UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA

DIPLOMOVÁ PRÁCE

Genetické algoritmy pro řešení VRP

Katedra matematické analýzy a aplikací matematiky
Vedoucí práce: **RNDr. Pavel Ženčák, Ph.D.**
Vypracoval(a): **Bc. Zuzana Vranová**
Studijní program: B1103 Aplikovaná matematika
Studijní obor Aplikace matematiky v ekonomii
Forma studia: prezenční
Rok odevzdání: 2016

BIBLIOGRAFICKÁ IDENTIFIKACE

Autor: Bc. Zuzana Vranová

Název práce: Genetické algoritmy pro řešení VRP

Typ práce: Diplomová práce

Pracoviště: Katedra matematické analýzy a aplikací matematiky

Vedoucí práce: RNDr. Pavel Ženčák, Ph.D.

Rok obhajoby práce: 2016

Abstrakt: Rozvozní problém (VRP) je velmi známý optimalizační problém, jehož cílem je optimálně naplánovat rozvoz požadavků zákazníků z centrálního skladu. Obsluha je zajištěna dostupnou flotilou vozidel tak, aby byly minimalizovány náklady přepravy, většinou minimalizací celkové délky tras vozidel. Tato diplomová práce se zabývá řešením rozvozního problému pomocí genetických algoritmů a jejich implementací v Matlabu.

Klíčová slova: VRP, rozvozní problém, genetické algoritmy

Počet stran: 94

Počet příloh: 1

Jazyk: český

BIBLIOGRAPHICAL IDENTIFICATION

Author: Bc. Zuzana Vranová

Title: Genetic algorithms for solution of VRP

Type of thesis: Master's

Department: Department of Mathematical Analysis and Application of Mathematics

Supervisor: RNDr. Pavel Ženčák, Ph.D.

The year of presentation: 2016

Abstract: Vehicle routing problem belongs to very well known optimization problems which aims to plan delivery of goods from central depot to the customers. This delivery is served by a float of vehicles and the objective of this task is to minimize the total route costs. This thesis is focused on solving VRP with genetic algorithms and its implementation in Matlab.

Key words: VRP, vehicle routing problem, genetic algorithms

Number of pages: 94

Number of appendices: 1

Language: Czech

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracovala samostatně pod vedením pana RNDr. Pavla Ženčáka, Ph.D. a všechny použité zdroje jsem uvedla v seznamu literatury.

V Olomouci dne ....

podpis

Obsah

Seznam obrázků

Seznam tabulek

Poděkování

Ráda bych poděkovala svému vedoucímu diplomové práce panu RNDr. Pavlu Ženčákovi, Ph.D. za jeho cenné rady a připomínky, které mi velmi pomohly při vypracování této práce. Dále bych chtěla poděkovat rodině a přátelům za jejich podporu při mém studiu.

Úvod

Tématem mé diplomové práce je řešení rozvozního problému, respektive varianty s kapacitním omezením, pomocí genetických algoritmů. Cílem této práce bylo nastudovat jak fungují genetické algoritmy pro řešení rozvozního problému a jaká jsou jejich specifika. Praktickou část tvořila implementace algoritmu v Matlabu a porovnání výpočetní efektivity jednotlivých metod.

Rozvozní problém neboli Vehicle routing problem je optimalizační úloha, která se zabývá plánováním flotily vozidel vyjíždějících ze skladu (depa) za účelem rozvozu požadavků zákazníků. Rozvozní problém patří do kategorie NP-těžkých problémů a byl poprvé formulován Dantzigem a Ramserem roku 1959, kteří se zabývali optimalizací rozvozu paliva do benzínových pump. Řešení rozvozní úlohy má bohaté využití v praxi a s použitím počítačových modelů umožnilo podstatné snížení nákladů firem. Zahrnutí optimalizace rozvozních tras do procesu plánování vede k větší časové i nákladové efektivitě prováděných činností firmy. V posledních letech byly také zavedeny programy umožňující elektronickou komunikaci mezi řidiči vozidel a "plánovači", což umožňuje rychlejší reakci na dynamičnost daného systému a zmírnění nebo předejití problémům dodávek způsobených nehodami vozidel nebo hustým provozem na silnici. Úspěch těchto algoritmů je dán nejen rozvíjející se počítačovou technikou, ale také zvyšujícím se počtem matematických modelů, které umožňují zohlednit všechny charakteristiky VRP.

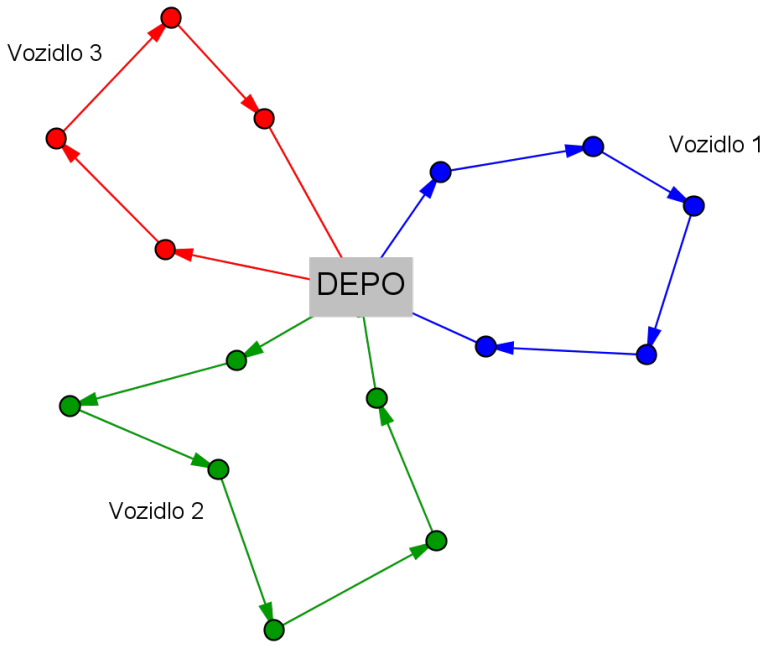
1. Obecná formulace VRP

Při tvorbě této kapitoly jsem vycházela z [1], [2], [4], [5], [6] a [7].

Vehicle routing problem neboli rozvozní problém je optimalizační úloha, která se zabývá plánováním flotily vozidel vyjíždějících ze skladu (depa) k rozvozu požadavků zákazníků.

Model rozvozního problému můžeme popsat takto:
Máme dáno: Množinu zákazníků včetně jejich pozice a výše požadavku.
Úloha: Cílem je určit která vozidla z dostupného vozového parku obslouží požadavky zákazníků, tak aby výsledná cena za přepravu byla co nejnižší. Výsledné řešení musí jednoznačně určit, která vozidla obslouží které zákazníky a v jakém pořadí, tak aby nebyla porušena omezení úlohy. Na obrázku 1.1 je znázorněno možné řešení jednoduchého rozvozního problému s třemi vozidly.

Cílem úlohy může být mimo minimalizace celkových nákladů za přepravu také minimalizace celkové vzdálenosti ujeté vozidly, počtu vozidel potřebných pro přepravu zboží nebo minimalizace času stráveného na cestě.



Obrázek 1.1: Jednoduchý příklad řešení VRP

Optimální řešení rozvozního problému musí spolňovat následující podmínky:

- každé město je navštíveno právě jednou a právě jedním vozidlem
- trasy všech vozidel začínají a končí v depu
- jsou dodržena předem stanovená omezení úlohy (kapacita vozidla, maximální délka jedné trasy, maximální množství vozidel, atd.)

1.1. Kapacitní rozvozní problém

Obecný rozvozní problém může být modifikován přidáním různých podmínek a parametrů. Každá rozvozní úloha je jednoznačně určena svými omezeními, která stanoví, zda je možné dané trasy realizovat nebo ne. Omezení mohou být dána například na maximální kapacitu vozidla, délku trasy jednoho vozidla, povolení opakovaného využití vozidel, stanovený časový interval pro obsloužení zákazníků nebo i kombinace uvedených omezení. Detailnímu popisu různých variant VRP

se věnuje například [1].

Jednou z nejvíce studovaných forem rozvozního problému je kapacitní rozvozní problém (CVRP), u kterého je kladen požadavek na maximální náklad vozidla. Standardní kapacitní dopravní problém můžeme popsat následující definicí:

Definice 1. Kapacitní rozvozní problém

Necht' $G = (V, A)$ je orientovaný graf, kde $V = \{v_0, v_1, \ldots, v_n\}$ je množina uzlů a $A = \{(v_i, v_j)|\ v_i, v_j \in V, i \leq j\}$ je množina hran. Depo je reprezentováno uzlem v_0, $v_0 = 0$ a množina uzlů $\{v_1, \ldots, v_n\}$ reprezentuje místa odběru neboli jednotlivé zákazníky. Uspokojení požadavků $q_i, i = 1 \ldots n$ pocházejících od n zákazníků je zajištěno m vozidly, která mají stejnou kapacitu Q. S hranami $(v_i, v_j),\ i \neq j$ je spojena matice vzdáleností $C = (c_{ij}),\ c_{ij} \geq 0$. Úlohou VRP je najít sadu tras vozidel tak, aby celkové náklady na přepravu byly minimální. Řešením této úlohy je dělení $R_1, R_2, \ldots, R_m$ množiny V, reprezentující trasy vozidel, kde $R_k = \left\{v_{k_0}, v_{k_1} \ldots v_{k_{l+1}}\right\}$, $v_{k_l} \in V$ a $v_{k_0} = v_{k_{l+1}} = v_0 = 0$, za splnění podmínky $\sum_{v_{k_l} \in R_k} q_k \leq Q$. Celkové náklady N na přepravu jsou dány součtem nákladů jednotlivých tras $N(R_k)$ a platí:

$$N = \sum_{k=1}^{m} N(R_k) = \sum_{k=1}^{m} \sum_{k_l \in R_k} c_{k_l} \tag{1.1}$$

.

Úlohu VRP většinou bereme jako symetrickou, tj. $c_{ij} = c_{ji}$. V reálných aplikacích je matice vzdáleností často nesymetrická a musí být vypočítána pomocí geografických dat. V praktických aplikacích také může nastat situace, kdy flotila vozidel není homogenní a některé trasy mohou být přípustné pouze pro určitá vozidla.

2. Úvod do genetických algoritmů

Při přípravě této kapitoly byly použity zdroje [4] a [7].

S genetickými algoritmy jako řešením optimalizačních úloh poprvé přišel John Holland roku 1970. Genetické algoritmy jsou inspirovány Darwinovou teorií přirozeného výběru (1859) a aplikují principy biologické evoluce na řešení optimalizačních úloh. Genetické algoritmy napodobují evoluční procesy jako dědičnost, přirozený výběr, křížení a mutace a tak postupně „šlechtí“ řešení zadané úlohy. Stejně jako v přírodě zdatnější jedinci mají větší šanci se prosadit v konkurenci a přežít déle.

Genetické algoritmy pracují s populacemi, které jsou reprezentovány jedinci, neboli chromosomy, kde každý chromosom reprezentuje možné řešení úlohy. Typicky je populace na začátku simulace složena z náhodných prvků, aby byla zaručena její co největší rozmanitost. Noví jedinci jsou v populaci vytvořeni pomocí kombinace stávajících jedinců použitím operátorů křížení. S určitou pravděpodobností je na potomky použit také operátor mutace, který provede malé náhodné změny v chromosomu a tím pomáhá zachovat rozdílnost jedinců v populaci. Použití operátoru mutace ve většině případů zvyšuje šanci na dosažení globálního minima. Potomci vzniklí pomocí křížení a mutace jsou poté podle pravidel stanovené náhradové strategie začleněni do stávající populace a je tak vytvořena nová generace.

Úspěch genetického algoritmu závisí na mnoha faktorech, jako je například

selekční tlak a nehomogenost populace. Selekční tlak způsobuje, že jsou v populaci upřednostňováni kvalitnější jedinci, přičemž bychom se ale měli snažit o zachování co největší diverzifikace populace. Oba faktory jsou na sobě závislé, pokud roste selekční tlak, klesá rozmanitost populace a naopak. Příliš malý selekční tlak často vede k dosažení neefektivního řešení. Na druhou stranu je ale nejlepších výsledků dosaženo, pokud je co nejdéle zachována co největší rozmanitost populace.

Genetické algoritmy jsou často využívány v praktických aplikacích, a to především z oblasti inženýrství. Mezi nejznámější aplikace patří:

1. **Návrh motorů pro Boeing 777:** Genetické algoritmy byly použity při vývoji proudového motoru pro letadlo Boeing 777. Samotný motor byl navržen běžným způsobem, nicméně pro optimalizaci a doladění parametrů bylo využito genetických algoritmů. Optimalizovaný motor umožnil snížit náklady na palivo o 2,5 procent.

2. **Identifikace zločinců:** Genetické algoritmy mohou být použity také při rekonstrukci tváří osob podezřelých ze spáchání zločinu. Algoritmus pracuje s určitou populací obličejů a pomocí procesu křížení generuje další nákresy tváří. Svědek poté určí pořadí jednotlivých obličejů na základě jejich podobnosti s podezřelou osobou.

Základní postup genetických algoritmů je pro různé optimalizační úlohy vždy stejný. Aplikace algoritmu na různé problémy ale může vyžadovat například jiné způsoby reprezentace jedinců nebo také specifické metody křížení a mutace.

3. Genetický algoritmus pro rozvozní problém

Při přípravě této kapitoly jsem vycházela zo zdrojů [1], [3], [4], [5], [6], [7], [12] a [13].

Pro řešení kapacitního rozvozního problému existuje celá řada heuristik a metaheuristik, například metoda Tabu Search nebo heuristika Clarka a Wrigtha, jejichž přehled byl popsán například v [1] nebo v [4]. Dle [5] již případy s množstvím zákazníků mezi 30 až 40 nemohou být často vyřešeny optimálně, ale pouze přibližně pomocí metaheuristik. Proto pro řešení úlohy dává smysl využití genetických algoritmů. Jelikož pro VRP je charakteristická jeho podobnost velmi dobře prozkoumanému problému obchodního cestujícího (TSP) a může být tedy považován za jeho zobecněnou verzi, je možné pro řešení použít metod původně navržených pro TSP.

Implementace algoritmu pro TSP, respektive VRP má svá určitá specifika. Při generování jedinců musí být zaručeno, že se ve výsledném řešení nebudou jednotlivé geny (zákazníci) opakovat a proto je potřeba aplikovat vhodné metody reprezentace a křížení.

Při řešení úlohy VRP pouze pomocí samotných genetických algoritmů není často možné dosažení konkurenceschopných výsledků. Proto se při řešení úlohy často spoléhá na hybridní přístupy. Hybridní genetické algoritmy kombinují níže

uvedný postup genetických algoritmů s aplikací zlepšujících heuristik, například lokálních optimalizačních metod.

Algoritmus 1. Postup genetického algoritmu

1. Vytvoření počáteční populace, kde každý jedinec je vyjádřen pomocí předem určené reprezentace
2. Ohodnocení jedinců na základě jejich kvality
3. Výběr rodičů z populace, jejichž kombinací budou vytvořeni potomci
4. Vytvoření nových potomků
5. Aplikace mutace na vygenerované potomky
6. Vyhodnocení jedinců a výběr jedinců pro novou generaci

Kroky 3 - 6 se opakují tak dlouho, dokud není splněna stanovená ukončovací podmínka.

Schopnost genetického algoritmu najít přípustné a zároveň dobré řešení je ovlivněna mnoha faktory, například:

- Forma reprezentace jedinců
- Metoda konstrukce počáteční populace
- Zvolená hodnotící funkce jedinců
- Zvolené operátory simulující reprodukci - křížení a mutace
- Metoda náhradové strategie pro vytvoření nové populace
- Hodnoty parametrů, tj. velikost populace, maximální počet iterací, pravděpodobnost s jakou dochází v populaci k mutaci
- Zvolená ukončovací podmínka

3.1. Reprezentace jedinců

Každý kandidát na řešení úlohy CVRP musí jednoznačně určit počet potřebných vozidel, specifikovat kteří zákazníci budou obslouženi kterým vozidlem a pořadí obsluhy jednotlivých zákazníků. Každý jedinec je reprezentován příslušným chromosomem (udává pořadí zákazníků) a ten se skládá z genů (jednotliví zákazníci). V genetických algoritmech existuje mnoho možných způsobů reprezentace jedinců, ne všechny jsou ale vhodné pro reprezentaci jedinců ve VRP.

3.1.1. Permutace zákazníků

Jednou z nejjednoduších a nejsnadněji aplikovatelných metod je reprezentace pomocí řetězce daného permutací zákazníků, který určuje pořadí jejich obsluhy. Každý jedinec má v této reprezentaci dvě formy - kódovanou (pouze pořadí zákazníků, bez rozdělení mezi vozidla) a dekódovanou (pořadí zákazníků včetně rozdělení mezi vozidla). Před výpočtem hodnotící funkce je potřeba chromosom dekódovat a permutaci rozdělit do cest tak, aby řešení bylo přípustné z hlediska omezení úlohy. V případě, kdy se zákazníci rozdělí do tras tak, aby byl minimalizován počet vozidel má každý kódovaný jedinec právě jedno dekódované řešení. Předpokládáme, že první vozidlo obslouží co nejvíce zákazníků od počátku chromosomu, dokud není naplněna jeho kapacita. Podobně pokračujeme pro další vozidla.

Operátory křížení pro rozvozní problém můžeme rozdělit do dvou skupin - ty které pracují pouze s kódovanými jedinci (lze využít operátory křížení a mutace převzaté z TSP) a je potřeba je dekódovat a poté ty, které pracují po celou dobu s dekódovanými jedinci a odpadá proces kódování (specifické operátory pro VRP).

Řešení VRP může vypadat například takto: První vozidlo začíná v depu, obslouží zákazníky $1, 3, 5, 7, 2$ a poté se znovu vrátí do depa. Druhé vozidlo obslouží zákazníky $4, 8, 6, 9$ a vrátí se do depa. Níže jsou uvedeni příslušní kódovaní a

dekódovaní jedinci.

Kódovaný jedinec:

$$1\ 3\ 5\ 7\ 2\ 4\ 8\ 6\ 9$$

Dekódovaný jedinec:

$$0\ \underbrace{1\ 3\ 5\ 7\ 2}_{\text{první vozidlo}}\ 0\ \underbrace{4\ 8\ 6\ 9}_{\text{druhé vozidlo}}\ 0$$

V uvedeném chromosomu 0 značí depo, nicméně v literatuře (např. [6]) je možné se setkat i s rozdělovači trasy očíslovanými $\{n+1, n+2, ...\}$, kde $\{1, 2, ..., n\}$ představují zákazníky.

Reprezentace pomocí permutace zákazníků s depem označeným číslem 0 je později také použita při realizaci genetického algoritmu v Matlabu.

3.1.2. Permutace vozidel

Reprezentace pomocí permutace vozidel byla pro řešení VRP použita například v [11]. Každému zákazníkovi je přiřazeno jedno vozidlo, kterým bude obsloužen. Jednotlivci jsou poté dáni řetězcem čísel vozidel příslušných jednotlivým zákazníkům. Jelikož netrváme na neopakování prvků v chromosomech není nutné používat operátory křížení specifické pro TSP/VRP.

Řešení VRP může vypadat například takto: První vozidlo obslouží zákazníky $1, 2, 3, 5, 7$ a druhé vozidlo obslouží zákazníky $4, 6, 8, 9$. Níže je uvedena podoba příslušného jedince.

zákazníci: 1 2 3 4 5 6 7 8 9
chromosom: 1 1 1 2 1 2 1 2 2

V tomto případě jednotlivé geny chromosomu udávají, kterým vozidlem bude obsloužen příslušný zákazník.

3.2. Přípustnost řešení

Genetické algoritmy mohou být nastaveny tak, aby generovaly pouze řešení, která jsou realizovatelná nebo povolit i generování jedinců, kteří nejsou přípustní vzhledem k omezením úlohy.

Pro úlohy s omezeními existují přípustné a nepřípustné množiny řešení, F a U. Pro celkovou množinu řešení S platí $F \subseteq S$, $U \subseteq S$, $U \cup F = S$ a $U \cap F = \emptyset$. Metody, které vygenerují jedince z množiny nepřípustných řešení U, vyžadují další postupy, které opraví nepřípustné jedince nebo zajistí, aby tito jedinci nebyli vybráni pro reprodukci.

3.2.1. Penalizační funkce

Penalizační funkce je jedním z přístupů, který řeší nepřípustnost jedinců a je často využívána pro genetické algoritmy. Cílem je přidat dostatečně velkou hodnotu k hodnotící funkci v případě, že by byl vytvořen nepřípustný jedinec. Tím je zaručeno, že jedinec bude mít jen velmi malou, téměř nulovou, šanci být vybrán pro další reprodukci.

3.2.2. Oprava jedinců

Metoda opravy jedinců definuje funkci $y = R(x)$, kde y je opravená verze jedince x, taková že $y \in F$ a $x \in U$. Opravený jedinec poté nahradí původního jedince v populaci.

Vzhledem k tomu, že naprogramované soubory v Matlabu umožňují pouze vytvoření přípustných jedinců, jsem se dále opravou nepřípustných řešení nezabývala. Detailnější popis penalizace a opravy jedinců jsou uvedeny například v [6].

3.3. Počáteční populace

Vhodná počáteční populace je velmi důležitá pro efektivní výkon algoritmu. Je nutné aby byla zaručena její co největší diverzita, aby nedošlo k předčasné konvergenci algoritmu. Počáteční populace je většinou generována náhodně, může být ale vytvořena i pomocí konstruktivních heuristik.

V [9] jsou současně vytvářeny a šlechtěny dvě populace řešení, které se vyvíjejí nezávisle na sobě. Při vytváření další populace je umožněna migrace jedinců, tj. určitý počet jedinců je mezi populacemi vyměněn. Tento přístup klade zvláštní důraz na zachování diverzity populace.

3.3.1. Náhodné generování

Jedinci jsou generování jako náhodná permutace zákazníků. Pokud by sekvence obsloužených zákazníků způsobila převýšení kapacity vozidla je do jedince vložen rozdělovač cesty.

Algoritmus 2. Náhodné generování počáteční populace

1. Nechť $V = \{v_1, v_2, \ldots, v_n\}$ je množina všech zákazníků, $\{q_1, q_2, \ldots, q_n\}$ jsou odpovídající velikosti požadavků a Q je kapacita vozidla.
2. Náhodně vyber zákazníka $v_i \in V$, vytvoř cestu $S = \{0, v_i, 0\}$, polož náklad cesty $Q_S = q_i$ a odstraň v_i z množiny zákazníků V
3. Náhodně vyber $v_j \in V$
   (a) pokud je překročena kapacita vozidla, tj. $Q_S + q_j > Q$ vytvoř trasu pro další vozidlo, tj. $S = \{S, v_j, 0\}$ a polož $Q_S = q_j$
   (b) pokud kapacita překročena není, tj. $Q_S + q_j \leq Q \rightarrow$ vlož v_j na předposlední pozici cesty S, polož $Q_S = Q_S + q_j$ a odstraň v_j z množiny zákazníků V
4. Opakuj krok 3 dokud nebudou všichni zákazníci přiřazeni, tj. dokud V není rovno prázdné množině

I náhodná populace se může generovat konstruktivně, a to například tak, že budeme náhodně vybírat pouze uzly, které mají k sobě blíže. Tímto způsobem

získáme hned na začátku několik kvalitních jedinců a tím zrychlíme konvergenci algoritmu.

Algoritmus 3. Náhodné generování počáteční populace pomocí shluků zákazníků

1. Nechť $V = \{v_1, v_2, \ldots, v_n\}$ je množina všech zákazníků, $\{q_1, q_2, \ldots, q_n\}$ jsou odpovídající velikosti požadavků a Q je kapacita vozidla.
2. Vytvoř shluky zákazníků (například pomocí shlukování K-průměrů) $C = \{C_1, \ldots, C_k\}$, $C_i = \{v_{i_1}, v_{i_2}, \ldots, v_{i_n}\}$, kde k je minimální počet vozidel potřebných k obsluze zákazníků. Minimální počet vozidel vypočteme tak, že sečteme poptávky zákazníků a vydělíme kapacitou vozidel.
3. Náhodně vyber shluk $C_i \in C$
4. Náhodně vyber zákazníka $v_{i_l} \in C_i$, vytvoř cestu $S = \{0, v_{i_l}, 0\}$, polož $Q_S = q_{i_l}$ a odstraň v_{i_l} z množiny zákazníků C_i
5. Pokud $C_i \neq \emptyset$ náhodně vyber $v_{i_j} \in C_i$.
   (a) pokud je překročena kapacita vozidla tj. $Q_S + q_{i_j} > Q$ vytvoř trasu pro další vozidlo, tj. $S = \{S, v_{i_j}, 0\}$ a polož $Q_s = q_{i_j}$
   (b) pokud kapacita překročena není tj. $Q_S + q_{i_j} \leq Q \rightarrow$ vlož v_{i_j} do cesty S, polož $Q_S = Q_S + q_{i_j}$ a odstraň v_{i_j} z množiny zákazníků C_i
6. Pokud $C_i = \emptyset$ polož $C = C \setminus C_i$ a vrať se ke kroku 3
7. Opakuj kroky 3. až 6. dokud nebudou všichni zákazníci přiřazeni, tj. dokud C není rovno prázdné množině

3.3.2. Sweep algoritmus

Další možností konstruktivního vytváření jedinců je použití algoritmu Sweep. Touto metodou nicméně získáme pouze jednoho jedince, slouží tedy pouze jako doplnění předchozích přístupů.

Nejdříve je potřeba stanovit počáteční přímku, tj. přímku spojující depo a zvoleného zákazníka. Tento zákazník je poté přidán na začátek cesty. Další zákazníci jsou postupně přidávání na následující pozice tak, že počáteční přímku otáčíme po směru nebo proti směru hodinových ručiček a pokaždé když narazí na nějakého

zákazníka, je tento zákazník vložen do trasy. Prakticky se jednotliví zákazníci vyjádří pomocí polárních souřadnic a poté se jednoduše seřadí podle velikosti úhlu. Takto získáme kódovaného jedince, kterého je potřeba rozdělit do tras, buď již v průběhu výpočtu nebo na jeho konci.

3.4. Výpočet fitness hodnoty

Abychom zjistili kvalitu jedince, je potřeba je nějakým způsobem ohodnotit. Každý jedinec má přiřazenu svou fitness funkci, která udává jeho kvalitu. Kvalita jedince v dopravním problému je většinou dána jeho celkovou délkou tras pro všechna vozidla, může být ale udána také cenou za přepravu nebo množstvím potřebných vozidel.

Pro fitness hodnotu f_i jedince i platí:

$$f_i = \sum_m c_{i,m}$$

kde $c_{i,m}$ je cena m-té trasy i-tého jedince. V této práci je cena jedince dána délkou trasy vypočítanou pomocí Euklidovské vzdálenosti.

3.5. Selekce

Selekce neboli výběr rodičů určuje, kteří jedinci z populace budou vybráni pro vytvoření nových jedinců. Při výběru rodičů se většinou snažíme zohlednit jejich kvalitu, aby platilo evoluční pravidlo přežití „silnějšího“. Kvalitnější jedinci tak mají větší šanci, že budou vybráni pro reprodukci.

3.5.1. Metoda ruletového kola

Metoda ruletového kola simuluje hru rulety a kvalita jedince udává velikost výseče na ruletovém kole. Kvalitnější jedinci dostanou větší část kola a ti nekva-

litní pouze nepatrnou část. Tato metoda má několik variant, lišících se výpočtem pravděpodobnosti vybrání jedince (velikostí výseče).

Pravděpodobnost podle fitness hodnoty

Pro výpočet velikosti výseče na ruletovém kole se v této metodě používá fitness hodnota jedince. Pravděpodobnost, že bude jedinec vybrán je tedy přímo úměrná jeho kvalitě.

Pravděpodobnost, že jedinec bude vybrán se vypočte pomocí následujícího vzorečku:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

kde f_i značí fitness hodnotu i-tého jedince.

Pravděpodobnost podle normované fitness hodnoty

Postup výpočtu je stejný jako u předchozího případu, s tím, že pro výpočet pravděpodobností používáme normovaných fitness hodnot jedinců.

Pravděpodobnost, že jedinec bude vybrán je rovna:

$$p_i = \frac{f_{max} - f_i}{f_{max} - f_{min}}$$

kde f_i značí fitness hodnotu i-tého jedince.

Tento přístup uplatňuje velmi silný selekční tlak, tedy ti nejkvalitnější jedinci mají na ruletovém kole výrazně větší podíl než ti méně kvalitní.

Pravděpodobnost podle pořadí

V pořadové metodě je populace seřazena od nejlepšího po nejhoršího jedince a každému jedinci je přiřazeno jeho pořadové číslo. Na pořadová čísla jedinců se poté aplikuje stejný princip jako v předchozích dvou případech.

3.5.2. Turnajová metoda

K dalším možným metodám výběru patří turnajová metoda, kde se náhodně vybere skupina jedinců a mezi nimi se uspořádá turnaj. Vítězem turnaje se pak stává jedinec s vyšší fitness hodnotou. Selekci můžeme provést také kombinací metody ruletového kola a turnaje, pak se jedná o pravděpodobnostní turnaj.

3.5.3. Metoda ořezávání

V metodě ořezávání provádíme křížení pouze pro ty nejkvalitnější jedince. Jedinci z populace se seřadí podle jejich fitness hodnoty a poté populaci rozdělíme na dvě části. Rodiče se vybírají pouze z první části, tj. z kvalitnějších jedinců.

3.5.4. Náhodná selekce

Jedinci se mohou vybírat také náhodně, bez zohlednění jejich kvality. Tato metoda je nejjednodušší, ale pro řešení není příliš vhodná jelikož neuplatňuje žádný selekční tlak.

3.6. Křížení

Když máme vybrány jedince pro křížení, musíme stanovit, jakým způsobem budeme z rodičů vytvářet nové jedince. Pro rozvozní problém můžeme použít operátory vypůjčené z úlohy obchodního cestujícího, existují ale také algoritmy speciálně navržené pro VRP. Všechny zmíněné operátory pracují s chromosomy, kteří jsou reprezentování pomocí permutace jednotlivých zákazníků.

Různým typům operátorů křížení jsou věnovány samostatné kapitoly 4.1 a 4.2.

3.7. Mutace

Operátor mutace je s určitou pravděpodobností aplikován na vygenerované potomky proto, aby se zabránilo „zaseknutí“ v lokálním minimu a zvýšila se šance na dosažení globálního minima úlohy. Mutace jsou malé, náhodné změny genů uvnitř chromozomu a jejich hlavní funkcí je přinášet nové charakteristické vlastnosti populace. Nejlepších výsledků genetických algoritmů je dosaženo, pokud se co nejdéle zachová různorodost populace, proto jsou důležité operátory mutace, které zaručí rozmanitost v případě, že populace se v důsledku selekčních tlaků stala příliš homogenní. Může se jednat o změny uvnitř cesty jednoho vozidla (změna pořadí návštěv zákazníků) nebo mezi cestami (záměna zákazníků mezi různými vozidly).

Různým operátorům mutace je věnována samostatná kapitola 4.3.

3.8. Lokální optimalizace

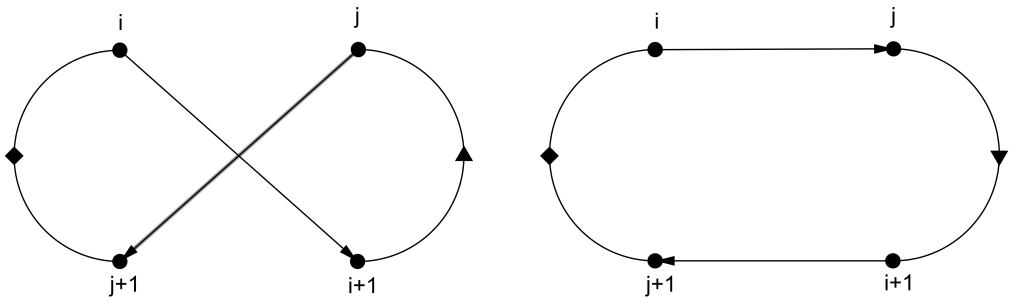
Lokální optimalizace je často součástí genetického algoritmu pro VRP a spočívá v lokálním vylepšení jedince. Lokální optimalizace je přidána proto, aby byla urychlena konvergence algoritmu. Spočívá v záměně zákazníků takovým způsobem, aby bylo dosaženo zkrácení tras ujetých vozidly. Změny lze aplikovat dvěma způsoby, a to udělat pouze jednu změnu v rámci jedné iterace nebo po změně hledat znovu vylepšení a opakovat to tak dlouho, dokud je to možné, což je samozřejmě časově náročnější, byť je dosaženo většího zlepšení.

Dále při aplikaci lokální optimalizace existují dvě strategie a to buď provést tu nejlepší možnou změnu nebo první možnou změnu. Při strategii nejlepší změny jsou vyzkoušeny všechny možné záměny genů a poté provedena ta nejlepší, naopak u strategie první změny je provedena první možná záměna, která vede k vylepšení řešení a zároveň je přípustná z hlediska omezení úlohy.

3.8.1. Lokální optimalizace uvnitř tras vozidel

Nejjednodušší je optimalizovat řešení pouze v rámci tras, jelikož v případě CVRP není potřeba dávat pozor aby nebyla překročena kapacita vozidla. Lokální optimalizace řešení, která mohou být získána z původního řešení pomocí záměny k hran za jiných k hran se nazývají k-opt. Cesta, která již dále nemůže být vylepšena pomocí k-opt se nazývá k-optimální.

Nejčastěji používanou metodou je optimalizace 2-opt, která zamění 2 hrany, za jiné dvě hrany. Tato optimalizace se často používá v TSP a ve VRP se aplikuje na trasy jednotlivých vozidel. Je jednodušší, jelikož nemusíme brát v úvahu možné překročení kapacity vozidla, které může nastat pokud by byly zaměněni zákazníci z různých tras. Princip optimalizace 2-opt je ilustrován na obrázku 3.1.



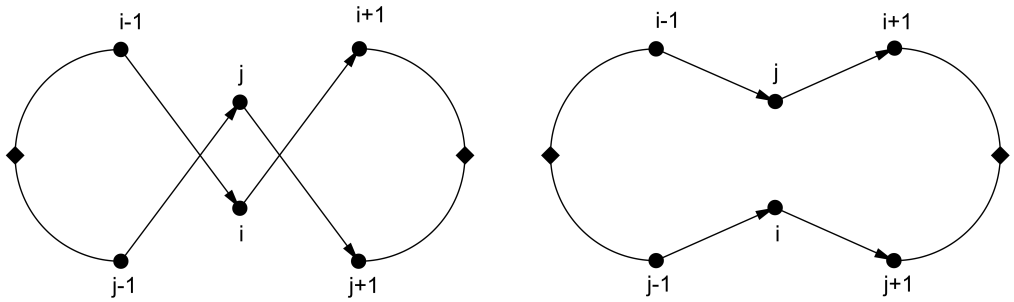
Obrázek 3.1: Lokální optimalizace pomocí 2-opt. Konce trasy jsou označeny symbolem kosočtverce.

Při aplikaci 2-opt hrany $(i, i+1)$ a $(j, j+1)$ jsou nahrazeny hranami (i, j) a $(i+1, j+1)$.

3.8.2. Lokální optimalizace mezi trasami vozidel

Pro lokální optimalizaci mezi trasami může být použita například heuristika Exchange. Tato metoda zamění dva zákazníky ve dvou různých cestách. Při apli-

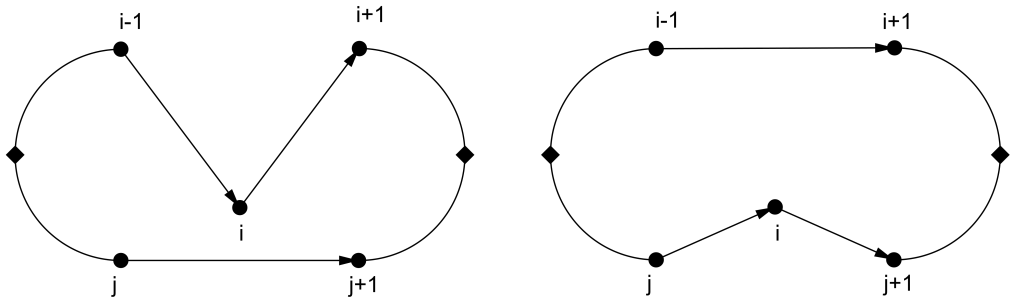
kaci Exchange je ale potřeba dát si pozor, aby nebyla překročena kapacita vozidla.



Obrázek 3.2: Lokální optimalizace pomocí Exchange. Konce trasy jsou označeny symbolem kosočtverce.

Při aplikaci Exchange hrany $(i-1,i)$, $(i,i+1)$, $(j-1,j)$ a $(j,j+1)$ jsou nahrazeny $(i-1,j)$, $(j,i+1)$, $(j-1,i)$ a $(i,j+1)$, tj. dva zákazníci z různých tras jsou současně přesunuti do jiných tras.

Dalším možnou metodou lokální optimalizace je operátor Relocate. Tento operátor jednoduše přemístí zákazníka z jedné trasy do jiné trasy.



Obrázek 3.3: Lokální optimalizace pomocí Relocate. Konce trasy jsou označeny symbolem kosočtverce.

Při aplikaci Relocate hrany $(i-1,i)$, $(i,i+1)$ a $(j,j+1)$ jsou nahrazeny

$(i-1, i+1)$, (j, i) a $(i, j+1)$, tj. zákazník i je z původní trasy přemístěn do konečné trasy.

3.9. Náhradová strategie

Po provedení operací křížení a mutace musíme vybrat, kteří jedinci budou použiti pro vytvoření následující generace. Jedinci se většinou vybírají opět podle hodnoty jejich fitness funkce a při aplikaci selekčního tlaku by postupně měli být jedinci s horšími hodnotami z populace vyřazeni.

3.9.1. Elitismus

Elitismus vybere z populace n nejlepších jedinců a poté se populace náhodně doplní zbývajícími jedinci tak, aby bylo dosaženo stanoveného počtu N jedinců v populaci.

3.9.2. Extrémní elitismus

Extrémní elitismus je speciální případ elitismu, kdy se jedinci uspořádají podle kvality a do další populace je poté vybráno N nejlepších jedinců. Extrémní elitismus často vede k předčasné konvergenci, jelikož populace se brzy stává příliš homogenní.

3.9.3. Prázdná strategie

Prázdná strategie vybírá jedince pro novou populaci zcela náhodně, bez ohledu na jejich kvalitu. Prázdná strategie zachovává z uvedených strategií největší variabilitu v populaci. Selekční tlak je zde realizován pouze prostřednictvím selekce a nikoliv výběrem rodičů a náhradovou strategií současně.

3.10. Ukončovací kritéria

Genetické algoritmy jsou stochastické metody, které by mohly počítat do nekonečna, pokud by nebyla nastavena ukončovací podmínka. Algoritmus může být zastaven na základě různých ukončovacích kritérií nebo i jejich kombinací.

Možná ukončovací kritéria mohou být:

- Dosažení předepsaného počtu generací
- Dosažení předepsaného výpočetního času
- Dosažení předem zadané hodnoty fitness
- Dosažení maximálního počtu iterací od poslední změny nejlepšího řešení
- Pokud se populace stane příliš homogenní a není tedy příliš pravděpodobné, že nastanou změny.
- Dosažení stanoveného počtu jedinců se stejnou fitness hodnotou. Můžeme použít například kritéria rozdílu q-kvantilu a minimální hodnoty

4. Genetické operátory

Při tvorbě této kapitoly byly využity zdroje [3], [7] a [8].

Správná volba genetických operátorů je klíčová pro výkon genetického algoritmu. Předpokládejme populaci velikosti N. Z této populace je vybráno n jedinců, jejichž kombinací je vytvořen jeden nebo více nových jedinců - potomků. Potomci jsou získání pomocí dvou typů genetických operátorů - operátorů křížení a operátorů mutace. Operátory vhodné pro řešení VRP musí být schopny změnit pořadí zákazníků v rámci trasy, změnit vozidla přiřazená požadavkům a také změnit počet vozidel potřebných k obsluze požadavků (přidání nebo smazání tras). Není nezbytně nutné aby operátory generovaly pouze přípustná řešení, musí se ale počítat s tím, že řešení není realizovatelné a dále jej do výpočtu nezahrnovat - použitím penalizační funkce nebo zlepšovacích technik.

4.1. Operátory křížení převzaté z TSP

Při použití operátorů převzatých z TSP křížení probíhá stejně jako u TSP a dostaneme kódované jedince, kteří musí být rozděleni do tras (vložíme rozdělovač trasy - depo na ta místa, kde by aktuální trasa přesáhla kapacitu vozidla). Vzhledem k tomu, že kódovaní jedinci nezachovávají veškerou informaci (chybí například informace kdy je výhodné zajíždět do depa), některé metody mohou vést k neefektivnímu řešení. Většinou je nezbytné aplikovat některé zlepšovací techniky, aby bylo výsledné řešení alespoň blízko k optimálnímu.

4.1.1. Uniform Order Crossover (UOX)

Operátor Uniform Order Crossover je pro řešení VRP použit například v [7]. Operátor Uniform Order Crossover pracuje s náhodně vygenerovaným binárním řetězcem stejné délky jako jednotliví jedinci z populace. Binární řetězec udává, které části rodičů budou zachovány ve vytvořených potomcích.

Pomocí určené metody selekce vybereme z populace dva rodiče určené pro křížení a z prvního rodiče poté zachováme ty geny (čísla zákazníků), kteří se nacházejí na stejné pozici jako číslo „0“ v binárním řetězci. Na zbylé pozice poté vložíme zbývající geny v pořadí v jakém se nacházejí ve druhém rodiči. Podobně se vytvoří druhý potomek, kde se geny nejdříve kopírují z druhého rodiče z míst, kde je v binárním řetězci 1 a poté se zbylé geny doplní ve stejném pořadí v jakém se nacházejí v prvním rodiči.

Algoritmus 4. Uniform Order Crossover

1. Vytvoř potomka p_1, kde p_1 je vektor stejné délky jako je počet zákazníků
2. Náhodně vygeneruj binární řetězec stejné délky jako je počet zákazníků
3. Okopíruj do potomka p_1 geny z prvního rodiče r_1, které se nacházejí na stejné pozici jako 0 v binárním řetězci a vytvoř vektor z, který obsahuje geny druhého rodiče, které ještě nejsou obsaženy v potomkovi, tj. $z = r_2 \setminus p_1$
4. Na prázdné pozice v p_1 postupně doplň zbylé geny z
5. Druhý potomek se vytvoří analogicky z druhého rodiče a pozice z binárního řetězce rovny 1

V následující tabulce je popsán příklad generování potomků pomocí operátoru Uniform Order Crossover.

Binární řetězec	0	1	1	0	1	1	0	0
Rodič 1	1	2	3	4	5	6	7	8
Rodič 2	2	5	1	4	8	7	3	6
Částečné řešení								
Potomek 1	1	–	–	4	–	–	7	8
Potomek 2	–	5	1	–	8	7	–	–
Generovaní potomci								
Potomek 1	1	2	5	4	3	6	7	8
Potomek 2	2	5	1	3	8	7	6	4

Tabulka 4.1: Uniform Order Crossover - příklad

4.1.2. Order Crossover (OX)

Operátor Order Crossover je pro řešení VRP použit například v [5]. Tento operátor náhodně vybere část chromosomu prvního rodiče a zkopíruje ji do potomka. Zbylé geny se doplní z druhého rodiče ve stejném pořadí, v jakém se nacházely v tomto rodiči. Jinak řečeno náhodně zvolíme dva rozdělující body a část cesty, která se nachází mezi těmito body je poté zkopírována do potomka. Zbylé pozice v chromosomu potomka se poté doplní geny z druhého rodiče. Obdobně se vytvoří druhý potomek, pouze s tím rozdílem, že se nejdříve náhodně vybírá část chromosomu z druhého rodiče a zbylé geny se doplňují z prvního rodiče.

Algoritmus 5. Order Crossover

1. Vytvoř potomka p_1, kde p_1 je vektor stejné délky jako je počet zákazníků
2. Náhodně vyber dva dělící body b_1 a b_2, $b_2 > b_1$
3. Na pozice b_1 až b_2 potomka p_1 okopíruj geny z r_1, nacházející se mezi dělícími body b_1 a b_2 a polož $z = r_2 \setminus p_1$
4. Na prázdné pozice v p_1 postupně doplň zbylé geny z
5. Analogicky pro druhého potomka

V následující tabulce je popsán příklad generování potomků pomocí operátoru Order Crossover.

Rodič 1	1	2	3	4	5	6	7	8
Rodič 2	3	5	1	8	4	7	2	6
Částečné řešení								
Potomek 1	–	–	–	4	5	6	–	–
Potomek 2	–	–	–	8	4	7	–	–
Generovaní potomci								
Potomek 1	3	1	8	4	5	6	7	2
Potomek 2	1	2	3	8	4	7	5	6

Tabulka 4.2: Order Crossover - příklad. Dělící body jsou v příkladu znázorněny znakem |.

4.1.3. Partially Mapped Crossover (PMX)

Operátor Partially Mapped Crossover je pro řešení VRP použit například v [5] nebo v [6]. Tento operátor je velmi podobný operátoru Order Crossover. Znovu chromosomy rodičů náhodně rozdělíme na tři části. Část chromosomu z prvního rodiče uvnitř dělících bodů se okopíruje do potomka. Zbylé pozice poté obsadíme tak, aby se chromosom co nejvíce podobal druhému rodiči. Nejdříve tedy dosadíme geny, které mohou být vloženy na stejné pozice, jako byly v druhém rodiči. Zbylé pozice obsadíme geny z druhého rodiče tak, aby se nacházely ve stejném pořadí.

Algoritmus 6. Partially Mapped Crossover

1. Vytvoř potomka p_1, kde p_1 je vektor stejné délky jako je počet zákazníků
2. Náhodně vyber dva dělící body b_1 a b_2, $b_2 > b_1$
3. Na pozice b_1 až b_2 potomka p_1 okopíruj geny z r_1, nacházející se mezi dělícími body b_1 a b_2 a polož $z = r_2 \setminus p_1$
4. Najdi geny ze z, které mohou být vloženy do potomka na stejnou pozici jako se nacházejí v rodiči a okopíruj je do potomka, poté polož $z = r_2 \setminus p_1$
5. Na prázdné pozice v p_1 postupně doplň zbylé geny z
6. Analogicky pro druhého potomka

V následující tabulce je popsán příklad generování potomků pomocí operátoru Partially Mapped Crossover.

Rodič 1	**1**	**2**	**3**	|	4	5	6	|	7	8
Rodič 2	**3**	5	**1**	|	8	4	7	|	**2**	6
Částečné řešení po kroku 3.										
Potomek 1	–	–	–	|	4	5	6	|	–	–
Potomek 2	–	–	–	|	8	4	7	|	–	–
Částečné řešení po kroku 4.										
Potomek 1	**3**	–	**1**	|	4	5	6	|	**2**	–
Potomek 2	**1**	**2**	**3**	|	8	4	7	|	–	–
Generovaní potomci										
Potomek 1	**3**	8	**1**	|	4	5	6	|	**2**	7
Potomek 2	**1**	**2**	**3**	|	8	4	7	|	5	6

Tabulka 4.3: Partially Mapped Crossover - příklad. Dělící body jsou v příkladu znázorněny znakem | a geny, které mohou být okopírovány do potomka na stejnou pozice tučně.

4.1.4. Edge Recombination Crossover (ERX)

Operátor Edge Recombination Crossover je pro řešení VRP použit například v [5] nebo v [6]. Tento operátor ignoruje směr obsluhy zákazníků a interpretuje jednotlivé jedince (chromosomy) jako neřízené cykly hran, tj. není důležitý směr obsluhy zákazníků, ale vzdálenost mezi dvěma zákazníky (ohodnocení hrany). Hlavní myšlenkou je, že potomek by měl zdědit co nejvíce hran, které jsou společné pro oba rodiče.

Operátor Edge Recombination Crossover vygeneruje ze dvou rodičů pouze jednoho potomka. Na začátku algoritmu je nutné vytvořit matici sousedních prvků, tj. pro každého zákazníka vytvoříme seznam zákazníků, kteří jsou v obou rodičích obslouženi bezprostředně před ním nebo jsou v obsluze následující. Matice sousedních prvků bude velikosti $n \times 4$, kde řádky představují jednotlivé zákazníky a poté první 2 sloupce sousední uzly z prvního rodiče, resp. poslední 2 sloupce sousední uzly z druhého rodiče. Geny, které vkládáme do potomka, poté primárně

vybíráme z genů, které jsou v matici sousedních prvků uvedeny jako sousední s předchozím vloženým genem. Pokud rodiče nemají společný žádný sousední gen, vybereme jej ze zbylých sousedních genů náhodně.

Algoritmus 7. Edge Recombination Crossover

1. Vytvoř potomka $p_1 = \emptyset$
2. Vytvoř matici sousedních prvků H, tj. pro oba rodiče a každý gen najdi, se kterými geny sousedí.
3. Polož h rovno prvnímu genu prvního rodiče
4. Gen h vlož na konec potomka p_1 a z matice H vymaž všechny prvky rovny h
5. Pokud h-tý řádek matice H obsahuje zatím nepoužité geny vyber z jeho prvků $h*$. Pokud je některý gen v řádku obsažen dvakrát, má tento gen při výběru prioritu, jinak je výběr proveden náhodně.
6. Pokud h-tý řádek matice H je prázdný, vyber $h*$ náhodně, tak aby nebyl zatím obsažen v potomkovi p_1
7. Polož $h = h*$
8. Opakuj kroky 4. - 7. tak dlouho, dokud se délka potomka nerovná délce rodičů, tj. dokud zbývají neobsloužení zákazníci

V následujících tabulkách je popsán příklad generování potomka pomocí operátoru Edge Recombination Crossover.

Rodič 1	1	2	3	4	5	6	7	8
Rodič 2	3	5	1	8	4	7	2	6

Nejdříve je potřeba vytvořit matici sousedních prvků H.

Gen	Rodič 1		Rodič 2	
1	8	2	5	8
2	1	3	7	6
3	2	4	6	5
4	3	5	8	7
5	4	6	3	1
6	5	7	2	3
7	6	8	4	2
8	7	1	1	4

První gen potomka je vybrán náhodně, v našem případě například zákazník číslo 1. Tento zákazník sousedí v obou rodičích se zákazníkem číslo 8. Na další pozici potomka je tedy vložen tento zákazník. Poté je potřeba aktualizovat matici H tak, že odstraníme první řádek a poté zákazníka číslo 1 ze všech pozic.

Gen	Rodič 1		Rodič 2	
2		3	7	6
3	2	4	6	5
4	3	5	8	7
5	4	6	3	
6	5	7	2	3
7	6	8	4	2
8	7			4

Jelikož zákazník 8 nemá společného souseda v obou rodičích kromě 1 (ten jsme již ale použili a byl vymazán z matice sousedních prvků), bude následující gen potomka vybrán náhodně z genů 7 a 4. V našem případě například zákazník 4. Takto pokračujeme tak dlouho, dokud zbývají volní zákazníci. Postup generování potomků je uveden v následující tabulce.

Rodič 1	1	2	3	4	5	6	7	8
Rodič 2	3	5	1	8	4	7	2	6
Částečné řešení								
Potomek	1	–	–	–	–	–	–	–
Potomek	1	8	–	–	–	–	–	–
Potomek	1	8	4	–	–	–	–	–
Potomek	1	8	4	7	–	–	–	–
Potomek	1	8	4	7	6	–	–	–
Potomek	1	8	4	7	6	2	–	–
Potomek	1	8	4	7	6	2	3	–
Generovaný potomek								
Potomek	1	8	4	7	6	2	3	5

Tabulka 4.4: Edge Recombination Crossover - příklad

4.1.5. Cycle Crossover (CX)

Operátor Cycle Crossover je pro řešení VRP použit například v [5]. Tento operátor se snaží zachovat některé pozice z prvního rodiče a jiné zase z druhého rodiče pomocí cyklů. Při konstrukci cyklů nejdříve vezmeme první gen prvního rodiče a vložíme jej do potomka. Poté se podíváme, který zákazník je na stejné pozici, tj. první, v rodiči druhém a tohoto zákazníka okopírujeme na tu pozici, na které se nachází opět v rodiči prvním. Takto pokračujeme tak dlouho, dokud se nevrátíme na začátek cyklu, tj. první gen prvního rodiče. Další cyklus bude začínat genem z druhého rodiče.

Algoritmus 8. Cycle Crossover

1. Vytvoř potomka p_1, kde p_1 je vektor stejné délky jako je počet zákazníků a polož $i = 1$
2. Polož $h = r_1(i)$, tj. h je rovno i-tému genu prvního rodiče, dále polož $p_1(i) = h$
3. Polož $h = r_2(i)$
   (a) Pokud pokud h není prvkem p_1 najdi index j určující pozici genu h v prvním rodiči r_1, polož $i = j$, a zatím nepoužité geny $z = r_1 \setminus p_1$
   (b) Pokud h je prvkem p_1, vyber h náhodně z množiny zbývajících genů z a zaměň rodiče, tj. další cyklus bude kopírovat geny a jejich pozice z druhého rodiče
4. Opakuj kroky 2-3 tak dlouho, dokud neplatí, že $z = \emptyset$

V následující tabulce je popsán příklad generování potomka pomocí operátoru Cycle Crossover. V uvedeném příkladu nejdříve okopírujeme první gen z prvního rodiče, tj. 3. Na stejné pozici jako 3 ve druhém rodiči je zákazník 4. Z pvního rodiče tedy okopírujeme gen 4 na stejnou pozici. Na stejné pozici jako 4 v prvním rodiči je zákazník 8 ve druhém rodiči. Zákazník 8 bude opět okopírován na stejné pozici jako se nachází v rodiči 1. Pokračujeme tak dlouho, dokud ve druhém rodiči nenarazíme na gen, který je již v jedinci obsažen. Poté se náhodně vybere gen, který ještě v jedinci není obsažen a cyklus začíná u druhého rodiče. Pokud již žádný další cyklus v rodiči neexistuje, okopírujeme zbylé geny z prvního nebo druhého rodiče (záleží u kterého rodiče skončil poslední cyklus) ve stejném pořadí v jakém se nacházejí v tomto rodiči.

Rodič 1	3	5	2	4	1	8	7	6
Rodič 2	4	1	2	8	7	6	5	3
Cyklus 1								
Potomek	3	–	–	–	–	–	–	–
Potomek	3	–	–	4	–	–	–	–
Potomek	3	–	–	4	–	8	–	–
Potomek	3	–	–	4	–	8	–	6
Cyklus 2								
Potomek	3	1	–	4	–	8	–	6
Potomek	3	1	–	4	–	8	5	6
Potomek	3	1	–	4	7	8	5	6
Generovaný potomek								
Potomek	3	1	2	4	7	8	5	6

Tabulka 4.5: Cycle Crossover - příklad

4.2. Operátory křížení uzpůsobené pro VRP

Vedle metod křížení, které byly jednoduše převzaty z úlohy obchodního cestujícího existují i algoritmy, které byly navrženy speciálně pro řešení rozvozního problému. Tyto algoritmy jsou schopny pracovat s depy a odpadá tedy proces dekódování jedince, tj. rozdělení do jednotlivých tras. Tyto algoritmy jsou schopny dojít k dobrému řešení i bez použití hybridního přístupu ve formě například lokální optimalizace získaných řešení.

4.2.1. Best Route Crossover (BRX)

Operátor Best Route Crossover byl pro řešení VRP použit například v [6]. Tento operátor se snaží zachovat "dobré" cesty z obou rodičů a zahrnout je do potomka. "Dobrá" cesta je definována jako taková cesta vozidla, která co nejlépe využívá kapacity vozidla a zároveň není příliš dlouhá. Jednotlivé cesty jsou uspořádány podle rozdílu mezi nákladem vozidla a jeho celkovou kapacitou a podle jejich délky. Předpokládáme-li m vozidel, je poté vybráno několik tras (maximálně $m/2$), které mají co nejvíce naplněnou kapacitu a zároveň nejsou delší než je median cest v rodiči. Tyto cesty jsou poté vloženy na první pozice potomka.

Zbylé pozice potomka jsou doplněny ve stejném pořadí v jakém se nacházejí v druhém rodiči.

Algoritmus 9. Best Route Crossover

1. Vytvoř potomka $p_1 = \emptyset$
2. Pro všechny trasy vozidel z prvního rodiče $r_1 = (t_1, \ldots, t_m)$ spočítej jejich délku a náklad vozidla
3. Vyber nejlepší trasy $t_{i_1}, \ldots, t_{i_t} \in r_1$, pro které platí, že délka trasy d_{t_i} je co nejkratší a kapacita vozidla Q_{t_i} je co nejvíce naplněna, tj. například s využitím mediánu musí platit: $d(t_i) \leq \text{med}(d_{t_i})$ a $Q_{t_i} \geq \text{med}(Q_{t_i})$
4. Tyto trasy okopíruj do potomka, $p_1 = (t_{i_1}, \ldots, t_{i_t})$
5. Najdi zákazníky, kteří zatím nebyli přidáni do potomka, $z = r_1 \setminus p_1$
6. Zbylí zákazníci z se poté postupně doplní za okopírované trasy do potomka p_1 ve stejném pořadí, v jakém se nacházejí v druhém rodiči r_2. Pokud by v průběhu doplňování zákazníků došlo k překročení kapacity, je vložen rozdělovač trasy.
7. Analogicky pro druhého potomka

V následujících tabulkách je ilustrován příklad generování potomka pomocí operátoru Best Route Crossover. Nejdříve musíme spočítat délku jednotlivých tras a náklad vozidel pro prvního rodiče.

Trasa	**Zákazníci**	**Délka trasy**	**Náklad**
1	4 - 5 - 6	131	70
2	10 - 7	229	30
3	3 - 9 - 8	290	40
4	1 - 2	169	45

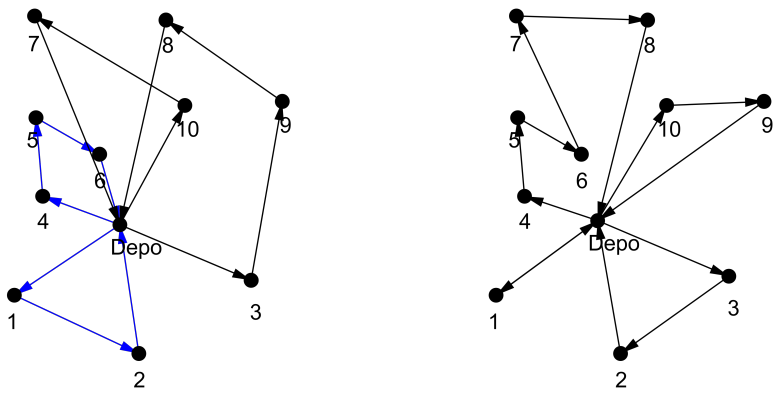
Tabulka 4.6: Best Route Crossover - příklad (rodič 1)

Z tabulky je zřejmé, že trasy 1 a 4 splňují podmínku na délku trasy i náklad vozidla, tj. délka trasy je menší než median délek tras (199) a náklad vozidla je větší než median nákladů vozidel (42,5). Tyto trasy budou poté okopírovány do potomka a zbylí zákazníci budou doplněni ve stejném pořadí v jakém se nacházejí v druhém rodiči.

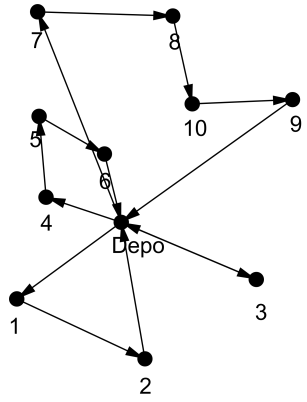
Rodič 1	0	**4**	**5**	**6**	0	10	7	0	3	9	8	0	**1**	**2**	0
Rodič 2	0	1	0	4	5	6	7	8	0	10	9	0	3	2	0
Částečné řešení															
Potomek	0	**4**	**5**	**6**	0	**1**	**2**	0							
Generovaný potomek															
Potomek	0	4	5	6	0	1	2	0	7	8	10	9	0	3	0

Tabulka 4.7: Best Route Crossover - příklad (řešení)

Na následujících obrázcích je graficky znázorněn postup řešení předchozího příkladu pomocí operátoru Best Route Crossover.



Obrázek 4.1: Best Route Crossover - rodiče. Modře jsou vyznačeny trasy prvního rodiče, které budou okopírovány do potomka.



Obrázek 4.2: Best Route Crossover - potomek

4.2.2. Common Edge Crossover (CEX)

Operátor Common Edge Crossover byl pro řešení VRP použit například v [7]. Operátor Common Edge Crossover generuje ze dvou rodičů jednoho potomka. Operátor je založen na myšlence zachovat "dobré" části rodičů, které jsou společné pro oba rodiče a tím dosáhnout kvalitnějšího řešení. Operátor tak zachová v potomku hrany z prvního rodiče, které se nacházejí i v druhém rodiči.

Algoritmus 10. Common Edge Crossover

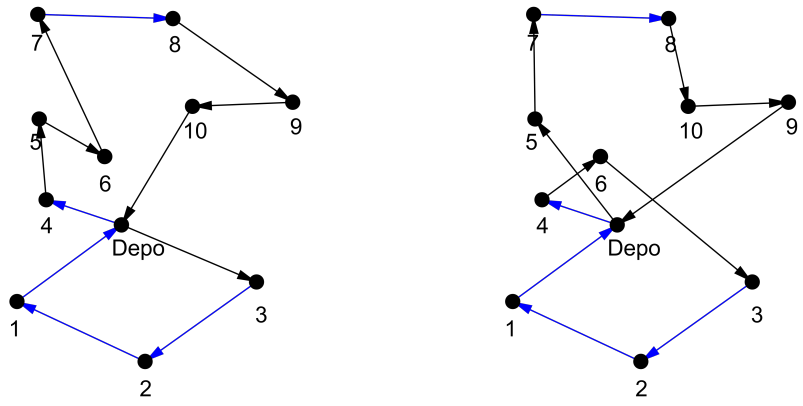
1. Vytvoř potomka $p_1 = \emptyset$
2. Pro $\forall v_i \in r_1$, pokud je uzel v_i počátečním nebo koncovým bodem hrany společné oběma rodičům, přidej tento uzel na konec potomka p_1
3. Po zhodnocení všech zákazníků $v_i \in r_1$ se do p_1 vloží zbylí zákazníci tak, aby prodloužení trasy způsobené jejich vložením bylo co nejmenší, tj. do existujících tras se gen vloží na takovou pozici, která výslednou délku tras prodlouží nejméně ze všech možností. Pokud gen není možné vložit do žádné existující trasy kvůli překročení kapacity vozidla, je vytvořena trasa nová.

V následující tabulce je popsán příklad generování potomka pomocí operátoru Common Edge Crossover. Tučně jsou vyznačeny hrany společné obou rodičům.

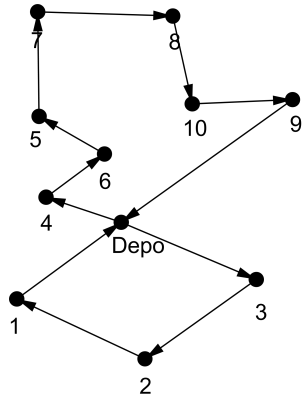
Rodič 1	0	**3**	**2**	**1**	**0**	**4**	5	6	**7**	**8**	9	10	0
Rodič 2	**0**	**4**	5	**3**	**2**	**1**	**0**	5	**7**	**8**	10	9	0
Částečné řešení													
Potomek	0	3											
Částečné řešení													
Potomek	0	3	2										
Částečné řešení													
Potomek	0	3	2	1									
Částečné řešení													
Potomek	0	3	2	1	0								
Částečné řešení													
Potomek	0	3	2	1	0	4							
Částečné řešení													
Potomek	0	3	2	1	0	4	7						
Částečné řešení													
Potomek	0	3	2	1	0	4	7	8					
Generovaný potomek													
Potomek	0	3	2	1	0	4	6	5	7	8	10	9	0

Tabulka 4.8: Common Edge Crossover - příklad

Na následujících obrázcích je graficky znázorněn postup řešení předchozího příkladu pomocí operátoru Common Edge Crossover.



Obrázek 4.3: Common Edge Crossover - rodiče. Modře jsou vyznačeny části tras, které jsou společné oběma rodičům.



Obrázek 4.4: Common Edge Crossover - potomek

4.2.3. Route Based Crossover (RBX)

Operátor Route Based Crossover byl pro řešení VRP použit například v [8]. Tento operátor generuje potomka tak, že nahradí cestu jednoho rodiče cestou z druhého rodiče, při zachování zbylých hran rodiče v potomku. Zbylé geny se poté doplní tak, aby prodloužení trasy způsobené jejich vložením bylo co nejmenší.

Algoritmus 11. Route Based Crossover

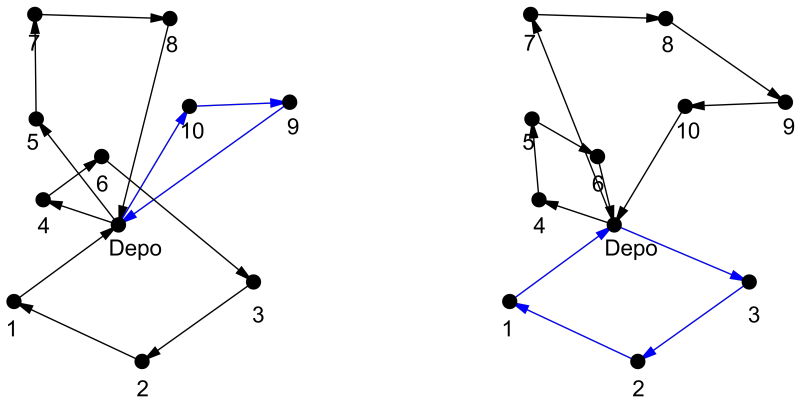
1. Polož $p_1 = r_1$
2. Náhodně vyber cestu s_2 z r_2
3. Z potomka p_1 se odstraní zákazníci, kteří patří do trasy s_2
4. Náhodně vyber cestu s_1 z p_1, odstraň tuto cestu z p_1
5. Cestu s_2 přidej do potomka p_1 na stejnou pozici jako se nacházela trasa s_1
6. Doplň zbylé zákazníky tak, aby cena za jejich vložení byla co nejmenší, tj. do existujících tras se gen vloží na takovou pozici, která výslednou délku tras prodlouží nejméně ze všech možností. Pokud gen není možné vložit do žádné existující trasy kvůli překročení kapacity vozidla, je vytvořena trasa nová.
7. Analogicky se vytvoří druhý potomek p_2

V následující tabulce je popsán příklad generování potomka pomocí operátoru Best Route Crossover. Tučně jsou vyznačeny trasy, které budou zaměněny, tj. trasa $s_1 = 0, 10, 9, 0$ a trasa $s_2 = 0, 3, 2, 1, 0$.

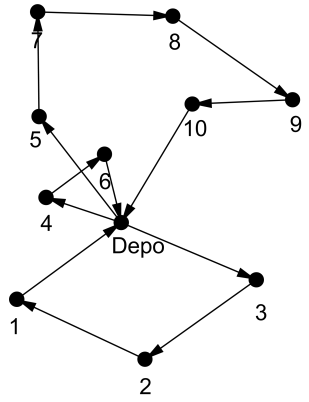
Rodič 1	0	4	6	3	2	1	0	5	7	8	**0**	**10**	**9**	**0**
Rodič 2	0	4	5	6	0	7	8	9	10	**0**	**3**	**2**	**1**	**0**
Částečné řešení po kroku 3.														
Potomek	0	4	6	0	5	7	8	0	10	9	0	–	–	–
Částečné řešení po kroku 4.														
Potomek	0	4	6	0	5	7	8	0	–	–	–	–	–	–
Částečné řešení po kroku 5.														
Potomek	0	4	6	0	5	7	8	0	3	2	1	0	–	–
Generovaný potomek														
Potomek	0	4	6	0	5	7	8	9	10	0	3	2	1	0

Tabulka 4.9: Best Route Crossover - příklad

Na následujících obrázcích je graficky znázorněn postup řešení předchozího příkladu pomocí operátoru Route Based Crossover.



Obrázek 4.5: Route Based Crossover - rodiče. Modře jsou vyznačeny trasy, které budou zaměněny.



Obrázek 4.6: Route Based Crossover - potomek

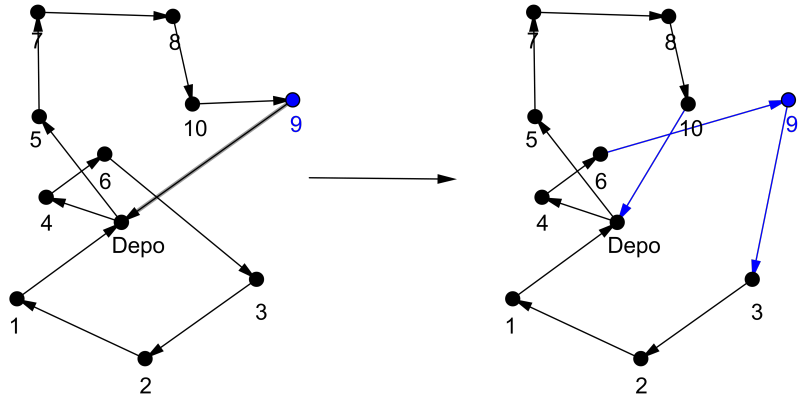
4.3. Operátory mutace

Potomci vzniklí křížením mohou dále s určitou pravděpodobností podstoupit mutaci. Zda potomek bude zmutován nebo ne je dáno stanovenou pravděpodobností. Operátory mutace umožňují záměnu zákazníku v rámci jedné trasy nebo i mezi trasami, může se také vybrat část trasy a přesunout ji na jiné místo. Pokud by byla vybrána k přemístění celá trasa jednoho vozidla, může v některých případech dojít i ke změně počtu vozidel. Při aplikaci operátorů mutace je potřeba dát si pozor na to, aby nebyla porušena omezení úlohy. V případě, že by změna porušila omezení, je potřeba provést mutaci záměnou jiných zákazníků.

4.3.1. Gene Insertion

Operátor Gene Insertion byl pro řešení VRP použit například v [6]. Tento operátor náhodně vybere gen z chromosomu (zákazník nebo i depo) a přesune jej na jiné místo v trase stejného vozidla nebo jej přiřadí do trasy jiného vozidla. V případě přesunutí genu do jiné trasy je potřeba dát pozor na to, aby nebyla porušena omezení úlohy.

Na následujícím obrázku je znázorněna mutace potomka pomocí operátoru Gene Insertion.

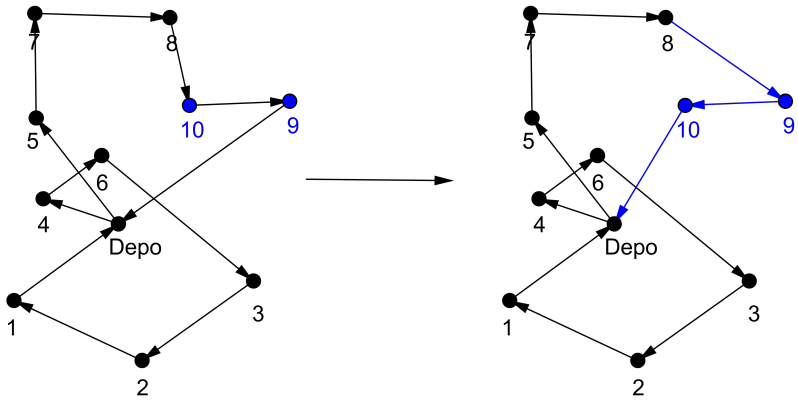


Obrázek 4.7: Operátor mutace Gene Insertion. Zákazník číslo 9 byl přesunut z jedné trasy do jiné trasy. Modře je vyznačen zaměněný zákazník a všechny části tras, které byly mutací změněny.

4.3.2. Gene Swap

Operátor Gene Swap byl pro řešení VRP použit například v [6]. Tento operátor náhodně vybere dva geny, které zamění, přičemž se dbá na to, aby nebyla porušena omezení úlohy. Zobecněním operátoru Gene Swap je Gene Sequention Swap, který zamění pořadí více genů jedince.

Na následujícím obrázku je znázorněna mutace potomka pomocí operátoru Gene Swap.



Obrázek 4.8: Operátor mutace Gene Swap. Zákazníci číslo 9 a 10 byly zaměněni. Modře jsou vyznačeni zaměnění zákazníci a všechny části tras, které byly mutací změněny.

5. Realizace genetického algoritmu pro VRP v Matlabu

Praktickou část mé diplomové práce představovala programová realizace algoritmu v Matlabu a testování vlivu hodnot parametrů na výkon genetického algoritmu. Následující kapitola uvádí a popisuje všechny funkce, které byly pro výpočty použity. Všechny uvedené m-soubory jsou dostupné na přiloženém CD.

5.1. Hlavní funkce

`ga_vrp.m` - Hlavní soubor genetického algoritmu pro VRP.

Funkce `ga_vrp.m` je jediná funkce z funkcí týkajících se genetického algoritmu, kterou uživatel musí volat.

5.1.1. Vstupní argumenty hlavní funkce

Vstupní argumenty funkce `ga_vrp.m` musí být zadány uživatelem. Následující seznam uvádí všechny argumenty a jejich možné hodnoty.

1. `path` - cesta k datovému souboru se zadaním úlohy
2. `file` - název datového souboru se zadáním úlohy
3. `max_iter` - maximální počet iterací
4. `pop_random` - velikost populace, která bude generována náhodně

5. `pop_cluster` - velikost populace, která bude generována pomocí náhodného shlukování

6. `pop_sweep` - stanoví, zda bude vygenerován jedinec pomocí sweep algoritmu

   Možné hodnoty: `true,false`

7. `selection_method` - vybere metodu selekce

   Možné hodnoty: `roulette_fitness,roulette_normfitness, roulette_rank`

8. `n_parents` - počet dvojic rodičů vybraných pro křížení

9. `cross_over_method` - metoda křížení.

   Možné hodnoty: `@uniform_order_ox,@order_ox,@partially_mapped_ox, @edge_recombination_ox,@cycle_ox,@best_route_ox,@common_edge_ox, @route_based_ox`

10. `mutation_method` - metoda mutace

    Možné hodnoty: `@gene_swap,@gene_insertion`

11. `p_mutation` - pravděpodobnost provedení mutace

12. `new_popul` - vybere metodu náhradové strategie

    Možné hodnot: `@choose_np_elitism,@choose_np_empty`

13. `exchange` - stanoví, zda bude na konci výpočtu provedena lokální optimalizace operátorem Exchange

    Možné hodnoty:`true,false`

14. `two_opt` - stanoví, zda bude na konci výpočtu provedena lokální optimalizace metodou 2-opt

    Možné hodnoty: `true,false`

15. `graph_routes` - stanoví, zda bude na konci výpočtu vykreslen graf cest

    Možné hodnoty: `true,false`

16. `graph_fitness` - stanoví, zda bude na konci výpočtu vykreslen graf s minimální, maximální a průměrnou hodnotou fitness pro každou iteraci

    Možné hodnoty: `true,false`

17. `graph_fitness_routes` - stanoví, zda bude na konci výpočtu vykreslen graf s vývojem cest a statistickými charakteristikami fitness hodnot pro všechny iterace.

    Možné hodnoty: `true,false`

5.1.2. Výstup

1. `result_solution` - nejlepší získaný jedinec
2. `min_fitness` - minimální fitness hodnota

5.2. Funkce hybridního genetického algoritmu

Funkce hybridního genetického algoritmu jsou volány automaticky z hlavní funkce `ga_vrp.m` a uživatel je přímo volat nemusí. Proto je u následujících funkcí popsán pouze jejich účel bez vstupních parametrů.

1. `create_initial_popul_random.m` - vytvoří počáteční populaci pomocí náhodných permutací
2. `create_initial_popul_cluster.m` - vytvoří počáteční populaci pomocí náhodných permutací uvnitř shluků. Shlukování je provedeno pomocí K-průměrů. Jednotliví zákazníci jsou poté primárně vybírání z jednotlivých shluků.
3. `create_initial_popul_sweep.m` - vytvoří počáteční populaci pomocí algoritmu sweep

4. `routes.m` - provede rozdělení kódovaného jedince do cest
5. `fitness_value.m` - vypočítá hodnotu fitness funkce
6. `roulette_fitness.m` - provede selekci rodičů pomocí metody ruletového kola podle fitness hodnot
7. `roulette_normfitness.m` - provede selekci rodičů pomocí metody ruletového kola podle normovaných fitness hodnot
8. `roulette_rank.m` - provede selekci rodičů pomocí metody ruletového kola podle pořadí
9. `choose_np_elitism.m` - výběr následující populace pomocí strategie elitismu
10. `choose_np_empty.m` - výběr následující populace pomocí prázdné strategie
11. `load_vehicle.m` - vypočítá náklad vozidel
12. `exchange.m` - provede lokální optimalizace pomocí Exchange
13. `two_opt.m` - provede lokální optimalizaci pomocí 2-opt

5.3. Operátory křížení

Následující funkční M-soubory představují operátory křížení. Tyto funkce jsou obdobně jako předchozí volány automaticky.

1. `uniform_order_ox.m`
2. `order_ox.m`
3. `partially_mapped_ox.m`
4. `edge_recombination_ox.m`
5. `cycle_ox.m`

6. `best_route_ox.m`

7. `common_edge_ox.m`

8. `route_based_ox.m`

5.4. Operátory mutace

Následující funkční M-soubory představují operátory mutace. Tyto funkce jsou obdobně jako předchozí volány automaticky.

1. `gene_swap.m`

2. `gene_insertion.m`

5.5. Pomocné funkce

Zbylé funkce nejsou přímo součástí genetického algoritmu, jsou ale nutné pro náš genetický algoritmus. Všechny funkce jsou volány automaticky a uživatel je sám volat nemusí.

1. `nacti_vrp.m` - načte datový soubor se zadáním úlohy

2. `distance_matrix.m` vypočítá matici vzdáleností na základě souřadnic zákazníků

3. `kmeansshluk.m` - shlukování pomocí K-Means

4. `graph_fitness.m` - vykreslí graf vývoje minimální, maximální, průměrné hodnoty a mediánu fitness funkce v průběhu výpočtu

5. `graph_routes.m` - vykreslí trasy nejlepšího získaného řešení, včetně výše poptávek, délek tras, celkové délky a pokud je známo tak i celkové délky tras nejlepšího známého řešení

6. `graph_fitness_routes.m` - vykreslí graf s vývojem cest a statistickými charakteristikami fitness hodnot pro všechny iterace

5.6. Testovací skripty

1. `test1.m` - test vlivu metod a parametrů ovlivňujících selekční tlak, tj. velikost populace, metoda selekce a náhradové strategie
2. `test2.m` - test pro metodu a pravděpodobnost mutace
3. `test3.m` - test metod lokální optimalizace
4. `Pn101k4.m` - řešení příkladu P-n101-k4.vrp

6. Numerické testování algoritmu

Cílem této kapitoly je prozkoumat chování genetických algoritmů, otestovat vliv různých variant výpočtů na jeho rychlost a kvalitu řešení a také najít vhodná nastavení parametrů genetického algoritmu tak, aby bylo dosaženo co nejlepších výsledků. Nejdříve bude testována velikost populace a vhodná kombinace metody selekce a náhradové strategie. V dalším testu bude zkoumán vliv a výkon dvou mutačních operátorů a pravděpodobností mutace. Poslední část je věnována lokální optimalizaci.

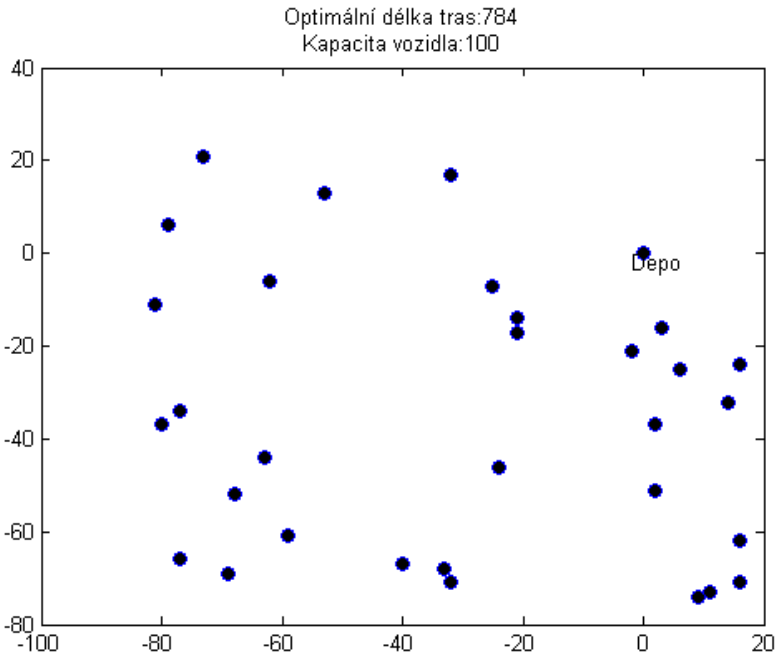
6.1. Použité příklady

Všechny testy byly provedeny pro 3 různé příklady - příklad s 31 zákazníky (A-n-32-k5), 54 zákazníky (A-n55-k9) a 79 zákazníky (A-n80-k10). Data pro příklady byla čerpány z webové stránky [14] a to konkrétně soubory Augerat et al., set A. Základní charakteristiky úloh jsou uvedeny v tabulce 6.1.

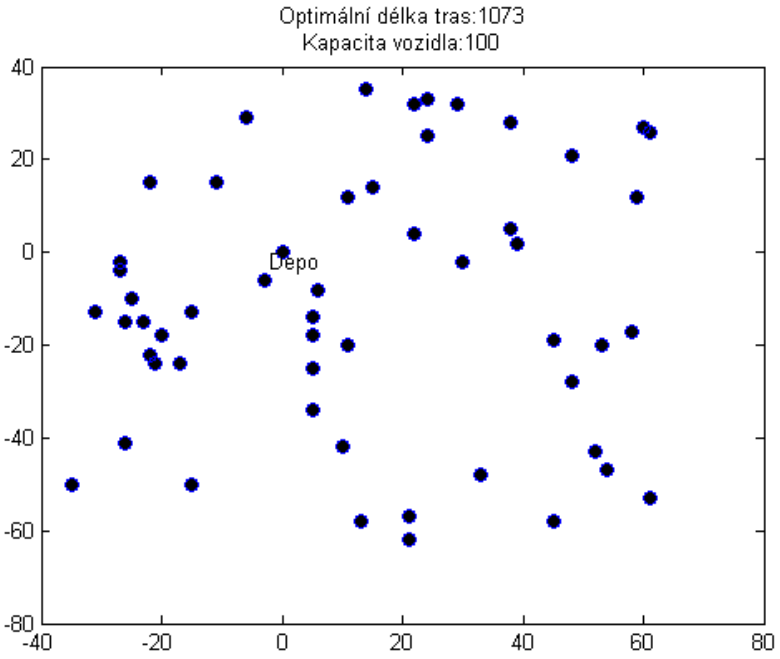
Název	**Počet zákazníků**	**Počet vozidel (min)**	**Optimální hodnota**
A-n32-k5	31	5	784
A-n55-k9	54	9	1073
A-n80-k10	79	10	1764

Tabulka 6.1: Použité příklady

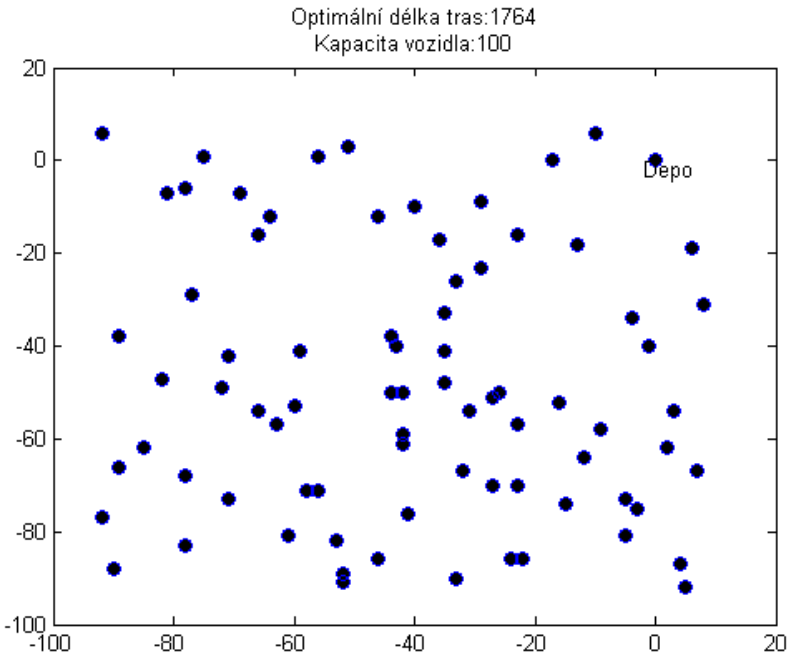
Obrázky 6.1, 6.2 a 6.3 zobrazují rozmístění zákazníků pro testované příklady. U všech příkladů bylo z důvodu zjednodušení některých výpočtů depo přesunuto na pozici $[0, 0]$.



Obrázek 6.1: Rozmístění zákazníků pro příklad A-n32-k5



Obrázek 6.2: Rozmístění zákazníků pro příklad A-n55-k9



Obrázek 6.3: Rozmístění zákazníků pro příklad A-n80-k10

6.2. Metody a parametry ovlivňující selekční tlak

6.2.1. Porovnání metod selekce

Nejdříve byly porovnány různé metody selekce pomocí ruletového kola - ruleta podle fitness hodnoty jedince, ruleta podle normované fitness hodnoty jedince a ruleta podle pořadí.

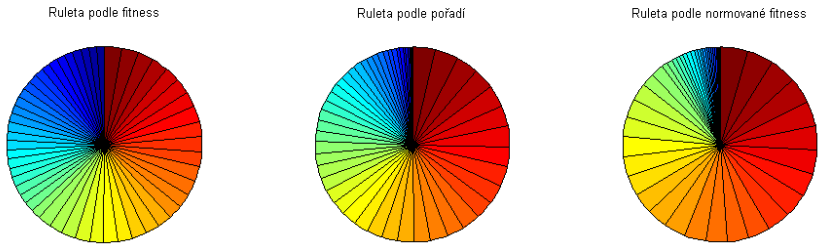
Princip je pro všechny typy ruletového kola stejný, jednotlivé metody se nicméně liší rozdělením pravděpodobností s jakou budou jedinci vybráni. Nejspravedlivější je metoda ruletového kola podle hodnoty fitness jedince. Tato metoda přiřazuje všem jedincům přibližně stejnou pravděpodobnost, nejlepší jedinci nejsou nijak zvlášť preferování a i ti nejhorší jedinci mají šanci být vybráni.

U ruletového kola podle pořadí nejsou nijak zohledněny rozdíly mezi fitness hodnotami jedinců, jelikož je jim přiřazeno pouze pořadí na základě jejich kvality.

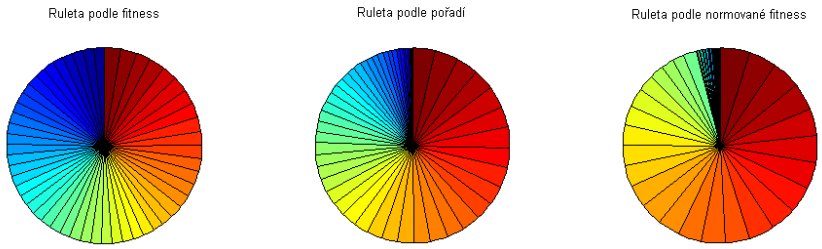
Nejkvalitnější jedinci mají o něco větší pravděpodobnost vybrání, resp. nejhorší jedinci o něco nižší než u ruletového kola podle fitness hodnoty.

Ruletové kolo podle normované fitness hodnoty je extrémním případem ruletové selekce. Nejlepší jedinci jsou silně upřednostňováni a pravděpodobnost vybrání nejlepších jedinců se v průběhu výpočtu ještě zvyšuje.

Na obrázcích 6.4 a 6.5 je ilustrováno rozdělení pravděpodobností po první a šestisté iteraci pro všechny zmíněné typy ruletového kola.



Obrázek 6.4: Rozdělení pravděpodobností ruletového kola po 1 iteraci



Obrázek 6.5: Rozdělení pravděpodobností ruletového kola po 600 iteracích

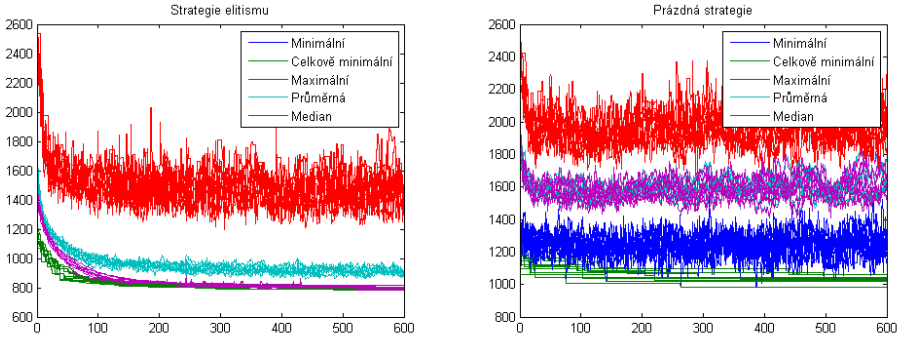
6.2.2. Porovnání náhradových strategií

V této sekci jsou porovnány dva typy náhradové strategie, strategie elitismu s počtem vybraných jedinců $N/2$, kde N je velikost populace a prázdná strategie.

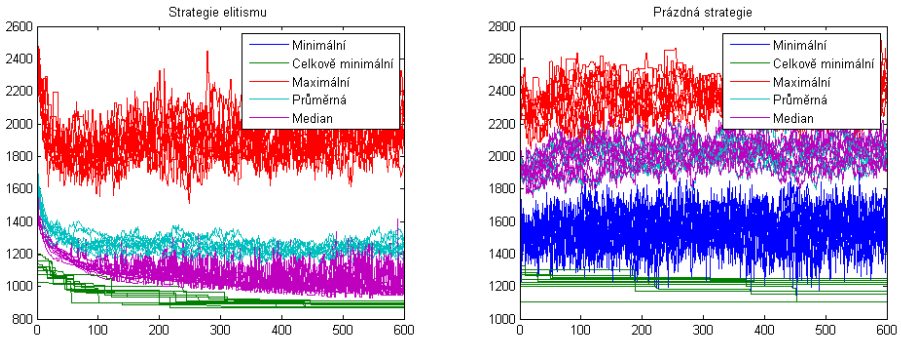
Strategie elitismu upřednostňuje kvalitnější jedince. Jelikož je uplatňován selekční tlak, populace se postupně šlechtí a její jedinci jsou čím dál tím kvalitnější. Strategie elitismu vede k lepším výsledkům než prázdná strategie, nicméně populace se stává rychleji homogenní a může dojít k předčasné konvergenci. Jelikož nejlepší jedinec musí být vždy zahrnut v populaci je hodnota fitness v poslední iteraci také tou nejmenší.

Prázdná strategie zachovává větší rozmanitost v populaci. Často nejlepší hodnota není do nové generace vybrána a proto je potřeba si zapamatovat celkovou minimální hodnotu a jí příslušné řešení.

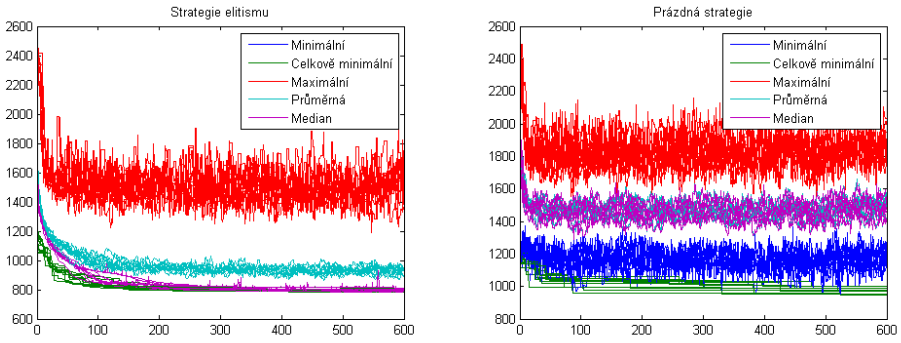
Na obrázcích 6.6, 6.7 a 6.8 je ilustrována změna fitness hodnot v průběhu iterací při deseti pokusech. Z obrázků je patrná velká míra náhodnosti při použití prázdné strategie. Výpočet při strategii elitismu se naopak chová pokaždé velmi podobně. U prázdné strategie také ke zlepšování výsledku dochází postupně, kdežto u strategie elitismu je největšího zlepšení dosaženo během prvních iterací.



Obrázek 6.6: Porovnání náhradových strategií pomocí základních statistických charakteristik fitness hodnot populace v jednotlivých iteracích pro 10 výpočtů při použití rulety podle fitness a pro strategii elitismu (vlevo) a při prázdné náhradové strategii (vpravo)



Obrázek 6.7: Porovnání náhradových strategií pomocí základních statistických charakteristik fitness hodnot populace v jednotlivých iteracích pro 10 výpočtů při použití rulety podle normované fitness a pro strategii elitismu (vlevo) a při prázdné náhradové strategii (vpravo)



Obrázek 6.8: Porovnání náhradových strategií pomocí základních statistických charakteristik fitness hodnot populace v jednotlivých iteracích pro 10 výpočtů při použití rulety podle pořadí a pro strategii elitismu (vlevo) a při prázdné náhradové strategii (vpravo)

Jednotlivé metody ruletového kola sice fungují velmi podobně, nicméně dosažené výsledky fitness hodnot se velice liší. Nejlepších výsledků dosahuje metoda ruletového kola podle fitness hodnoty nebo podle pořadí fitness hodnot v kombinaci s náhradovou strategií elitismu. Prázdná strategie se pro metodu ruletového kola absolutně nehodí, jelikož dosažené výsledky jsou mnohem horší než při použití elitismu.

V tabulce 6.2 jsou uvedeny fixní a testované hodnoty parametrů provedeného testu.

Příklad	A-n32-k5
Velikost populace	101
Typ ruletového kola	podle fitness / normované fitness / pořadí
Počet vybraných rodičů	40
Metoda křížení	RBX
Pravděpodobnost mutace	$0,2$
Metoda mutace	Gene Swap
Náhradová strategie	elitismus / prázdná
Počet iterací	600

Tabulka 6.2: Parametry testu selekce a náhradové strategie

V tabulce 6.3 jsou uvedeny výsledky provedeného testu.

Typ rulety	Náhrada	Min f_{min}	Avg f_{min}	Max f_{min}	t
Podle fitness	Elitismus	785,0	795,8	818,0	27
	Prázdná	983,0	1029,1	1062	22
Podle normované fitness	Elitismus	840,0	888,9	935,0	27
	Prázdná	1056,0	1179,5	1247,0	25
Podle pořadí	Elitismus	788,0	791,2	795,0	30
	Prázdná	993,0	994,5	1034,0	25

Tabulka 6.3: Porovnání minimální, průměrné, maximální hodnoty fitness f_{min} a výpočetního času (s) z 10 výpočtů pro testované typy rulety a náhradových strategií.

6.2.3. Test parametrů pro různé operátory křížení

V tomto testu jsem testovala vliv různých hodnot parametrů na výkon genetického algoritmu a pokusila jsem se o nalezení optimálních hodnot parametrů pro uvedená křížení. Testovanými parametry byla velikost populace, metoda selekce a náhradová strategie. Stanovená populace je vždy rozdělena tak, aby byl jeden jedinec vygenerován metodou Sweep algoritmu a poté polovina zbývajících jedinců zcela náhodně a druhá polovina pomocí náhodného shlukování. Výpočet byl proveden bez použití mutace a lokální optimalizace. Na základě výsledných fitness hodnot testování byla vybrána vždy optimální kombinace parametrů, tedy ta, která vedla k nejnižším průměrným hodnotám výsledných fitness hodnot f_{min}. Výsledky všech kombinací tohoto testu jsou dostupné na přiloženém CD.

Tabulka 6.4 uvádí testované a fixní hodnoty parametrů.

Příklad	A-n-32-k5 / A-n55-k9 / A-n80-k10
Velikost populace	51 / 101 / 201 / 401
Metoda ruletového kola	podle fitness / podle pořadí
Náhradové strategie	elitismus / prázdná
Počet vybraných rodičů	40
Počet iterací	1200

Tabulka 6.4: Parametry testu operátorů křížení

Velikost populace je pro výkon algoritmu důležitá, nicméně její vhodná velikost je ovlivněna mnoha faktory a nelze proto jednoznačně určit, jaká velikost populace je nejlepší pro řešení VRP. Z výsledků testování je zjevné, že optimální velikost populace se liší pro různá křížení, není ale příliš závislá na velikost úlohy. Pro většinu operátorů (OX, PMX, ERX, CX, BRX, RBX) se jeví výhodnější používat větší množství jedinců v populaci - 401. Jiné operátory (UOX a CEX) ale dosáhly výrazně lepších výsledků s populací menší, tj. 101 - 201 jedinců.

V předchozí sekci byly testovány různé metody selekce a náhradových strategií pro křížení RBX. Z výsledků testů křížení je zjevné, že předchozí výsledky lze aplikovat na všechny operátory křížení. Opět ruleta podle fitness a pořadí dosahují zhruba stejných výsledků, s mírně lepšími výsledky dosaženými metodou rulety podle fitness. Pouze operátory BRX a CEX dosahovaly ve dvou ze tří případů lepších výsledků s použitím rulety podle pořadí. Stejně jako v předchozím testu také náhradová strategie elitismus dosahuje mnohem lepších výsledků pro všechna křížení.

Tabulka 6.5 uvádí nalezené optimální parametry pro křížení.

Křížení	**Příklad**	**Populace**	**Ruleta**	**Náhrada**	**Min f_{min}**	**Avg f_{min}**	**t**
UOX	An32k5	201	Fitness	Elitismus	850,0	905,1	29
	An55k9	101	Fitness	Elitismus	1180,0	1343,4	40
	An80k10	51	Fitness	Elitismus	2294,0	2432,9	51
OX	An32k5	201	Fitness	Elitismus	1037,0	1081,7	29
	An55k9	401	Fitness	Elitismus	1393,0	1485,9	49
	An80k10	401	Pořadí	Elitismus	2418,0	2511,9	58
PMX	An32k5	401	Fitness	Elitismus	849,0	876,5	60
	An55k9	401	Fitness	Elitismus	1164,0	1237,9	72
	An80k10	401	Fitness	Elitismus	2123,0	2269,4	88
ERX	An32k5	201	Fitness	Elitismus	948,0	1006,7	45
	An55k9	401	Fitness	Elitismus	1375,0	1468,2	114
	An80k10	401	Fitness	Elitismus	2501,0	2545,4	157
CX	An32k5	401	Fitness	Elitismus	951,0	992,8	20
	An55k9	401	Fitness	Elitismus	1356,0	1425,9	31
	An80k10	401	Fitness	Elitismus	2437,0	2508,5	45
BRX	An32k5	201	Fitness	Elitismus	872,0	953,3	35
	An55k9	401	Pořadí	Elitismus	1176,0	1245,3	54
	An80k10	401	Pořadí	Elitismus	2125,0	2201,3	72
CEX	An32k5	401	Pořadí	Elitismus	791,0	805,0	79
	An55k9	101	Pořadí	Elitismus	1150,0	1190,9	172
	An80k10	101	Fitness	Elitismus	1971,0	2045,6	430,5
RBX	An32k5	401	Fitness	Elitismus	**789,0**	**791,6**	52
	An55k9	401	Fitness	Elitismus	**1088,0**	**1108,5**	87
	An80k10	201	Fitness	Elitismus	**1906,0**	**1950,2**	158

Tabulka 6.5: Porovnání minimální a průměrné hodnoty fitness f_{min} a průměrného výpočetního času (s) z 10 výpočtů pro nalezenou optimální kombinaci parametrů (velikost populace, typ ruletové selekce a náhradová strategie). Tučně jsou vyznačeny nejlepší dosažené hodnoty při porovnání výsledků jednotlivých křížení.

6.3. Pravděpodobnost a operátory mutace

V tomto testu jsem testovala vliv dvou metod mutace - Gene Insertion a Gene Swap a jejich pravděpodobností na výkon genetického algoritmu. Na základě výsledných fitness hodnot testování byla vybrána vždy optimální hodnota pravděpodobnosti, tedy ta, která vedla k nejnižším průměrným hodnotám výsledných fitness hodnot f_{min}.

Pro zjednoduššení výpočtů a lepší porovnání výsledků byly pro všechna křížení zvoleny stejné fixní parametry pro velikost populace, typ selekce a náhradové strategie. Tabulka 6.6 uvádí testované a fixní hodnoty parametrů.

Příklad	A-n-32-k5 / A-n55-k9 / A-n80-k10
Velikost populace	401
Typ ruletového kola	podle fitness
Počet vybraných rodičů	40
Pravděpodobnost mutace	$0,1$ / $0,2$ / $0,5$ / $1,0$
Metoda mutace	Gene Swap / Gene Insertion
Náhradová strategie	elitismus
Počet iterací	1200

Tabulka 6.6: Parametry testu operátorů mutace

Operátor Gene Swap dosahuje nejlepších výsledků při použití s nízkou pravděpodobností, tj. 0,1 - 0,2. Druhý testovaný operátor, Gene Insertion, pro změnu dosahuje lepších výsledků při použití s vyšší pravděpodobností, tj. 0,5-1,0. Z uvedených výsledků nicméně nelze jednoznačně určit, s jakou pravděpodobností by měla být mutace na potomcích prováděna.

Pro operátory křížení PMX, CX a BRX se jeví výhodnější využití metody mutace Gene Swap. Pro operátor OX, CEX a RBX zase Gene Insertion, rozdíly mezi výsledky obou metod mutací ale jsou u těchto křížení pouze minimální. U ostatních operátorů křížení (UOX, ERX) není možné vybrat metodu mutace, podle které by byly výsledky lepší pro všechny testované příklady. Také u těchto operátorů jsou výsledné minimální fitness hodnoty velmi podobné pro oba operátory mutace.

Tabulka 6.7 uvádí optimální pravděpodobnosti pro obě metody mutace, průměrnou hodnotu výsledné fitness a minimální fitness.

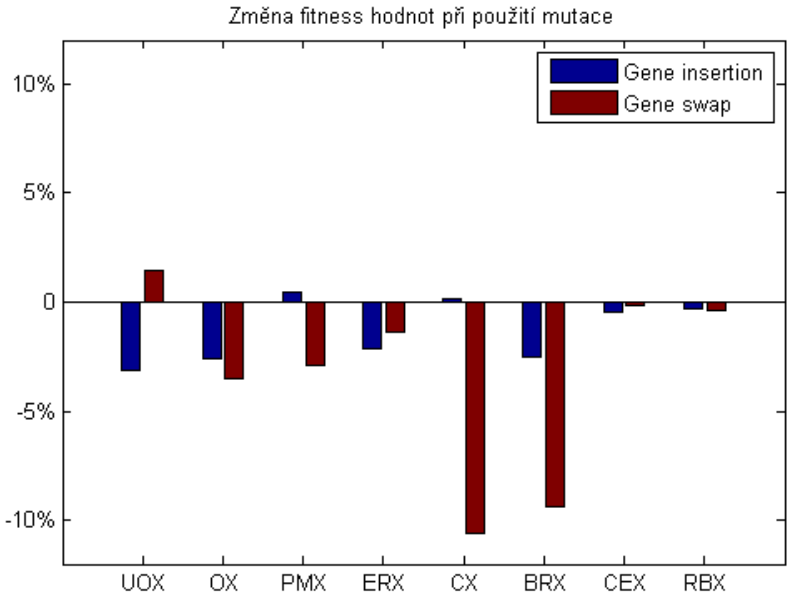
		Gene Swap			Gene Insertion		
Křížení	**Příklad**	**Pst**	**Min f_{min}**	**Avg f_{min}**	**Pst**	**Min f_{min}**	**Avg f_{min}**
UOX	An32k5	0,1	844,0	940,7	1,0	**841,0**	**898,3**
	An55k9	0,1	**1390,0**	1497,7	0,2	1402,0	**1481,9**
	An80k10	0,1	**2385,0**	**2545,1**	0,5	2482,0	2545,7
OX	An32k5	0,5	1028,0	**1061,3**	0,1	**1012,0**	1071,1
	An55k9	0,2	1420,0	1469,8	1,0	**1339,0**	**1467,5**
	An80k10	0,1	2446,0	2532,8	0,5	**2396,0**	**2525,4**
PMX	An32k5	0,5	**821,0**	**851,2**	0,5	849,0	880,8
	An55k9	0,2	**1140,0**	**1174,3**	0,2	1189,0	1269,6
	An80k10	0,2	**1971,0**	**2071,9**	1,0	2118,0	2222,7
ERX	An32k5	0,1	960,0	1001,7	1,0	**891,0**	**994,2**
	An55k9	0,1	**1404,0**	1476,1	0,1	1409,0	**1468,3**
	An80k10	0,5	**2468,0**	**2528,6**	0,1	2472,0	2533,1
CX	An32k5	0,2	**846,0**	**887,5**	0,5	957,0	994,6
	An55k9	0,1	**1235,0**	**1329,9**	0,2	1347,0	1433,9
	An80k10	0,2	**2224,0**	**2376,9**	0,5	2417,0	2471,5
BRX	An32k5	0,2	**807,0**	**837,8**	1,0	858,0	901,4
	An55k9	0,5	**1111,0**	**1154,0**	0,1	1158,0	1239,4
	An80k10	0,2	**1972,0**	**2021,7**	1,0	2135,0	2221,2
CEX	An32k5	0,1	796,0	804,0	0,5	**794,0**	**801,5**
	An55k9	0,2	**1220,0**	1297,8	0,5	1246,0	**1275,8**
	An80k10	0,2	2323,0	**2367,4**	0,5	**2264,0**	2369,1
RBX	An32k5	0,2	784,0	**788,4**	1,0	784,0	789,4
	An55k9	0,2	1097,0	1123,5	0,5	**1095,0**	**1113,4**
	An80k10	0,1	1924,0	1983,4	0,2	**1897,0**	**1952,3**

Tabulka 6.7: Porovnání minimální a průměrné hodnoty fitness f_{min} z 10 výpočtů pro nalezenou optimální pravděpodobnost obou metod mutace. Tučně je pro každý příklad a operátor křížení vyznačeno která metoda mutace vede k lepšímu výsledku.

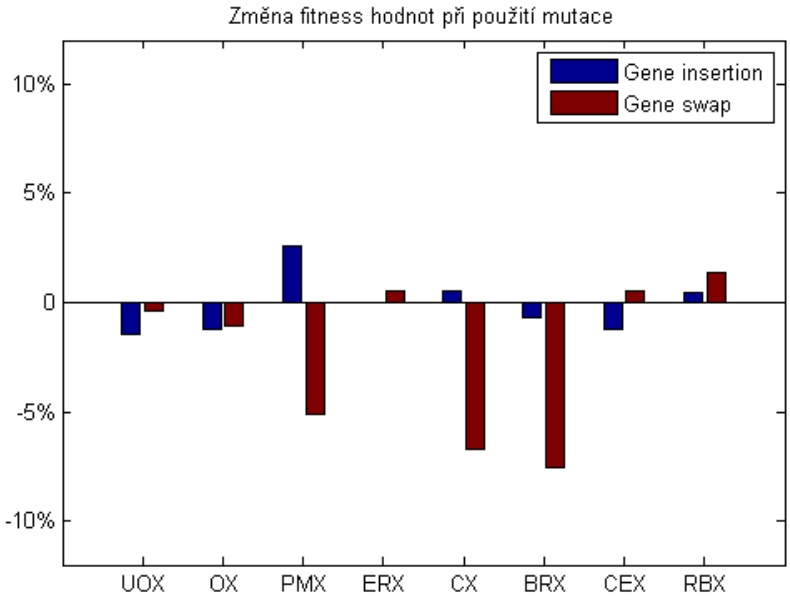
Mutace se do genetických algoritmů zařazuje proto, aby pomocí malých změn v chromosomech rozšířila oblast prohledávání a tím zaručila větší rozmanitost populace. Použití mutace ve většině případů opravdu zaručí dosažení lepšího výsledku než genetický algoritmus bez použití mutace. Nicméně z výsledků testování je zřejmé, že tomu tak nemusí být vždy. Výsledek závisí na mnoha faktorech jako je vybraný operátor křížení, pravděpodobnost mutace a ostatní parametry. Například operátory PMX a CX v kombinaci s operátorem Gene Insertion dosahuje u dvou ze tří testovaných příkladů mírně horších výsledků než bez použití

mutace. Tyto operátory ale dosahují výrazně lepších výsledků při použití mutace Gene Swap. Operátor Gene Swap v kombinaci s operátory křížení CEX a RBX zase dosáhl u dvou ze tří příkladů mírně horších výsledků než bez použití mutace.

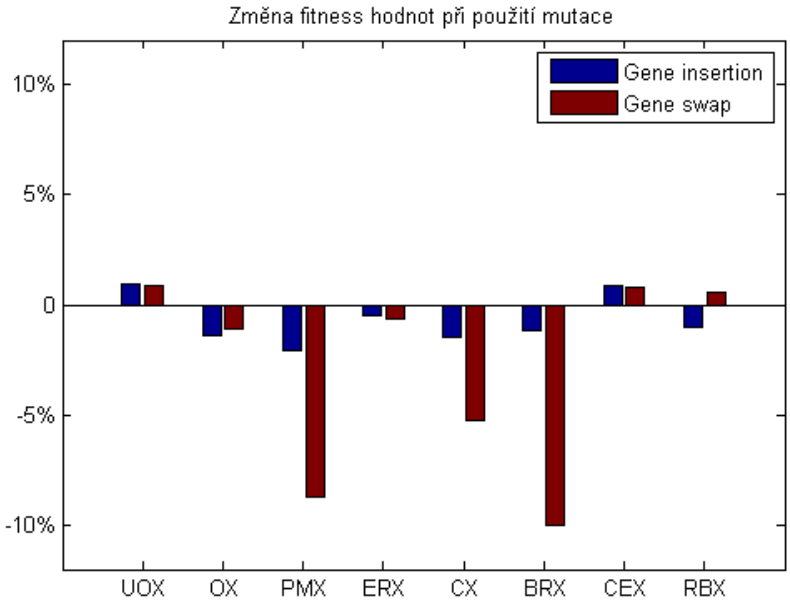
Na obrázcích 6.9, 6.10 a 6.11 je znázorněna změna výsledné průměrné minimální fitness hodnoty při použití mutace oproti výpočtu bez použití mutace.



Obrázek 6.9: A-n32-k5 - Procentuelní změna fitness hodnot při použití mutace



Obrázek 6.10: A-n55-k9 - Procentuelní změna fitness hodnot při použití mutace



Obrázek 6.11: A-n80-k10 - Procentuelní změna fitness hodnot při použití mutace

6.4. Metody lokální optimalizace

6.4.1. Porovnání metod lokální optimalizace

V předchozích testech z důvodu zkrácení výpočetního času a lepšího porovnání výsledků nebyla zahrnuta žádná lokální optimalizace. Z výsledků předchozích testů je ale zřejmé, že pro větší úlohy (více než 32 zákazníků) samotný genetický algoritmus není schopen najít optimální řešení pro ani jednu metodu křížení. Proto se do algoritmu často zahrnuje i nějaká metoda lokální optimalizace.

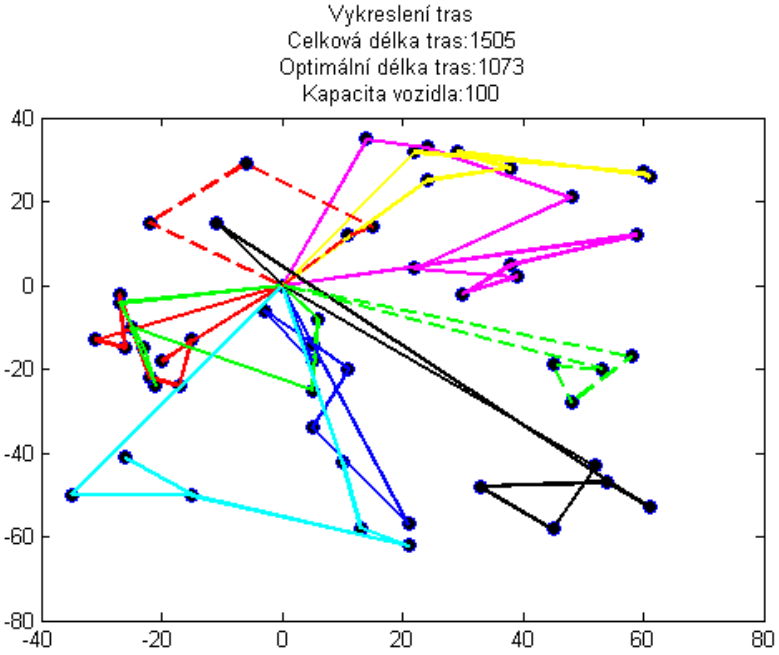
Lokální optimalizace byla použita po skončení samotného genetického algoritmu na získané nejlepší řešení. Tedy nejdříve bylo provedeno 1200 iterací genetického algoritmu a až poté aplikován algoritmus lokální optimalizace. V každé iteraci algoritmu lokální optimalizace byla provedena první možná změna, která vedla k vylepšení výsledku a největší počet těchto změn byl nastaven na 1000. Tento způsob příliš neprodlužuje výpočetní čas a zároveň je schopný dojít k dobrým výsledkům. Lokální optimalizaci je ale možné aplikovat i průběžně na získané potomky. Pro testované úlohy se výsledky získané tímto způsobem příliš neliší od výsledků získaných předchozím způsobem, dochází ale k výraznému prodloužení výpočetního času. Pro větší úlohy by nejspíše bylo vhodnější lokální optimalizaci používat průběžně na vygenerované potomky. Následující obrázky znázorňují zlepšení řešení s použitím lokální optimalizace Exchange, 2-opt nebo obou metod současně.

Tabulka 6.8 uvádí testované a fixní hodnoty parametrů pro řešení příkladu A-n55-k9.

Příklad	A-n55-k9
Velikost populace	401
Typ ruletového kola	Podle fitness
Počet vybraných rodičů	40
Metoda křížení	UOX / RBX
Pravděpodobnost mutace	0,2
Metoda mutace	Gene Swap
Náhradová strategie	Elitismus
Lokální optimalizace	2-opt / Exchange / obě
Počet iterací	1200

Tabulka 6.8: A-n55-k9 - Parametry testu metod lokální optimalizace

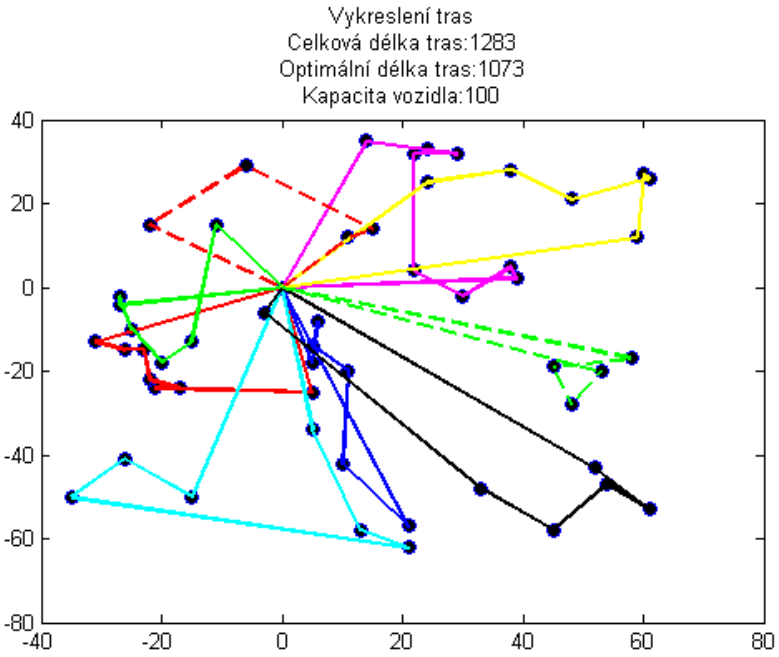
Obrázek 6.12 zobrazuje výsledné trasy získané křížením UOX a mutací Gene Swap pro příklad s 54 zákazníky před použitím lokální optimalizace.



Obrázek 6.12: A-n55-k9 - Výsledné trasy bez lokální optimalizace (UOX)

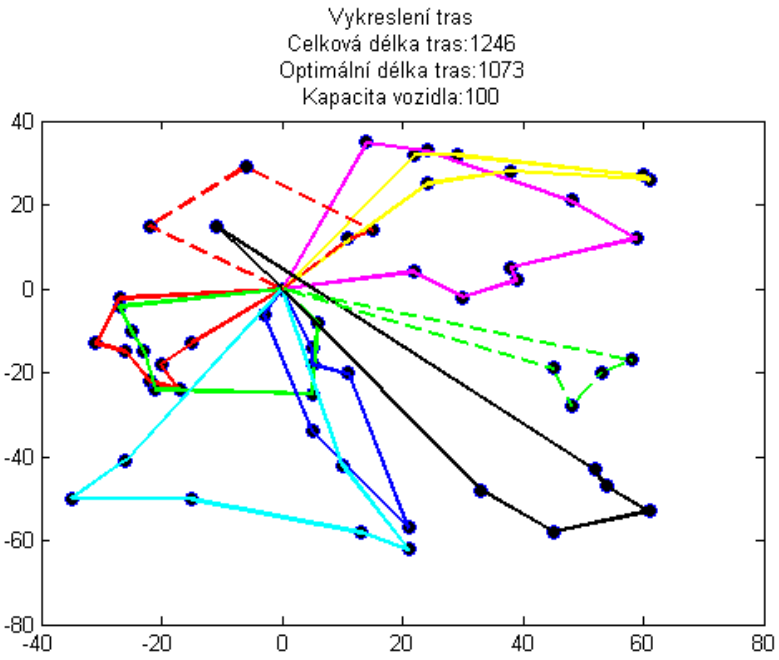
Obrázek 6.13 zobrazuje zlepšení výsledného řešení z předchozího obrázku 6.12 pokud je použit operátor Exchange. Použití operátoru vylepšilo výslednou délku tras z 1505 na 1283. Řešení ale ještě není optimální, některé trasy jsou překřížené,

jelikož Exchange nedokáže odstranit překřížení, pokud uzly následují příliš blízko po sobě .



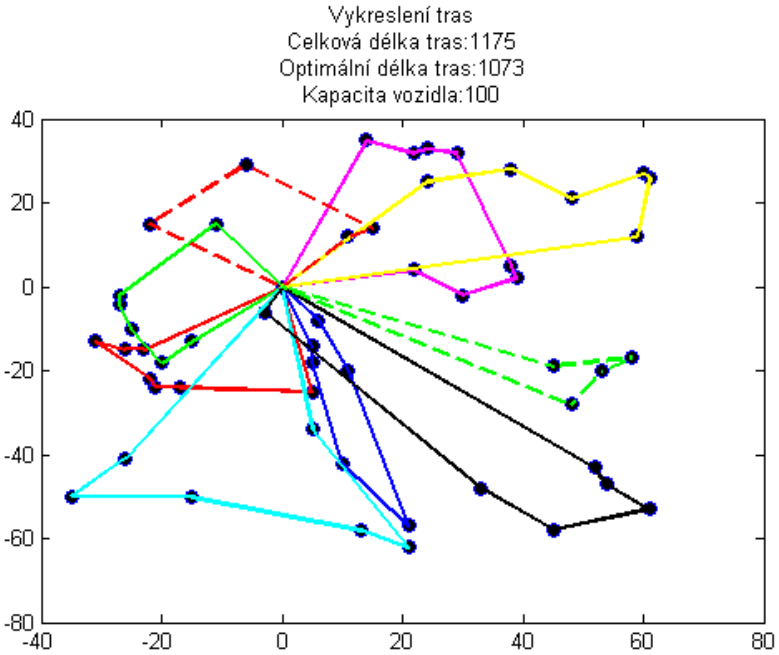
Obrázek 6.13: A-n55-k9 - Výsledné trasy s použitím operátoru Exchange (UOX)

Obrázek 6.14 zobrazuje zlepšení výsledku z obrázku 6.12 vzniklé použitím lokální optimalizace 2-opt. Výsledné řešení je mírně lepší než při použití Exchange, jelikož již neobsahuje žádné překřížené trasy. Operátor 2-opt optimalizuje trasy pouze v rámci jednoho vozidla, takže nedokáže změnit přiřazení vozidel zákazníkovi.



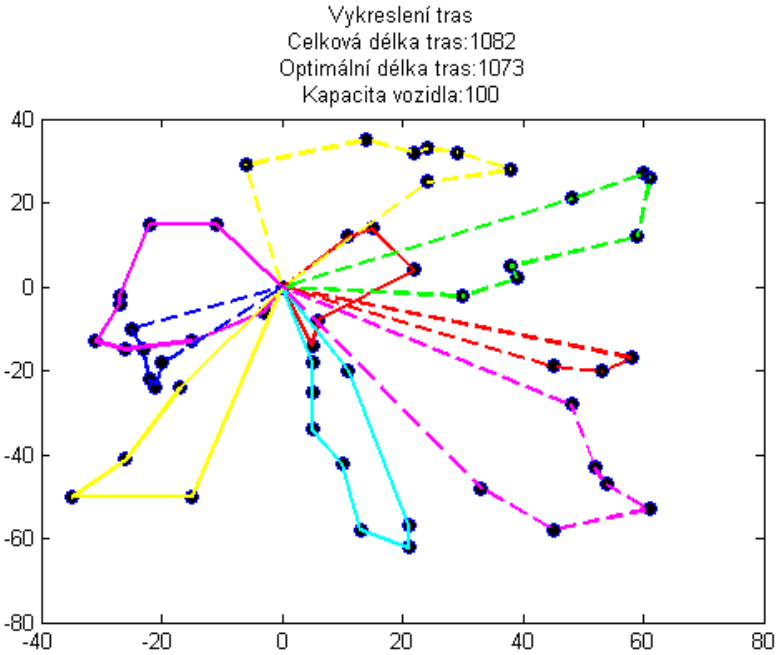
Obrázek 6.14: A-n55-k9 - Výsledné trasy s použitím operátoru 2-opt (UOX)

Jako nejlepší se jeví použít obě metody lokální optimalizace dohromady. Nejdříve aplikujeme operátor Exchange, který optimalizuje přiřazení vozidel jednotlivým zákazníkům a poté odstraníme překřížené trasy v rámci trasy jednoho vozidla pomocí operátoru 2-opt.



Obrázek 6.15: A-n55-k9 - Výsledné trasy s použitím operátorů Exchange a 2-opt (UOX)

Úspěch lokální optimalizace silně závisí na kvalitě řešení, které bylo získáno genetickým algoritmem. Řešení získané křížením UOX není ani zdaleka optimální, proto ani použití lokální optimalizace nezaručí dosažení optimálního řešení. Výsledek získaný při použití operátoru křížení RBX je mnohem kvalitnější a tedy s pomocí lokální optimalizace dosáhne téměř optima. Obrázek 6.16 znázorňuje výsledné řešení získané kombinací operátoru RBX, lokální optimalizace Exchange a 2-opt.



Obrázek 6.16: A-n55-k9 - Výsledné trasy s použitím operátorů Exchange a 2-opt(RBX)

6.4.2. Test výkonu lokální optimalizace pro různé operátory křížení

V následující části jsem se zabývala porovnáním výsledků při použití lokální optimalizace pro všechny operátory křížení. Pro zjednoduššení výpočtů a lepší porovnatelnost výsledků byly pro všechna křížení zvoleny stejné fixní parametry pro velikost populace, typ selekce, pravděpodobnost a metodu mutace a náhradové strategie. Tabulka 6.9 uvádí testované a fixní hodnoty parametrů.

Příklad	A-n-32-k5 / A-n55-k9 / A-n80-k10
Velikost populace	401
Typ ruletového kola	Podle fitness
Počet vybraných rodičů	40
Pravděpodobnost mutace	0,2
Metoda mutace	Gene Swap
Náhradová strategie	Elitismus
Lokální optimalizace	2-opt / Exchange / obě
Počet iterací	1200

Tabulka 6.9: Parametry testu metod lokální optimalizace

Z výsledků testování je zřejmé, že použití lokální optimalizace výrazně vylepší řešení získané genetickým algoritmem. Zlepšení je patrné zvláště pro operátory křížení, které zatím nedosahovaly příliš dobrých výsledků, například operátory UOX, OX a ERX. Alespoň malého zlepšení v délce výsledných tras je ale dosaženo vždy. Můžeme také říci, že čím větší úloha, tím větší zlepšení zaručí použití lokální optimalizace.

Tabulka 6.10 uvádí výsledné průměrné hodnoty minimální fitness pro obě metody lokální optimalizace a jejich kombinaci.

Křížení	Příklad	Bez		Exchange		2-opt		Obě	
		Avg f_{min}	**t**	**Avg** f_{min}	**t**	**Avg** f_{min}	**t**	**Avg** f_{min}	**t**
UOX	An32k5	980,3	35	918,7	37	896,1	35	874,1	37
	An55k9	1474,9	45	1280,8	47	1299,3	47	1209,2	46
	An80k10	2563,2	62	2123,9	64	2069,6	64	1999,4	65
OX	An32k5	1071,9	33	937,0	34	917,6	33	868,6	38
	An55k9	1481,6	45	1236,2	46	1256,8	47	1174,3	46
	An80k10	2539,4	63	2069,8	65	2088,5	63	1975,6	65
PMX	An32k5	867,8	61	848,4	81	860,4	63	854,2	71
	An55k9	1196,0	73	1187,6	75	1169,9	85	1173,5	75
	An80k10	2112,6	93	2032,1	96	2002,4	94	1950,0	96
ERX	An32k5	1004,6	55	917,8	58	918,2	55	861,0	59
	An55k9	1506,6	99	1289,4	102	1288,1	107	1191,8	104
	An80k10	2555,7	161	2104,0	162	2131,0	174	1991,0	162
CX	An32k5	902,0	20	888,5	23	884,4	22	857,1	21
	An55k9	1363,1	32	1272,8	33	1228,4	35	1167,7	33
	An80k10	2361,5	48	2090,6	49	2031,0	48	1975,3	49
BRX	An32k5	856,9	40	865,9	41	846,6	40	856,8	41
	An55k9	1167,7	57	1159,0	60	1160,2	64	1155,7	60
	An80k10	2055,9	76	1994,2	81	1970,4	81	1940,6	81
CEX	An32k5	807,3	96	805,8	98	798,7	100	798,8	99
	An55k9	1313,4	297	1240,4	299	1187,9	307	1166,9	301
	An80k10	2368,2	580	2125,7	608	2110,3	590	1997,5	641
RBX	An32k5	789,8	55	789,4	58	788,9	57	786,1	56
	An55k9	1121,4	90	1112,1	92	1103,3	94	1098,5	92
	An80k10	1999,4	165	1954,4	166	1886,4	161	1881,4	178

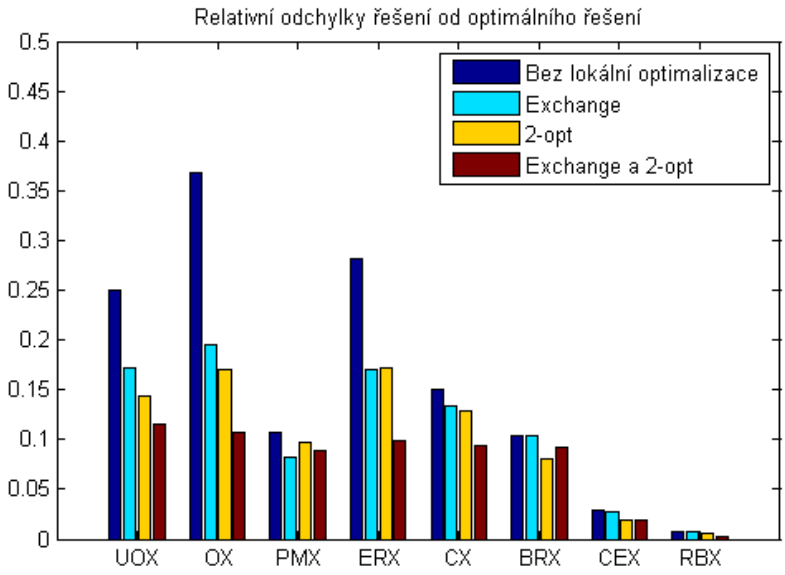
Tabulka 6.10: Porovnání průměrných hodnot f_{min} z 10 výpočtů pro varianty genetického algoritmu bez lokální optimalizace, s lokální optimalizací Exchange, 2-opt a obou současně.

Jelikož známe optimální řešení pro všechny testované příklady, můžeme snadno porovnat kvalitu získaných řešení pomocí relativní odchylky od optimální hodnoty. Relativní odchylku ρ od optimálního řešení vypočteme pomocí následujícího vzorečku:

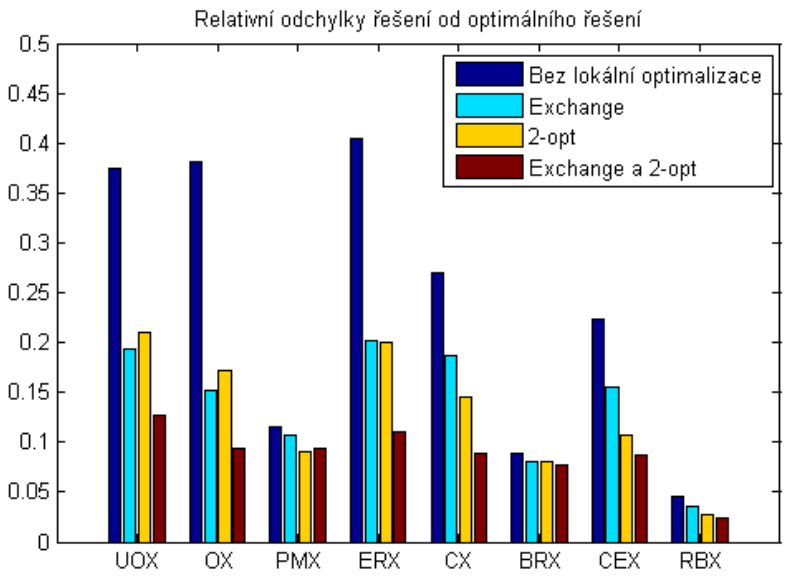
$$\rho = \frac{f_{min} - f_{opt}}{f_{opt}}$$

kde f_{min} značí minimální délku tras (v tomto případě průměrnou minimální délku tras) a f_{opt} optimální délku tras.

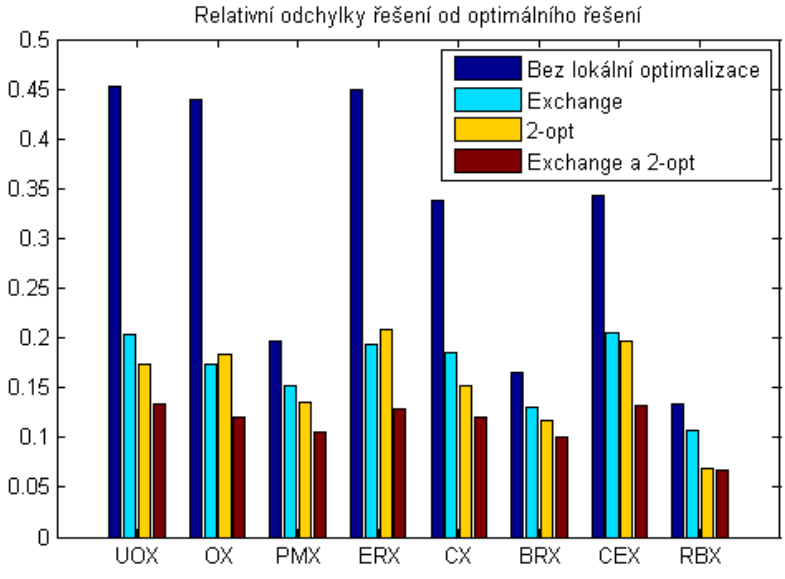
Obrázky 6.17, 6.18 a 6.19 znázorňují relativní odchylky od optimálního řešení při výpočtu bez lokální optimalizace, s použitím lokální optimalizace Exchange nebo 2-opt a při použití obou metod současně.



Obrázek 6.17: A-n32-k5 - Relativní odchylka průměrného řešení od optimálního řešení



Obrázek 6.18: A-n55-k9 - Relativní odchylka průměrného řešení od optimálního řešení



Obrázek 6.19: A-n55-k9 - Relativní odchylka průměrného řešení od optimálního řešení

6.5. Zhodnocení provedených testů

6.5.1. Zhodnocení testů parametrů

V této kapitole byly uvedeny výsledky testů a chování genetického algoritmu při různých hodnotách parametrů a vlivu parametrů na různé operátory křížení. Konkrétně byla testována velikost populace, metoda selekce, náhradová strategie, pravděpodobnost mutace, metoda mutace a metody lokální optimalizace.

Jako první byly testovány různé kombinace velikosti populace, metody selekce a náhradové strategie. Z výsledků testů se ukázalo, že pro rozvozní problém je nejvhodnější metoda rulety podle selekce eventuelně pořadová metoda v kombinaci se strategií elitismu. Tato kombinace dosáhla výrazně lepších výsledků pro všechny operátory křížení. Optimální velikost populace se z naměřených výsledků nedá jednoznačně určit, pro většinu příkladů a křížení ale vycházely nejlépe větší populace, tj. 200-400 jedinců. Menší populaci je ale vhodné zvolit pro operátory křížení UOX nebo CEX.

Dále byly pro všechny operátory křížení testovány dva operátory mutace - Gene Swap a Gene Insertion a pravděpodobnost mutace potomků. Z výsledků testování je zřejmé, že různá křížení pracují lépe s různými metodami mutace. Například křížení PMX, CX a BRX v kombinaci s mutací Gene Swap dosahuje výrazně lepších výsledků. Tento operátor mutace je vhodný aplikovat s menší pravděpodobností, tj. 0,1 - 0,2. Na druhou stranu křížení OX a RBX jsou výkonnější v kombinaci s mutací Gene Insertion. Operátor Gene Insertion je vhodnější aplikovat s větší pravděpodobností, tj. 0,5 - 1,0. Přestože mutace by měla zaručit dosažení lepších výsledků, v případě nevhodné kombinace křížení, pravěpodobností mutace a její metody může použití mutace výsledná řešení dokonce zhoršit. Je tedy potřeba dbát na vhodné zvolení parametrů a otestovat jejich různé kombinace.

Jako poslední byla otestována lokální optimalizace výsledného řešení. Lokální optimalizace se ukázala nezbytnou pro dosažení výsledků blízkých k optimu pro úlohy větší než 32 zákazníků. Testovány byly dvě metody lokální optimalizace - Exchange a 2-opt a také jejich současné použití. Lokální optimalizace 2-opt dosahovala mírně lepších výsledků než metoda Exchange. Nejlepších výsledků ale bylo dosaženo pokud byly obě metody použity současně a to nejdříve operátor Exchange a poté na řešení znovu aplikována metoda 2-opt. Výkon lokální optimalizace je závislý na kvalitě vstupního řešení, čím lepší je původní řešení, tím lepšího výsledků je dosaženo po aplikaci lokální optimalizace.

6.5.2. Porovnání výkonu operátorů křížení

Ve všech předchozích testech jsme se zaměřili pouze na průměrné hodnoty minimálních fitness hodnot, jelikož průměrná řešení lépe charakterizují výsledky testů, které obsahují náhodné vlivy. V následující sekci jsou porovnány nejlepší dosažené hodnoty pro všechny provedné testy.

Pro zjednodušení porovnání výsledků byly pro všechna křížení zvoleny stejné parametry. Tabulka 6.11 uvádí hodnoty fixních parametrů.

Příklad	A-n-32-k5 / A-n55-k9 / A-n80-k10
Velikost populace	401
Typ ruletového kola	Podle fitness
Počet vybraných rodičů	40
Pravděpodobnost mutace	0,2
Metoda mutace	Gene Swap
Náhradová strategie	Elitismus
Lokální optimalizace	Exchange a 2-opt
Počet iterací	1200

Tabulka 6.11: Optimální hodnoty parametrů

Prvním z testovaných příkladů je příklad s 31 zákazníky (A-n32-k5). Optimální délka tras je dle [14] 784. Optimálního výsledku dosáhlo pouze křížení

RBX a to i bez použití lokální optimalizace. Velmi dobrých výsledků dosáhlo také křížení CEX a PMX. Tabulka 6.12 uvádí nejnižší dosažené hodnoty pro všechna křížení a relativní odchylky od optimálního řešení.

	Křížení		**Mutace**		**Lok. opt.**	
Křížení	f_{min}	ρ	f_{min}	ρ	f_{min}	ρ
UOX	867	0,11	858	0,09	840	0,07
OX	1057	0,35	1017	0,30	832	0,06
PMX	849	0,08	794	0,01	807	0,03
ERX	984	0,26	1004	0,28	816	0,04
CX	951	0,21	846	0,08	832	0,06
BRX	855	0,09	807	0,03	818	0,04
CEX	790	0,01	796	0,02	791	0,01
RBX	789	0,01	784	0,00	784	0,00

Tabulka 6.12: A-n32-k5 - Porovnání minimálních dosažených fitness hodnot. V prvním sloupci jsou uvedeny minimální hodnoty a relativní odchylky od optimálního řešení při použití pouze křížení, ve druhém sloupci výsledky při aplikaci křížení a mutace a ve třetím sloupci výsledky při aplikaci křížení, mutace i lokální optimalizace.

Dalším testovaným příkladem byl příklad s 54 zákazníky (A-n55-k9). Optimální délka tras je dle [14] 1073. Nejblíže se k této hodnotě opět přiblížilo křížení RBX s relativní odchylkou od optima pouze 0,01 po aplikaci lokální optimalizace. Velmi dobrých výsledků dosáhlo také křížení CX a BRX. Tabulka 6.13 uvádí nejnižší dosažené hodnoty pro všechna křížení a relativní odchylky od optimálního řešení.

	Křížení		**Mutace**		**Lok. opt.**	
Křížení	f_{min}	ρ	f_{min}	ρ	f_{min}	ρ
UOX	1432	0,33	1443	0,34	1184	0,10
OX	1393	0,30	1420	0,32	1138	0,06
PMX	1164	0,08	1140	0,06	1132	0,05
ERX	1375	0,28	1462	0,36	1134	0,06
CX	1356	0,26	1240	0,16	1119	0,04
BRX	1193	0,11	1130	0,05	1121	0,04
CEX	1267	0,18	1220	0,14	1127	0,05
RBX	1088	0,01	1097	0,02	1083	0,01

Tabulka 6.13: A-n55-k9 - Porovnání minimálních dosažených fitness hodnot. V prvním sloupci jsou uvedeny minimální hodnoty a relativní odchylky od optimálního řešení při použití pouze křížení, ve druhém sloupci výsledky při aplikaci křížení a mutace a ve třetím sloupci výsledky při aplikaci křížení, mutace i lokální optimalizace.

Posledním testovaným příkladem byl příklad se 79 zákazníky (A-n80-k10). Optimální délka tras je dle [14] 1764. Nejlepších výsledků dosáhlo opět křížení RBX s relativní odchylkou od optima 0,05. Velmi dobrého výsledku dosáhla také křížení ERX a BRX. Tabulka 6.14 uvádí nejnižší dosažené hodnoty pro všechna křížení a relativní odchylku od optimálního řešení.

	Křížení		**Mutace**		**Lok. opt.**	
Křížení	f_{min}	ρ	f_{min}	ρ	f_{min}	ρ
UOX	2469	0,40	2486	0,41	1923	0,09
OX	2497	0,42	2458	0,39	1911	0,08
PMX	2123	0,20	1971	0,12	1900	0,08
ERX	2501	0,42	2461	0,40	1872	0,06
CX	2437	0,38	2224	0,26	1914	0,09
BRX	2160	0,22	1972	0,12	1884	0,07
CEX	2298	0,30	2323	0,32	1936	0,10
RBX	1926	0,09	1942	0,10	1851	0,05

Tabulka 6.14: A-n80-k10 - Porovnání minimálních dosažených fitness hodnot. V prvním sloupci jsou uvedeny minimální hodnoty a relativní odchylky od optimálního řešení při použití pouze křížení, ve druhém sloupci výsledky při aplikaci křížení a mutace a ve třetím sloupci výsledky při aplikaci křížení, mutace i lokální optimalizace.

Cílem této kapitoly bylo nejen nalézt vhodné hodnoty parametrů genetického

algoritmu, ale také porovnat různé operátory křížení. Operátory křížení byly v teoretické části práce rozděleny do dvou skupin, podle toho zda v průběhu křížení a tvorby potomků pracují s depy (operátory křížení uzpůsobené pro VRP) nebo je vytvořené potomky potřeba rozdělit do tras (operátory křížení převzaté z TSP). Operátory křížení zvláštně navržené pro VRP obecně dosahují mnohem lepších výsledků, jelikož nedochází ke ztrátě informace o pozici depa a jeho vzdálenosti k zákazníkům. Nejlepších výsledků dosahuje pro všechny testované příklady operátor RBX. Z druhé kategorie operátorů se jako nejvhodnější pro VRP jeví operátor PMX. Ostatní operátory nejsou schopny dosáhnout konkurenceschopného řešení bez použití lokální optimalizace a i s lokální optimalizací výsledné řešení není příliš blízko tomu optimálnímu.

Operátory, které jsou založeny na vyhledávání společných částí obou rodičů (PMX, CEX) a jejich okopírování do potomka dosahují ve většině případů mnohem lepších výsledků než pokud jsou části rodičů kopírovány náhodně.

Jelikož byly vždy zvoleny takové parametry, které se jevily jako nejvhodnější pro většinu operátorů křížení, výkon některých operátorů křížení by jistě mohl být vylepšen, pokud by byly zvoleny hodnoty parametrů jiné. Tyto testy by ale byly příliš časově náročné a v této práci na tyto výpočty nebylo místo.

7. Řešený příklad se 100 zákazníky

V této kapitole je popsáno řešení příkladu se 100 zákazníky (P-n101-k4). Nejlepší dosažená hodnota této úlohy je 681 [14]. Pro řešení úlohy byl zvolen operátor křížení RBX, jelikož dosahoval nejlepších výsledků, spolu s vypozorovanými optimálními parametry.

Pro řešení byly použity hodnoty parametrů uvedené v tabulce 7.1.

Příklad	P-n101-k4.vrp
Velikost populace	401
Počet vybraných rodičů	40
Metoda křížení	RBX
Pravděpodobnost mutace	0,2
Metoda mutace	Gene Swap
Počet iterací	1200

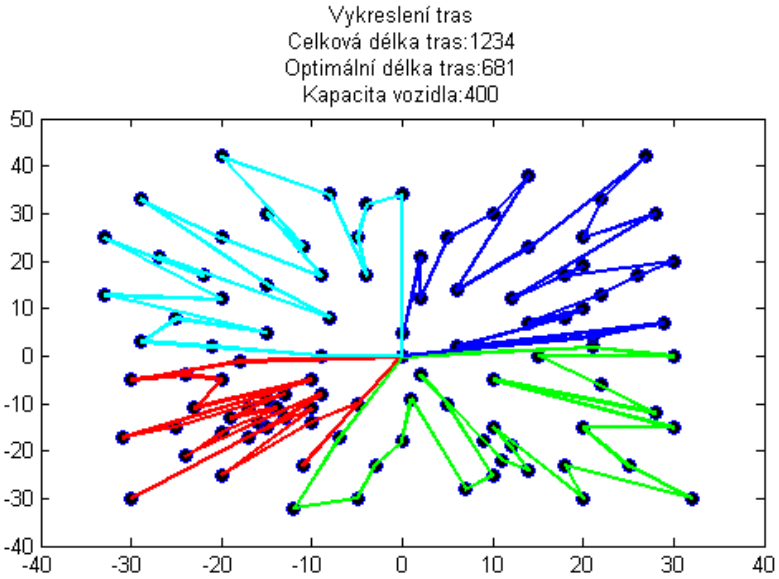
Tabulka 7.1: Parametry pro řešení příkladu P-n101-k4

Tabulka 7.2 uvádí dosažené hodnoty fitness, relativní odchylku od optimální hodnoty ρ a dobu výpočtu. Výpočet byl spuštěn desetkrát.

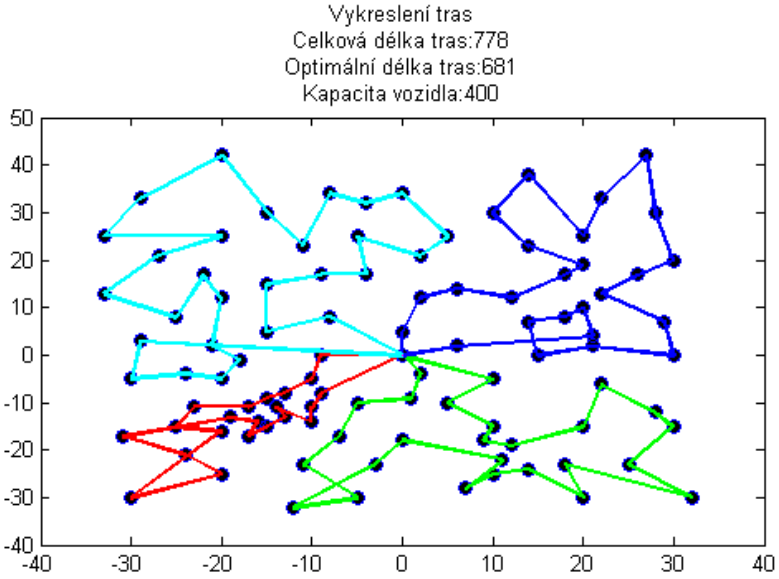
n	f_{min} **bez opt**	ρ	f_{min} **s opt**	ρ	**t**
1	805	0,18	719	0,06	231
2	771	0,13	768	0,13	231
3	789	0,16	753	0,11	295
4	791	0,16	769	0,13	275
5	799	0,17	742	0,09	268
6	794	0,17	725	0,06	309
7	776	0,14	729	0,07	259
8	778	0,14	709	0,04	272
9	751	0,10	746	0,10	262
10	775	0,14	730	0.07	321
Avg	782,9	0,15	739,0	0,09	272,3
σ	15,1	0,02	19,2	0,03	28,2

Tabulka 7.2: P-n101-k4 - Výsledné fitness hodnoty f_{min} a odchylky od optimálního řešení pro 10 výpočtů bez a s použitím lokální optimalizace.

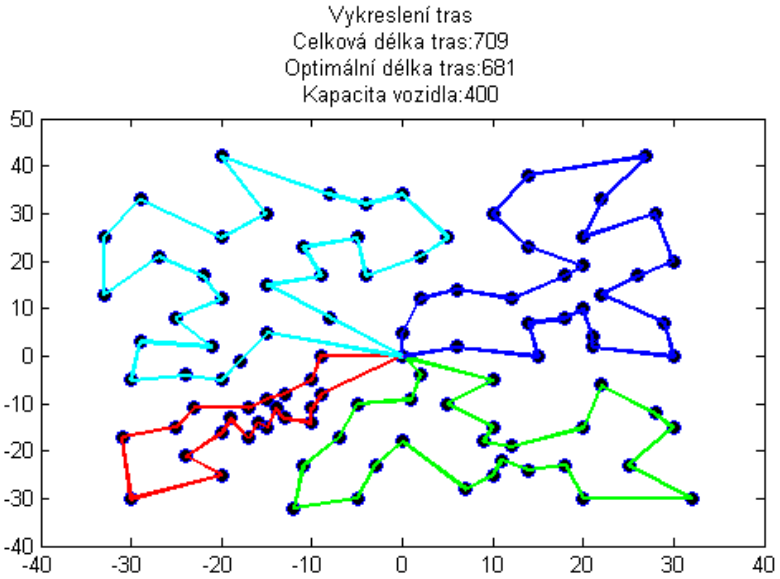
Nejlepší řešení bylo dosaženo během osmého výpočtu s výslednou fitness hodnotou 709. Po první iteraci nejlepší dosažené řešení mělo délku 1234. Po 1200 iteracích algoritmus došel k výsledku 778. Na toto řešení byly poté aplikovány zlepšovací operátory Exchange a 2-opt, které fitness hodnotu snížily na 709. Obrázky 7.1, 7.2 a 7.3 znázorňují postup výpočtu od první iterace až po výsledek získaný lokální optimalizací.



Obrázek 7.1: P-n101-k4 - Řešení po první iteraci

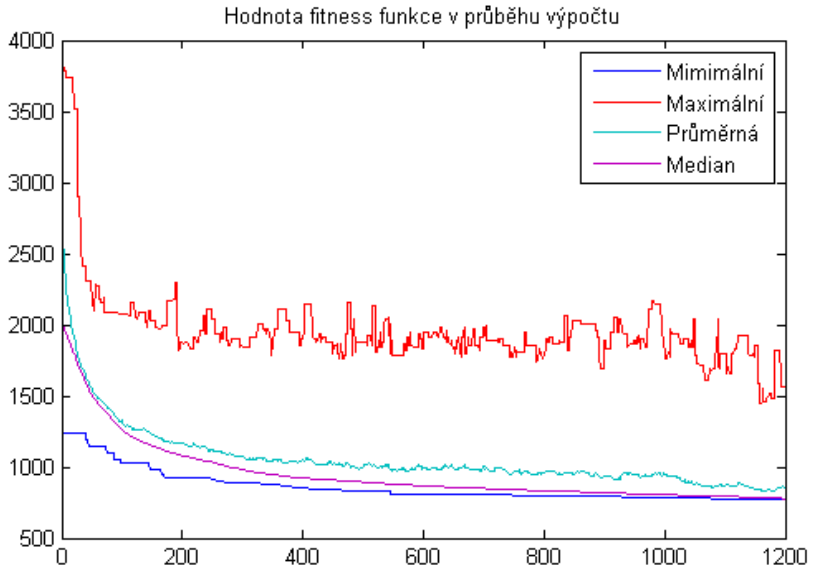


Obrázek 7.2: P-n101-k4 - Výsledné řešení bez lokální optimalizace



Obrázek 7.3: P-n101-k4 - Výsledné řešení s použitím lokální optimalizace

Obrázek 7.4 znázorňuje vývoj fitness hodnot jedinců v průběhu výpočtu. Je zřejmé, že hodnota fitness klesá nejvíce v prvních iteracích a po cca 500 iteracích se mění jen velmi pomalu.



Obrázek 7.4: P-n101-k4 - Vývoj základních statistických charakteristik fitness hodnot populace v jednotlivých iteracích.

Závěr

V teoretické části práce byla definována rozvozní úloha a stručný popis genetického algoritmu pro tuto úlohu. Dále byly podrobněji rozepsány všechny části výpočtu a operátory křížení a mutace vhodné pro VRP.

V praktické části byly otestovány všechny popsané fáze algoritmu, tj. populace a její velikost, metoda selekce a náhradové strategie a výkon různých kombinací těchto parametrů a metod. Dalším krokem bylo otestovat metody mutace a pravděpodobnosti mutace potomků. Jako poslední byly otestovány metody lokální optimalizace a jejich vliv na vylepšení řešení získaného pomocí genetického algoritmu. Pro všechny testy bylo uvedeno, které kombinace a metody dosahují nejlepších výsledků a mohou tedy být brány jako doporučení pro stanovení parametrů genetického algoritmu.

CD příloha

Na přiloženém CD jsou obsaženy všechny funkce vytvořené v Matlabu a použité pro testování parametrů. Podrobný popis funkcí a jejich parametrů byl uveden v kapitole 5.

Druhou část CD přílohy tvoří soubor `GA_VRP_vysledky.xls` ve kterém jsou uvedeny všechny naměřené hodnoty získané testováním algoritmu.

Literatura

[1] Paolo Toth, Daniele Vigo : *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, SIAM and the Mathematical Optimization Society, USA, 2014.

[2] Gilbert Laporte : *The Vehicle Routing Problem: An overview of exact and approximate algorithms.*, European Journal of Operational Research 59, 1992.

[3] Francisco B. Pereira, Jorge Tavares, Penousal Machado, Ernesto Costa: *GVR: a New Genetic Representation for the Vehocle Routing Problem.*

[4] Habibeh Nazif, Lai Soon Lee: *Optimised crossover genetic algorithm for capacitated vehicle routing problem.* Applied Mathematical Modelling 36, strana 2110–2117, 2012.

[5] Krunoslav Puljić, Robert Manger: *Comparison of Eight Evolutionary Crossover Operators for the Vehicle Routing Problem.* Mathematical Communications 18, strana 359-375, 2013.

[6] Graglia P., Stark N., Salto C., Alfonso H.: *A Comparison of Recombination Operators for Capacitate Vehicle Routing Problem.*

[7] Gintaras Vaira, Olga Kuraskova: *Genetic Algorithms and VRP: The Behaviour of a Crossover Operator.* Baltic J. Modern Computing, Vol. 1, strana 161-185, 2013.

[8] Jean-Yves Potvin, Samy Bengio: *The Vehicle Routing Problem with Time Windows - Part II: Genetic Search.* INFORMS Journal on Computing, strana 165-172, 1996.

[9] Jean Berger, Mohamed Barkaoui: *A Hybrid Genetic Algorithm for the Capacited Vehicle Routing Problem.*

[10] B. M. Ombuki, M. Nakamura, M. Osamu: *A Hybrid Search Based on Genetic Algorithms and Tabu Search for Vehicle Routing.* 6th IASTED Intl. Conf. On Artificial Intelligence and Soft Computing, strana 176-181, 2002.

[11] Qingfang Ruan, Qi Ruo, Kevin Woghiren, Lixin Miao: *A Hybrid Genetic Algorithm for the Vehicle Routing Problem with Three-Dimensional Loading Constraints.*

[12] David E. Goldberg, Kalyanmoy Deb: *A Comparative Analysis of Selection Schemes Used in Genetic Algorithms.*

[13] Olli Bräysy, Michel Gendreau: *Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms.*

[14] Networking and Emerging Optimization [online]. [cit. 2016-03-01]. Dostupné z: http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/.