlení potomka od rodiče pak lze nastavit pro jednotlivé prvky, čímž je vyřešen první problém.

Pokud jsou tyto parametry navíc samoadaptivní, je vyřešen i druhý problém. Samoadaptivní parametry v průběhu prohledávání mění svoji hodnotu. Jedním z možných způsobů, jak vypočítat hodnoty parametrů a aplikovat mutaci na vektor je ukázán v následujících rovnicích:

$$\sigma_i(t+1) = \sigma_i(t) \cdot e^{\tau' r' + \tau r} \tag{3.1}$$

$$x_i(t+1) = x_i(t) + \sigma_i(t+1) r_i, \quad i = 1, ..., n \tag{3.2}$$

kde $\sigma_i(t)$ je parametr strategie prvku x_i v generaci t, τ a τ' jsou parametry učení a r, r' a $r_1, ..., r_n$ jsou nezávisle náhodně vybrané proměnné ze standardního normálního rozdělení pravděpodobnosti (se středem v 0 a standardní odchylkou 1). τ a τ' ovlivňují rychlost samoadaptace a typicky jsou nastaveny na hodnoty $\tau' = \frac{1}{\sqrt{n}}$ a $\tau = \frac{1}{\sqrt{2\sqrt{n}}}$. Takto funguje tzv. mutace s velikostí kroku n (angl. *n-Step-Size Mutation*). Existují i další komplexnější způsoby adaptace parametrů [19].

3.3 Genetické programování

Původní motivací vývoje genetického programování (GP) bylo automatické vytváření počítačových programů na základě stanovených požadavků, s využitím evolučního procesu. GA byly pro tento úkol příliš omezující, protože typicky pracují s reprezentací řešení v podobě řetězců fixní délky, ale programy mají přirozeně hierarchickou strukturu s proměnnou

délkou [24]. GP podobně jako jiné evoluční algoritmy operuje nad populací potenciálních řešení, kterou iterativně vylepšuje pomocí selekce, křížení, mutace a nahrazování. Průběh GP je velice podobný průběhu GA, se dvěma důležitými rozdíly:

1. Liší se v podobě reprezentace řešení, která se využívá při evolučním procesu.
2. Evoluční proces v GP musí být otevřenější, protože velikost struktury řešení není předem známá.

GP nerozlišuje genotyp a fenotyp, evoluční proces se provádí přímo na řešeních (fenotypech), která obvykle mají strukturu stromu. Proto je potřeba před začátkem evolučního procesu definovat množiny základních stavebních bloků - *množinu funkcí* a *množinu terminálů*. Množina terminálů obsahuje položky s aritou 0, tedy takové, které nemají žádný vstup. Může se jednat například o konstanty, vstupy do programu či proměnné definované v programu. Terminály ve stromové struktuře odpovídají listům stromu. Množina funkcí obsahuje položky s aritou vyšší než 0, tedy vyžadují daný počet vstupů. Příkladem může být aritmetická funkce '+', která má aritu 2. Navíc délka řešení není předem známá, proto musí být počet prvků a propojení předmětem evoluce.

Pro ohodnocování jedinců se nejčastěji používá trénovací množina. Výstupy získané kandidátním programem se porovnají s požadovanými hodnotami a na základě velikosti odchýlení od těchto hodnot je pak vypočítána fitness hodnota. Mezi nejčastěji používané způsoby výpočtu fitness patří hrubá fitness a standardizovaná fitness. **Hrubá fitness** je udávaná v hodnotách, které jsou přirozené pro doménu řešeného problému. **Standardizovaná fitness** převádí hrubou fitness tak, že menší fitness znamená lepší řešení (0 je nejlepší). Cílem je pak fitness hodnotu minimalizovat. Výsledné řešení je po konci evoluce ověřeno na testovací množině [24].

Křížení

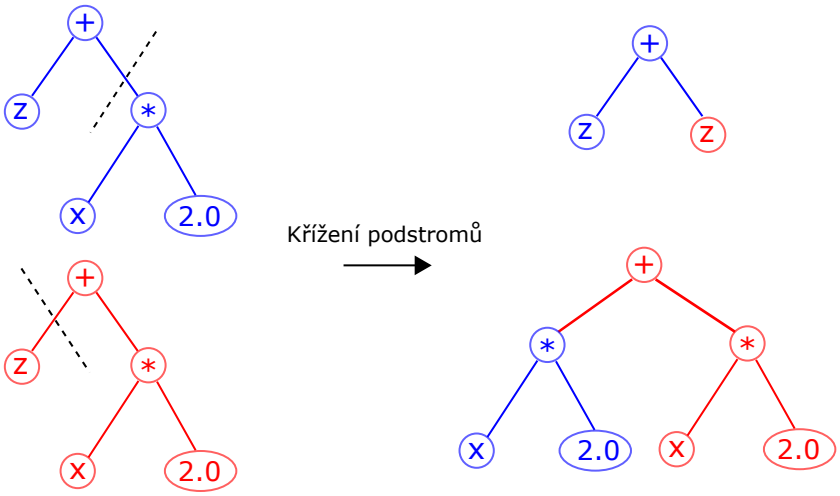
Klasickým operátorem křížení v GP je výměna podstromů. Vyberou se dva jedinci z populace (rodiče), a v každém z nich se náhodně vybere bod křížení. Podstromy vycházející z těchto bodů si rodiče vymění a vzniknou tak dva noví jedinci [1]. Jedním z hlavních rozdílů oproti křížení v GA je, že dva stejní rodiče můžou vygenerovat potomky, kteří se liší od obou rodičů, jak je ukázáno na obrázku 3.5.

Mutace

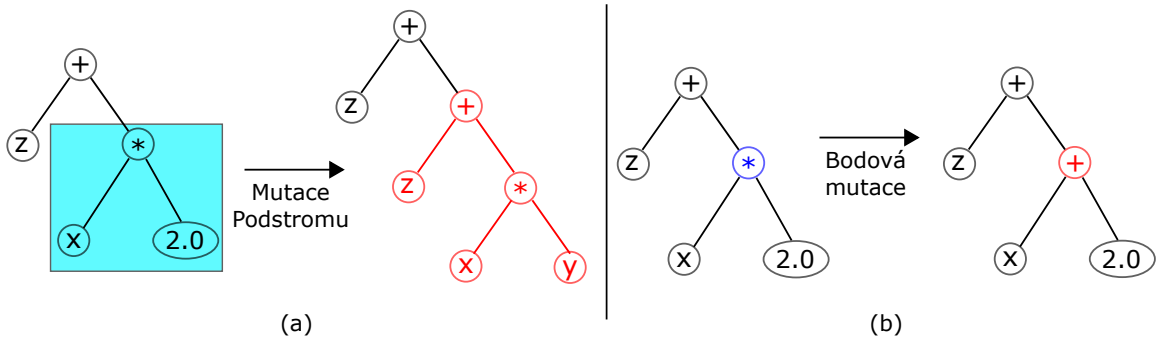
Existují dva základní typy mutace v GP. První z nich je mutace podstromu, kdy se náhodně vybere jeden neterminální uzel a je spolu s jeho podstromem nahrazen novým, náhodně vygenerovaným podstromem (obrázek 3.6a). Druhým typem je bodová mutace, kdy je náhodně vybrána jedna funkce a je nahrazena jinou náhodnou funkcí se stejnou aritou (obrázek 3.6b). Podobně lze mutovat i terminály.

Inicializace populace

Pro inicializaci počáteční populace existují 3 základní metody: metoda **full**, metoda **grow** a metoda **ramped half-and-half**. Každá z nich vyžaduje, aby byla předem specifikována maximální hloubka stromu *maxdepth*. Při použití metody *full* jsou stromy vytvářeny tak, že jsou vybírány pouze funkce do té doby, než je na všech větvích dosaženo hloubky $maxdepth - 1$. Poté jsou vybírány pouze terminály až do dokončení stromu. Takto vzniklé



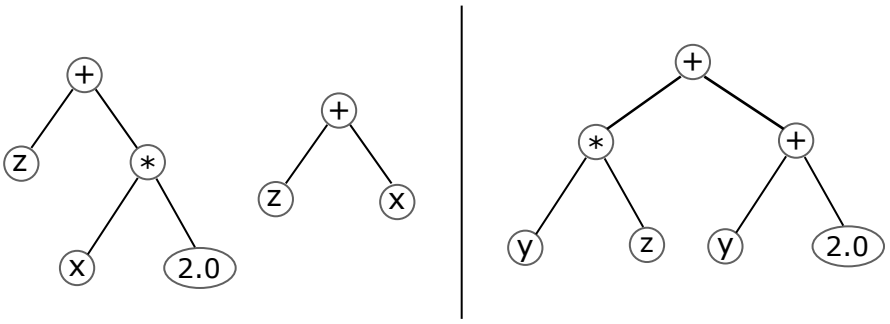
Obrázek 3.5: Křížení dvou identických stromů. Potomci jsou odlišní od rodičů.



Obrázek 3.6: Ukázka mutace podstromu (vlevo) a bodové mutace (vpravo).

stromy jsou vždy úplné. U metody *grow* se vybírá náhodně z terminálů i funkcí, dokud větev nedosáhne hloubky $maxdepth - 1$. Pak se vybírá pouze z terminálů, aby nebyla narušena maximální hloubka stromu [24]. Ukázka jedinců vzniklých oběma metodami je na obrázku 3.7.

Populace inicializované pouze jednou z výše popsaných metod postrádají rozmanitost. Proto se začala využívat metoda *ramped half-and-half*, která funguje následovně. Nechť je definovaná maximální hloubka stromu d. Pak je $\frac{1}{d}$ část populace generována s maximální hloubkou 1, $\frac{1}{d}$ s maximální hloubkou 2 a tak dále až do maximální hloubky d. Na každé úrovni je půlka jedinců generována metodou *grow* a druhá půlka metodou *full*. Tím je zajištěna populace různě velkých, úplných i neúplných, rozmanitých stromů [24].



Obrázek 3.7: Vlevo je ukázka jedinců vzniklých metodou *grow*, vpravo metodou *full* pro $maxdepth = 3$.

Kapitola 4

Evoluční řešení problému VRP

Využití evolučních algoritmů pro řešení problému VRP umožňuje efektivní prohledávání obsáhlého prostoru potenciálních řešení. Nejjednodušším způsobem, jak reprezentovat řešení, je pomocí jednoho celočíselného chromozomu, kde každé číslo představuje zákazníka a umístění v chromozomu určuje pořadí, ve kterém mají být zákazníci obslouženi. Jednotlivé trasy jsou odděleny například identifikátorem depa (nulou). Fitness funkce obvykle odpovídá účelové funkci, tedy celkové vzdálenosti, kterou vozidla urazila. Jednoduchost kódování a ohodnocení jedince vedla v posledních třiceti letech ke vzniku mnoha EA pro řešení VRP a jeho variant. V podkapitole 4.1 je shrnutí metod, které řeší problém s heterogenní flotilou vozidel. V další podkapitole 4.2 je podrobněji popsán jeden z moderních způsobů evolučního řešení VRP problému a v podkapitole 4.3 je popsáno konkrétní řešení problému HFVRP.

4.1 Existující řešení HVRP

Oproti jiným variantám VRP problému bylo pro VRP s heterogenními vozidly vytvořeno relativně malé množství populačních heuristik [8]. Důvodem můžou být problémy, které toto rozšíření přináší, a se kterými je potřeba se při návrhu řešení vypořádat. Prvním problémem je výrazné zvětšení prohledávaného prostoru s každým typem vozidla. Druhým je omezený počet jednotlivých typů vozidel, což sice zmenšuje prohledávaný prostor možných řešení, ale snižuje efektivitu genetických operátorů, které by při použití prohledávaly prostor nevalidních řešení. Nevalidní řešení v populaci lze sice potlačovat pomocí penalizace fitness funkce [11], ale nejedná se pak o efektivní prohledávání stavového prostoru.

Většina existujících řešení funguje tak, že nejprve seřadí zákazníky do vhodného pořadí a následně pomocí heuristiky přiřadí vozidla k úsekům seřazených zákazníků. Pořadí obsloužení zákazníků v trase je dáno pořadím v úseku přiřazeného vozidla [11].

První úspěchy zaznamenal algoritmus, který funguje na výše zmíněném principu [14]. Algoritmus počítá s neomezenou flotilou vozidel a pouze fixní cenou. Chromozom je seznam k genů, kde k je počet zákazníků. Hodnota každého genu je kladné celé číslo, které reprezentuje zákazníka. Využívá jeden operátor křížení (*ERX crossover*), který vytváří nové chromozomy změnou pořadí zákazníků v chromozomu. Tyto změny jsou založené na sousednosti zákazníků v trasách rodičů. Trasy jsou vytvořeny na základě pořadí zákazníků v chromozomu pomocí techniky pro vytváření „okvětních lístků“ a typ vozidla pro vytvořenou trasu je vybrán tak, aby byla kapacita dostatečná a cena nejmenší.

Dalším algoritmem, který také počítá s neomezenou flotilou heterogenních vozidel, je evoluční algoritmus, který využívá dvě techniky lokálního prohledávání: algoritmus GENIUS a λ-výměny (λ-interchange) [10]. Algoritmus udržuje populaci 500 jedinců. Na každého jedince je po inicializaci aplikován algoritmus GENIUS, který optimalizuje trasy jednotlivých vozidel. Rodiče jsou vybíráni turnajem dvou jedinců a je na ně aplikována varianta ERX křížení. Potomek je následně optimalizován pomocí λ-výměn.

Nejnovějším známým evolučním přístupem k řešení problému HVRP s omezenou či neomezenou flotilou vozidel je rozdělení sekvence zákazníků do tras pomocí procedury *split* s následnou optimalizací pomocí lokálního prohledávání [15]. Původní varianta metody split pracuje s neomezenou flotilou, ale později byla rozšířena pro problém s omezeným počtem vozidel [16]. Tyto techniky jsou podrobněji popsány v podkapitole 4.3. Novější metody buď upravují techniku lokálního prohledávání [11] nebo experimentují s jinými variantami procedury split [4].

4.2 Hybridní GA s lokálním prohledáváním pomocí gravitační emulace

Hybridní algoritmus založený na lokálním prohledáváním pomocí gravitační emulace (angl. *Gravitational Emulation Local Search* - GELS) a genetickém algoritmu, byl publikován v roce 2019 a je určený pro VRP s omezením kapacity [17]. Hlavní myšlenkou je využití GA primárně pro explorační fázi a využití GELS pro lepší exploitaci. Kombinace těchto algoritmů by měla vygenerovat škálu řešení z celého stavového prostoru, která jsou lokálně optimalizována prostřednictvím GELS.

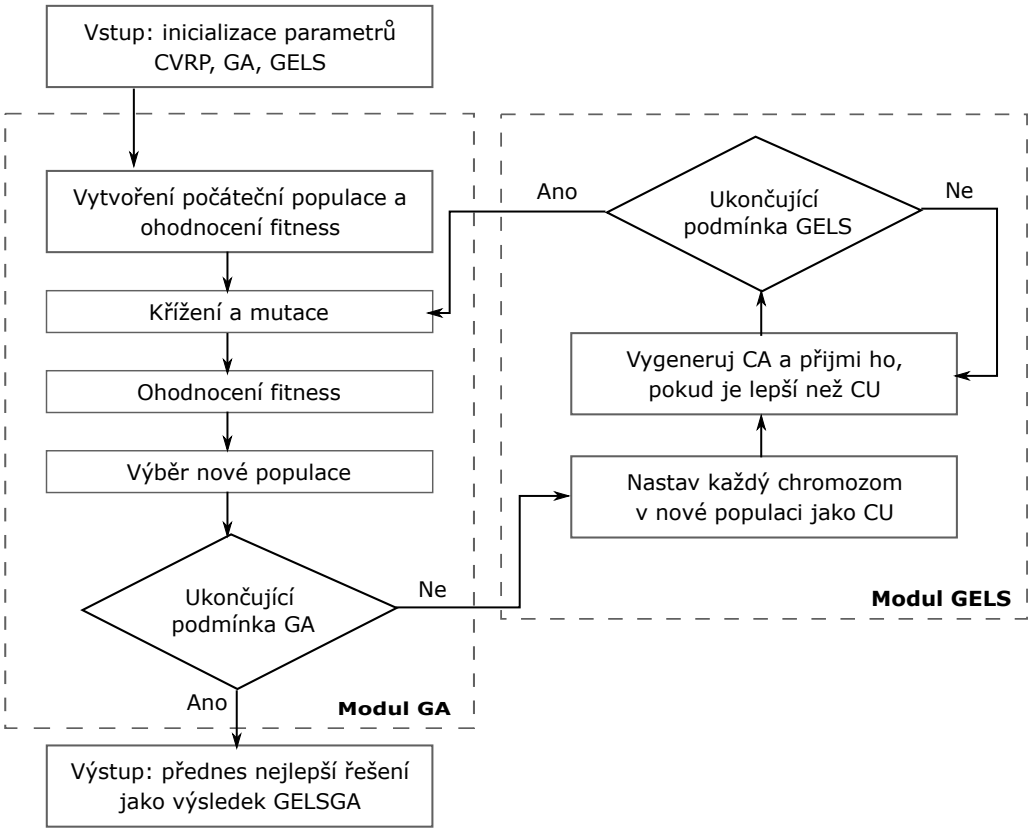
Na obrázku 4.1 je znázorněn průběh algoritmu GELSGA (GELS + GA). Řešení problému začíná GA, který v každé generaci provede mutaci, křížení a selekci pro další generaci. V tuto chvíli začíná GELS. Chromozomy vstupují do GELS po jednom jako CU a je na nich provedena lokální optimalizace. Optimalizované chromozomy se vrací do GA a celý proces se opakuje. Pseudo-kód algoritmu GELSGA je popsán v algoritmu 3, kde CU značí aktuální (*current*) řešení a CA kandidátní (*candidate*) řešení.

Genetický algoritmus

Řešení jsou reprezentována celočíselným chromozomem, kde geny odpovídají zákazníkům a jednotlivé trasy jsou odděleny nulou. Fitness funkce se spočítá jako celková vzdálenost všech tras, s penalizací za porušení podmínky kapacity. Algoritmus využívá dva operátory křížení a dva operátory mutace.

Při použití prvního operátoru křížení zdědí každý potomek pořadí zákazníků z jednoho rodiče a počet zákazníků každého vozidla z druhého rodiče. Druhý operátor funguje stejně jako jednobodové křížení, tedy potomek zdědí část chromozomu jednoho rodiče a část z druhého. Takto vzniklé řešení je potřeba upravit, protože se může vyskytovat více či méně než $k - 1$ nul (k je počet vozidel) a někteří zákazníci budou v chromozomu dvakrát. První problém se vyřeší přidáním či mazáním potřebného počtu nul. Druhý problém se vyřeší nahrazením opakovaného čísla odstraněným číslem.

První mutační operátor vymění dva zákazníky v chromozomu. Výměna může proběhnout v rámci jedné trasy, nebo mezi dvěma trasami. Druhý typ mutace mění pořadí pouze v rámci jedné trasy. Jeden nenulový gen je náhodně zvolen jako pivot a všichni zákazníci příslušného vozidla zmutují na jiného zákazníka tohoto vozidla.



Obrázek 4.1: Diagram průběhu algoritmu GELSGA.[17]

GELS

Algoritmus GELS je implementován třemi maticemi pro *rádius (radius)*, *rychlost (velocity)* a *hmotnost (mass)*. Matice pro rádius je konstantní a v problému CVRP je totožná s maticí vzdáleností zákazníků $distance_{(n \times n)}$. Matice rychlostí a hmotností se v průběhu algoritmu mění. Matice rychlostí označuje rychlost každého zákazníka k jinému zákazníkovi a všechny hodnoty v matici jsou na začátku nastaveny na 100. Položky v matici hmotností jsou vypočítány z matic vzdáleností a rychlostí:

$$mass(i,j) = \frac{distance(i,j)}{velocity(i,j)} \cdot 60 \tag{4.1}$$

Výpočet CA z CU probíhá iterativně, přičemž v každém cyklu je vytvořeno jedno CA. Pokud je lepší než CU, tak je akceptováno jako CU pro další iteraci. Po akceptování CA je upravena matice rychlostí rovnicemi 4.2 a 4.3

$$F = G \cdot \frac{fitness(CU) - fitness(CA)}{distance(i,j)^2} \tag{4.2}$$

$$velocity_{m+1}(i,j) = velocity_m(i,j) + F, \tag{4.3}$$

kde F je množství síly a G je gravitační konstanta nastavená na 6.672. Po aktualizování matice rychlostí je aktualizována i matice hmotností $mass(i,j)$ rovnicí 4.1. Pro aplikování gravitační síly na body i a j chromozomu CU, seřadí algoritmus všechny nenulové geny po j podle hmotnosti vzestupně, čímž vznikne CA. Proběhne aktualizace matic a po n

Algoritmus 3: Pseudo-kód algoritmu GELSGA [17]

```
Inicializuj parametry GA a GELS;
Vygeneruj prvotní populaci a ohodnoť jedince;
repeat
    for každý chromozom S_i v populaci do
        křížení(S_i);
        mutace(S_i);
    end
    Vyber novou populaci;
    for každý chromozom S_i v nové populaci do
        CU = S_i ;
        repeat
            Vypočítej CA z CU;
            if fitness(CA) < fitness(CU) then
                CU = CA;
                aktualizuj(matice rychlostí);
                aktualizuj(matice hmotností);
            end
        until nastane ukončující podmínka GELS;
    end
until nastane ukončující podmínka GA;
return nejlepší chromozom;
```

(počet zákazníků) iteracích algoritmus GELS skončí a vrátí nejlepší nalezené řešení. Na algoritmu 4 je ukázán pseudo-kód generování CA a aktualizací matic.

Algoritmus 4: Pseudo-kód fungování GELS na chromozomu Ch [17]

```
CU = Ch;
for každý nenulový gen j v CU do
    CA = sort(Ch(j...(n + k − 1))) na základě hmotností mass(j, :);
    if fitness(CU) < fitness(CA) then
        CU = CA;
        i = první nenulový gen před j;
        F = G · (fitness(CU) − fitness(CA)) / distance(i,j)^2;
        velocity_{m+1}(i, j) = velocity_m(i, j) + F;
        mass(i, j) = distance(i,j) / velocity(i,j) · 60;
    end
end
return CU;
```

Shrnutí experimentálních výsledků

Algoritmus byl testován na několika různých testovacích sadách (benchmarks) a je srovnáván s ostatními metodami, které využívají evoluční algoritmy, a to z pohledu kvality nalezených řešení, ale také potřebného výpočetního času [17]. Ve valné většině případů nalezl GELSGA nejlepší řešení ze všech srovnávaných algoritmů. Je to však na úkor výpo-

četního času, kterého potřebuje více než některé srovnatelné algoritmy, obzvlášť u problémů s menším počtem zákazníků. U problémů většího rozsahu je GELSGA schopen nalézt řešení lepší než dosud nejlepší známá řešení, což znamená, že je schopen nalézt opravdu kvalitní řešení ve srovnání s ostatními algoritmy.

4.3 Hybridní evoluční algoritmus pro HVFRP

Hybridní algoritmus kombinuje genetický algoritmus, který získává nová řešení pomocí operátorů mutace a křížení a lokálním prohledáváním, které vylepšuje získaná řešení [16]. Chromozom je permutací všech zákazníků bez rozdělení do jednotlivých tras. Na chromozom lze pohlížet jako na jednu velkou trasu pro vozidlo s nekonečnou kapacitou. Tato trasa je pomocí procedury *split* optimálně rozdělena do tras, které splňují zadané podmínky a dodržují pořadí zákazníků v chromozomu. Takto získané řešení může být následně optimalizováno pomocí lokálního prohledávání.

Generování nových jedinců probíhá pomocí operátoru křížení *order crossover*(OX) dvou jedinců [15], každý z nich vybraný metodou turnaje dvou náhodně vybraných jedinců. Nově vzniklý jedinec nahradí náhodného jedince v horší polovině populace.

Procedura split

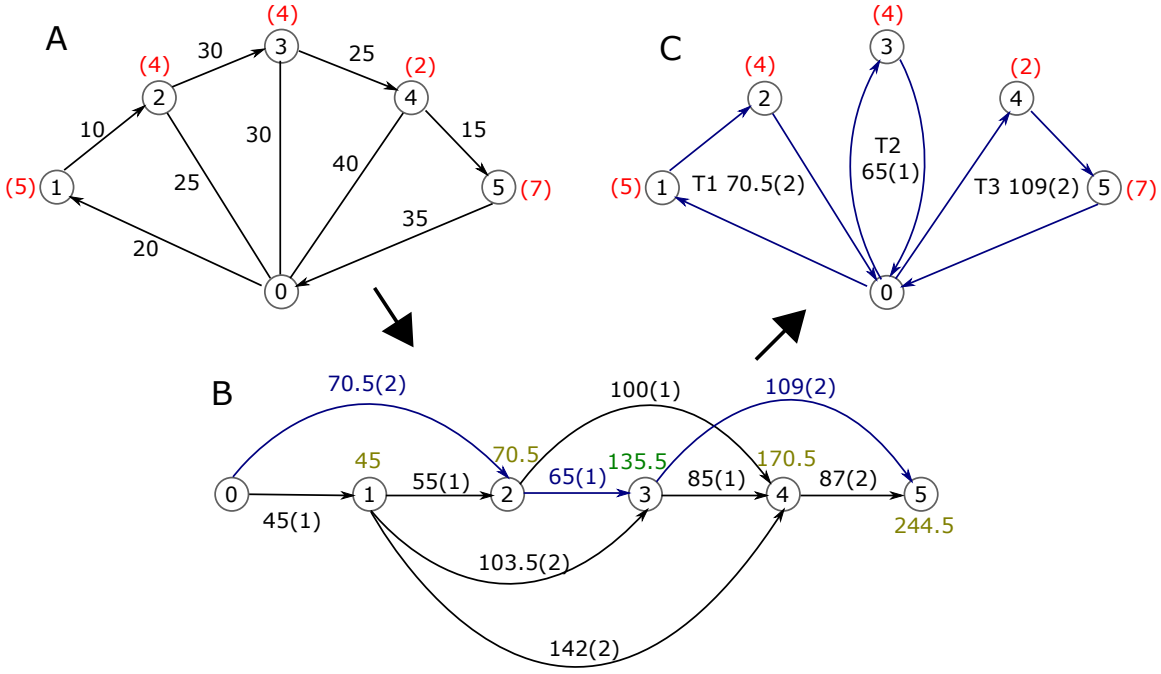
V této části je nejprve popsána základní procedura split, která je určená pro neomezenou flotilu vozidel, a poté úpravy potřebné pro funkčnost při omezené flotile vozidel. Každý chromozom C je permutací zákazníků 1 až n. Split nejdříve zkonstruuje pomocný acyklický ohodnocený graf $H = (N, A)$ s $n+1$ vrcholy, kde n je počet zákazníků. Vrcholy N obsahují jeden pomocný vrchol 0 a vrcholy 1 až n ve stejném pořadí jako v chromozomu, které reprezentují zákazníky. A obsahuje hranu $(i-1, j)$ právě tehdy, když posloupnost zákazníků $(C_i, C_{i+1}, ..., C_j)$ může být obsloužena v rámci jedné trasy. To znamená, že součet požadavků zákazníků v této posloupnosti je menší než kapacita alespoň jednoho typu vozidla. Celá trasa je pak dána posloupností $(0, C_i, C_{i+1}, ..., C_j, 0)$, kde 0 reprezentuje depo. K této hraně je pak přiřazeno vozidlo typu k takové, že pro cenu této trasy Z_{ij} platí

$$Z_{ij} = \min(Z_{ijk} = f_k + v_k L_{ij} : 1 \leq k \leq t, W_{ij} \leq Q_k), \tag{4.4}$$

kde Z_{ijk} je cena trasy při použití vozidla k, f_k a v_k jsou fixní a variabilní cena vozidla k, L_{ij} je celková vzdálenost trasy, t je počet typů vozidel, W_{ij} je součet požadavků zákazníků v trase a Q_k je kapacita vozidla k. Tato hodnota je přiřazena hraně $(i-1, j)$ v grafu H. Optimální rozdělení tras je pak dáno hranami, které tvoří nejkratší cestu v grafu z bodu 0 do bodu C_n. Nalézt nejkratší trasu lze např. Bellman-Fordovým algoritmem v polynomiálním čase. Příklad je ukázán na obrázku 4.2.

Výše popsaná procedura split funguje pro problém s neomezenou flotilou. Pokud je ale počet vozidel každého typu omezený, může procedura nalézat rozdělení, která porušují maximální počet vozidel některého typu. Aby optimální rozdělení do tras fungovalo i pro omezenou flotilu, je potřeba provést několik úprav.

Při konstrukci hran pomocného grafu není dopředu známo, jestli bude typ vozidla s nejnižší cenou k dispozici. Proto je potřeba pro každou trasu $(C_i, C_{i+1}, ..., C_j)$ vložit hranu $(i-1, j)$ pro každý typ vozidla, který má dostatečnou kapacitu pro obsloužení všech zákazníků.



Obrázek 4.2: Ukázka fungování procedury split s neomezenou flotilou vozidel dvou typů $k1$ a $k2$, s parametry $Q_{k1} = 6$, $Q_{k2} = 10$, $f_{k1} = 5$, $f_{k2} = 10$, $v_{k1} = 1$, $v_{k2} = 1.1$ na chromozomu [1,2,3,4,5]. V části A je zobrazena jedna velká trasa s depem v bodu 0, se vzdálenostmi mezi sousedními zákazníky a jejich vzdálenostmi od depa. U každého zákazníka je uvedena velikost požadavku (červené číslo v závorce). Tato trasa je transformována na acyklický graf (část B). Hodnoty u hran značí cenu příslušné hrany a číslo v závorce typ vozidla, zelené hodnoty u vrcholů cenu nejkratší trasy z vrcholu 0 k příslušnému vrcholu. Nejkratší cesta z 0 do 5 je vyznačena modře. V části C je získané optimální přiřazení vozidel.

Nalezení optimálního rozdělení tras odpovídá problému nalezení nejkratší cesty v grafu s omezenými zdroji, což je obecně NP-těžký problém, ale v praxi ho lze řešit poměrně rychle metodami dynamického programování [16].

Nechť t je počet typů vozidel a vektor $x = (x_1, x_2, ..., x_t)$ je vektor použitých vozidel, pro který platí $0 \leq x_k \leq a_k$, kde a_k je počet vozidel typu k, který je k dispozici. Pro počet ψ různých vektorů x platí

$$\psi = \prod_{k=1}^{t} (a_k + 1) \tag{4.5}$$

K výpočtu optimálního přiřazení zákazníků do tras se použije matice značek Y o velikosti $(n+1) \times \psi$. Každá značka Y_{jh} pro $j = 0...n$, $h = 0...(\psi - 1)$ obsahuje tyto informace:

- vektor $Y_{jh}x$ použitých vozidel
- nejkratší cestu $Y_{jh}P$ v pomocném grafu H z vrcholu 0 do vrcholu j při použití vozidel daných vektorem x
- cenu $Y_{jh}Z$ příslušné nejkratší cesty z 0 do j při použití vozidel x

Na začátku jsou všechny vektory x inicializovány na 0, ceny v prvním řádku matice na 0 a ceny v ostatních řádcích na nekonečno. Algoritmus pro každý vrchol i, každou hranu

(i, j) v H, každou Y_{ip} použitou v řádku i a každý možný typ vozidla k počítá hodnoty dalších značek v matici Y.

Nová značka Y_{jh} se vypočítá ze značky Y_{ip} tak, že se k hodnotě x_k přičte 1, k ceně Z se přičte cena hrany (i, j) při obsloužení vozidlem k a hrana (i, j) se přidá do nejkratší cesty P. Následně se vypočítá sloupec matice nově vzniklé značky, a pokud je cena této značky menší než aktuální cena v matici, značka v matici je nahrazena touto značkou. Sloupec h matice vektoru x lze vypočítat pomocí algoritmu 5. Proces výpočtu značek v matici je popsán v algoritmu 6.

Algoritmus 5: Výpočet sloupce matice z vektoru x

```
h = 0;
for k = 1..t do
    h = h × (a_k + 1) + x_k;
end
```

Algoritmus 6: Výpočet matice značek v proceduře split pro HFVRP

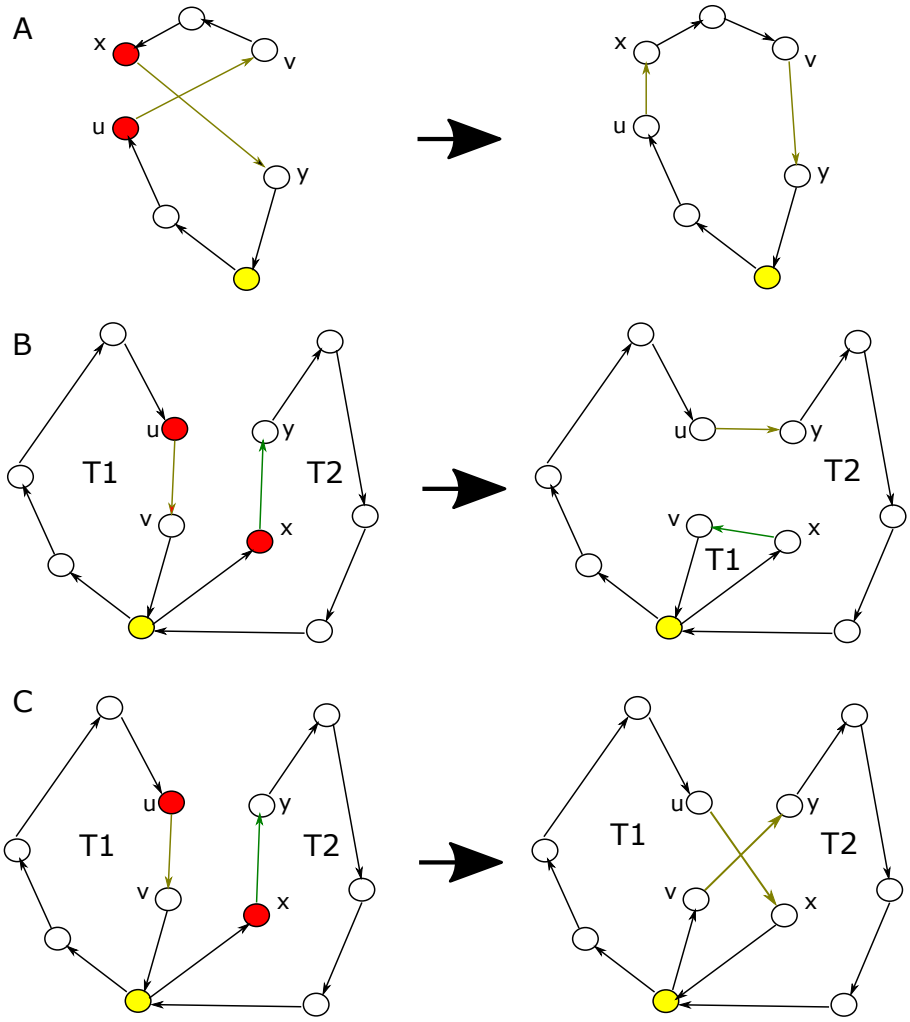
```
Inicializuj matici Y;
for i = 0..n do
    for každou hranu (i, j) v H do
        for p = 0..(ψ − 1), kde Y_ip Z < ∞ do
            for k = 1..t, kde (Y_ip x_k < a_k) a (W_ij ≤ Q_k) do
                U = Y_ip;
                U.x_k = U.x_k + 1;
                U.Z = U.Z + f_k + v_k L_ij;
                U.P = U.P ∪ {(i, j)};
                Vypočítej sloupec h pro vektor U.x;
                if U.Z < Y_jh.Z then
                    Y_jh = U;
                end
            end
        end
    end
end
```

Nejkratší cesta je po skončení výpočtu dána značkou v posledním řádku s nejnižší cenou. Pokud jsou všechny hodnoty cen v posledním řádku nekonečno, nejkratší cesta v grafu neexistuje, a tedy neexistuje pro tento chromozom žádné validní přiřazení vozidel.

Lokální prohledávání

Cílem metod lokálního prohledávání je nalézt co nejlepší řešení (lokální optimum) v blízkém okolí vstupního řešení. Toho se docílí tak, že algoritmus systematicky provádí operace, které nepatrně mění řešení. Pokud se řešení po provedení této změny zlepší, algoritmus začne znovu prohledávat okolí nově získaného řešení. Pokud se nezlepší, pokračuje další operací v předem daném pořadí. Pokud žádná operace nad řešením nepřinese zlepšení, lokální prohledávání skončí.

V rámci problému VRP se může jednat např. o přesun jednoho nebo dvou po sobě jdoucích zákazníků, výměnu dvou zákazníků či dvojic po sobě jdoucích zákazníků nebo tzv. 2-OPT přesun hran v rámci jedné či dvou tras, viz obrázek 4.3. Tyto operace jsou postupně prováděny pro každou možnou kombinaci zákazníků. Evoluční algoritmy pro řešení HFVRP problému využívají lokální prohledávání jako mutační operátor. To znamená, že je aplikováno na potomka s pravděpodobností p_{mut} (např. $p_{mut} = 0.1$) [11].



Obrázek 4.3: Ukázka přesunu 2-OPT na jedné trase (A) a na dvou trasách (B a C). Vybraní zákazníci jsou označení červeně, přesunované hrany zeleně, depo žlutě. U variant A a C je obrácena orientace některých částí tras.

Kapitola 5

Návrh řešení

Z přístupů popsaných v předchozí kapitole lze vyvodit, že nejúspěšnější evoluční algoritmy pro řešení různých variant problému VRP využívají kombinaci evoluce a techniky lokálního prohledávání. Tento princip zůstává zachován i v mém řešení. Narozdíl od algoritmu popsaném v podkapitole 4.3 jsem se rozhodl, že nebudu používat proceduru split, ale navrhnu několik „menších“ mutačních operátorů. Jedním z těchto mutačních operátorů bude i technika lokálního prohledávání.

V této kapitole jsou popsány všechny části řešení, které jsem navrhnul a implementoval. V podkapitole 5.1 je popsáno zvolené kódování jedinců a výpočet kvality řešení a v podkapitole 5.2 metoda inicializace populace. V dalších podkapitolách 5.3, 5.4 a 5.5 jsou charakterizovány navržené mutační operátory. V podkapitole 5.6 je návrh čtyř evolučních algoritmů, které využívají operátory z předchozích podkapitol. Na závěr jsou v podkapitole 5.7 popsány implementační detaily.

5.1 Reprezentace a kvalita jedinců

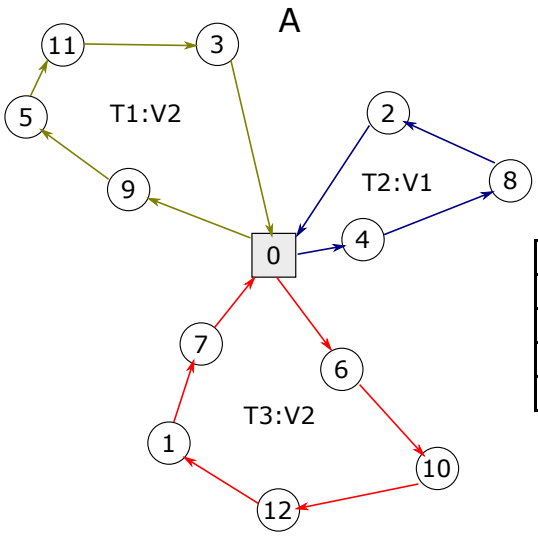
Řešením problému HFVRP je seznam tras pro jednotlivá vozidla. Každá trasa se skládá ze seznamu zákazníků, kteří jsou obslouženi v pořadí daném pořadím v seznamu, a typu vozidla, které tuto trasu obsluhuje. Jednotliví zákazníci v trase jsou reprezentováni kladnými celými čísly od 1 do n, kde n je počet zákazníků. Na začátku a konci každé trasy musí být hodnota reprezentující depo (0), jinak je trasa v kontextu HFVRP neplatná. Typy vozidel jsou zakódovány jako celá čísla v hodnotách od 0 do t, kde t je počet typů vozidel. Ukázka kódování řešení je na obrázku 5.1.

Kvalita každého jedince je určená jeho hodnotou fitness, která je dána fitness funkcí. Jako fitness funkci jsem se rozhodl použít přímo účelovou funkci, která je dána součtem cen tras všech použitých vozidel. Cena trasy je závislá na vzdálenosti, kterou je potřeba urazit. Je-li problém zadaný seznamem souřadnic, odpovídá vzdálenost d_{ij} mezi dvěma místy i a j euklidovské vzdálenosti:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}. \tag{5.1}$$

Pokud je problém zadaný pomocí úplného grafu, je vzdálenost rovna hodnotě v matici vzdáleností M:

$$d_{ij} = M[i][j]. \tag{5.2}$$



B

	Typ vozidla	Místa na trase						
Trasa 1:	2	0	9	5	11	3	0	
Trasa 2:	1	0	4	8	2	0		
Trasa 3:	2	0	6	10	12	1	7	0
Trasa 4:	0	0	0					

Obrázek 5.1: Ukázka reprezentace řešení v chromozomu. Nalevo (A) je konkrétní řešení problému HFVRP, se třemi trasami T1, T2, T3. Řešení využívá vozidla typu 1 a 2. Vpravo (B) je zobrazeno zakódování do chromozomu v podobě tabulky. Trasy 1, 2 a 3 odpovídají trasám T1, T2, T3. K dispozici je navíc jedno vozidlo typu 0, které je nevyužité. Některé mutační operátory mohou přesunovat zákazníky do prázdných tras, proto musí být i prázdná trasa reprezentována v chromozomu. Cena takových tras je nulová.

Cena c_r jedné trasy $r = \langle u_1, u_2, ..., u_L \rangle$, kde $u_1, u_2, ..., u_L$ reprezentuje posloupnost míst v pořadí, v jakém jsou navštíveny a obslouženy vozidlem typu k, se pak spočítá

$$c_r = f_k + v_k \sum_{i=1}^{L-1} d_{u_i u_{i+1}}, \tag{5.3}$$

kde f_k a v_k jsou fixní a variabilní cena vozidla k. Fitness hodnota F je pak rovna součtu cen C všech r tras:

$$F = C = \sum_{i=1}^{r} c_i \tag{5.4}$$

5.2 Inicializace populace

Aby byl algoritmus efektivní, je vhodné, aby prohledával pouze prostor validních řešení. Proto jsou mutační operátory v dalších podkapitolách navrženy tak, aby validitu neporušovaly. Pak stačí, aby počáteční populace sestávala pouze z validních, ale ideálně stále náhodných řešení.

Při inicializaci populace je každý jedinec vytvořen následujícím způsobem. Nejdříve se vygeneruje náhodná permutace P všech n zákazníků a prázdná trasa pro každé vozidlo, které je k dispozici. Poté se postupně pro každého zákazníka provedou tyto kroky:

1. Vytvoř seznam tras, kam lze zákazníka vložit bez porušení podmínky kapacity.
2. Pokud je seznam prázdný, přidej zákazníka do seznamu neobsloužených a skonči.
3. Jinak náhodně vyber jednu trasu ze seznamu kandidátů.

4. Vlož zákazníka na náhodné místo ve vybrané trase.

Pokud není seznam neobsloužených zákazníků prázdný, provede se ještě algoritmus 7, který by měl pomocí vhodného vyměňování zákazníků dosáhnout toho, že žádný zákazník nezůstane neobsloužený. Ve výjimečných případech může dojít k tomu, že i po této proceduře zůstane některý ze zákazníků neumístěný. Pak je tento jedinec zahozen a inicializace jedince proběhne od začátku.

Algoritmus 7: Umístění neobsloužených zákazníků

```
Vstup: seznam neobsloužených zákazníků unserved, seznam tras R;
Výstup: seznam neobsloužitelných zákazníků unserveable;
while unserved je neprázdný do
    C ← zákazník na začátku seznamu;
    if Lze C vložit do některé z tras v R bez porušení kapacity then
        Vlož C do náhodně vybrané vhodné trasy r;
        Odstraň C z unserved;
    end
    else
        Vytvoř seznam zákazníků s menším požadavkem než C;
        Vyber vhodné zákazníky, se kterými lze C vyměnit bez porušení validity;
        if Vhodný zákazník neexistuje then
            Vlož C do unserveable;
            Odstraň C z unserved;
        end
        else
            Vyber náhodného vhodného zákaníka U;
            Vlož C do trasy místo U;
            Vlož U do unserved místo C;
        end
    end
end
```

5.3 Náhodné mutační operátory

Všechny mutační operátory navržené v rámci této práce (kromě lokálního prohledávání) jsou pravděpodobnostního charakteru. V této části jsou popsány varianty operátorů, které provádí všechny náhodné výběry z diskrétního rovnoměrného (uniformního) rozdělení pravděpodobnosti, tedy každý kandidát má stejnou pravděpodobnost výběru.

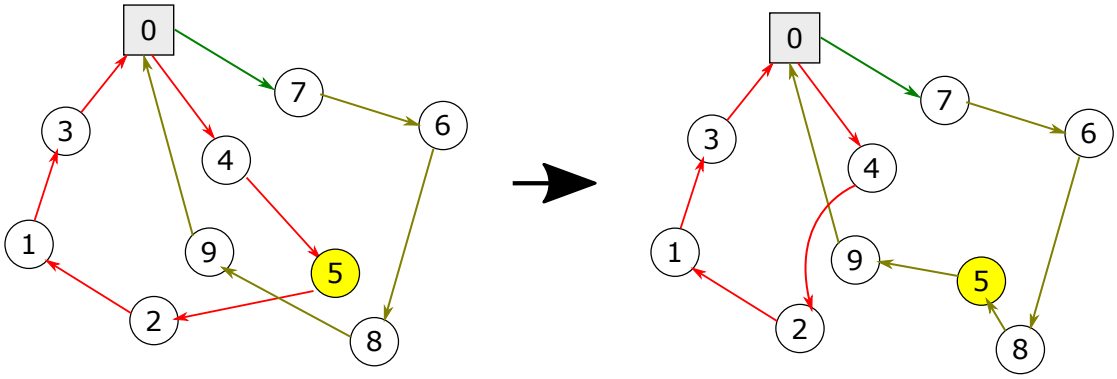
Náhodná výměna dvou zákazníků v rámci jedné trasy

Tento operátor `random_swap` nejprve náhodně vybere trasu s alespoň třemi zákazníky. Následně v této trase vybere dva zákazníky a prohodí jejich pořadí v této trase. Tato operace nemůže porušit žádné omezující podmínky (pokud již nejsou porušeny), protože množství požadavků všech tras zůstane konstatní.

Náhodný přesun zákazníka do jiné trasy

Mutační operátor `random_transfer` náhodně vybere jednoho ze zákazníků k přesunu. Protože při přesunu mezi dvěma trasami může dojít k porušení maximální kapacity, je nejprve vytvořen seznam kandidátních tras, které mají dostatek volné kapacity. Pokud žádná taková trasa neexistuje, operátor nic nezmění (potomek zůstane stejný jako rodič). Pokud není seznam kandidátních tras prázdný, náhodně se vybere jedna z tras v seznamu a vybraný zákazník je přesunut z původní trasy na takovou pozici ve zvolené trase, která nejméně zvýší cenu této trasy (nejlevnější pozice). Ukázka jednoho přesunu je na obrázku 5.2.

Tento operátor umožňuje přesun zákazníka do prázdné trasy a tudíž zvýšení celkového počtu tras a použitých vozidel. V kontextu HFVRP to znamená, že evoluční algoritmus může nalézat řešení, která využívají různé kombinace počtů a typů vozidel, což se od algoritmu očekává.

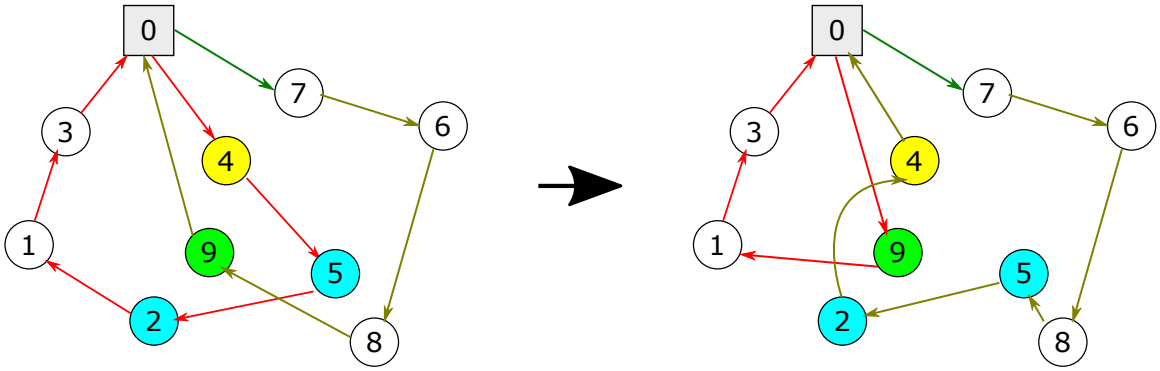


Obrázek 5.2: Ukázka mutačního operátoru přesunu (`random_transfer` nebo `guided_transfer`). Zákazník vybraný k přesunu je označený žlutě. Zelená trasa má dostatečnou volnou kapacitu pro obsloužení zákazníka.

Náhodná výměna více zákazníků mezi dvěma trasami

Operátor mutace `random_switch` provádí výměnu zákazníků mezi dvěma trasami. Aby zůstala zachována validita řešení, je potřeba, aby vyměnění zákazníci měli dostatečně podobný součet množství požadavků. Obě trasy po výměně musí splňovat omezení kapacity příslušného typu vozidla. Zákazníci mají ale velice rozdílná množství požadavků, a tak by pro některé instance problému HFVRP byla výměna zákazníků 1:1 buď nemožná, nebo cyklická (vyměňovali by se stále stejní zákazníci). Proto je umožněna výměna 1 až `MAX_SWITCH` po sobě jdoucích zákazníků jedné trasy za 1 až `MAX_SWITCH` po sobě jdoucích zákazníků druhé trasy. Parametr `MAX_SWITCH` jsem empiricky nastavil na hodnotu 4.

Nejprve je náhodně vybrán jeden zákazník C_i. Následně se určí maximální počet zákazníků p, které lze z této trasy vyměnit. Ten odpovídá hodnotě min(1 + počet zákazníků v trase po C_i, `MAX_SWITCH`). Poté se náhodně vybere počet zákazníků k v rozmezí 1 až p, který z této trasy bude vyměněn. Bude se jednat o část trasy $\langle C_i, C_{i+1}, ..., C_{p-1} \rangle$. Pro tento úsek se vytvoří seznam kandidátních jednotlivců a v trasách po sobě jdoucích dvojic, trojic a čtveřic zákazníků, za které lze vybranou část trasy vyměnit. Jedna z možností je náhodně vybrána a vyměňovaní zákazníci jsou odstraněni z jejich tras. Následně jsou postupně vkládáni na nejlevnější pozice v trase. Ukázka jedné výměny je na obrázku 5.3



Obrázek 5.3: Ukázka mutačního operátoru výměny (`random_switch` nebo `guided_switch`). Vybraný zákazník je označen žlutě. Náhodně jsou k výměně vybráni jeho dva následovníci. Seznam kandidátů pro validní výměnu obsahuje zeleného zákazníka. Trojice zákazníků 4,5,2 je vyměněna za zákazníka 9. Zákazníci jsou umístěni na nejlevnější pozice.

Náhodné promíchání několika tras

Operátor `randomize_routes` vybere několik tras z řešení, jejichž zákazníci budou promícháni. Pravděpodobnost, že bude trasa vybrána, je dána parametrem `P_SELECT`, který jsem empiricky nastavil na hodnotu 0.35. Všichni zákazníci jsou z těchto tras odstraněni a umístěni do seznamu neobsloužených zákazníků. Poté jsou náhodně rozdělováni mezi zvolené trasy, podobně jako při inicializaci, s tím rozdílem, že jsou umisťováni na nejlevnější pozice. Neobsloužení zákazníci jsou umístěni do tras pomocí algoritmu 7.

Cílem tohoto operátoru je zabránit uváznutí v lokálním optimu. Toho dosahuje vytvářením řešení, která jsou v prohledávaném prostoru potenciálně velmi vzdálená od rodičovského řešení.

5.4 Řízené mutační operátory

V této podkapitole jsou popsány mutační operátory, které využívají výběrů z nerovnoměrného diskrétního rozložení pravděpodobnosti. Operátory provádí výběry ze seznamu kandidátů na základě vybraných informací o těchto kandidátech. V případě operátorů v této části se bude jednat primárně o vzdálenosti vybíraných zákazníků od těžiště některé z tras, případně vzdálenost mezi těžišti dvou tras.

Pokud jsou zákazníci zadáni souřadnicemi, lze souřadnice těžiště trasy vypočítat jako aritmetický průměr souřadnic všech míst na této trase (včetně depa):

$$c_x = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad c_y = \frac{1}{n}\sum_{i=1}^{n} y_i, \tag{5.5}$$

kde (c_x, c_y) jsou souřadnice těžiště, n je počet míst v trase a (x_i, y_i) jsou souřadnice místa i. Vzdálenost zákazníka od těžiště i vzdálenost mezi dvěma těžišti odpovídá euklidovské vzdálenosti mezi těmito body.

V případě, že nejsou zadané souřadnice zákazníků, ale pouze matice vzdáleností M mezi nimi, nelze souřadnice těžiště explicitně vypočítat. Vzdálenost D_{ur} zákazníka u od těžiště trasy r lze pro účely řízených mutačních operátorů aproximovat jako průměr vzdáleností

zákazníka u od všech míst trasy r:

$$D_{ur} = \frac{1}{n} \sum_{i=1}^{n} M[u][r(i)], \tag{5.6}$$

kde $r(i)$ je i-té místo na trase r. Vzdálenost $D_{r_1 r_2}$ mezi těžišti dvou tras r_1, r_2 lze aproximovat jako aritmetický průměr vzdáleností všech míst z trasy r_1 od těžiště r_2, což odpovídá průměrné vzdálenosti každého místa na trase r_1 od každého místa na trase r_2:

$$D_{r_1 r_2} = \frac{1}{n \cdot m} \sum_{i=1}^{n} \sum_{j=1}^{m} M[r_1(i)][r_2(j)], \tag{5.7}$$

kde n a m jsou počty míst v trasách r_1, r_2.

Řízený přesun zákazníka v rámci jedné trasy

Prvním operátorem, který využívá vzdálenost zákazníka od těžiště, je `reinsert`. Ten nejprve vybere trasu alespoň se třemi zákazníky. Poté náhodně vybere jednoho ze zákazníků, přičemž pravděpodobnost výběru zákazníka je přímo úměrná jeho vzdálenosti od těžiště trasy. Vybraný zákazník je z trasy odstraněn a vložen na nejlevnější pozici v této trase.

Řízený přesun zákazníka do jiné trasy

Operátor `guided_transfer` náhodně vybere trasu na základě poměru její ceny k počtu zákazníků. Trasa s nejvyšší hodnotou poměru má největší šanci na výběr. Každému zákazníkovi v této trase je přiřazena váha podle jeho vzdálenosti od těžiště a výběrem z diskrétní distribuce pravděpodobnosti se vybere zákazník k přesunu. Podobně jako u operátoru `random_transfer` se vytvoří seznam tras, kam lze zákazníka přesunout se zachováním validity řešení. Každé trase se přiřadí váha nepřímo úměrná vzdálenosti jejího těžiště od zákazníka ($\frac{1}{D_{ur}}$). Proběhne výběr trasy, do které je zákazník vložen na nejlevnější pozici. Ukázka jednoho přesunu je na obrázku 5.2

Řízená výměna více zákazníků mezi dvěma trasami

Mutační operátor `guided_switch` náhodně vybere trasu s pravděpodobnostmi danými počtem zákazníků v trase. Ve vybrané trase se vybere zákazník stejně jako u `guided_transfer`, tedy nejvzdálenější zákazník od těžiště má nejvyšší pravděpodobnost. Podobně jako u operátoru `random_switch` se vybere počet zákazníků k přesunu a sestrojí seznam kandidátů. Každému kandidátovi je přiřazena váha w_k, která se odvíjí od vzdálenosti vybraného zákazníka u k těžišti kandidáta c_k a vzdálenosti kandidáta k od těžiště vybraného zákazníka c_u:

$$w_k = \frac{1}{D_{uc_k} + D_{kc_u}}. \tag{5.8}$$

V případě, že kandidát není jediný zákazník ale posloupnost více zákazníků, se bere vzdálenost prvního zákazníka v této části trasy. Vybraní zákazníci se odstraní z jejich tras a jsou postupně vkládáni do druhé trasy na nejlevnější pozice. Ukázka jedné výměny je na obrázku 5.3.

Výměna dvou vozidel

Operátor `vehicle_swap` provádí výměnu typů vozidel dvou tras. Pravděpodobnost výběru jednotlivých tras je dána váhou w_r, která se získá jako:

$$w_r = 2 - \frac{M_r}{C_r}, \tag{5.9}$$

kde M_r je součet požadavků všech zákazníků trasy r a C_r je kapacita vozidla, které obsluhuje trasu r.

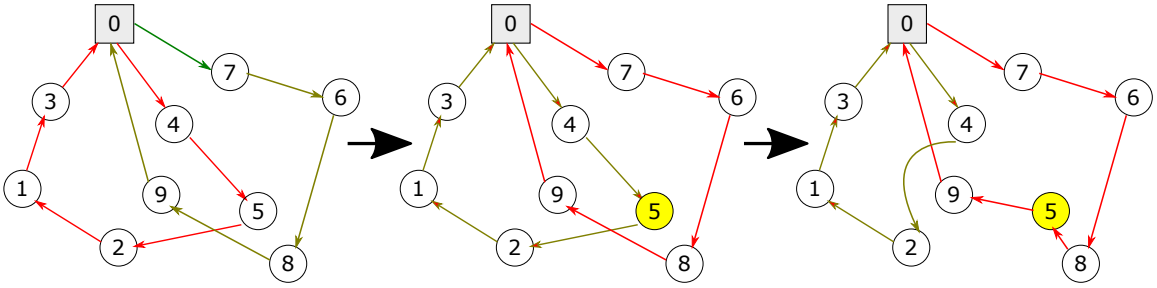
Seznam kandidátních tras pro výměnu sestává z tras, které obsluhuje typ vozidla podobný typu vozidla vybrané trasy. Nechť jsou typy vozidel $0, 1, ..., t$ seřazeny vzestupně podle kapacity a vozidlo, které obsluhuje vybranou trasu r je typu a. Pak jsou do seznamu kandidátních tras vloženy všechny trasy, které obsluhuje vozidlo typu $a + 1$ nebo $a - 1$. Každé kandidátní trase k je přiřazena váha w_k:

$$w_k = \frac{1}{D_{rk}}, \tag{5.10}$$

tedy převrácená hodnota vzdálenosti těžiště kandidáta od těžiště vybrané trasy. Vybere se jedna z kandidátních tras a vymění svůj typ vozidla s první vybranou trasou. Protože může být porušena podmínka maximální kapacity jedné z tras, začne oprava tras. Ta spočívá v náhodném přesunování zákazníků z trasy s porušenou podmínkou kapacity do druhé trasy, kde by měla být volná kapacita. Pokud existuje možnost přesunu jednoho zákazníka, která validitu obou tras zajistí, provede se tento přesun a oprava se ukončí. Jinak jsou zákazníci pro přesun vybíráni na základě poměru vzdáleností k těžištím tras:

$$p_i = \frac{D_{ir_1}}{D_{ir_2}}, \tag{5.11}$$

kde p_i je váha zákazníka i a D_{ir_1} a jsou vzdálenosti i od těžiště tras r_1 a r_2. Pokud po provedení deseti přesunů stále nejsou obě trasy validní, tak mutační operátor žádnou změnu neuloží a potomek bude stejný jako rodič. Ukázka jedné výměny vozidel je na obrázku 5.4.



Obrázek 5.4: Ukázka mutačního operátoru výměny (`vehicle_swap`. Nejprve se vymění typy vozidel dvou tras, vybraných popsaným způsobem. Zelené vozidlo nemá po výměně dostatečnou kapacitu pro obsloužení všech zákazníků, takže začne oprava trasy. Algoritmus opravy zjistí, že stačí vyměnit jediného zákazníka, a to zákazníka č. 5. Ten je vložen do červené trasy a oprava je ukončena.

5.5 Lokální prohledávání

Posledním mutačním operátorem, který budou navržené algoritmy využívat, je jedna z variant lokálního prohledávání (`local_search`). Technika lokálního prohledávání se snaží pomocí drobných systematických úprav co nejvíce vylepšit vstupní řešení. Pseudokód varianty lokálního prohledávání, kterou jsem zvolil a implementoval podle článku pro problém FSM-VRP [12], je popsána v algoritmu 8.

Algoritmus 8: Algoritmus lokálního prohledávání. Po každém kroku ve For cyklu proběhne ohodnocení nového řešení. Pokud je nové řešení lepší, For cyklus začne znovu nad nově získaným řešením. Pokud je horší, cyklus pokračuje dalším krokem s původním řešením.

```
Vstup: řešení Sol problému HFVRP;
while Proběhlo zlepšení v řešení do
    for Každou dvojici zákazníků u ≠ x do
        if Některý z dalších kroků vylepší řešení then
            Sol ← nové řešení;
            Ukonči For cyklus;
        end
        Zkus přesunout u za x;
        Zkus vyměnit u a x;
        if Po u následuje další zákazník v then
            Zkus přesunout u, v za x;
            Zkus Vyměnit u, v s x;
            if Po x následuje další zákazník y then
                Zkus vyměnit u, v a x, y;
                Zkus provést 2-OPT přesun hran u, v a x, y (obrázek 4.3);
            end
        end
    end
end
```

5.6 Návrh evolučních algoritmů

Pro účely srovnání navržených mutačních operátorů jsem navrhnul čtyři evoluční algoritmy pro řešení problému HFVRP. Jedná se o tři genetické algoritmy a jednu evoluční strategii. Varianty genetického algoritmu mají shodné nastavení evolučních parametrů, liší se pouze v použitých mutačních operátorech. Hodnoty společných evolučních parametrů jsou uvedeny v tabulce 5.1. Konfigurace těchto parametrů byla získána laděním evolučních algoritmů.

Genetický algoritmus s použitím náhodných operátorů

Genetický algoritmus GA-R využívá pouze náhodné operátory, konkrétně `random_swap`, `random_transfer`, `random_switch` a `randomize_routes`. Na začátku proběhne inicializace populace o velikosti POP_SIZE tak, jak je popsaná v sekci 5.2. Následně probíhá GENERATIONS iterací. V každé iteraci vznikne nová populace o velikosti POP_SIZE, která se

Parametr	hodnota pro GA	hodnota pro ES
GENERATIONS	100 000	100 000
POP_SIZE	30	-
P_MUT	50	50
TOURNAMENT	2	-
ELITISM	1	1
μ	-	5
λ	-	30
ES_PLUS	-	FALSE

Tabulka 5.1: Nastavení evolučních parametrů

skládá z ELITISM nejlepších jedinců z předchozí populace a POP_SIZE - ELITISM nových jedinců. Nový jedinec vznikne tak, že se náhodně vybere rodič z aktuální populace metodou turnaje s TOURNAMENT účastníky, a s pravděpodobností P_MUT (v procentech) je na něj aplikován některý z mutačních operátorů. Po provedení GENERATIONS iterací je řešením nejlepší nalezený jedinec v průběhu evoluce. Pokud je ELITISM nenulový, tak se jedná o nejlepšího jedince poslední populace.

Hodnoty pravděpodobností jednotlivých mutačních operátorů jsou k vidění v tabulce 5.2. Tyto hodnoty byly získány na základě experimentů s několika variantami nastavení.

Parametr	pravděpodobnost
P_RANDOM_SWAP	15
P_RANDOM_SWITCH	35
P_RANDOM_TRANSFER	35
P_RANDOMIZE_ROUTES	15

Tabulka 5.2: Nastavení pravděpodobností mutačních operátorů pro GA-R v procentech

Genetický algoritmus s použitím řízených operátorů

Genetický algoritmus GA-G funguje podobně jako GA-R, ale místo náhodných mutačních operátorů používá řízené operátory `reinsert`, `guided_transfer`, `guided_switch` a `vehicle_swap`. Z náhodných operátorů jsou ponechány operátory `randomize_routes` a `random_swap` pro zvýšení explorace. Hodnoty pravděpodobností operátorů jsou k vidění v tabulce 5.3. Průběh genetického algoritmu (inicializace populace, výběr jedinců atd.) je stejný jako u GA-R.

Parametr	pravděpodobnost
P_RANDOM_SWAP	10
P_REINSERT	15
P_GUIDED_SWITCH	30
P_GUIDED_TRANSFER	30
P_VEHICLE_SWAP	5
P_RANDOMIZE_ROUTES	10

Tabulka 5.3: Nastavení pravděpodobností mutačních operátorů pro GA-G v procentech

Genetický algoritmus s použitím řízených operátorů a lokálního prohledávání

Tento algoritmus GA-GLS využívá stejné mutační operátory jako GA-G, se stejným nastavením pravděpodobností (tabulka 5.3). Po provedení mutace existuje malá pravděpodobnost P_LOCAL_SEARCH, že bude na jedince aplikován operátor lokálního prohledávání `local_search`. Při experimentování s tímto algoritmem se ukázalo, že pro větší instance problému HFVRP je efektivnější algoritmus s nižší P_LOCAL_SEARCH. Prohledávaný prostor je totiž rozsáhlejší, a tak je potřeba provedení více mutačních operátorů k nalezení řešení v jiné části prostoru. Pokud je lokální prohledávání prováděno příliš často, uvázne algoritmus v lokálním minimu. Proto jsem zvolil adaptivní nastavení parametru P_LOCAL_SEARCH, které je závislé na počtu zákazníků vstupního problému. V tabulce 5.4 jsou experimentálně získané hodnoty parametru vzhledem k počtu zákazníků.

Počet zákazníků	P_LOCAL_SEARCH
< 51	1
51-75	0.5
76-100	0.1
101-125	0.05
126-150	0.02
151-175	0.01
176-200	0.005
>200	0.001

Tabulka 5.4: Nastavení pravděpodobnosti lokálního prohledávání v závislosti na počtu zákazníků v procentech

Evoluční strategie s použitím řízených operátorů a lokálního prohledávání

Evoluční strategie ES-GLS je typu (μ, λ), používá stejné nastavení mutačních operátorů jako GA-GLS, ale liší se v průběhu evolučního cyklu. Na začátku evoluce je inicalizována populace o velikosti μ způsobem popsaným v sekci 5.2. Poté proběhne GENERATIONS iterací. V každé iteraci se vytvoří $\lambda-$ELITISM potomků aplikováním operátorů mutace na rodiče, kteří jsou z aktuální populace vybíráni uniformně náhodně. Z těchto potomků se vybere $\mu-$ELITISM nejlepších jedinců, kteří přežijí do další generace. ELITISM nejlepších jedinců přežije do další generace beze změny. Nastavení evolučních parametrů je k vidění v tabulce 5.1, mutačních operátorů v tabulce 5.3 a parametru lokálního prohledávání v tabulce 5.4.

5.7 Implementace

V této podkapitole jsou uvedeny implementační detaily programu pro řešení problému HFVRP. Program je implementován v jazyce C++, konkrétně verze C++17, s použitím pouze standardních knihoven tohoto jazyka. Vývoj probíhal v prostředí operačního systému Windows, s překladem a spouštěním v podsystému Ubuntu 20.04 pomocí nástroje WSL (Windows Subsystem for Linux).

Program je implementován s prvky paradigmatu objektově orientovaného programování. Program se skládá z pěti tříd a několika dalších pomocných funkcí a datových struktur.

Třída `Route` reprezentuje jednu trasu v řešení a operace nad jednou (případně dvěma) trasami. Obsahuje vektor (C++ struktura `vector`) zákazníků reprezentovaných datovou strukturou `cust`, která nese informace o identifikátoru zákazníka, jeho souřadnicích a velikosti požadavku. Dále obsahuje datovou strukturu `vehicle`, která reprezentuje vozidlo obsluhující tuto trasu a nese informace o kapacitě, fixní a variabilní ceně vozidla a identifikátor typu vozidla.

Další třídou je `Solution`, která reprezentuje dekódované řešení a pomocné funkce potřebné pro fungování mutačních operátorů. Obsahuje vektor tříd `Route` a vektor neobsloužených zákazníků `cust`.

Třída `Singleton` obsahuje jednu instanci třídy `Solution`, vektor celých čísel pro reprezentaci chromozomu[1] a ukládá hodnotu fitness, která je znovu počítána pouze po změně jedince. V rámci této třídy jsou implementovány všechny mutační operátory, kódování a dekódování řešení a výpočet fitness.

Třída `Population` reprezentuje populaci jedinců a obsahuje vektor tříd `Singleton`. Obsahuje operace pro inicializaci populace a výběry jedinců (turnaj, ruleta, elitizmus).

Třída `EA` reprezentuje evoluční algoritmus. Obsahuje jednu instanci `Population`, ukládá a vypisuje informace o průběhu algoritmu (nejlepšího jedince, dobu trvání) a definuje metody, které provádí celý evoluční cyklus (genetický algoritmus nebo evoluční strategii). Dále obsahuje datovou strukturu `zadani`, která obsahuje vektor zákazníků (`cust`), vozidel (`vehicle`) a matici vzdáleností. Funkce pro zpracování vstupního souboru a převední do struktury `zadani` jsou implementovány v souboru `input_parse`.

Posledním souborem je `aux_functions`, který obsahuje implementaci různých pomocných funkcí, jako například výpis prvků vektoru, generování náhodných čísel apod.

[1] Tato struktura je ve finálních algoritmech využita pouze při inicializaci, protože mutační operátory pracují přímo nad třídou `Solution`.

Kapitola 6

Experimentální výsledky

V této kapitole jsou popsány experimentální výsledky dosažené metodami, které byly navrženy v rámci této práce. V podkapitole 6.1 jsou charakterizovány datové sady, na kterých byly experimenty prováděny. Experimenty jsou rozděleny do dvou částí. V první části (podkapitola 6.2) je srovnání algoritmů navržených v sekci 5.6. Ve druhém experimentu v podkapitole 6.3 je srovnání nejlepšího z navržených algoritmů s existujícími evolučními algoritmy [11][4]. V obou kapitolách je popsán princip experimentu, statisticky zpracované výsledky experimentů a jejich zhodnocení a diskuze.

Veškeré spouštění programu pro experimentální vyhodnocení probíhalo na superpočítači Salomon národního superpočítačového centra IT4Innovations. Pro spuštění jedné konfigurace se alokují 4 uzly po 24 jádrech a na každém jádru se spustí jedna instance programu. Celkově tedy proběhne 96 běhů pro každou konfiguraci.

6.1 Datové sady

Pro potřeby experimentů v této části byly použity tři datové sady. První dvě datové sady[1] byly poprvé uvedeny v práci É. D. Taillarda z roku 1999[21]. Obě datové sady jsou shodné s tím rozdílem, že v jedné mají vozidla pouze variabilní cenu, zatímco v druhé mají jak fixní tak i variabilní. Každá tato sada obsahuje 8 instancí, s 50, 75 nebo 100 zákazníky. Počet typů vozidel se pohybuje od 3 do 6, celkový počet vozidel od 9 do 17. Jedná se o uměle vytvořené instance a rozmístění zákazníků je zcela náhodné. Datová sada bez fixní ceny je označována jako HVRP, s fixní cenou HVFRP. Instance jsou číslované od 13 do 20[2].

Druhá datová sada pojmenovaná *Duhamel-Lacomme-Prodhon_HVRP*[3] (HVRP_DLP) je vytvořená na základě reálných míst ve Francii. Jednotlivé instance odpovídají reálným správním oblastem a zákazníci městům s maximálně 100 nebo 500 obyvateli, jak pro které oblasti. Tímto způsobem vzniklo 96 instancí s počtem zákazníků v rozsahu 20 až 256. Vzdálenosti mezi zákazníky jsou dány nejkratší cestou nalezenou pomocí nástroje Google web service a odpovídají reálným vzdálenostem mezi městy. Flotila vozidel pro každou instanci je náhodně vygenerovaná a umožňuje až 10 různých typů vozidel. Fixní a variabilní ceny nejsou závislé na kapacitě vozidel, což znamená, že vozidlo s nižší kapacitou může mít

[1] První je ke stažení na https://perso.isima.fr/~phlacomm/hvrp/hvrp.html, druhá vznikne úpravou fixních cen vozidel.[21]

[2] Datová sada původně vznikla pro FSMVRP problém a obsahovala 20 instancí, ale pro HFVRP byly vytvořeny parametry pouze pro 13-20.

[3] K dispozici na https://perso.isima.fr/~phlacomm/hvrp/hvrp.html

vyšší jak fixní, tak variabilní cenu než vozidlo s vyšší kapacitou. Zákazníci nemají explicitně zadané souřadnice, k dispozici je pouze matice vzdáleností mezi zákazníky[4].

6.2 Experiment 1 – srovnání navržených evolučních algoritmů

Předmětem experimentu popsaného v této kapitole je porovnání čtyř algoritmů navržených pro řešení problému HFVRP, charakterizovaných v sekci 5.6. Konkrétně se jedná o genetické algoritmy GA-R, GA-G, GA-GLS a evoluční strategii ES-GLS. Nastavení společných evolučních algoritmů pro všechny varianty je shrnuto v tabulce 5.1. Pravděpodobnosti jednotlivých mutačních operátorů jsou k vidění v tabulkách 5.2 (GA-R) a 5.3 (ostatní). Pravděpodobnost lokálního prohledávání u algoritmů GA-GLS a ES-GLS je popsána v tabulce 5.4. Výsledkem experimentu je statistické zpracování a analýza nalezených řešení a průběhu jednotlivých variant algoritmu. Porovnání je provedeno na datových sadách HVRP a HVFRP, celkem na 16 instancích problému HFVRP.

Prvním výsledkem experimentu je shrnutí celkových cen tras všech vozidel ze všech 96 běhů každé varianty na každé instanci. Výsledky pro datovou sadu HVRP jsou porovnány v tabulce 6.1 a pro datovou sadu HVFRP v tabulce 6.2. Každá položka v tabulce má 3 řádky. Na prvním řádku je celková cena všech tras nejlepšího nalezeného řešení ze všech 96 běhů příslušné varianty na dané instanci, na druhém řádku je průměrná cena řešení nalezených v rámci těchto 96 běhů a na třetím je cena nejhoršího nalezeného jedince během 96 běhů algoritmu na instanci.

Dalším výstupem experimentu je vykreslení výsledků běhů algoritmů pomocí boxplotů. Na obrázku 6.1 jsou boxploty z běhů na datové sadě HVRP a na obrázku 6.2 jsou vykresleny výsledky na datové sadě HVFRP. Pro každou instanci se opět jedná o data ze všech 96 běhů programu.

Na grafech v obrázku 6.3 a 6.4 je porovnána konvergence průměrného běhu každého algoritmu. Průměrný běh se pro každou instanci získá tak, že jsou ceny nejlepšího jedince v dané generaci ze všech 96 běhů zprůměrovány, a výsledné hodnoty jsou vykresleny do křivky.

Posledním výstupem jsou nejlepší nalezená řešení problémů z datové sady HVFRP, vykreslená na obrázku 6.5. Každá trasa je vyobrazena jinou barvou a je u ní uveden typ vozidla, které danou trasu obsluhuje.

6.2.1 Vyhodnocení výsledků experimentu 1

Z výsledků v tabulkách 6.1 a 6.2 lze vypozorovat, že varianta GA-G, která používá řízené operátory, dosahuje lepších výsledků, než varianta GA-R s náhodnými operátory. Zlepšení průměrného nalezeného řešení se na instancích HVRP a HVFRP pohybuje v rozsahu od 0.5 % do 6 %. K největší odchylce došlo u instance HVFRP19, a to o více než 10.5 %. Lze tedy říct, že řízené mutační operátory mají pozitivní vliv na výsledné řešení nalezené evolučním algoritmem. Ještě výraznější zlepšení přináší technika lokálního prohledávání. Algoritmy GA-GLS a ES-GLS nalézají v průměru o 3-9 % lepší řešení, než GA-G. Genetický algoritmus GA-GLS a evoluční strategie ES-GLS v průměru nalézají podobně dobrá řešení. Téměř u všech instancí HVRP a HVFRP dopadla lépe evoluční strategie ES-GLS. Výjimkou jsou instance HVRP20 a HVFRP20, u kterých nalézal lepší řešení genetický algoritmus GA-GLS.

Výše uvedené výsledky jsou podpořeny boxploty na obrázcích 6.1 a 6.2, ze kterých je patrné, že ceny výsledných řešení se při použití řízených operátorů pohybují v nižších hodnotách, než s náhodnými operátory. Z boxplotů lze vypozorovat, že použití lokálního prohledávání výrazně sníží rozsah cen nalézaných řešení, obzvlášť u datové sady HVFRP. To znamená, že tyto metody konvergují k lepším lokálním optimům, než metody bez lokálního prohledávání.

Algoritmy GA-GLS a ES-GLS jsou výrazně časově náročnější, než GA-R a GA-G, z důvodu náročnosti lokálního prohledávání, pro instance o 100 zákaznících je čas některých běhů až 8x delší. Z grafů na obrázcích 6.3 a 6.4 ale vyplývá, že i kdybychom zastavili algoritmy GA-GLS nebo ES-GLS už po 10 000 generacích, dosáhli bychom lepších řešení než GA-R a GA-G po 100 000 generací. Největší zlepšení u těchto algoritmů totiž probíhá právě v prvních 10 000 generacích. Je to dáno tím, že lokální prohledávání rychle dodává do populace poměrně dobrá řešení, která se prostřednictvím ostatních mutačních operátorů nadále zlepšují.

Instance	**GA-R**	**GA-G**	**GA-GLS**	**ES-GLS**
	1559.22	1539.01	1517.84	1517.84
HVRP13	1607.98	1599.92	1548.61	**1543.60**
	1664.06	1649.87	1594.90	1573.07
	633.57	616.57	607.53	609.17
HVRP14	660.32	645.04	619.16	**618.93**
	718.69	686.66	627.65	633.22
	1028.50	1017.58	1015.29	1015.29
HVRP15	1072.06	1057.15	1023.58	**1022.12**
	1135.88	1107.6	1038.36	1036.52
	1155.38	1155.13	1144.94	1144.94
HVRP16	1206.93	1190.67	1152.10	**1151.55**
	1295.88	1229.54	1168.62	1171.29
	1117.39	1092.33	1064.73	1064.07
HVRP17	1184.25	1159.40	1087.27	**1083.52**
	1298.67	1237.40	1117.29	1120.36
	1910.79	1877.77	1828.68	1833.37
HVRP18	2036.43	1996.83	1873.51	**1865.25**
	2222.21	2123.17	1956.38	1931.91
	1219.45	1193.95	1120.34	1120.34
HVRP19	1338.45	1260.56	1154.12	**1154.02**
	1428.45	1369.42	1202.56	1188.90
	1671.19	1618.83	1545.19	1544.52
HVRP20	1751.01	1688.42	**1573.08**	1574.28
	1872.90	1776.01	1610.72	1618.6

Tabulka 6.1: Porovnání výsledků algoritmů na instancích HVFRP. Horní hodnota je cena nejlepšího nalezeného jedince, prostřední je průměrná cena a spodní hodnota je nejhorší cena jedince nalezeného v některém z běhů algoritmu. Tučně jsou vyznačeny nejlepší průměry.

Instance	GA-R	GA-G	GA-GLS	ES-GLS
	3211.09	3219.24	3185.09	3185.09
HVFRP13	3289.00	3271.19	3208.05	**3201.99**
	3377.97	3358.59	3237.68	3236.15
	10169.70	10121.50	10107.50	10107.50
HVFRP14	11148.72	10844.45	10116.14	**10116.01**
	11195.40	11168.40	10132.3	10136.00
	3085.21	3068.08	3065.29	3065.29
HVFRP15	3116.74	3107.92	3068.48	**3067.22**
	3183.41	3148.67	3083.92	3079.08
	3356.49	3329.37	3265.41	3265.41
HVFRP16	3406.38	3390.84	3341.84	**3334.46**
	3476.72	3450.00	3364.14	3355.43
	2149.63	2106.54	2079.16	2077.97
HVFRP17	2227.21	2188.54	2105.01	**2100.30**
	2325.84	2273.23	2137.39	2146.06
	3822.90	3808.54	3748.68	3748.68
HVFRP18	3965.09	3911.19	3787.56	**3778.92**
	4128.60	4106.21	3881.90	3833.70
	10561.70	10474.70	10420.30	10420.30
HVFRP19	12497.92	11172.05	10694.90	**10570.22**
	13321.20	12682.10	11576.50	11074.10
	4964.84	4926.04	4836.70	4834.17
HVFRP20	5064.65	4990.67	**4865.40**	4865.52
	5169.31	5106.02	4905.68	4897.76

Tabulka 6.2: Porovnání výsledků algoritmů na instancích HVFRP. Horní hodnota je cena nejlepšího nalezeného jedince, prostřední je průměrná cena a spodní hodnota je nejhorší cena jedince nalezeného v některém z běhů algoritmu. Tučně jsou vyznačeny nejlepší průměry.

6.3 Experiment 2 – srovnání mého řešení s existujícími algoritmy

Předmětem experimentu 2 je porovnání nejlepšího z mnou navržených algoritmů s existujícími evolučními řešeními problému HFVRP. Z výsledků v experimentu 1 vyplynulo, že nejlepší evoluční technikou je ES-GLS. Výsledky tohoto algoritmu budou porovnány s výsledky dvou existujících evolučních algoritmů pro řešení HFVRP.

Prvním algoritmem, se kterým bude ES-GLS srovnáván, je algoritmus Liu-hybrid [11]. Tato metoda vychází z technik popsaných v sekci 4.3, se dvěma výraznými odlišnostmi. Metoda i pro problém HFVRP používá variantu procedury `split` pro problém s neomezenou flotilou vozidel. Nevalidní řešení pak penalizuje ve fitness funkci tak, aby byla vždy horší než validní řešení. Druhým rozdílem je, že technika lokálního prohledávání s postupem evoluce zvětšuje okolí vstupního řešení, které prohledává. To znamená, že s postupem času provádí v rámci optimalizace čím dál větší změny (přesun více zákazníků, výměna více zákazníků apod.).

Liu-hybrid uvádí výsledky při použití na problémy z datových sad HVRP a HVFRP. Většina algoritmů využívající kombinaci procedury `split` a lokálního prohledávání je schopna

nalézt řešení blízká nejlepšímu známému (a pravděpodobně optimálnímu) řešení. Srovnávání nejlepších nalezených řešení na těchto instancích tudíž nemá příliš velký význam. Proto budu porovnávat průměry nalezených řešení z 10 běhů Liu-hybrid s průměrem z 96 běhů ES-GLS. Porovnání je k vidění v tabulkách 6.3 a 6.4. U každé instance je uveden poměr celkových požadavků k celkové kapacitě vozidel, cena průměrného nalezeného řešení pomocí ES-GLS a Liu-hybrid a procentuální odchylka těchto hodnot.

Druhým algoritmem je hybridní algoritmus GRASPxELS[4], který využívá jinou variantu procedury `split`, konkrétně *Depth First Search Split*. Porovnání proběhlo na datové sadě DLP, která byla vytvořena v rámci práce na algoritmu GRASPxELS. Instance jsou pro přehlednější zpracování rozděleny do 4 sad podle počtu zákazníků. SET_DLP_HVRP1 obsahuje instance o velikosti do 100 zákazníků, sada SET_DLP_HVRP2 od 100 do 150, SET_DLP_HVRP3 od 150 do 200 a SET_DLP_HVRP4 200 zákazníků a více. Porovnání výsledků na těchto sadách je v tabulce 6.5. Tabulky výsledků pro jednotlivé instance jsou uvedeny v příloze A.

6.3.1 Vyhodnocení výsledků experimentu 2

Z tabulek 6.3 a 6.4 lze vypozorovat, že algoritmus ES-GLS nalézá v průměru podobně dobrá řešení, jako Liu-hybrid. Na 9 z 16 instancí Liu-hybrid nalézá průměrně lepší řešení. Na datové sadě HVRP bez fixní ceny se odchylka průměrné ceny ES-GLS od Liu-hybrid pohybuje v rozmezí 0.6-1.6 %. Na instancích HVFRP s fixní cenou dopadl algoritmus ES-GLS o něco lépe, odchylky se pohybují v rozmezí 0.02-0.46 %. Na 6 instancích nalézá algoritmus ES-GLS v průměru lepší řešení než Liu-hybrid. Největší zlepšení je u instancí HVRP19 a HVFRP19, která se vyznačuje tím, že má nejmenší poměr součtu požadavků všech zákazníků k celkové kapacitě vozidel. Bohužel se jedná o jedinou takovou instanci v datových sadách a nelze tedy říct, zda-li je zlepšení dáno tímto poměrem nebo jinými charakteristikami této konkrétní instance. Autoři uvádí, že každý běh terminují po 1 000 sekundách. Doba běhů algoritmu ES-GLS se na těchto instancích při 100 000 generacích pohybuje v rozmezí 1 200-1 500 sekund, což považuji za srovnatelné.

Porovnání výsledků ES-GLS na datové sadě DLP s výsledky získanými algoritmem GRASPxELS je k vidění v tabulce 6.5. Na sadách 1 a 2 se algoritmu povedlo přiblížit řešením získaným algoritmem GRASPxELS. Odchylka průměrných cen řešení nalezených algoritmem ES-GLS je menší než 2.5 %. Průměr nejlepších řešení z 96 běhů je v případě sady 2 dokonce lepší, než průměr řešení získaných GRASPxELS. To znamená, že v některých bězích ES-GLS nalezl lepší řešení, než která jsou uvedena ve výsledcích GRASPxELS. U instancí větších rozměrů dosahuje ES-GLS horších výsledků než GRASPxELS.

Nevýhodou algoritmu ES-GLS je jeho časová náročnost, protože ukončení algoritmu proběhne vždy až po 100 000 generacích i v případech, kdy je nejlepší řešení nalezeno už na začátku evoluce. Doba evoluce se pohybuje v rozmezí 100 sekund pro nejmenší instance, po 4 500 sekund pro největší a je úměrná počtu zákazníků. Čas výpočtu GRASPxELS je v průměru 5-10 krát menší než ES-GLS. Bohužel nelze určit, jak velký podíl na tom má rychlost použitého procesoru (Opteron 2.1Ghz). Dobu evoluce by bylo možné zkrátit zavedením dalších ukončujících podmínek, jako např. maximální omezení času, ukončení při nalezení dostatečně dobrého řešení či ukončení, pokud dlouho nedošlo ke zlepšení. To by ale způsobilo i zhoršení získaných výsledků.

Výsledky k jednotlivým instancím datové sady DLP jsou k dispozici v příloze A. Algoritmus ES-GLS nalezl v rámci 96 běhů pro některé instance řešení lepší než GRASPxELS. V některých případech, například u instance č. 44 (tabulka A.3), je i průměrné nalezené řešení

lepší, než řešení získané GRASPxELS. Existují tedy instance problému HFVRP, u kterých ES-GLS nalézá lepší řešení, než GRASPxELS.

Instance	Dem:Cap	ES-GLS	Liu-hybrid	Odchylka
HVRP13	0.95	1543.60	1534.04	0.6 %
HVRP14	0.88	618.93	609.17	1.6 %
HVRP15	0.95	1022.12	1025.14	-0.3 %
HVRP16	0.95	1151.55	1153.14	-0.13 %
HVRP17	0.95	1083.52	1083.55	0.00 %
HVRP18	0.95	1865.25	1843.16	1.2 %
HVRP19	0.77	1154.02	1160.98	-0.6 %
HVRP20	0.96	1574.28	1560.62	0.88 %

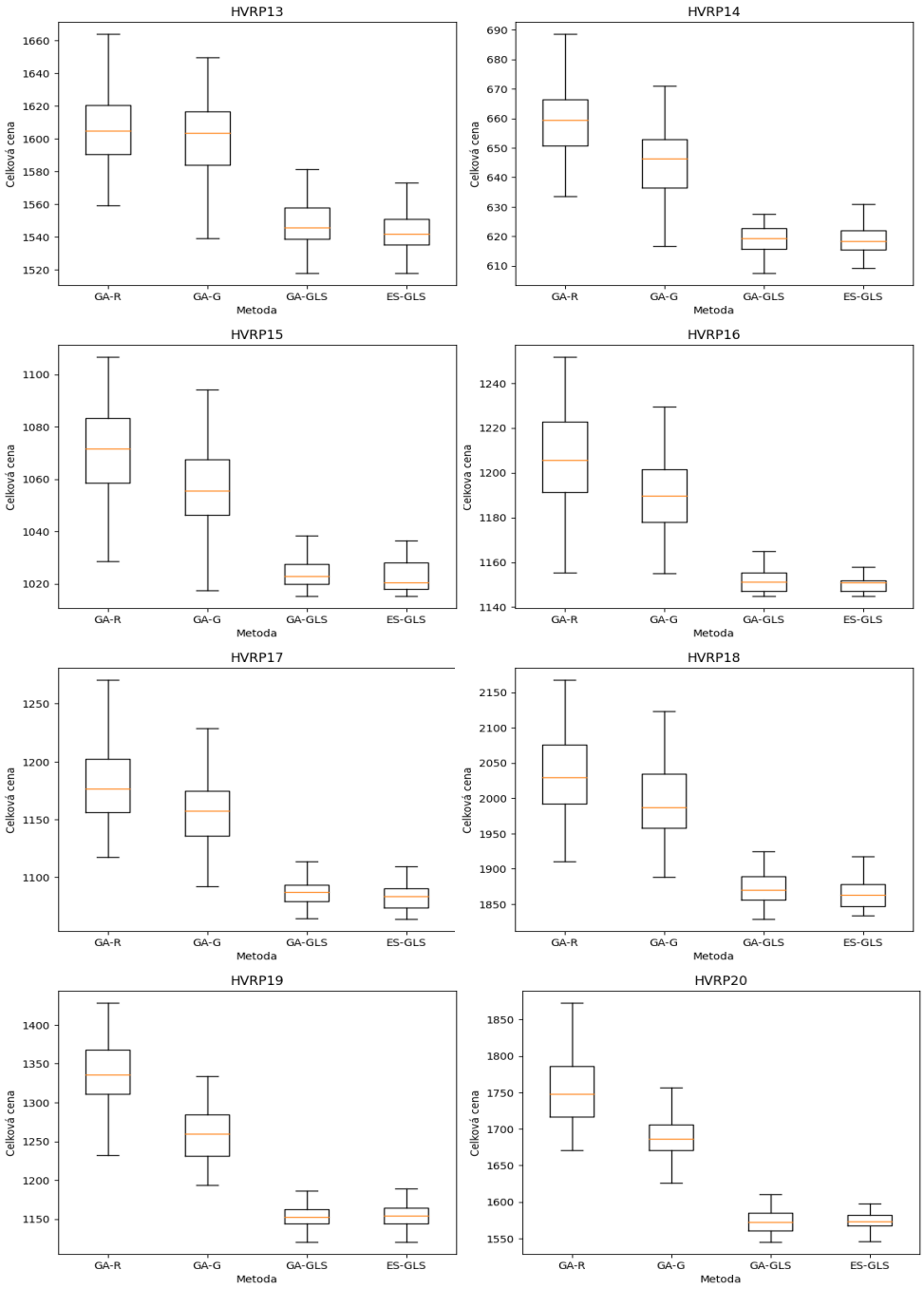
Tabulka 6.3: Srovnání ES-GLS s Liu-hybrid na datové sadě HVRP. Modrá barva značí lepší výsledek, červená horší.

Instance	Dem:Cap	ES-GLS	Liu-hybrid	Odchylka
HVFRP13	0.95	3201.99	3196.12	0.18 %
HVFRP14	0.88	10116.01	10109.17	0.06 %
HVFRP15	0.95	3067.22	3073.12	-0.19 %
HVFRP16	0.95	3334.46	3351.60	-0.51 %
HVFRP17	0.95	2100.30	2098.73	0.07 %
HVFRP18	0.95	3778.92	3761.68	0.46 %
HVFRP19	0.77	10570.22	10786.55	-2.01 %
HVFRP20	0.96	4865.52	4864.40	0.02 %

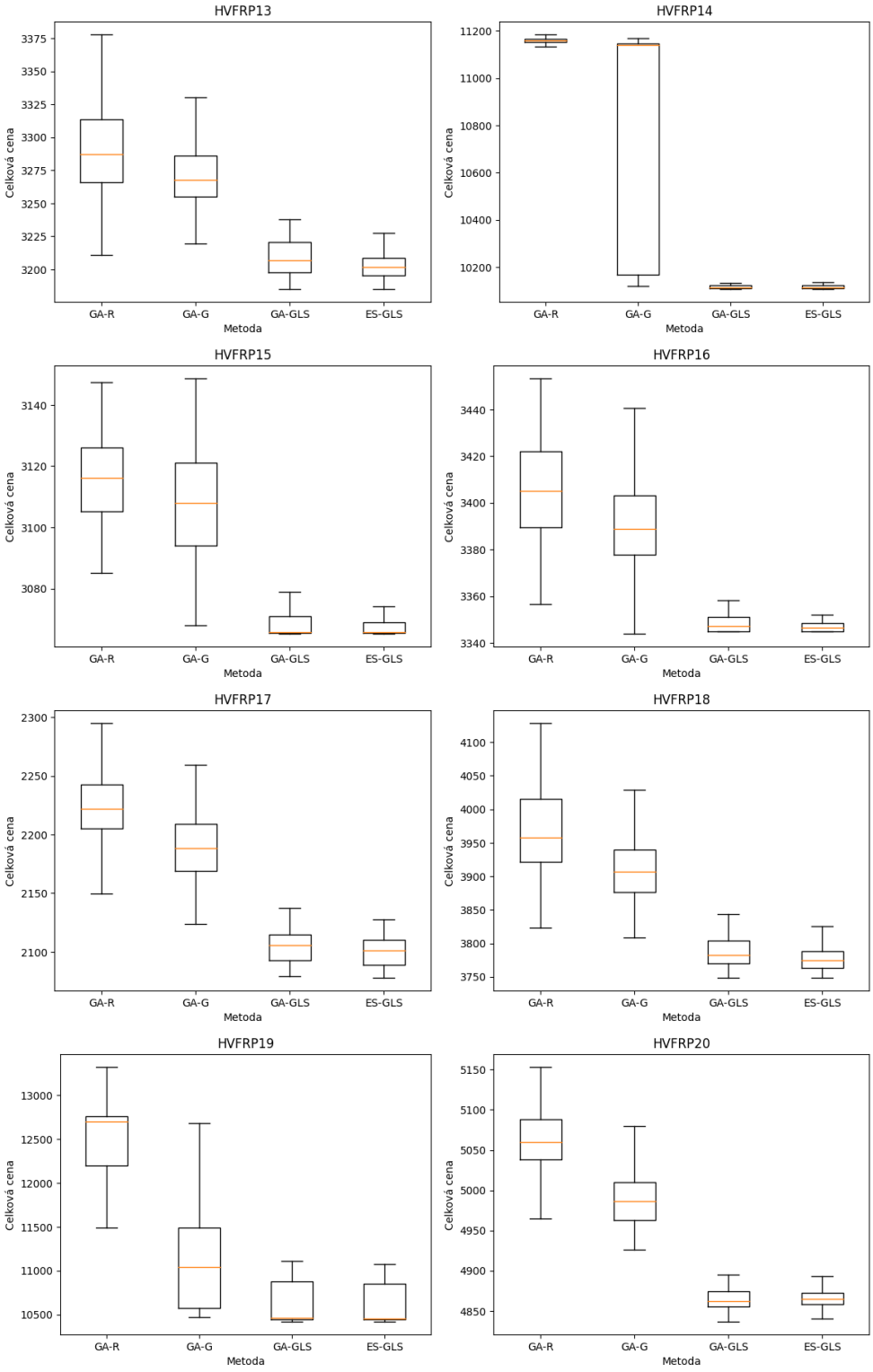
Tabulka 6.4: Srovnání ES-GLS s Liu-hybrid na datové sadě HVRFP. Modrá barva značí lepší výsledek, červená horší.

Instance	ES-GLS best	ES-GLS avg	GRASPxELS	Odchylky
`SET_DLP_HVRP1`	4401.79	4500.23	4393.64	0.19 %, 2.42 %
`SET_DLP_HVRP2`	8685.27	8916.84	8704.26	-0.22 %, 2.44 %
`SET_DLP_HVRP3`	11941.94	12324.01	11791.08	1.28 %, 4.52 %
`SET_DLP_HVRP4`	15013.68	15739.74	14444.81	3.94 %, 8.96 %

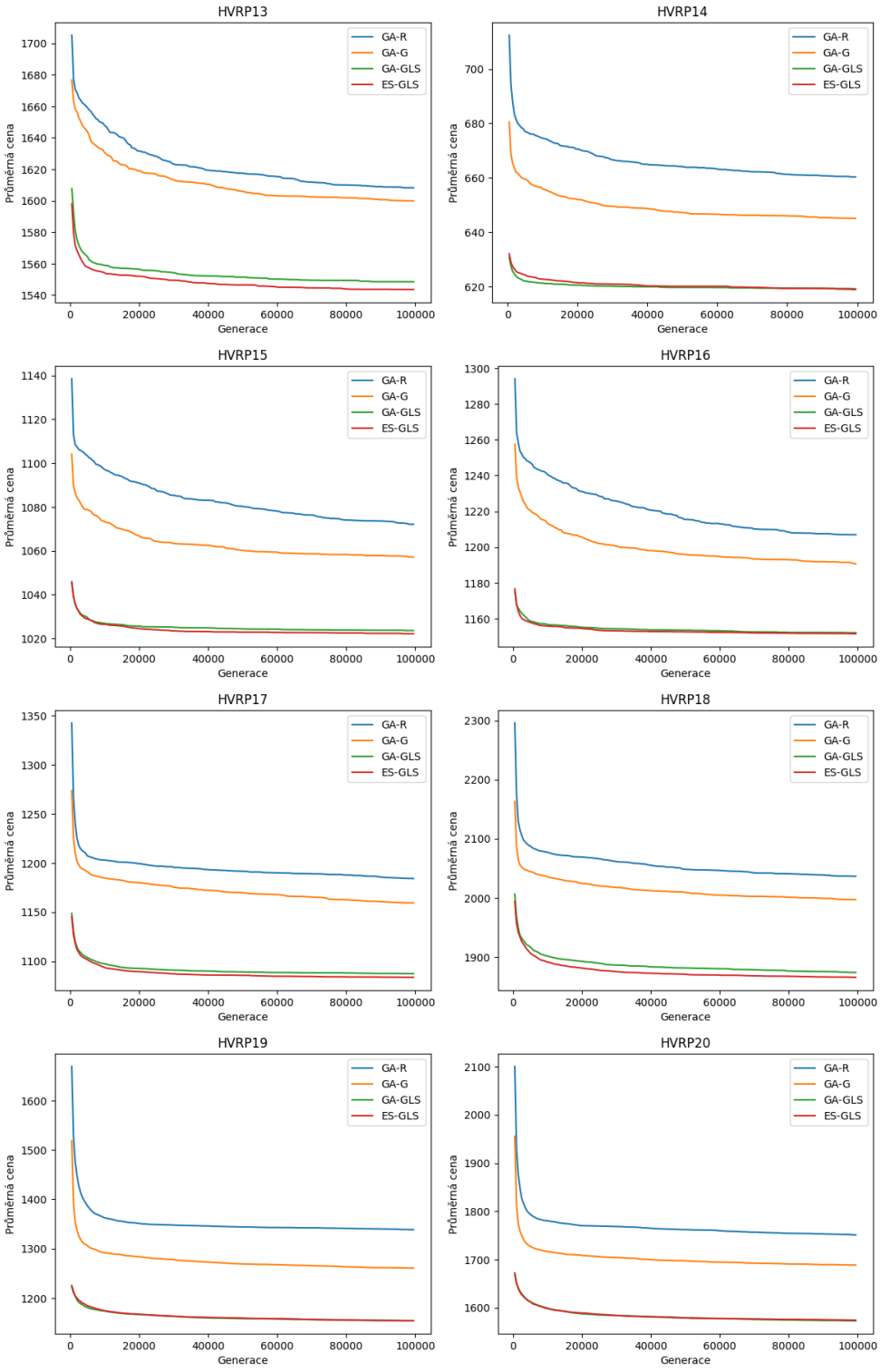
Tabulka 6.5: Srovnání ES-GLS s GRASPxELS algoritmem na datové sadě DLP. První odchylka je od průměru nejlepších řešení každého problému, druhá od průměru všech řešení. Modrá barva značí lepší výsledek, červená horší.



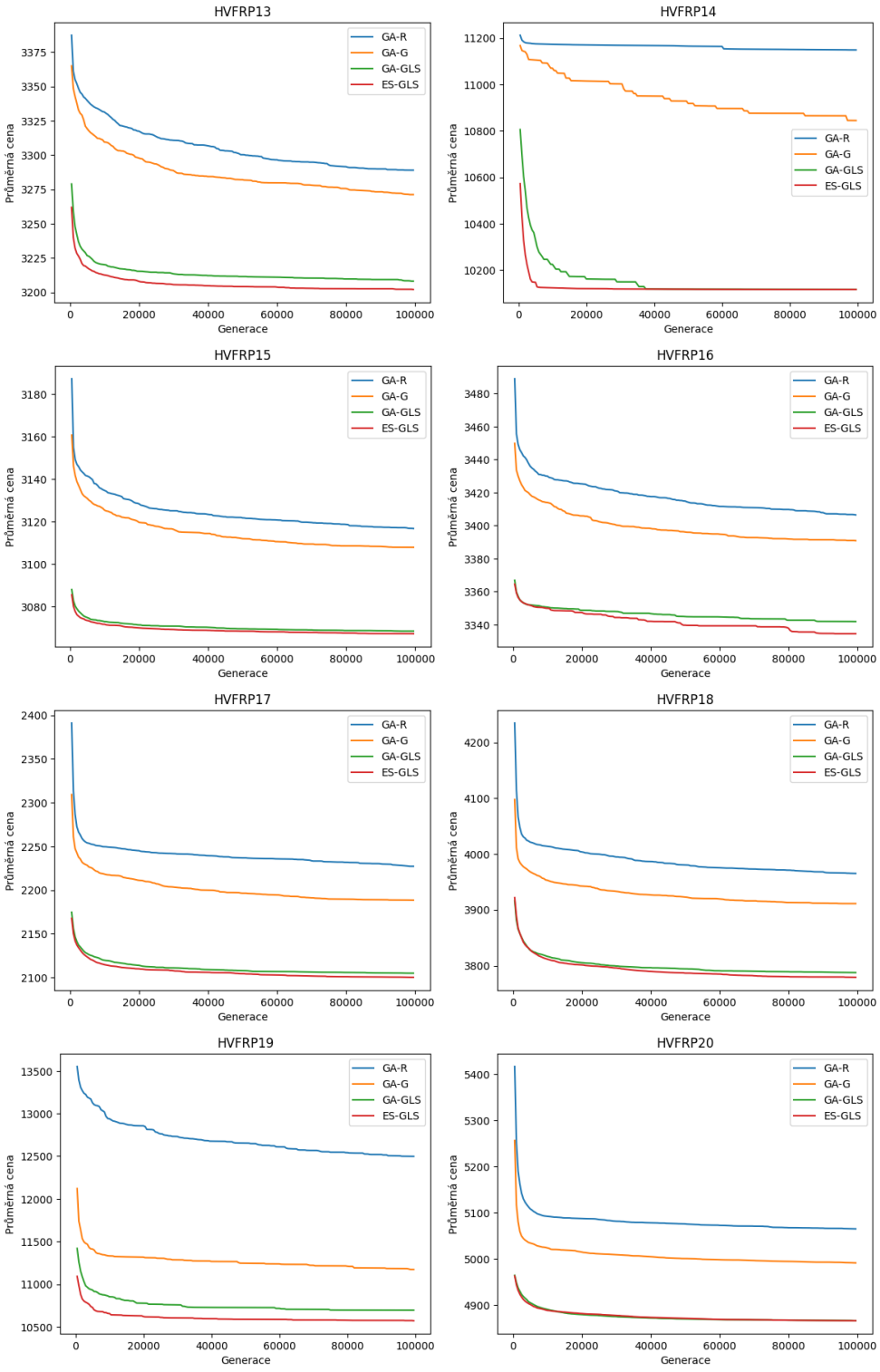
Obrázek 6.1: Statistické vyhodnocení ceny nejlepšího nalezeného jedince během evoluce na instancích z datové sady HVRP.



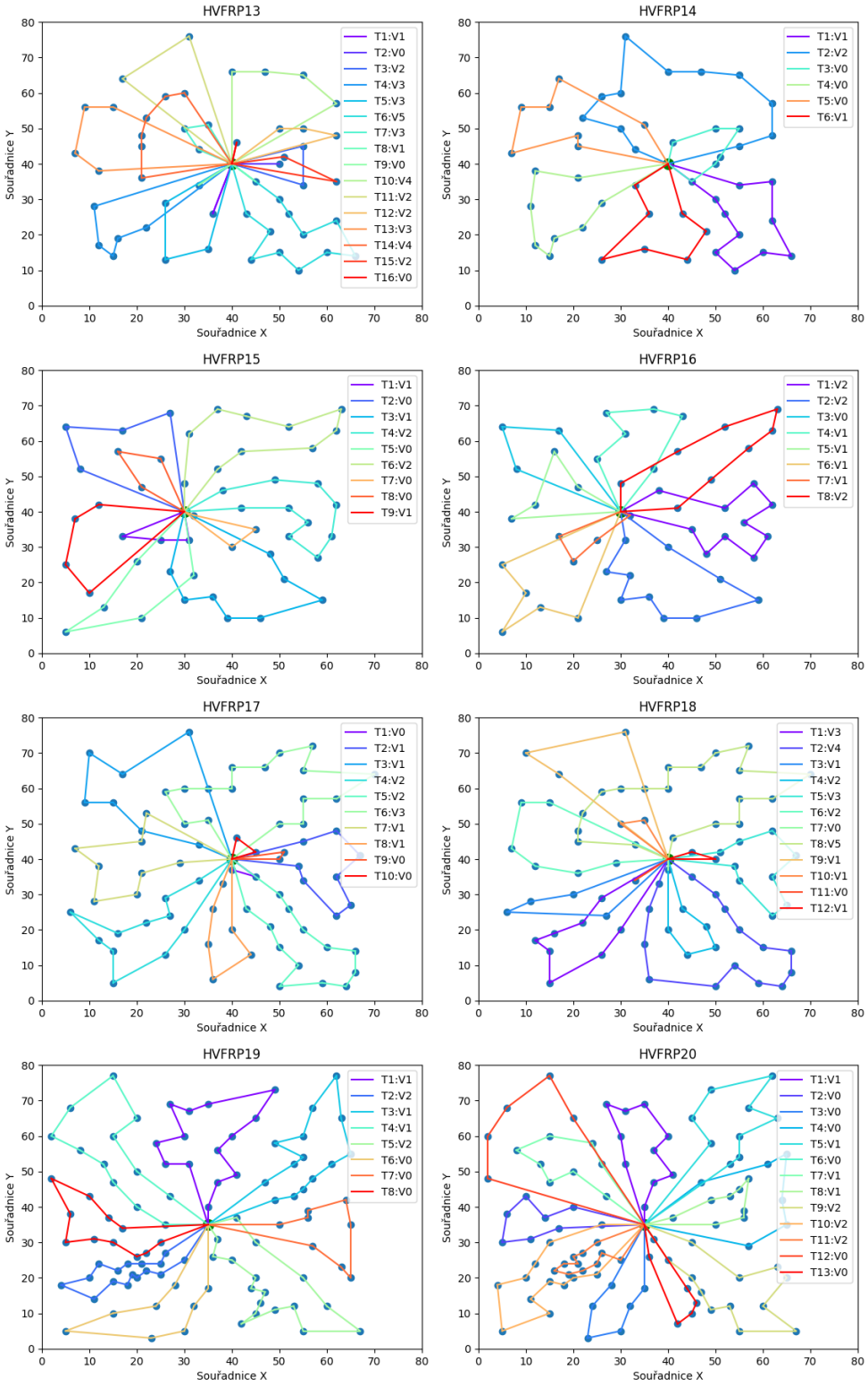
Obrázek 6.2: Statistické vyhodnocení ceny nejlepšího nalezeného jedince během evoluce na instancích z datové sady HVFRP.



Obrázek 6.3: Konvergenční analýza navržených algoritmů na instancích z datové sady HVRP, průměrovaná přes všechny běhy na dané instanci. Křivky jsou z důvodu přehlednosti vykresleny až od 500 generace.



Obrázek 6.4: Konvergenční analýza navržených algoritmů na instancích z datové sady HVFRP, průměrovaná přes všechny běhy na dané instanci. Křivky jsou z důvodu přehlednosti vykresleny až od 500 generace.



Obrázek 6.5: Nejlepší nalezená řešení pro jednotlivé instance HVFRP. Formát označení tras v legendě je T[číslo trasy]:V[typ vozidla].

Kapitola 7

Závěr

Předmětem této práce je řešení jedné z variant problému směrování vozidel, konkrétně problému s heterogenní omezenou flotilou vozidel. Cílem bylo navrhnout a implementovat evoluční algoritmus, který je schopný nalézt dostatečně dobrá řešení, provést experimenty a porovnat získané výsledky s existujícími algoritmy.

V rámci této práce bylo navrženo několik mutačních operátorů, metoda inicializace jedinců a 4 různé evoluční algoritmy, jejichž cílem je efektivní optimalizace zadaného problému. Algoritmy byly implementovány a otestovány na testovacích sadách. Výsledky těchto algoritmů byly statisticky zpracovány a porovnány. Nejúspěšnějším z těchto algoritmů byla evoluční strategie, která používá řízené mutační operátory a lokální prohledávání (ES-GLS). Výsledky této metody byly konfrontovány s výsledky dvou existujících evolučních přístupů pro řešení stanoveného problému.

Algoritmus byl porovnán na třech testovacích sadách. Na dvou z nich dosahuje metoda podobně dobrých výsledků jako existující algoritmy a v podobném čase. Průměrná odchylka řešení získaného ES-GLS se pro různé instance pohybuje v intervalu $\langle -2, 1.6 \rangle$ %, přičemž záporná hodnota znamená lepší výsledek. U třetí datové sady dosahuje ES-GLS podobných výsledků pro menší instance jako srovnávaný algoritmus, ale s delší dobou výpočtu. Pro největší instance problému dochází i přes delší výpočetní čas k získávání horších řešení, v průměru o 9 %. I přesto považuji výsledný algoritmus za poměrně úspěšný, protože u některých instancí problému z třetí testovací sady dokázal ES-GLS najít i výrazně lepší řešení než srovnávaný algoritmus.

Z experimentů provedených v této práci vyplynulo, že evoluční algoritmy pro problém optimalizace přepravy, které využívají více menších mutačních operátorů, mají potenciál. K tomu, aby byly efektivní, potřebují pomocnou metodu pro optimalizaci získaných řešení. V této práci byla využita technika lokálního prohledávání blízkého okolí. Za zvážení by stálo vyzkoušet jiné metody, například tabu prohledávání nebo lokální prohledávání pomocí gravitační emulace. Druhou možností dalšího zlepšování vytvořených metod je použít jinou sadu mutačních operátorů.

Literatura

[1] Brabazon, A., O'Neill, M. a McGarraghy, S. *Natural computing algorithms*. Springer, 2015. ISBN 978-3-662-43630-1.

[2] Dantzig, G. B. a Ramser, J. H. The truck dispatching problem. *Management science*. Informs. 1959, sv. 6, č. 1, s. 80–91.

[3] Deb, K. An introduction to genetic algorithms. *Sadhana*. 1999, sv. 24, 4–5, s. 293–315.

[4] Duhamel, C., Lacomme, P. a Prodhon, C. A GRASPxELS with Depth First Search Split Procedure for the HVRP. Červen 2012.

[5] Eshelman, L. J. Genetic algorithms. In: Back, T., Fogel, D. B. a Michalewicz, Z., ed. *Handbook of Evolutionary Computation*. 1st. GBR: IOP Publishing Ltd., 1997. ISBN 0750303921.

[6] Holland, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992. ISBN 0262082136.

[7] Jacobsen-Grocott, J., Yi Mei, Gang Chen a Zhang, M. Evolving heuristics for Dynamic Vehicle Routing with Time Windows using genetic programming. In: *2017 IEEE Congress on Evolutionary Computation (CEC)*. 2017, s. 1948–1955. DOI: 10.1109/CEC.2017.7969539.

[8] Koç Çağrı, Bektaş, T., Jabali, O. a Laporte, G. Thirty years of heterogeneous vehicle routing. *European Journal of Operational Research*. 2016, sv. 249, č. 1, s. 1–21. DOI: https://doi.org/10.1016/j.ejor.2015.07.020. ISSN 0377-2217. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0377221715006530.

[9] Laporte, G. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research*. Elsevier. 1992, sv. 59, č. 3, s. 345–358.

[10] Lima, C., Goldbarg, M. a Goldbarg, E. A Memetic Algorithm for the Heterogeneous Fleet Vehicle Routing Problem. *Electronic Notes in Discrete Mathematics*. 2004, sv. 18, s. 171–176. DOI: https://doi.org/10.1016/j.endm.2004.06.027. ISSN 1571-0653. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1571065304010844.

[11] Liu, S. A hybrid population heuristic for the heterogeneous vehicle routing problems. *Transportation Research Part E: Logistics and Transportation Review*. 2013, sv. 54,

s. 67–78. DOI: https://doi.org/10.1016/j.tre.2013.03.010. ISSN 1366-5545. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1366554513000604.

[12] Liu, S., Huang, W. a Ma, H. An effective genetic algorithm for the fleet size and mix vehicle routing problems. *Transportation Research Part E: Logistics and Transportation Review*. 2009, sv. 45, č. 3, s. 434–445. DOI: https://doi.org/10.1016/j.tre.2008.10.003. ISSN 1366-5545. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1366554508001324.

[13] Mester, D. a Bräysy, O. Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers & Operations Research*. 2005, sv. 32, č. 6, s. 1593–1614. DOI: https://doi.org/10.1016/j.cor.2003.11.017. ISSN 0305-0548. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0305054803003605.

[14] Ochi, L. S., Vianna, D. S., Drummond, L. M. a Victor, A. O. A parallel evolutionary algorithm for the vehicle routing problem with heterogeneous fleet. *Future Generation Computer Systems*. 1998, sv. 14, č. 5, s. 285–292. DOI: https://doi.org/10.1016/S0167-739X(98)00034-X. ISSN 0167-739X. Bio-inspired solutions to parallel processing problems. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0167739X9800034X.

[15] Prins, C. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*. 2004, sv. 31, č. 12, s. 1985–2002. DOI: https://doi.org/10.1016/S0305-0548(03)00158-8. ISSN 0305-0548. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0305054803001588.

[16] Prins, C. Two memetic algorithms for heterogeneous fleet vehicle routing problems. *Engineering Applications of Artificial Intelligence*. 2009, sv. 22, č. 6, s. 916–928. DOI: https://doi.org/10.1016/j.engappai.2008.10.006. ISSN 0952-1976. Artificial Intelligence Techniques for Supply Chain Management. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0952197608001693.

[17] Rahmani Hosseinabadi, A. A., Slowik, A., Sadeghilalimi, M., Farokhzad, M., Babazadeh shareh, M. et al. An Ameliorative Hybrid Algorithm for Solving the Capacitated Vehicle Routing Problem. *IEEE Access*. 2019, sv. 7, s. 175454–175465. DOI: 10.1109/ACCESS.2019.2957722.

[18] Rechenberg, I. *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, 1973.

[19] Schwefel, H.-P. *Evolution and Optimum Seeking*. Leden 1995. ISBN 978-0-471-57148-3.

[20] Sloss, A. N. a Gustafson, S. 2019 Evolutionary Algorithms Review. *CoRR*. 2019, abs/1906.08870. Dostupné z: http://arxiv.org/abs/1906.08870.

[21] Taillard, E. D. A heuristic column generation method for the heterogeneous fleet VRP. *RAIRO - Operations Research - Recherche Opérationnelle*. EDP-Sciences. 1999, sv. 33, č. 1, s. 1–14. Dostupné z: http://www.numdam.org/item/RO_1999__33_1_1_0/.

[22] Thangiah, S. Vehicle Routing with Time Windows using Genetic Algorithms. Srpen 1995. DOI: 10.1201/9781420050073.ch11.

[23] Toth, P. a Vigo, D. *The vehicle routing problem.* SIAM, 2002. ISBN 0-89871-579-2.

[24] Vanneschi, L. a Poli, R. Genetic Programming — Introduction, Applications, Theory and Open Issues. In: Rozenberg, G., Bäck, T. a Kok, J. N., ed. *Handbook of Natural Computing.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 709–739. ISBN 978-3-540-92910-9.

Příloha A

Výsledky experimentů na datové sadě DLP

Instance	**ES-GLS best**	**ES-GLS avg**	GRASPxELS
HVRP_DLP_1	9235.89	9456.64	9219.65
HVRP_DLP_8	4596.52	4671.56	4603.92
HVRP_DLP_10	2107.55	2127.07	2107.55
HVRP_DLP_11	3374.33	3417.28	3369.91
HVRP_DLP_36	5809.33	6186.37	5730.58
HVRP_DLP_39	2925.47	2970.69	2934.55
HVRP_DLP_43	8818.06	9131.46	8743.13
HVRP_DLP_52	4029.42	4108.28	4029.42
HVRP_DLP_55	10244.3	10341.09	10244.34
HVRP_DLP_70	6685.24	6794.52	6685.24
HVRP_DLP_75	452.853	452.85	452.85
HVRP_DLP_82	4768.21	4792.61	4768.21
HVRP_DLP_92	564.39	611.95	564.39
HVRP_DLP_93	1036.99	1060.89	1038.24
HVRP_DLP_94	1378.25	1380.17	1378.25

Tabulka A.1: Instance s méně než 100 zákazníky (set 1)

Instance	ES-GLS best	ES-GLS avg	GRASPxELS
HVRP_DLP_3	11035.8	11625.92	11320.58
HVRP_DLP_5	11010.5	11306.59	10963.62
HVRP_DLP_6	11899.4	12244.65	11792.94
HVRP_DLP_7	8153.96	8415.41	8130.50
HVRP_DLP_12	3545.0	3710.3	3543.99
HVRP_DLP_13	6730.52	7012.18	6713.14
HVRP_DLP_16	4178.02	4335.28	4161.61
HVRP_DLP_17	5425.65	5591.44	5370.05
HVRP_DLP_2A	8034.46	8240.11	7885.93
HVRP_DLP_2B	8674.12	9043.14	8537.31
HVRP_DLP_21	5186.64	5364.89	5154.38
HVRP_DLP_25	7287.18	7570.59	7228.54
HVRP_DLP_26	6596.74	6751.23	6481.93
HVRP_DLP_28	5575.9	5652.74	5542.76
HVRP_DLP_30	6362.83	6450.96	6321.69
HVRP_DLP_31	4328.15	4741.24	4103.88
HVRP_DLP_34	5885.16	6081.69	5800.12
HVRP_DLP_40	11262.7	11444.17	11172.98
HVRP_DLP_41	7764.16	8150.6	7679.32
HVRP_DLP_47	16274.5	16559.67	16222.94
HVRP_DLP_48	21504.1	21806.9	21413.92
HVRP_DLP_51	7890.3	8176.68	7780.88
HVRP_DLP_53	6491.08	6627.73	6470.49
HVRP_DLP_60	17163.1	17460.29	17067.85
HVRP_DLP_61	7300.62	7451.57	7300.10
HVRP_DLP_66	13060.8	13401.78	13319.73
HVRP_DLP_68	9175.88	9386.88	9135.23
HVRP_DLP_73	10285.4	10422.89	10243.66
HVRP_DLP_74	11739.4	11870.71	11732.54
HVRP_DLP_79	7343.31	7657.74	7314.89
HVRP_DLP_81	10732.0	10845.56	10715.28
HVRP_DLP_83	10045.9	10184.21	10019.83
HVRP_DLP_84	7264.32	7315.6	7269.55
HVRP_DLP_85	8900.17	9001.62	8874.31
HVRP_DLP_87	3771.86	4046.68	3753.87
HVRP_DLP_88	12554.5	12920.94	12443.41
HVRP_DLP_89	7246.82	7491.47	7135.36
HVRP_DLP_90	2359.42	2477.91	2360.83

Tabulka A.2: Instance se 100 až 150 zákazníky (set 2)

Instance	**ES-GLS best**	**ES-GLS avg**	**GRASPxELS**
HVRP_DLP_2	12127.5	12483.42	12102.01
HVRP_DLP_4	11187.3	11565.37	11276.45
HVRP_DLP_9	7728.69	7865.36	7647.59
HVRP_DLP_14	5714.23	5889.03	5679.80
HVRP_DLP_15	8456.64	8624.31	8301.63
HVRP_DLP_24	9261.49	9467.96	9183.78
HVRP_DLP_29	9301.05	9926.49	9147.39
HVRP_DLP_33	9787.92	10253.37	9543.17
HVRP_DLP_35	9870.93	10061.11	9640.80
HVRP_DLP_37	7081.74	7308.57	6921.19
HVRP_DLP_42	11132.2	11616.75	11713.90
HVRP_DLP_44	12437.6	13002.07	14418.00
HVRP_DLP_45	10687.9	11276.81	10519.25
HVRP_DLP_50	13037.3	13708.87	12508.77
HVRP_DLP_54	10623.1	11112.42	11511.62
HVRP_DLP_56	31444.0	31949.59	31292.81
HVRP_DLP_57	45373.8	45926.84	45152.42
HVRP_DLP_59	14661.3	15184.4	14367.14
HVRP_DLP_63	20640.0	21261.04	20241.72
HVRP_DLP_64	17308.9	17923.36	17157.37
HVRP_DLP_67	11174.9	11628.42	11854.61
HVRP_DLP_69	9441.38	9680.59	9276.93
HVRP_DLP_71	10161.2	10465.62	9960.84
HVRP_DLP_72	6118.43	6406.61	5976.54
HVRP_DLP_76	12187.4	12513.86	12098.66
HVRP_DLP_77	7112.04	7561.41	6991.59
HVRP_DLP_78	7241.81	7688.65	7069.82
HVRP_DLP_80	6920.87	7223.67	6839.96
HVRP_DLP_86	9188.61	9290.08	9076.63
HVRP_DLP_91	6471.75	6645.73	6437.14
HVRP_DLP_95	6318.26	6532.62	6244.13

Tabulka A.3: Instance se 150 až 200 zákazníky

Instance	**ES-GLS best**	**ES-GLS avg**	**GRASPxELS**
HVRP_DLP_18	10071.2	10377.64	9782.58
HVRP_DLP_19	12159.2	12686.28	11870.28
HVRP_DLP_22	13559.7	13901.06	13213.34
HVRP_DLP_23	7994.12	8346.59	7849.75
HVRP_DLP_27	8740.47	9332.77	8539.77
HVRP_DLP_32	10180.4	11017.3	9402.54
HVRP_DLP_38	11800.4	12538.67	11397.37
HVRP_DLP_46	25796.7	26669.02	25066.46
HVRP_DLP_49	17004.8	17553.7	16569.06
HVRP_DLP_58	24707.6	25819.91	23826.88
HVRP_DLP_62	24604.3	26462.08	23881.43
HVRP_DLP_65	13545.3	14171.9	13091.74

Tabulka A.4: Instance s více jak 200 zákazníky

Příloha B

Obsah přiloženého paměťového média

Přiložené paměťové médium obsahuje text práce a její zdrojový kód, použité datové sady a zdrojové kódy implementace algoritmů.

Médium obsahuje následující složky a soubory:

- `datasets/` - obsahuje použité datové sady HVRP, HVFRP a DLP
- `src/` - obsahuje zdrojové kódy implementace
- `text/` - obsahuje text práce
  - `source/` - obsahuje zdrojový kód této práce
  - `thesis.pdf` - text této práce ve formátu pdf
- `Makefile` - soubor sloužící k překladu zdrojových kódů
- `README` - stručný manuál ke spuštění programu
- `xberan38_DIP` - binární spustitelný soubor