VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

POKROČILÁ EVOLUČNÍ OPTIMALIZACE ÚLOH TYPU TSP

ADVANCED EVOLUTIONARY OPTIMISATION OF TSP-BASED PROBLEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. VADYM HLADYUK**
AUTHOR

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

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

144022

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Hladyuk Vadym, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Bioinformatika a biocomputing
Název:	**Pokročilá evoluční optimalizace úloh typu TSP**
Kategorie:	Algoritmy a datové struktury
Akademický rok:	2022/23

Zadání:

1. Seznamte se s problematikou úloh typu Travelling Salesman Problem (TSP) a možnostmi jejich řešení pomocí evolučních a jiných přírodou inspirovaných algoritmů. Zvolte jednu z variant TSP, jejíž optimalizace bude předmětem řešení této práce.
2. Pro vybranou variantu TSP prostudujte existující možnosti reprezentace a optimalizace. Zaměřte se na metody používané pro netriviální velikosti instancí TSP.
3. Zvolte vhodný algoritmus pro optimalizaci řešení TSP z předchozího bodu. Tento algoritmus realizujte ve vhodném prostředí s důrazem na dosažení co nejvyšší výkonnosti.
4. Proveďte sadu experimentů a analyzujte jejich průběh a výsledky za účelem identifikace částí systému vhodných k modifikaci s potenciálem vylepšení existujícího přístupu.
5. Navrhněte a implementujte modifikace z bodu 4 a proveďte pokročilou sadu experimentů s cílem dosažení efektivních řešení pro komplexní instance TSP.
6. Zhodnoťte dosažené výsledky a diskutujte přínosy a možnosti rozšíření vašeho řešení.

Literatura:

- Dle pokynů vedoucího projektu.

Při obhajobě semestrální části projektu je požadováno:
Splnění bodů 1 a 2 zadání, demonstrace prototypu algoritmu z bodu 3.

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

Vedoucí práce:	**Bidlo Michal, Ing., Ph.D.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2022
Termín pro odevzdání:	17.5.2023
Datum schválení:	31.10.2022

Abstrakt

Práce řeší problém obchodního cestujícího pomocí evolučního algoritmu, konktrétně pomocí genetického algoritmu. Jedná se o hybrid genetického algoritmu s využitím lokálního prohledávacího algoritmu a dalších vylepšení, které nám pomohou vylepšit výsledky. Problémy obchodního cestujícího budou řešeny od 20 měst až po 25 tisíc měst. V kapitole s experimenty jsem zjistil nejvhodnější nastavení všech parametrů v programu a řádně otestoval jejich přínos. V další části kapitoly s experimenty jsem zjistil jakých výsledků dosahují genetické algoritmy. V poslední části jsem porovnal vývoj hodnoty fitness různých variant genetických algoritmů a různých variant operátorů křížení, také jsem porovnal časovou náročnost. Navrhnul jsem další možná vylepšení ať už lokálních prohledávacích algoritmů či jiného přístupu k řešení TSP.

Abstract

This paper solves the traveling salesman problem using an evolutionary algorithm, specifically a genetic algorithm. It is a hybrid of the genetic algorithm, using a local search algorithm and other enhancements that further improve the results obtained. The traveling salesman problem will be solved from 20 cities to 25,000 cities. In the experiments chapter, I have determined the best settings for all the parameters in the program and properly tested their appropriateness. In the next part of the experiments chapter, I found out the performance of the full version of the genetic algorithm and its variants. In the last section, I compared the evolution of fitness values of different variants of genetic algorithms and different variants of crossover operators, I also compared the time consumption. I suggested further possible improvements either to the local search algorithm or to another approach to solve the TSP.

Klíčová slova

evoluční algoritmy, genetický algoritmus, selekce, turnajový výběr, operátory křížení, problém obchodního cestujícího, TSP, optimalizační algoritmus, 2-opt, 3-opt, hybrid genetického problému, experimenty optimalizačního problému

Keywords

evolutionary algorithms, genetic algorithm, selection, tournament selection, crossover operators, traveling salesman problem, TSP, optimization algorithm, 2-opt, 3-opt, hybrid genetic problem, optimization problem experiments

Citace

HLADYUK, Vadym. *Pokročilá evoluční optimalizace úloh typu TSP*. Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Bidlo, Ph.D.

Pokročilá evoluční optimalizace úloh typu TSP

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Michala Bidla PhD. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

....
Vadym Hladyuk
15. května 2023

Poděkování

Chtěl bych vyslovit své upřímné poděkování Ing. Michalovi Bidlovi Ph.D za jeho cenné rady a podporu při psaní této diplomové práce. Dále bych chtěl vyjádřit svou vděčnost rodině a mé přítelkyni Alžbětě za jejich nekonečnou trpělivost, podporu a motivaci, kterou mi poskytli v průběhu celého procesu psaní této práce a patří jim můj nejupřímnější dík.

Obsah

1 Úvod **5**

2 Problém obchodního cestujícího **6**
2.1 Definice TSP 6
2.1.1 Formální definice 6
2.2 Problémy optimalizace TSP 7
2.3 Existující reprezentace řešení TSP 8
2.4 Využití řešení TSP 9

3 Evoluční algoritmy **11**
3.1 Genetický algoritmus 11
3.1.1 Vlastnosti prvků genetického algoritmu 13

4 Přehled vybraných technik optimalizace TSP **20**

5 Nové metody generování řešení TSP **23**
5.1 Redukovaná matice nejbližších měst 23
5.2 Návrhy vylepšení operátorů křížení 24
5.3 Návrh hybridního genetického algoritmu 25

6 Experimentální výsledky **27**
6.1 Experimenty pro nastavení parametrů genetického algoritmu 28
6.2 Úplné výsledky experimentů 35
6.3 Vývoj hodnot fitness funkcí a časové náročnosti 38

7 Závěr **43**

Literatura **44**

A Struktura odevzdaného adresáře **47**

B Návod na spuštění programu **48**

Seznam obrázků

2.1 Příklad cesty obchodního cestujícího vybranými německými městy. [1] 7
2.2 Binární matice cest jistého řešení TSP, kde $C(ij) = 1$ znamená, že cestující prošel z města i do města j kde $C(ij) = 0$ znamená, že přechod z města i do j nebyl v tomto řešení využit. 8
2.3 Binární matice pořadí, kde $C(ij) = 1$, znamená to, že město i je v pořadí v daném řešení na j pozici a kde $C(ij) = 0$ znamená, že přechod z města i do j nebyl v tomto řešení využit. 9
2.4 Řešení TSP je zakódováno pomocí zapsaných měst v určitém pořadí. 9

3.1 Princip evoluce, kdy jeden jedinec (genotyp) je vyřazen pomocí selekce podle hodnoty fitness funkce a druhý jedinec se dostane k reprodukci, kdy je následně možnost nějaké mutace předtím, než vznikne nová populace. [4] 13
3.2 Snímek znázorňující turnajový výběr při selekci jedince, který se bude reprodukovat. 14
3.3 Snímek znázorňující váženou ruletu, kdy hodnota fitness funkce prvního jedince je nejnižší a hodnota fitness funkce čtvrtého jedince je nejvyšší. Roztočením vybereme s vyšší pravděpodobností kvalitnější jedince, kteří se zúčastní reprodukce. [5] 15
3.4 Ukázka jednobodového a dvoubodového křížení. Genotyp rodiče se náhodně rozdělí na jednom nebo dvou místech. Rozdělením nám vznikly části genotypů rodičů. Pro vytvoření potomků poskládáme tyto části z obou rodičů, kdy nám vznikne nový jedinec. [6] 16
3.5 Ukázka fungování VGX. Náhodně jsme zvolili město, v našem případě 1 a dali jsme ho na začátek generovaného řešení. Následně jsem nalezli sousedy města 1. Z nalezených sousedů (2,6,5,1) je nejbližší 2, tudíž 2 vkládáme do řešení a hledáme sousedy města 2. 17
3.6 Double-bridge operátor vymění náhodně vybranou pozici dvou genů mezi sebou.[7] 17

[1] https://miro.medium.com/max/1400/1*NJEshcz7pqTwJ4zwO4oBlA.webp

[4] https://www.generativedesign.org/02-deeper-dive/02-04_genetic-algorithms/02-04-01_what-is-a-genetic-algorithm

[5] https://www.researchgate.net/profile/Jhielson-Montino-Pimentel/publication/320236456/figure/fig5/AS:775837079048197@1561985382212/Decision-maker-function-based-on-the-Genetic-Algorithm-Roulette-Wheel-Selection-The.ppm

[6] https://i.iinfo.cz/images/282/bio-algoritmy-3-4.png

[7] https://brandinho.github.io/images/swap_mutation.png

3.7 Ukázka fungování PMX. Náhodně jsme zvolili v prvním kroku dvě místa řezů a následně jsme provedli mapování 2 ↔ 1, 7 ↔ 6 a 1 ↔ 8 a tyto oblasti prohodili do potomků. V druhém kroku jsme doplnili oblasti mimo řezy městy, které jsou v původních rodičích. Poslední krok je doplnění měst, které jsem nemohl doplnit v druhém kroku. Zde využijeme mapování sekcí mezi řezy. První × v prvním potomkovi by mělo být město 8, které pochází od prvního rodiče, ale 8 je již v tomto potomkovi, takže zkontrolujeme mapování 1 ↔ 8 a opět vidíme, že v tomto potomkovi je město 1 již obsazeno, opět tedy zkontrolujeme mapování 2 ↔ 1, a získáme město 2, které ještě není v prvním potomkovi obsaženo, tudíž na první × doplníme 2. Stejným způsobem získáme město pro všechny × v obou potomcích. Konkrétní příklad je převzat z [10]. 18
3.8 Ukázka fungování OX. Náhodně jsme zvolili v prvním kroku dvě místa řezů a následně jsme tyto podtrasy vložili do potomků. V druhém kroku jsme vzali celou sekvenci se začátkem po druhém místě řezu z druhého rodiče. Takto nám vznikla trasa 3 → 7 → 4 → 2 → 5 → 1 → 6 → 8. Následně jsme z ní odstranili města, která již existují v prvním potomku a zbyla nám sekvence 3 → 4 → 5 → 6 → 8. Města ze vzniklé sekvence vkládáme po jednom do volných míst potomka od druhého řezu. Takto nám vznikne nový jedince. Stejný postup opakujeme pro druhého potomka. Konkrétní příklad je převzat z [10]. 18
3.9 Ukázka fungování CX. V prvním kroku jsme náhodně zvolili rodiče, ze kterého budeme generovat potomka. Zvolili jsme prvního rodiče pro generování, tudíž na první index potomka dáme město 1. V druhém kroku vytváříme cyklus, kdy na prvním indexu druhého rodiče je město 8. Přidáme město 8 a pokračujeme. Ve druhém rodiči na indexu měst 8 z prvního rodiče se nachází město 7. Nyní přidáme město 7 na index, kde se nachází v prvním rodiči. To stejné uděláme pro město 4. Následně ovšem na indexu města 4 se ve druhém rodiči nachází město 1, které se už v potomkovi nachází. Tímto je ukončen cyklus a pokračujeme krokem 3, a to doplněním zbytku měst přímo ze druhého rodiče. Pro příklad vidíme, jak by dopadlo křížení, kdybychom zvolili na začátku jiného rodiče. Musíme počítat s tím, že je možné, že existují rodiče, ze kterých se vygenerují totožní potomci. Konkrétní příklad je převzat z [10]. 19

4.1 V prvním kroku vidíme, jak vypadají možné cesty mezi mraveništěm a potravu a směry cest mravenců. V druhém vidíme, jak zprvu mravenci chodí náhodně a využívají všech možných cest jak se dostat k jídlu. Po několik generací vidíme, že feromonová cesta, kterou zanechali mravenci na prostřední cestě je nejsilnější a drtivá většina mravenců využívá právě tuto cestu, která je i nejkratší cesta do místa s potravou.[2] 21
4.2 Na snímku můžeme vidět, jak 2-opt existující cestu (A, B, E, D, C, F, G) upraví na cestu (A, B, C, D, E, F, G). Došlo k výměně hran mezi b → e a c → f za hrany b → c a e → f, což je zjevně výhodnější trasa. [3] 22

[2] https://forum.1hive.org/uploads/default/original/2X/7/7448c9ae3fce8c5fcb21f2f404ebde21805adb95.jpeg
[3] https://upload.wikimedia.org/wikipedia/commons/thumb/b/b8/2-opt_wiki.svg/440px-2-opt_wiki.svg.png

5.1 Na příkladu vidíme, že při 100 městech rozdíl ještě není zásadní. Postupně ovšem rozdíl roste a při posledním příkladu, je redukovaná matice 10 tisíc krát menší než úplná varianta matice. Redukovaná matice roste lineárně oproti úplné matici, která roste exponenciálně. Konkrétní příklad je převzat z [20]. 24

6.1 Schéma zobrazuje zjednodušenou verzi programu. Oproti klasickému genetickému algoritmu je zde navíc vytvoření nebo načtení redukované matice nejbližších měst. Také jsou zde vložené jeden cyklus 2-opt algoritmu, který se spustí v případě, že genetický algoritmus se zasekne v lokálním optimu. Na samotném konci se spustí 2-opt nebo 3-opt algoritmus, který finálně vylepší řešení TSP. 27
6.2 Na grafu vidíme výsledky experimentu. Statistické vyhodnocení vlivu velikosti matice nejbližších měst na délku trasy TSP. 29
6.3 Na grafu vidíme výsledky experimentu. Statistické vyhodnocení vlivu velikosti matice nejbližších měst při zmenšení limitu iterací na délku trasy TSP. 30
6.4 Na grafu vidíme výsledky experimentu. Statistické vyhodnocení vlivu procenta vylepšení genu na délku trasy TSP. 31
6.5 Na snímku vidíme výsledky experimentů. Vidíme zde 3 grafy, ukazují vliv počtu mutací při různých velikostech populace na délku tras TSP. 32
6.6 Na snímku vidíme výsledky experimentů. Vidíme zde 4 grafy, které ukazují vliv velikosti populace a počet double-bridge mutací na délku tras TSP. . . 33
6.7 Na grafu vidíme výsledky experimentu. Graf ukazuje vliv pravděpodobnosti řešení na výsledky TSP. 34
6.8 Na grafu vidíme výsledky experimentu. Osa X, která znázorňuje počet iterací, je logaritmická. Vidíme zde průběh fitness funkcí různých operátorů křížení. 39
6.9 Na grafu vidíme výsledky experimentu. Můžeme pozorovat vliv výběru varianty hybridního genetického algoritmu na vývoj křivky historie fitness funkcí. 41

Kapitola 1

Úvod

Pokročilá evoluční optimalizace úloh typu TSP (Traveling Salesman Problem) se zabývá hledáním nejkratší cesty, kterou musí prodejce projít aby navštívil všechna města. V této práci se zabýváme řešením TSP pomocí evolučních algoritmů. Popíšeme principy fungování evolučních algoritmů, ukážeme si příklady použití evolučních algoritmů pro řešení TSP, a dále se zaměříme na genetický algoritmus, který je použit pro řešení TSP v této práci, který si popíšeme podrobněji. Popíšeme si i 2-opt a 3-opt algoritmus, který budeme také využívat pro řešení TSP.

Cílem této práce bude hledání možností pokročilé optimalizace TSP o velikostech instancí v řádu desítek až desítek tisíc měst. Zkusíme navrhnout vylepšení genetického algoritmu, a jiná vylepšení, která nám pomohou získat co nejlepší výsledky. Všechny parametry programu si nastavíme a vyzkoušíme jejich přínos pomocí experimentů, které vykreslíme. V další časti pustíme plnou verzi genetického algoritmu na sadě problémů obchodního cestujícího od velikosti 29 měst do 25 tisíc měst. Provedeme experimenty, která nám prozradí jakých výsledků jsou algoritmy schopny dosáhnout. V dalších experimentech si následně ukážeme vývoj hodnot fitness funkcí různých genetických algoritmů a různých druhů operátorů křížení, kdy si také porovnáme časovou náročnost těchto různých variant. Zhodnotíme všechny experimenty.

Kapitola 2

Problém obchodního cestujícího

Při řešení problému obchodního cestujícího musíme navštívit všechna požadovaná města právě jednou a vrátit se do města, kde jsme začali cestu. Cílem je učinit tuto trasu v nejkratší vzdálenosti. Příklad takové cesty německými městy můžeme vidět na obrázku 2.1.

V této kapitole si probereme definice TSP a formální definici TSP. Probereme, proč je hledání optimálních řešení problémů TSP náročné a proč je neumíme řešit optimálně. Ukážeme si různé reprezentace cest v TSP a podíváme se na reálné využití programu, který řešení TSP.

2.1 Definice TSP

Problém obchodního cestujícího (TSP) je jedním z klasických NP-úplných problémů kombinatorické optimalizace. Lze jej vyjádřit takto: nechť $G = (V, A)$ je graf, kde V je množina N vrcholů a A je množina hran. N je počet měst, které musí obchodní cestující navštívit. Nechť $C = (c_{ij})$ je daná matice vzdáleností přidružená k A. TSP spočívá v určení minimální délky hamiltonovské trasy, tj. okruhu o minimální vzdálenosti procházejícího každým vrcholem pouze jednou s návratem do počátečního vrcholu. V této práci se zabýváme pouze symetrickými problémy obchodního cestujícího, tj. kdy $c_{ij} = c_{ji}$ pro všechny $i, j \in V$ [15]. Příklad vyřešeného zadání TSP je na snímku 2.1.

2.1.1 Formální definice

Jedná se o Dantzig–Fulkerson–Johnson (DFJ) definici [15]. Máme množinu skládající se z n měst vypadající $\mathrm{V} = \{0, 1, ..., n-1\}$, která musíme navštívit, se vzdáleností mezi každým párem měst i a j danou v c_{ij}. Zavedeme si další rozhodovací proměnnou y_{ij} pro každou z dvojci (i, j), která je definovaná jako:

$$y_{ij} = \begin{cases} 1 & \text{pokud město } j \text{ je navštíveno ihned po městu } i \\ 0 & \text{v opačném případě} \end{cases} \tag{2.1}$$

Účelová funkce je dána pomocí:

$$F = min \sum_i \sum_j c_{ij} y_{ij} \tag{2.2}$$

Pro zajištění validní cesty je třeba přidat několik omezení.

Omezení pro počet měst, kam může obchodní cestující jít

Po návštěvě města i, musí obchodní cestující navštívit pouze jedno další město:

$$\sum_j y_{ij} = 1, i = 0, 1, ..., n-1 \tag{2.3}$$

Omezení pro počet měst, odkud může obchodní cestující přijít

Když navštěvujeme město, musí obchodní cestující přijít pouze z jednoho města:

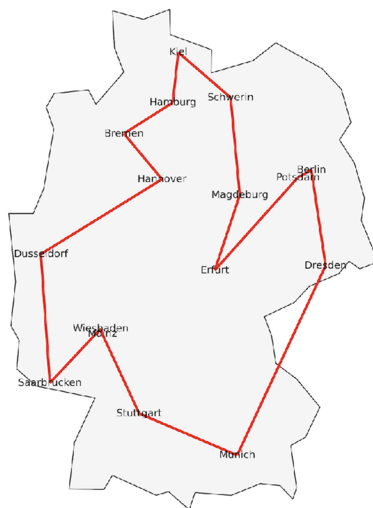
$$\sum_j y_{ij} = 1, j = 0, 1, ..., n-1 \tag{2.4}$$

Zajištění, že cesta obchodního cestujícího je kompletní

Zajištění, že cesta je plně propojená a neexistují zde subcesty:

$$\sum_i \sum_j y_{ij} \leq \mid S \mid -1 S \subset V, 2 \leq \mid S \mid \leq n-2 \tag{2.5}$$

kde S je sada všech cest G.



Obrázek 2.1: Příklad cesty obchodního cestujícího vybranými německými městy. [1]

2.2 Problémy optimalizace TSP

Při prvním pohledu a analýze TSP nám nemusí být jasné, jak velký a komplexní tento problém je. Jedná se o permutační problém, tudíž musíme poskládat do vhodného pořadí všechna daná dostupná města. Na první město, které navštívíme, máme n možností, kdy n je počet měst, na další město v pořadí máme $n-1$ možností, a tak dále. Celkový počet všech kombinací je tedy $n!$. V případě, že se jedná o asymetrické TSP, je počet možností

skutečně $n!$. V případě, že se jedná o symetrické TSP je počet možností redukován na polovinu, a tudíž počet možností je $\frac{n!}{2}$. Například v případě symetrického TSP pro 10 měst existuje 1 814 400 možností, pro 100 měst je to $4,66 \times 10^{157}$ možností, což už pro výčet takového počtu řešení přesahuje možnosti současných počítačů.

Hrubou silou jsme schopni spočítat pouze relativně malé TSP, které pro praxi nejsou příliš zajímavé. Existuje několik jiných způsobů řešení TSP. Například se jedná o řešení pomocí dynamického programování (Held–Karp algoritmus [7]), pomocí programu Concorde TSP Solver [1], které různě aproximují řešení problémů nebo využití heuristik.

Optimalizace TSP patří mezi NP-těžké úlohy. Není známo, jak nalézt optimální řešení v čase úměrném nějaké mocnině počtu uzlů a zdali vůbec takový algoritmus existuje. TSP je NP-úplný, tedy existuje nedeterministický Turingův stroj (nedeterministický algoritmus).

2.3 Existující reprezentace řešení TSP

Máme několik způsobů, jak reprezentovat možná řešení TSP. Každé řešení musí být jednoznačné a pokud možno co nejjednodušeji prezentovatelné. Je více způsobů reprezentace a nyní se podíváme na možná řešení. Pro porovnání budou všechny reprezentace zobrazovat stejnou cestu, aby byly zřetelné rozdíly. První dvě reprezentace jsou převzaté z [13].

Přímá binární reprezentace založena na cestách mezi městy

Tato reprezentace je binární matice o rozměrech $n \times n$, kdy n je počet měst v daném problému TSP. Pokud se $\mathrm{C}(ij) = 1$, znamená to, že mezi městy i a j vede cesta v daném řešení TSP. V této reprezentaci bude mít každé město v daném řádku i sloupci matice C právě dvě hodnoty nastavené 1. Nyní si ukážeme, jak tato reprezentace vypadá. Matice na obrázku 2.2 zobrazuje příklad cesty (1, 2, 3, 6, 8, 5, 7, 4).

0	1	0	1	0	0	0	0
1	0	1	0	0	0	0	0
0	1	0	0	0	1	0	0
1	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	1	0	0	0	0	1
0	0	0	1	1	0	0	0
0	0	0	0	1	1	0	0

Obrázek 2.2: Binární matice cest jistého řešení TSP, kde $\mathrm{C}(ij) = 1$ znamená, že cestující prošel z města i do města j kde $\mathrm{C}(ij) = 0$ znamená, že přechod z města i do j nebyl v tomto řešení využit.

Přímá binární reprezentace založená na pořadí měst

Reprezentace cesty v předchozí sekci není jednoznačná, ale pro řešení TSP to nevadí. Cesta (1,2,3) a naprosto stejná jako (2,3,1), jenom začneme v jiném městě. Tento problém řeší

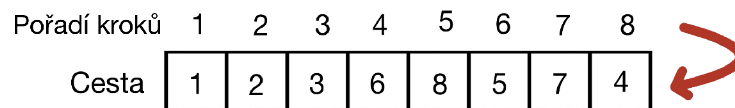
upravená matice, která udává i pořadí měst. Pokud se C(ij) = 1, znamená to, že město i je v daném řešení na pozici j. Nyní si ukážeme, jak tato reprezentace vypadá. Matice na obrázku 2.3 zobrazuje cestu (1, 2, 3, 6, 8, 5, 7, 4).

1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0

Obrázek 2.3: Binární matice pořadí, kde C(ij) = 1, znamená to, že město i je v pořadí v daném řešení na j pozici a kde C(ij) = 0 znamená, že přechod z města i do j nebyl v tomto řešení využit.

Reprezentace permutací

Asi nejčastější použitou reprezentací řešení TSP je zápis pomocí permutace. Je zvyklostí, že označení měst je P $\in (1, ..., n)$ kdy n je počet měst. Tato prezentace je jednoznačná, stejně jako reprezentace z předchozí sekce. Výhodou této reprezentace je její jednoduchost, čitelnost pro lidi, a také jednoduchý způsob vytváření nových permutací například i z částí předchozích permutací a kombinací více permutací, kdy můžeme rozdělit množinu měst na dvě části od 1 po $\frac{n}{2}$ a $\frac{n}{2} + 1$ po n a vytvořit permutace a tyto permutace spojit. Tyto vlastnosti mohou být výhodné např. pro evoluční algoritmy. Příklad cesty (1, 2, 3, 6, 8, 5, 7, 4) permutační reprezentací zobrazujeme na obrázku 2.4.



Obrázek 2.4: Řešení TSP je zakódováno pomocí zapsaných měst v určitém pořadí.

Toto byly nejznámější způsoby zakódování řešení TSP. Z důvodů jednoduchosti a čitelnosti jsem si zvolil pro použití v této práci reprezentaci pomocí permutace.

2.4 Využití řešení TSP

Kromě využití při plánování obchodní cesty, které je zřejmé už z názvu samotného problému, nám řešení TSP pomůže vyřešit a ušetřit mnoho času i financí. První známý problém, kde

se využilo řešení TSP bylo plánování cesty školního autobusu [16]. V dnešní době ovšem řešíme mnohonásobě větší TSP.

V dnešní době jsme obklopeni elektrickými spotřebiči a jejich počet stále roste. Při výrobě komponent do těchto spotřebičů se dají problémy namapovat na TSP. Například musíme vyvrtat několik tisíc děr do základní desky. Díry, které chceme vyvrtat, jsou města, která musí obchodní cestující navštívit. Pokud bychom naplánovali cestu, která je 3krát delší, než je optimální, znamenalo by to, že na výrobní lince by byl proces, který trvá 3krát déle a celý výrobní proces zdržuje. Dále si představme, že potřebujeme pospojovat všechny komponenty v čipech. Opět, kdybychom zvolili 3krát delší cestu, zabralo by to 3krát více času a také bychom použili 3krát více materiálu, než při optimálním způsobu.

Vědecké skupiny, které zkoumají vesmírnou oblohu k tomu používají teleskopy. Těch nejvýkonnějších teleskopů je velice omezené množství. Všechny výzkumné skupiny musí nějak využívat tyto teleskopy pro zkoumaní hlubin našeho Vesmíru. Samotný pohyb těchto teleskopů je zdlouhavý a nákladný. Pokud by si výzkumné skupiny uměli naplánovat optimální cestu přes objekty ve Vesmíru, které chtějí zkoumat pomocí teleskopu, ušetřili by za pohyb teleskopu a ušetřili by také čas strávený používám teleskopu.

Toto jsou jenom některé možnosti využití TSP. V dnešní době, kdy se dbá na efektivitu ve zdrojích i času, je toto oblast, kterou stojí za to zkoumat.

Kapitola 3

Evoluční algoritmy

Teorie evoluce, kterou zformuloval Charles Darwin [5], stojí na přirozeném výběru a dědičnosti. Jedinci, kteří jsou schopni se lépe přizpůsobit prostředí, mají vyšší pravděpodobnost přežití a další reprodukce než jedinci, kteří mají tuto schopnost horší. Jelikož se reprodukují jedinci, jejichž geny jsou vhodnější pro prostředí, jsou jejich geny dále distribuovány v populaci. Postupem času se populace generace po generaci vylepšuje a jejich schopnost přizpůsobovat se prostředí se vylepšuje.

Evoluční algoritmy vycházejí z této myšlenky evoluce populace a reprodukují její principy v počítačích. Existují i evoluční algoritmy, které nejsou založené na populacích, ty mají pouze jednoho jedince a s tím nějak pracují, ovšem nejsou tak časté jako ty populační. Evoluční algoritmy nejsou konkrétní algoritmy, ale řadí se mezi tzv. metaheuristiky, to znamená, že tyto algoritmy jsou velice obecné a neřeší žádný konkrétní problém. Evoluční algoritmy jsme tedy schopni aplikovat na mnohé optimalizační problémy, nikoliv pouze na problém obchodního cestujícího. Optimalizační problémy prohledávají prostor možných řešení a hledají řešení, pro které je účelová funkce v globálním maximu, či minimu, záleží na problému a definici účelové funkce. Evoluční algoritmy simulují procesy, které známe z teorii evoluce a aplikují je na tyto optimalizační problémy. Jelikož se jedná o obecný algoritmus, pro každý problém, který jsme schopni řešit evolučním algoritmem, je potřeba tento algoritmus upravit. Úpravy se většinou týkají nastavení parametrů, přizpůsobení účelové funkce, nastavení ukončujících podmínek algoritmu atd. Evoluční algoritmus, stejně jako evoluce, obsahuje prvek náhody. Občas i nejslabšímu jedinci se podaří reprodukovat a občas i nejsilnějšímu jedinci z řešení se reprodukovat nepodaří. Evoluční algoritmy mají tedy stochastické vlastnosti a je nutno s tímto prvkem počítat. Evoluční algoritmy negarantují nalezení řešení, které hledáme. Nyní si definujeme přesněji genetický algoritmus a pojmy, které se vyskytují v evolučních i genetických algoritmech. Informace do této části jsem čerpal z [23] a [27].

3.1 Genetický algoritmus

Genetické algoritmy jsou inspirované přírodou, konktrétněji přirozeným výběrem a procesem reprodukce pozorovaným v přírodě. Genetické algoritmy patří z evolučních algoritmů k nejpopulárnějším. Profesor Holland je prvním, kdo se ve své knize zmiňuje o genetických algoritmech [9]. Genetické algoritmy řeší především optimalizační problémy. Za pomoci populace kandidátních řešení se přirozeným výběrem optimalizuje problém, který řešíme. Řešení problémů zakódujeme do genotypu, se kterým GA pracuje. Genetické algoritmy simulují evoluci, kdy lepší geny, v našem případě řešení našich problémů, přežívají a zvy-

šují pravděpodobnost reprodukce a tím pádem i sdílení své genetické informace, kdežto ty horší geny nemají takovou pravděpodobnost reprodukce. My tento proces pouze nastavíme, vhodně zakódujeme možná řešení, vhodně nastavíme všechny parametry a vyhodnocujeme výsledky. Nejkvalitnější řešení se reprodukuje s nejvyšší pravděpodobností a tato kvalita se dědí do příštích generací s nadějí na další vylepšení. Proces evoluce můžeme vidět na snímku 3.1. Evoluce funguje na stejném principu. V ideálním případě současná populace obsahuje kvalitnější řešení než populace předchozí, zároveň budoucí populace, vzniklá ze současné populace, by měla obsahovat ještě kvalitnější řešení. Tento proces samozřejmě nepokračuje do nekonečna a existuje nějaká ukončovací podmínka, která proces evoluce ukončí. Všechny tyto procesy musí být vhodně nastaveny abychom dosáhli na konci evoluce co nejlepších výsledků. Je nutné si vyjmenovat a charakterizovat pojmy, které se v genetických algoritmech vyskytují.[6] [24] [3] [9]

Jedinec (genotyp) Jedinec je vhodným způsobem zakódované řešení našeho problému. Například se může jednat o binární matici kódující cestu obchodního cestujícího.

Fenotyp Fenotyp už je samotné řešení problému. Fenotyp je rozkódovaný genotyp, který hodnotíme fitness funkcí. Například se jedná o konkrétní cestu obchodního cestujícího mezi městy.

Fitness funkce(účelová funkce) Fitness funkce ohodnocuje, jak kvalitní řešení je zakódované v genotypu. Hlavním cílem genetických algoritmů je maximalizace či minimalizace fitness funkce. Je nutné si dávat pozor na lokální extrémy, jelikož cílem genetických algoritmů je nalezení globálních extrémů.

Populace Jedná se o soubor jedinců, kteří spolu tvoří populaci. Velikost populace se nastavuje jako vstupní parametr.

Iniciální populace První populace genetického algoritmu. Generování iniciální populace může být náhodné nebo podle nějaké heuristiky.

Selekce Selekce je výběr jedinců, ze kterých proběhne reprodukce neboli vytváření potomků. Selekce upřednostňuje jedince s nejlepší fitness hodnotu v populaci, aby jejich genotyp byl rozšířen co nejvíce do budoucích populací. Selekce probíhá algoritmy, které si popíšeme níže.

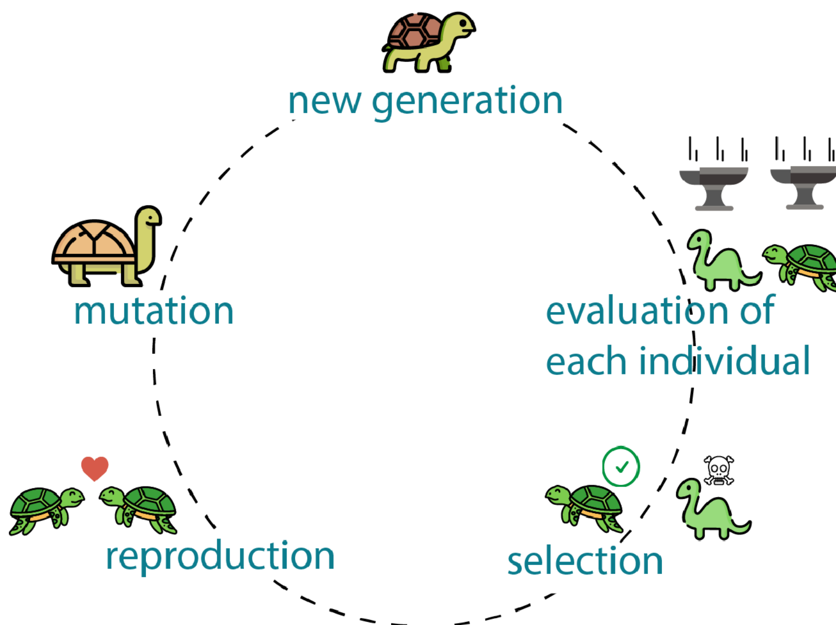
Křížení(reprodukce) Vytváření potomků z rodičů vzniká křížením, kdy jsou informace z genotypů sdíleny a namíchany do nově vznikajícího jedince.

Mutace Mutace je náhodná změna v genotypu jedince. Z evoluce je známo, že v některých případech náhodné mutace vytvořili vlastnost jedince, která nebyla v populaci, ale jedince přizpůsobila prostředí a zvýšila jeho šanci na další reprodukci.

Ukončující podmínka genetického algoritmu Každý algoritmus musí být ukončen. V případě genetických algoritmů je dáno, že po nějakém počtu populací nebo po splnění nějaké hodnoty fitness je genetický algoritmus ukončen a vrací řešení s nejlepší hodnotou fitness funkce.

Předchozí část s pojmy z genetického algoritmu je převzata z [3] a [25]. Po seznámení se s pojmy uvedenými výše můžeme nyní představit úplnou podobu metaheuristiky genetického algoritmu. Pseudokód algoritmu ukazuje algoritmus 1. V prvním kroku je inicializována počáteční populace. Následuje zahájení evolučního cyklu, který vyvíjí jedince tak

dlouho, dokud není splněna ukončující podmínka. Jedinci jsou ohodnocení pomocí fitness funkce. Poté proběhne selekce jedinců, kteří se zúčastní reprodukce. Poté proběhne samotná reprodukce a následné obnovení populace. S jistou pravděpodobností proběhne poté mutace jedinců.



Obrázek 3.1: Princip evoluce, kdy jeden jedinec (genotyp) je vyřazen pomocí selekce podle hodnoty fitness funkce a druhý jedinec se dostane k reprodukci, kdy je následně možnost nějaké mutace předtím, než vznikne nová populace. [4]

Algorithm 1 Genetický algoritmus [3]

Inicializace populace
while není splněna ukončující podmínka genetického algoritmu **do**
    ohodnocení jedinců fitness funkcí
    selekce jedinců, kteří se zúčastní reprodukce
    křížení jedinců (reprodukce) s pravděpodobností p_cross
    mutace jedince s pravděpodobností p_mut
    obnovení jedinců v populaci
end while

3.1.1 Vlastnosti prvků genetického algoritmu

Je nutné si blíže specifikovat vlastnosti prvků, se kterými pracujeme v genetických algoritmech. Další část, kde spekulujeme podrobněji pojmy z GA, je převzata z [3], [6] a [25].

Genotyp a fenotyp

Jak jsme zmínili v předchozí sekci, genotyp je způsob zakódování řešení našeho problému a fenotyp je dekódované řešení genotypu představující objekt reálného světa, který je před-

mětem ohodnocení fitness. Celý prohledávaný prostor musí být reprezentovatelný zvoleným způsobem zakódování a zároveň musí tak činit co nejefektivnějším způsobem. V ideálním případě by malá změna genotypu měla způsobit malou změnu v řešení, naopak velké změny by měly způsobit velkou změnu v řešení. Tento aspekt nám pomůže odhadovat, jak se řešení změní při použití nějakého operátoru. Musíme také vyvážit efektivnost a duplicitu zakódování řešení. V ideálním případě je jeden fenotyp možný reprezentovat pouze jedním genotypem.

Selekce

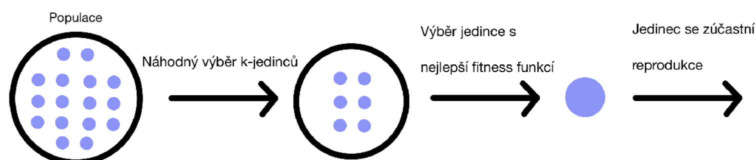
Máme dva aspekty selekce v genetických algoritmech. Jeden je výběr jedinců jako rodičů do reprodukce a druhý je výběr jedinců do budoucí populace. Nejdříve blíže specifikujeme výběr jedinců pro reprodukci.

Výběr jedinců pro reprodukci nám vybírá jedince, kteří budou vhodní pro křížení a mutaci. Způsobem selekce ovlivníme, jak moc preferujeme kvalitní jedince oproti méně kvalitním, přičemž ve smyslu kvality je myšlena hodnota fitness funkce jedinců. Tento jev označujeme jako selekční tlak, kdy platí, že čím vyšší selekční tlak tím pravděpodobněji si pro reprodukci zvolíme kvalitní jedince, a tím méně pravděpodobněji si zvolíme nekvalitní jedince. Existují různé varianty selekce, nejčastěji se používají následující:

Náhodný výběr Nejjednodušší způsob pro výběr rodiče pro reprodukci je náhodný výběr. Tento způsob není nejvhodnější, jelikož zde nevzniká žádný selekční tlak a konvergence k nějakému kvalitnějšímu řešení by byla čistě náhodná.

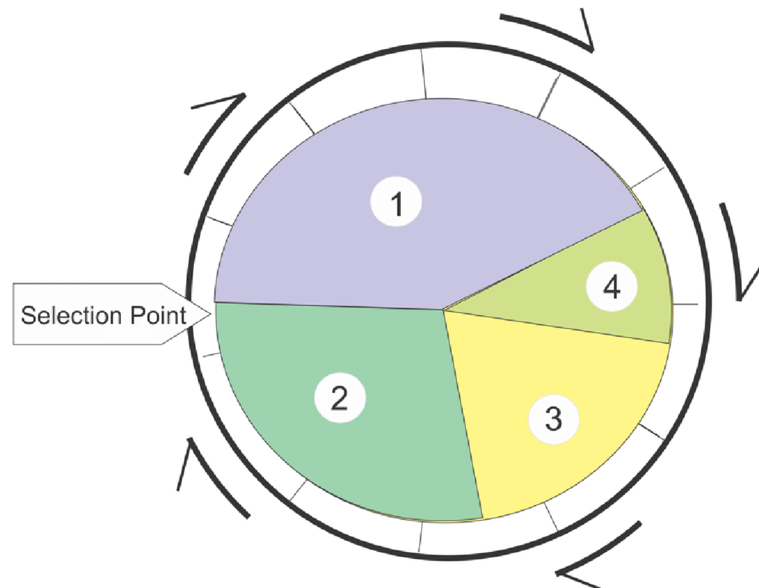
Podle pořadí Jedinci se seřadí podle fitness hodnoty od nejkvalitnějších po nejméně kvalitní. Nyní si tento soubor jedinců rozdělíme na dvě části a vybíráme jedince pouze z té kvalitnější části populace.

Turnaj Náhodně si vybereme k-jedinců z populace, kdy k je menší než velikost populace. Následně uspořádáme turnaj mezi těmito k jedinci, kdy vyhraje turnaj ten jedinec, jehož fitness hodnota je nejlepší. Tento jedinec se stává rodičem do reprodukce. Ilustraci turnajového výběru vidíme na obrázku 3.2.



Obrázek 3.2: Snímek znázorňující turnajový výběr při selekci jedince, který se bude reprodukovat.

Vážená ruleta Celá populace je zastoupena v ruletě. Kolo rulety je rozděleno na p částí, kde p je velikost populace. Následně je velikost každé části rulety upravena. Velikost dílků odpovídá fitness hodnotě. Čím blíže je fitness hodnota jedince optimální hodnotě, ke které směřujeme, tím větší část rulety přísluší jedinci. Ruleta se poté roztočí a vybere se jedinec. V případě, že chceme zvolit více jedinců, bude ruleta roztočena víckrát nebo máme více hodnot, které určují výběr. Příklad rulety s jedním výběrovým bodem je na obrázku 3.3.



Obrázek 3.3: Snímek znázorňující váženou ruletu, kdy hodnota fitness funkce prvního jedince je nejnižší a hodnota fitness funkce čtvrtého jedince je nejvyšší. Roztočením vybereme s vyšší pravděpodobností kvalitnější jedince, kteří se zúčastní reprodukce. [5]

Křížení

Křížením vznikají nová řešení problému z existujících řešení. Křížení probíhají v genetických algoritmech s nějakou pravděpodobností. Křížení neboli reprodukce je výsledkem operátorů křížení, který ze dvou rodičů vytvoří potomka, který sdílí části z obou rodičů. Takto docílíme, že části řešení, obsažené v rodičích, přenesou svoji genetickou informaci do potomka. Způsoby, kterými vznikají potomci mohou být různé. Je to způsobeno tím, že reprezentace řešení má různé formy, a tudíž se na genotypy aplikují jiné druhy operátorů křížení. My se věnujeme problému, jehož genotypy reprezentujeme permutací, která je zapsána jako indexy v poli čísel, a tudíž si ukážeme příklady operátorů na snímku 3.4, který pracuje s touto formou. Operátory křížení na snímku 3.4 jsou nejjednodušší varianty. Pro TSP se používají složitější metody, které si představíme níže.

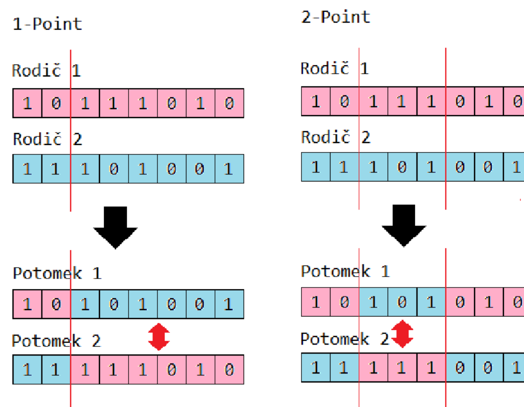
VGX Very greedy crossover je speciální variantou greedy crossoveru [12]. Základní myšlenkou operátoru je, že se ze začátku zvolí náhodně město a vloží se na začátek potomka. Následně najdeme pravého a levého souseda našeho města v obou rodičích a vložíme do řešení na další pozici souseda, který měl k městu nejkratší vzdálenost. V článku [12] je více variant, jak se má operátor křížení zachovat. V našem případě, pokud jsou už všechny sousedy obsaženy v prozatímním řešení, zvolíme město, které je co nejblíže původnímu městu. Proto se jedná o very greedy crossover. Nákres fungování VGX je na snímku 3.5.

PMX Partly mapped crossover [10] je další pokročilejší operátor křížení, který ovšem nijak neprohledává ani nepoužívá heuristiku při křížení jako tomu bylo u VGX. Dalším rozdílem je, že zde vzniknou vždy dva potomci místo jednoho ze dvou rodičů. Zde je princip výměny informací mezi dvěma rodiči následující. Uděláme dva náhodné řezy řešením v obou rodičích. Tyto oblasti mezi řezy namapujeme a následně je do potomků prohodíme. V oblastech mimo náhodné řezy dosadíme města, která se zatím nenacházejí

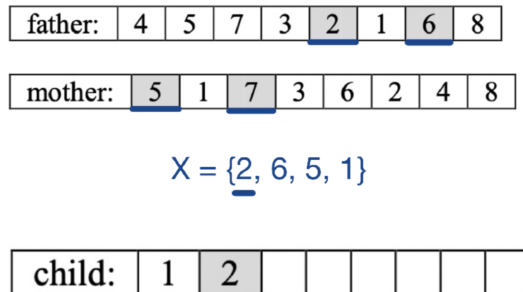
v prozatimních řešeních. Pro zbylá města využijeme namapování oblastí mezi řezy a dostaneme se iterativně k městu, které tam můžeme vložit. Může se totiž stát, že hned první namapované město nás může vést na město, které je také už obsažené v potomkovi. Pro lepší pochopení je PMX vysvětleno na ilustraci 3.7.

OX Order crossover [10] je operátor křížení, který vytvoří potomky výběrem podtrasy jednoho rodiče a zachováním relativního pořadí genů druhého rodiče. Opět se tedy nejedná o operátor křížení, který by využíval heuristiku nebo prohledávání. Oblast podtrasy je vybrána, stejně jako u operátoru VGX, dvěma náhodnými řezy. Zbytek jedince získáme z jiného rodiče, kdy vezmeme posloupnost genů za druhým řezem, odstraníme z posloupnosti města, která už jsou obsažena v jedinci a postupně tam vložíme města za druhý náhodný řez. Tímto způsobem opět vytvoříme dva potomky, jak tomu bylo u PMX. Pro lepší pochopení je OX vysvětleno na ilustraci 3.8.

CX Cycle crossover [10] je operátor křížení, který vytváří potomky způsobem, kdy každý gen pochází od jednoho rodiče. Náhodně zvolíme město jednoho z rodičů na prvním indexu. Další město doplníme podle toho, jaké město se nachází na prvním indexu druhého rodiče a takto pokračujeme, dokud nenalezneme cyklus. Poté co nalezneme cyklus ukončíme vkládání a vložíme zbytek měst z druhého rodiče. Nevýhodou tohoto operátoru je, že v některých případech vygenerují stejné potomky jako jsou rodiče. Pro lepší pochopení je OX vysvětleno na ilustraci 3.9.



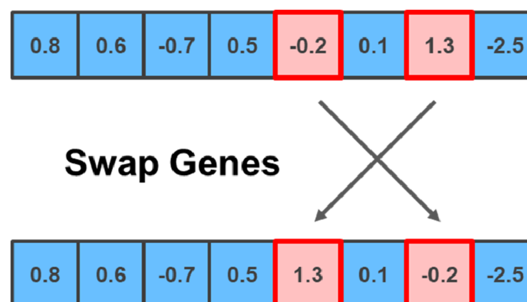
Obrázek 3.4: Ukázka jednobodového a dvoubodového křížení. Genotyp rodiče se náhodně rozdělí na jednom nebo dvou místech. Rozdělením nám vznikly části genotypů rodičů. Pro vytvoření potomků poskládáme tyto části z obou rodičů, kdy nám vznikne nový jedinec. [6]



Obrázek 3.5: Ukázka fungování VGX. Náhodně jsme zvolili město, v našem případě 1 a dali jsme ho na začátek generovaného řešení. Následně jsem nalezli sousedy města 1. Z nalezených sousedů (2,6,5,1) je nejbližší 2, tudíž 2 vkládáme do řešení a hledáme sousedy města 2.

Mutace

Mutace je posledním krokem před vznikem nové generace. Mutace je nějaká náhodná změna v genotypu. Mutace probíhají v genetických algoritmech s nějakou pravděpodobností. Například u lidí došlo k náhodné mutaci genu pro hemoglobin, která na jednu stranu zhoršuje vlastnosti červených krvinek, ovšem lidé, kteří mají tuto mutaci jsou imunní vůči malárii, což je v Africe výhodou [21].Tuto charakteristiku replikujeme mutacemi v genetických algoritmech. Kdybychom pouze používali operátor křížení, nebyli bychom časem schopni vytvářet nové jedince. Toto by omezovalo prohledávací schopnost algoritmu. Mutace na rozdíl od křížení může přinést potomkům rysy, které se nevyskytovaly u rodičů a tím vytvořit kvalitnější řešení. Příkladem mutačního operátoru použitého pro permutační reprezentaci je double-brigde [11]. Operátor vymění dva náhodně vybrané geny mezi sebou. Můžeme provést těchto výměn několik v řadě. Ilustraci double-bridge operátoru můžeme vidět na snímku 3.6.



Obrázek 3.6: Double-bridge operátor vymění náhodně vybranou pozici dvou genů mezi sebou.[7]

1. $P_1 = (3\ 4\ 8 \mid 2\ 7\ 1 \mid 6\ 5)$
$P_2 = (4\ 2\ 5 \mid 1\ 6\ 8 \mid 3\ 7)$
$O_1 = (\times\ \times\ \times \mid 1\ 6\ 8 \mid \times\ \times)$
$O_2 = (\times\ \times\ \times \mid 2\ 7\ 1 \mid \times\ \times)$

2. $O_1 = (3\ 4\ \times \mid 1\ 6\ 8 \mid \times\ 5)$
$O_2 = (4\ \times\ 5 \mid 2\ 7\ 1 \mid 3\ \times)$

3. $O_1 = (3\ 4\ 2 \mid 1\ 6\ 8 \mid 7\ 5)$
$O_2 = (4\ 8\ 5 \mid 2\ 7\ 1 \mid 3\ 6)$

Obrázek 3.7: Ukázka fungování PMX. Náhodně jsme zvolili v prvním kroku dvě místa řezů a následně jsme provedli mapování $2 \leftrightarrow 1$, $7 \leftrightarrow 6$ a $1 \leftrightarrow 8$ a tyto oblasti prohodili do potomků. V druhém kroku jsme doplnili oblasti mimo řezy městy, které jsou v původních rodičích. Poslední krok je doplnění měst, které jsem nemohl doplnit v druhém kroku. Zde využijeme mapování sekcí mezi řezy. První $\times$ v prvním potomkovi by mělo být město 8, které pochází od prvního rodiče, ale 8 je již v tomto potomkovi, takže zkontrolujeme mapování $1 \leftrightarrow 8$ a opět vidíme, že v tomto potomkovi je město 1 již obsazeno, opět tedy zkontrolujeme mapování $2 \leftrightarrow 1$, a získáme město 2, které ještě není v prvním potomkovi obsaženo, tudíž na první $\times$ doplníme 2. Stejným způsobem získáme město pro všechny $\times$ v obou potomcích. Konkrétní příklad je převzat z [10].

1. $P_1 = (3\ 4\ 8 \mid 2\ 7\ 1 \mid 6\ 5)$
$P_2 = (4\ 2\ 5 \mid 1\ 6\ 8 \mid 3\ 7)$
$O_1 = (\times\ \times\ \times \mid 2\ 7\ 1 \mid \times\ \times)$
$O_2 = (\times\ \times\ \times \mid 1\ 6\ 8 \mid \times\ \times)$

2. $3 \to 7 \to 4 \to 2 \to 5 \to 1 \to 6 \to 8$

$3 \to 4 \to 5 \to 6 \to 8$

3. $O_1 = (5\ 6\ 8 \mid 2\ 7\ 1 \mid 3\ 4)$
$O_2 = (4\ 2\ 7 \mid 1\ 6\ 8 \mid 5\ 3)$

Obrázek 3.8: Ukázka fungování OX. Náhodně jsme zvolili v prvním kroku dvě místa řezů a následně jsme tyto podtrasy vložili do potomků. V druhém kroku jsme vzali celou sekvenci se začátkem po druhém místě řezu z druhého rodiče. Takto nám vznikla trasa $3 \to 7 \to 4 \to 2 \to 5 \to 1 \to 6 \to 8$. Následně jsme z ní odstranili města, která již existují v prvním potomku a zbyla nám sekvence $3 \to 4 \to 5 \to 6 \to 8$. Města ze vzniklé sekvence vkládáme po jednom do volných míst potomka od druhého řezu. Takto nám vznikne nový jedince. Stejný postup opakujeme pro druhého potomka. Konkrétní příklad je převzat z [10].

1.

$P_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8)$

$P_2 = (8\ 5\ 2\ 1\ 3\ 6\ 4\ 7)$

$O_1 = (1\ \times\ \times\ \times\ \times\ \times\ \times\ \times)$

2.

$O_1 = (1\ \times\ \times\ \times\ \times\ \times\ \times\ 8)$

↓

$O_1 = (1\ \times\ \times\ \times\ \times\ \times\ 7\ 8)$

↓

$O_1 = (1\ \times\ \times\ 4\ \times\ \times\ 7\ 8)$

3.

$O_1 = (1\ 5\ 2\ 4\ 3\ 6\ 7\ 8)$

$O_2 = (8\ 2\ 3\ 1\ 5\ 6\ 4\ 7)$

Obrázek 3.9: Ukázka fungování CX. V prvním kroku jsme náhodně zvolili rodiče, ze kterého budeme generovat potomka. Zvolili jsme prvního rodiče pro generování, tudíž na první index potomka dáme město 1. V druhém kroku vytváříme cyklus, kdy na prvním indexu druhého rodiče je město 8. Přidáme město 8 a pokračujeme. Ve druhém rodiči na indexu měst 8 z prvního rodiče se nachází město 7. Nyní přidáme město 7 na index, kde se nachází v prvním rodiči. To stejné uděláme pro město 4. Následně ovšem na indexu města 4 se ve druhém rodiči nachází město 1, které se už v potomkovi nachází. Tímto je ukončen cyklus a pokračujeme krokem 3, a to doplněním zbytku měst přímo ze druhého rodiče. Pro příklad vidíme, jak by dopadlo křížení, kdybychom zvolili na začátku jiného rodiče. Musíme počítat s tím, že je možné, že existují rodiče, ze kterých se vygenerují totožní potomci. Konkrétní příklad je převzat z [10].

Kapitola 4

Přehled vybraných technik optimalizace TSP

Podíváme se blíže vybrané práce, které se zabývají řešením problému obchodního cestujícího pomocí evolučních algoritmů. Mezi existující řešení obchodního cestujícího patří řešení pomocí mravenčích algoritmů, simulovaného žíhání nebo genetického algoritmu. Další algoritmus, který ovšem není evoluční, ale rád bych ho zde zmínil, je algoritmus 2-opt, 3-opt nebo jeho obecnější varianta k-opt.

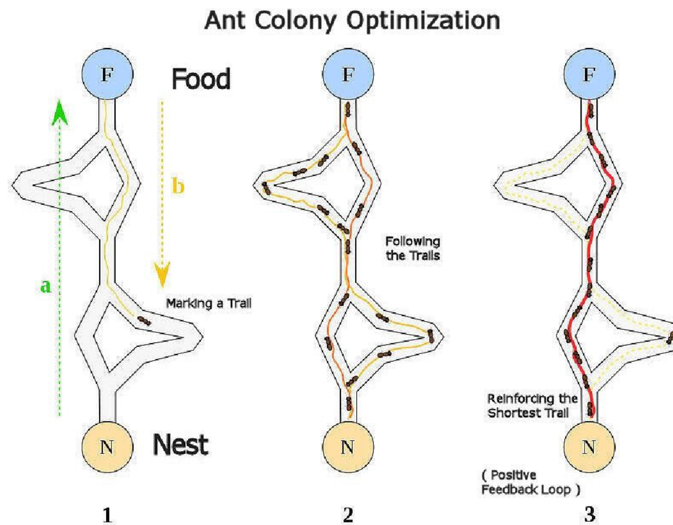
Optimalizace TSP mravenčí kolonií

Optimalizace mravenčí kolonií (ACO) [2] je metoda, která využívá principů chování mravenců k řešení optimalizačních problémů. Nejčastěji ACO simuluje pohyb mravenců pozorovaný při hledání potravy a jejím transportu do mraveniště. Každý mravenec tvoří trasu, kterou jde, a zanechává feromony po cestě, což zvyšuje pravděpodobnost, že další mravenci půjdou stejnou trasou. Další generace mravenců se dále řídí feromonovými značkami, které mají nějaké charakteristiky vypařování. V práci [22] je představena vylepšená varianta ACO algoritmu, nazvaná Focusesd ACO (FACO). Práce se zaměřuje na paralelizaci procesů a restrikci velikosti datových struktur, které jsou pro ACO jinak náročné. Ve výsledku se více zaměřuje na kvalitní cesty, tím pádem je rychlejší a méně paměťově náročný. Optimalizace mravenčí kolonií je zobrazena na obrázku 4.1.

Optimalizace TSP pomocí simulovaného žíhání

Simulované žíhání (Simulated Annealing nebo SA) [14] je metaheuristická metoda pro řešení optimalizačních problémů. Metoda využívá principu žíhání kovů, kdy se materiál ohřívá na vysokou teplotu a pak se postupně ochlazuje. Tímto procesem se kov dostává do nejstabilnějšího stavu, což je také cílem simulovaného žíhání.

SA začíná náhodným výběrem počátečního stavu a postupně generuje nové stavy pomocí jistého operátoru. Řešení je ohodnoceno pomocí fitness funkce, poté na základě výsledku fitness funkce a teplotě simulovaného žíhaní je nový stav přijat nebo odmítnut. Teplota se postupně ochlazuje, což v případě SA znamená, že se snižuje pravděpodobnost přijetí horší stavu, než který je současný. Studie [26] prezentuje více operátorů pro vytváření nových řešení a také prezentuje druhy opatření, aby algoritmus SA se nezasekl v lokálním optimu.



Obrázek 4.1: V prvním kroku vidíme, jak vypadají možné cesty mezi mraveništěm a potravu a směry cest mravenců. V druhém vidíme, jak zprvu mravenci chodí náhodně a využívají všech možných cest jak se dostat k jídlu. Po několik generací vidíme, že feromonová cesta, kterou zanechali mravenci na prostřední cestě je nejsilnější a drtivá většina mravenců využívá právě tuto cestu, která je i nejkratší cesta do místa s potravou.[1]

Optimalizace TSP pomocí genetického algoritmu

Genetický algoritmus (GA) [6] je metaheuristická metoda pro řešení optimalizačních problémů, která se inspirovala principy darwinovské evoluce. GA pracuje s populací náhodně generovaných řešení, která se postupně vyvíjí pomocí principů dědičnosti, selekce, křížení a následně ještě mutací. Řešení problému je reprezentováno jako genotyp. Selekce pak na základě hodnocení fitness funkcí určuje pravděpodobnosti jednotlivých řešení na další reprodukci. Nové kombinace genotypů vznikají právě křížením a mutací při reprodukci.

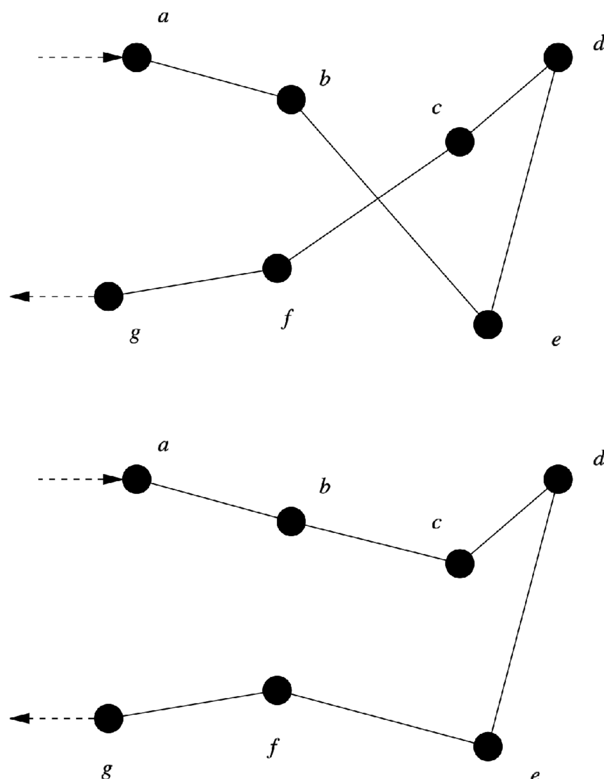
V práci [17] je prezentovaný hybridní genetický algoritmus, který prezentuje heuristiku generování iniciální populace. Představuje lokální operátor křížení a způsob mutace řešení TSP.

Optimalizace TSP pomocí 2-opt a 3-opt

Algoritmy 2-opt a 3-opt [4], obecně k-opt, sice nejsou evoluční algoritmy, ovšem některé výše zmíněné články je v nějaké formě využívají a jedná se o velice silný algoritmus, kterým bývají problémy obchodního cestující řešeny, nebo je to pomocný algoritmus evolučním algoritmům, které řeší TSP.

Jedná se o jednoduchý algoritmus lokálního prohledávání. Číslo v algoritmu určuje, s kolika hranami pracuje. Algoritmus je navržen přímo na řešení problému obchodního cestujícího. Algoritmus už pracuje s navrženou cestou a vyzkouší všechny možné kombinace přehození dvou hran. Pro představu, máme cestu (A, B, D, C, E) a 2-opt vyzkouší vyměnit hrany B $\rightarrow$ D a D $\rightarrow$ C a vytvoří nové hrany B $\rightarrow$ C a C $\rightarrow$ D. Ve výsledku nám vznikne nová cesta (A, B, C, D, E). Práci algoritmu můžeme vidět na obrázku 4.2. Dalším krokem algoritmu je zjistit, jestli jsme prohozením dosáhli zmenšení délky trasy. Pokud ano, změna

hran zůstane, v případě že ne, vrátíme se k původnímu stavu. V obou případech algoritmus pokračuje až do průchodu celé trasy, kdy už nejsme schopni trasu vylepšit žádným prohozením. Algoritmy 3-opt, 4 a další fungují stejně, jenom pracují s více hranami, tudíž je více možností, jak tyto hrany poskládat.



Obrázek 4.2: Na snímku můžeme vidět, jak 2-opt existující cestu (A, B, E, D, C, F, G) upraví na cestu (A, B, C, D, E, F, G). Došlo k výměně hran mezi b → e a c → f za hrany b → c a e → f, což je zjevně výhodnější trasa. [3]

Pro řešení problému obchodního cestujícího jsem si vybral genetický algoritmus, respektive jeho hybridní formu. Článek [17] mi posloužil jako inspirace pro moji prvotní implementaci.

Kapitola 5

Nové metody generování řešení TSP

V této kapitole navrhuji několik vylepšení, ať už operátorů křížení nebo návrh matice, která bude obsahovat nejbližší města, což nám pomůže v některých operátorech nebo při inicializaci populace.

5.1 Redukovaná matice nejbližších měst

V matici nejbližších měst je uložena informace, která města se určitým městům nacházejí nejblíže. Do určité velikosti je možné používat úplnou matici, ovšem s rostoucí velikostí TSP, už to není možné a ani to není efektivní. Nejkratší řešení TSP vznikají spojováním měst, která se nacházejí co možná nejblíže. Samozřejmě toto pravidlo neplatí vždy, ale v zásadě není obvyklé, že by nějaké řešení TSP blízko domnělého minima obsahovalo spojení měst, které se nenachází v blízkém okolí. Pro naše potřeby ovšem nepotřebujeme znát vzdálenosti mezi těmito nejbližšími městy určitému městu, nýbrž nám postačí pouze pořadí měst od nejbližšího po nejvzdálenější. Ovšem po inspiraci z článku [20], jsem si uvědomil, že není nutné aby matice nejbližších měst obsahovala všechna města v pořadí, ale spíš nějaké omezené množství těchto měst. Budeme znát n nejbližších měst, ke každému městu. Vystavujeme se riziku, že tato města už všechna budou obsažena v řešení TSP, ale vzhledem k paměťové náročnosti úplné matice je použití redukované matice jediný způsob, jak mít alespoň část nejbližších měst. Pro představu paměťové náročnosti je na ilustraci 5.1 vidět, jak paměťová náročnost roste. V mé implementaci má redukovaná matice nejbližších měst dvoje použití v podkapitolách níže.

Optimalizace iniciální populace

Při inicializaci populace například pro genetický algoritmus, se náhodně vygeneruje iniciální populace. Pro urychlení konvergence k nějakému řešení a pro dodání kvalitní informace do genu ovšem můžeme použít vylepšení nějakého procenta jedinců v populaci. V článku [18] po inicializaci populace projde 20 % genomu optimalizací. V našem případě vybereme 20 % náhodných genů, kdy následně vylepšíme jejich sousední město, kdy dodáme do řešení TSP informaci, která pomůže při konvergenci. V článku ovšem autoři neprovedli experimenty, jestli tento postup pomáhá optimalizovat řešení TSP, případně jaké procento genu je nejvhodnější optimalizovat.

Porovnání velikostí matic nejbližších měst

Velikost TSP	Úplná matice nejbližších měst	Redukovaná matice nejbližších měst
100 měst	40 KB	4 KB
1 000 měst	4 MB	40 KB
10 000 měst	400 MB	400 KB
100 000 měst	40 GB	4 MB

Velikost redukovaná matice je 10 nejbližších měst

Obrázek 5.1: Na příkladu vidíme, že při 100 městech rozdíl ještě není zásadní. Postupně ovšem rozdíl roste a při posledním příkladu, je redukovaná matice 10 tisíc krát menší než úplná varianta matice. Redukovaná matice roste lineárně oproti úplné matici, která roste exponenciálně. Konkrétní příklad je převzat z [20].

Heuristické operátory křížení

V operátoru VGX, a dalších operátorů, které vycházejí z VGX, v jistých případech musíme dosadit na další pozici město, které je nejblíže. V naše případě VGX, nejbližší možné město, což znamená, že dosadíme postupně vyzkoušíme všechny města v matici, dokud nenalezneme město, které není obsažené v prozatímním řešení. Pokud i všechny města jsou z redukované matice vyčerpána, musíme dosadit město náhodné.

5.2 Návrhy vylepšení operátorů křížení

Pro operátory křížení, představené v 3.1.1, jsem vymyslel několik změn, které by mohly směřovat k vylepšení schopnosti hledat optimálnější řešení TSP. Jelikož operátory z kapitoly 3.1.1 fungují různými způsoby, také způsoby vylepšení se budou lišit.

LGX

Less greedy crossover vychází z operátoru křížení VGX. V některých případech vede rychlá konvergence k zaseknutí v lokálním optimu, kterému bychom se mohli vyhnout, kdyby operátor křížení byl méně hladový. Z tohoto důvodu jsem vytvořil operátor LGX, který se liší ve výběru následujícího města, které bude dosazeno do prozatímního řešení. Ve VGX operátor zvolí souseda, který má nejkratší vzdálenost k městu. Tuto vlastnost jsem změnil a zvolil méně hladovou variantu. V operátoru LGX jsem změnil výběr souseda, který bude vložen do prozatímního řešení na výběr turnajovou metodou. Turnajová metoda je rychlá a jednoduchá, tudíž nedojde k zásadnímu zpomalení operátoru a mohli bychom docílit toho, že operátor křížení by byl méně hladový.

VGXG

Operátor křížení VGXG vychází z operátoru VGX. Jelikož máme redukovanou matici nejbližších měst, zkusíme využít těchto dat a místo hledání sousedů vyzkoušíme přímo s určitou pravděpodobností dosadit nejbližší možné město, které se nachází v redukované matici nejbližších měst. Je nutné si dát pozor na pravděpodobnost zvolení přímo nejbližších měst, jelikož se může stát, že algoritmus bude příliš hladový a brzy se zasekne v lokální optimu.

LGXG

Jelikož operátor LGX vychází z VGX operátoru, zkusíme využít i zde ve verzi LGXG dosazení s určitou pravděpodobností jednoho z nejbližších měst, nacházející se v redukované matici nejbližších měst.

PMX2-opt

PMX patří do stochastických operátorů křížení. To znamená, že konvergence k potencionálnímu minimu bude pomalejší oproti například VGX, který aktivně prohledává prostor a hledá minimální délky cest. Z tohoto důvodu, jsem chtěl dodat PMX operátoru větší sílu pro rychlejší konvergenci. Jelikož máme lokální prohledávací algoritmus 2-opt, dal by se zde využít. Na začátku křížení pomocí PMX operátoru jsme zvolili dva náhodné místa řežu v rodičích. Následně mezi těmito dvěma místy řezu se provede mapování. Ve vylepšené variantě PMX-2-opt se tato oblast mezi řezy optimalizuje pomocí lokálního prohledávacího algoritmu 2-opt. Tímto se rychleji dostane kvalitnější informace do genu a tím pádem i do populace a může se urychlit konvergence. Jelikož se 2-opt neprovede na celém genomu, ale pouze na jeho části, nemusí dojít k zaseknutí moc brzo v lokálním optimu.

OX2-opt

Na stejném principu jsme vylepšili operátor OX. Jelikož se též jedná o stochastický operátor křížení a jeho síla je slabší oproti heuristickým, je zde opět potencionál využít lokální prohledávací algoritmus 2-opt v oblasti mezi řezy. Opět může dojít k urychlení konvergence.

Vylepšení operátorů křížení a redukovaná matice nejbližších měst nám pomůže dosáhnout lepší řešení, a také urychlit konvergenci genetického operátoru. Všechny aspekty vylepšení se otestují v následující kapitole.

5.3 Návrh hybridního genetického algoritmu

Hybridní algoritmus vychází z genetického algoritmu [17]. V klasickém genetickém algoritmu probíhá cyklus, dokud není splněna ukončující podmínka nebo limit iterací. Zde je změna, kdy při zaseknutí fitness hodnoty po několik generací se spustí jeden cyklus 2-opt algoritmu na všech jedincích v populaci. Tímto bychom měli dostat řešení z lokálního minima a pokračovat v genetickém algoritmu. Schéma algoritmus je zde 2.

Algorithm 2 Hybridní genetický algortimus

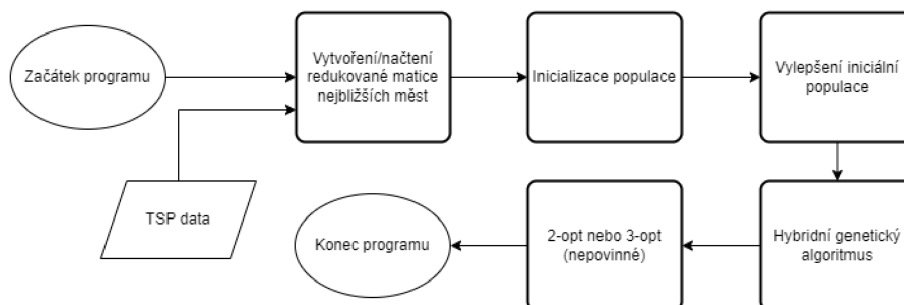
```
Inicializace populace
while není splněna ukončující podmínka genetického algoritmu do
    ohodnocení jedinců fitness funkcí
    if fitness hodnota se nezměnila po x generací then
        if neproběhl jeden cyklus 2-opt algoritmu then
            jeden cyklus 2-opt algoritmu na celé generaci
        else
            ukončení hybridního genetického algoritmu
        end if
    end if
    selekce jedinců, kteří se zúčastní reprodukce
    křížení jedinců (reprodukce) s pravděpodobností p_cross
    mutace jedince s pravděpodobností p_mut
    obnovení jedinců v populaci
end while
```

V dalším kapitole si představíme celé schéma programu, který používáme k řešení problému obchodního cestujícího, a také všechny provedené experimenty.

Kapitola 6

Experimentální výsledky

V této kapitole budou veškerá testování a experimenty. Nejdříve se zaměříme na nastavení parametrů samotného genetického algoritmu. V další části zjístíme jakých výsledků jsme s hybridními genetickými algoritmy schopni dosáhnout. Před samotným testováním charakterizujeme vstupní data a schéma programu, následně si určíme všechny nastavitelné parametry, pro každý provedeme testování, abychom jej vhodně nastavili. Zjednodušené schéma programu můžeme vidět na ilustraci 6.1.



Obrázek 6.1: Schéma zobrazuje zjednodušenou verzi programu. Oproti klasickému genetickému algoritmu je zde navíc vytvoření nebo načtení redukované matice nejbližších měst. Také jsou zde vložené jeden cyklus 2-opt algoritmu, který se spustí v případě, že genetický algoritmus se zasekne v lokálním optimu. Na samotném konci se spustí 2-opt nebo 3-opt algoritmus, který finálně vylepší řešení TSP.

Vstupní data

Zadání TSP lze reprezentovat více způsoby. Zvolená datová sada pochází z tohoto webu[1], který se věnuje TSP. Jelikož zde byla data TSP uložena pro knihovnu TSPLIB95[2] v jazyce Python, zvolil jsem tento formát. Prototypy programu vznikaly v jazyce Python, kde jsem i využíval knihovnu TSPLIB95, která se mi osvědčila, z důvodu kvalitní dokumentace a všech funkcí, které obsahuje. Formát vstupních dat obsahuje pro tuto knihovnu některé informace na začátku souboru. Jako například proměnou `NAME`, která udává název TSP, `TYPE`, která udává, jestli se jedná o symetrické či asymetrické TSP, počet měst v TSP a `COMMENT`.

[1] Odkaz na stránku, která se zabývá TSP. Autor: William J. Cook Ph.D. https://www.math.uwaterloo.ca/tsp/

[2] Dokumentace TSPLIB95 https://tsplib95.readthedocs.io/en/stable/

Nejdůležitější informace je ovšem `EDGE_WEIGHT_TYPE`, ta udává, jakým způsobem se pro toto TSP musejí počítat vzdálenosti mezi městy. Samotná města a jejich souřadnice jsou následně uložená na dalších řádcích, každé město na novém. Začátek řádku je pořadí měst, kdy následují dvě float čísla, kdy se jedná o souřadnice konkrétního měst. Datová sada obsahuje pouze `EUC_2D`, tedy vzdálenost se počítá Eukleidovskou metrikou. Tuto vlastnost za Vás vyřeší knihovna TSPLIB95, pokud programujete v jazyce Python. V mém programu jsem si naimplementoval všechny potřebné funkce v jazyce C++, kdy jsem využil podoby knihovny TSPLIB95. Pro vlastní potřebu jsem si zjednodušil i podobu vstupního souboru, kdy vstupní data mého programu mají pouze číslo města a jeho souřadnice, jelikož velikost TSP je uložená v názvu TSP a další informace nepotřebuji.

Parametry programu

Kromě klasických pravděpodobností křížení a mutací se v mém hybridním genetickém algoritmu objevují i jiné parametry, které si vyjmenujeme a specifikujeme, kdy následně ukážeme experimenty, které hledají nejvhodnější nastavení těchto parametrů.

- Velikost matice nejbližších měst.

  Tento parametr nám určí, kolik budeme znát nejbližších měst ke každému městu. Musí se zvolit dostatečně velká, aby obsahovala dostatečné množství informací ovšem není vhodné, aby byla zbytečně příliš velká a neměla přínos.

- Procento genomu, které projde vylepšením.

  Tento parametr nám určí, kolik procent genů se z genomu vylepší pomocí matice nejbližších měst. Toto vylepšení proběhne po inicializaci jedinců ovšem před vstupem do genetického algoritmu.

- Velikost populace.
- Limit iterací.
- Pravděpodobnost mutace.
- Počet double-bridge mutací při použití mutačního operátoru.
- Pravděpodobnost křížení.
- Operátor křížení.

6.1 Experimenty pro nastavení parametrů genetického algoritmu

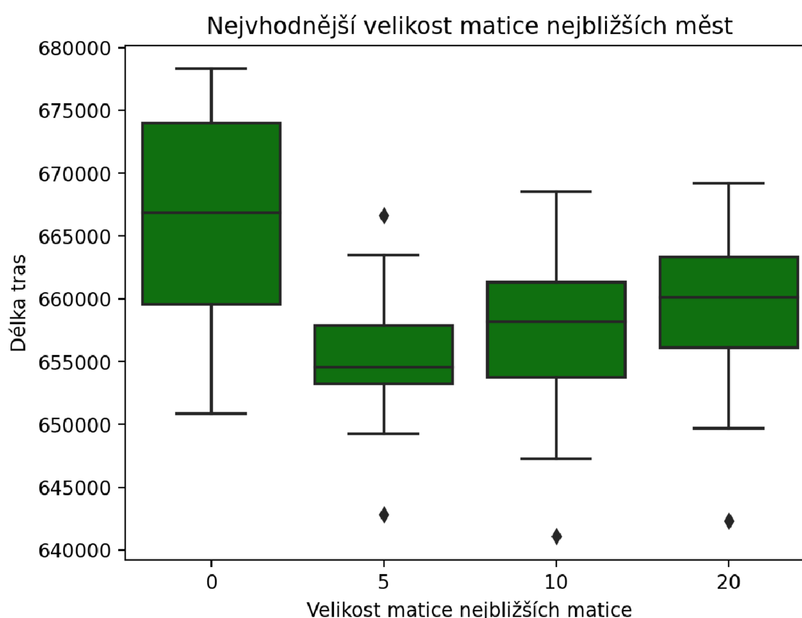
V této části představím všechny experimenty, které jsem provedl pro nalezení nejvhodnějšího nastavení parametrů. Prozatím můj program obsahuje pouze mutační operátor, v dalších částech budeme hledat i vhodné nastavení pro operátor křížení. Takto vhodně nastavíme všechny parametry a ve finálním testování provedeme porovnání všech operátorů křížení.

Velikost matice nejbližších měst

V tomto experimentu hledáme vhodnou matici nejbližších měst. Uvedeme si i variantu bez použití matice nejbližších měst, abychom pozorovali případný vliv na výsledek. Genetický algoritmus je zde bez operátorů křížení. Nastavení tohoto experimentu je následující:

- Velikost TSP: 10 150 (XMC10150),
- Limit iterací: 400 000,
- Velikost populace: 10 jedinců,
- Počet double-bridge mutací: 5,
- Počet nezávislých běhů: 20,
- Procento vylepšených genů: 20 %,
- Velikost matice nejbližších měst: 0, 5, 10, 20.

Výsledky experimentu vidíme na grafu 6.2.

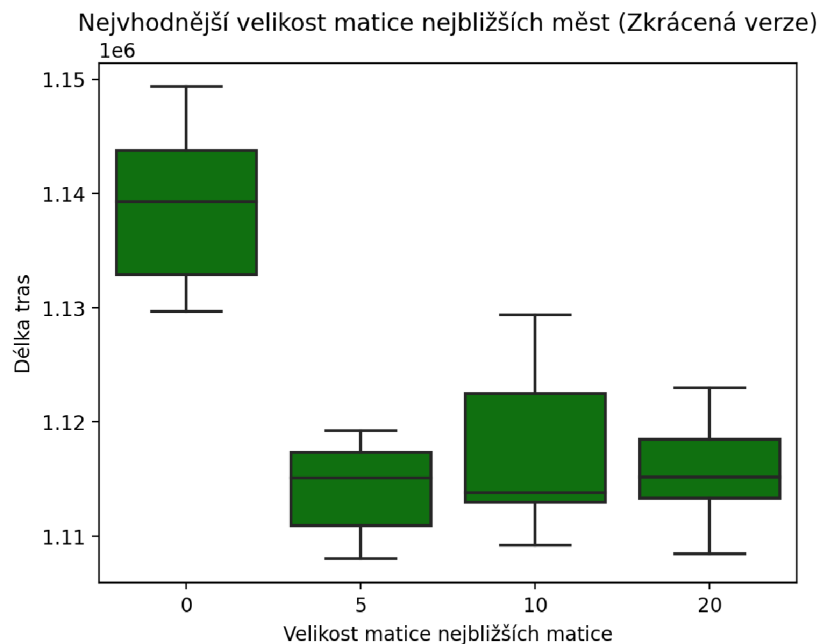


Obrázek 6.2: Na grafu vidíme výsledky experimentu. Statistické vyhodnocení vlivu velikosti matice nejbližších měst na délku trasy TSP.

Je vidět, že algoritmus získá lepší výsledky v průměru s použitím matice nejbližších měst, ovšem některé výsledky byli i lepší bez použití matice nejbližších měst než s použitím matice nejbližších měst. Větší rozdíly mezi výsledky při různých velikostech matice nelze pozorovat. Z důvodu, že nebylo zřetelnou výhodou použít matice nejbližších měst, rozhodl jsem se ještě udělat další experiment, který změní limit iterací. Nastavení zkrácené verze se pouze změní v limitu iterací:

- Velikost TSP: 10 150 (XMC10150),
- Limit iterací: 40 000,
- Velikost populace: 10 jedinců,
- Počet double-bridge mutací: 5,
- Počet nezávislých běhů: 20,
- Procento vylepšených genů: 20 %,
- Velikost matice nejbližších měst: 0, 5, 10, 20.

Výsledky experimentu vidíme na grafu 6.3.



Obrázek 6.3: Na grafu vidíme výsledky experimentu. Statistické vyhodnocení vlivu velikosti matice nejbližších měst při zmenšení limitu iterací na délku trasy TSP.

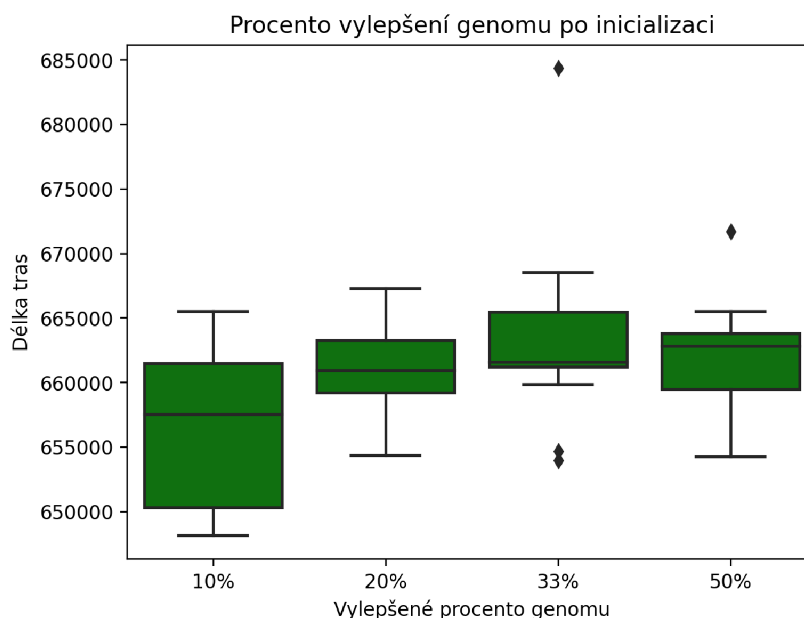
Z experimentů nám vyšlo, že použití matice nejbližších měst má vliv na výsledky algoritmu. Při menším počtu iterací se ukázalo, že konvergence k lepším výsledkům je rychlejší při použití matice nejbližších měst. Každá velikost matice nejbližších měst překonává variantu bez použití matice nejbližších měst. Dále vyšlo, že větší matice nezískávají nijak větší výhodu, tudíž můžeme zvolit variantu s pěti nejlepšími městy, kdy dosáhneme kvalitních výsledků a ušetříme místo v paměti.

Procento vylepšených genů po inicializaci

Samotné vylepšení jsem přebral z článku [18]. V článku dojde k vylepšení 20 % genů v genomu. Pro svůj program jsem se rozhodl tuto vlastnost otestovat a případně najít nejvhodnější procento genů, které vylepšíme po inicializaci. Genetický algoritmus je zde opět bez použití operátoru křížení. Nastavení tohoto experimentu je následující:

- Velikost TSP: 10 150 (XMC10150),
- Limit iterací: 400 000,
- Velikost populace: 10 jedinců,
- Počet double-bridge mutací: 5,
- Velikost matice nejbližších měst: 5,
- Počet nezávislých běhů: 20,
- Procento vylepšených genů: 10 %, 20 %, 33 %, 50 %.

Výsledky experimentu vidíme na grafu 6.4.



Obrázek 6.4: Na grafu vidíme výsledky experimentu. Statistické vyhodnocení vlivu procenta vylepšení genu na délku trasy TSP.

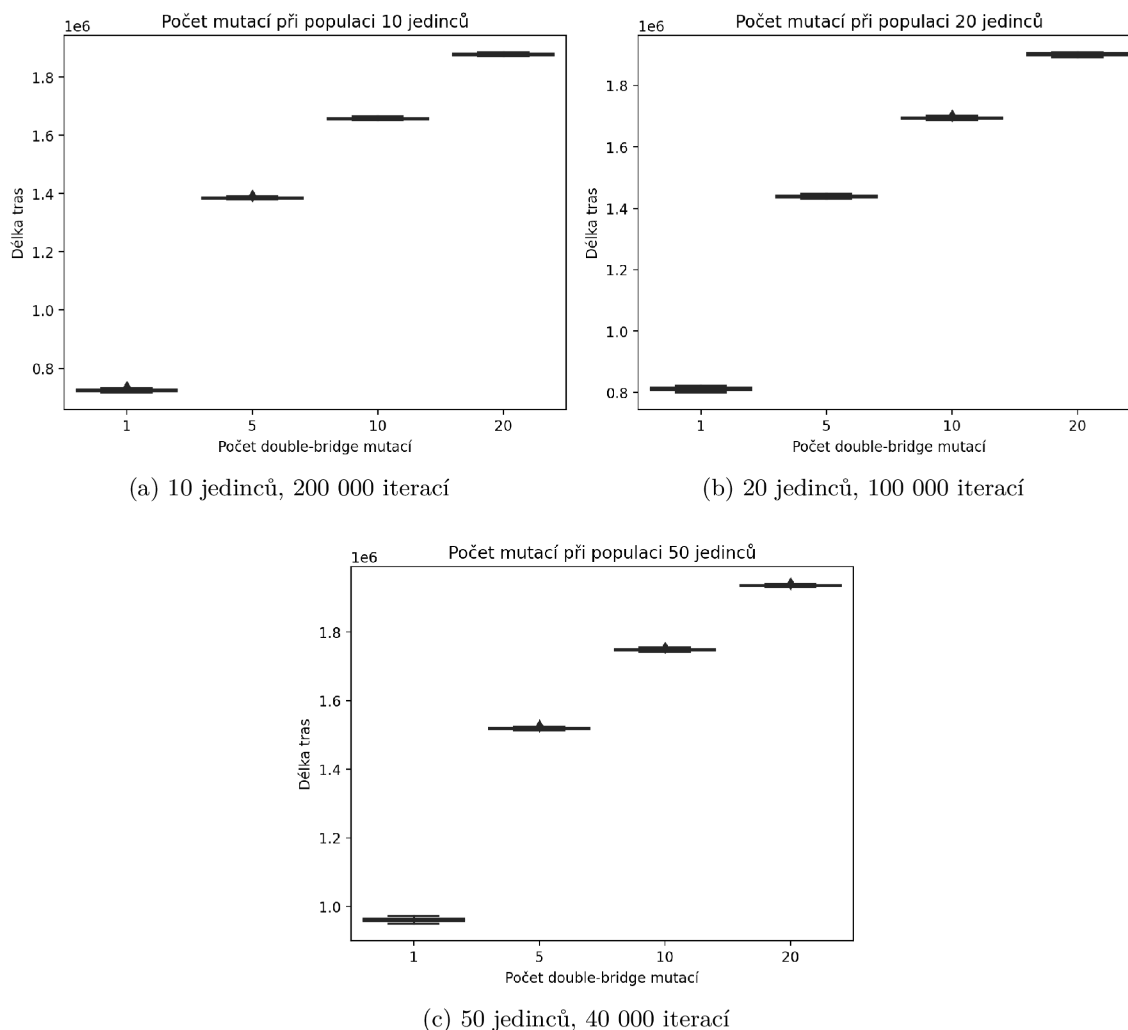
Z experimentu nám vyšlo, že nejvhodnější bude upravit 10 % genů po inicializaci. S vyšším procentem genů se výsledky postupně zhoršují, kdy přijde zlom až u 50 % vylepšených genů. Nastavením 10 % vylepšení genomu dosáhneme vnesení kvalitní informace v množství, které pomůže našemu programu dosáhnout co nejkvalitnějších výsledků.

Velikost populace a počet double-bridge mutací

V tomto experimentu porovnáme velikosti populace a počet mutací s výkonností genetického algoritmu bez operátorů křížení. Jelikož budeme experimentovat s velikostí populace, je nutné ošetřit, aby počet iterací a prohledávaný prostor byl vždy stejně velký. Nejdříve porovnáme, jak se projeví počet double-bridge mutací při různých velikostech populace. Nastavení tohoto experimentu je následující:

- Velikost TSP: 10 150 (XMC10150),
- Limit iterací: 40 000, 100 000, 200 000,
- Velikost populace: 50, 20, 10,
- Počet double-bridge mutací: 1, 5, 10, 20,
- Velikost matice nejbližších měst: 5,
- Počet nezávislých běhů: 20,
- Procento vylepšených genů: 10 %.

Výsledky experimentu vidíme na grafu 6.5.



(a) 10 jedinců, 200 000 iterací

(b) 20 jedinců, 100 000 iterací

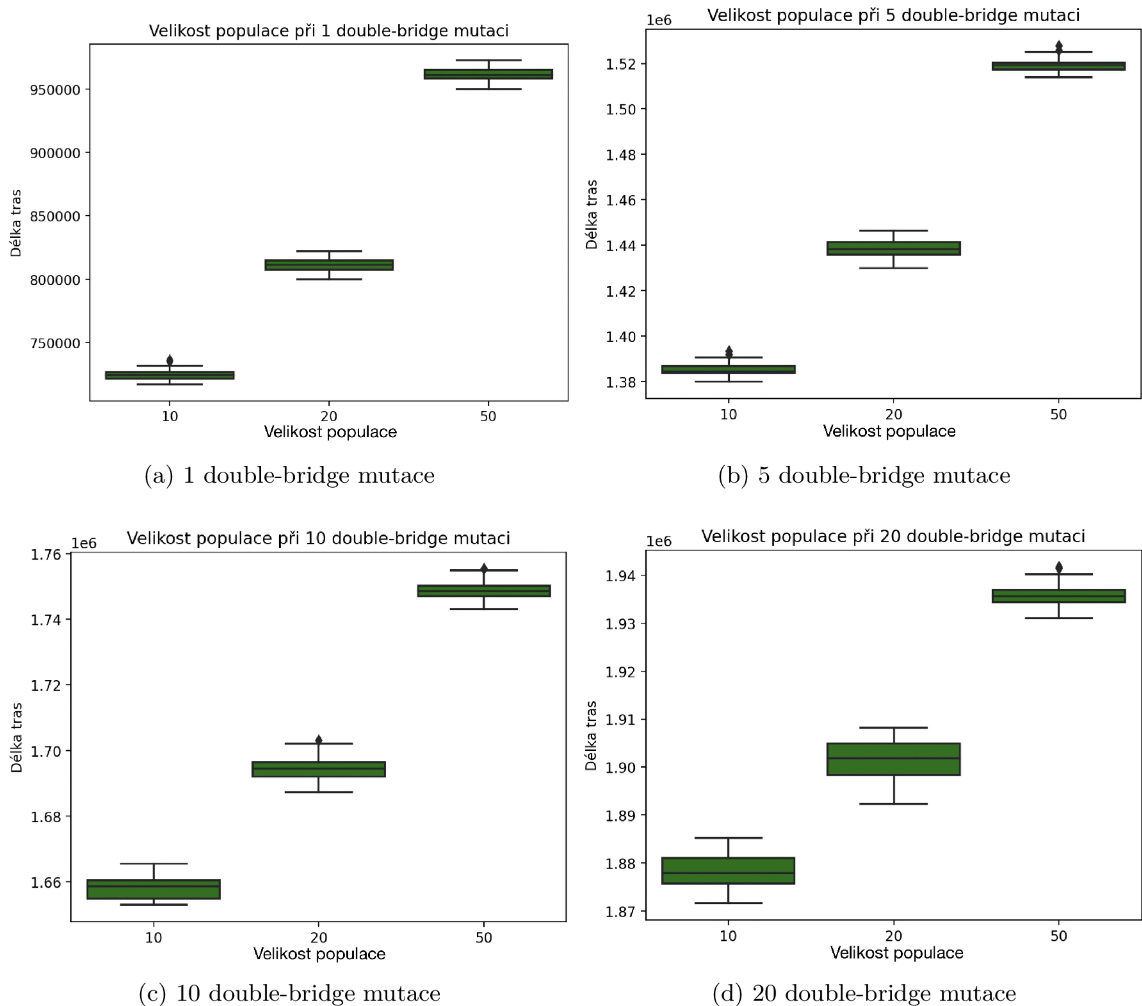
(c) 50 jedinců, 40 000 iterací

Obrázek 6.5: Na snímku vidíme výsledky experimentů. Vidíme zde 3 grafy, ukazují vliv počtu mutací při různých velikostech populace na délku tras TSP.

Z experimentu nám vyšlo, že nejvhodnější je mít mutační operátor nastavený na 1 double-bridge mutací, pro získání nejlepších výsledků. Lze na to pohlížet, že 1 double-bridge mutace

je nejcitlivější možnost, jak získat na konci nejlepší výsledek. Tato varianta je nejlepší ve všech testovaných. Také vidíme, že 1 double-bridge mutace je nejlepší variantou v každé variantě experimentu.

Nyní se zaměříme na experiment, který nám nalezne nejvhodnější počet jedinců v populaci. Tudíž porovnáme, jak se projeví změny velikosti populace při zachování počtu double-bridge mutací. Jedná se o stejné nastavení experimentu jako předtím, pouze zobrazíme jinou variantu grafů 6.6, podle které rozhodneme nejvhodnější počet jedinců v populaci.



(a) 1 double-bridge mutace

(b) 5 double-bridge mutace

(c) 10 double-bridge mutace

(d) 20 double-bridge mutace

Obrázek 6.6: Na snímku vidíme výsledky experimentů. Vidíme zde 4 grafy, které ukazují vliv velikosti populace a počet double-bridge mutací na délku tras TSP.

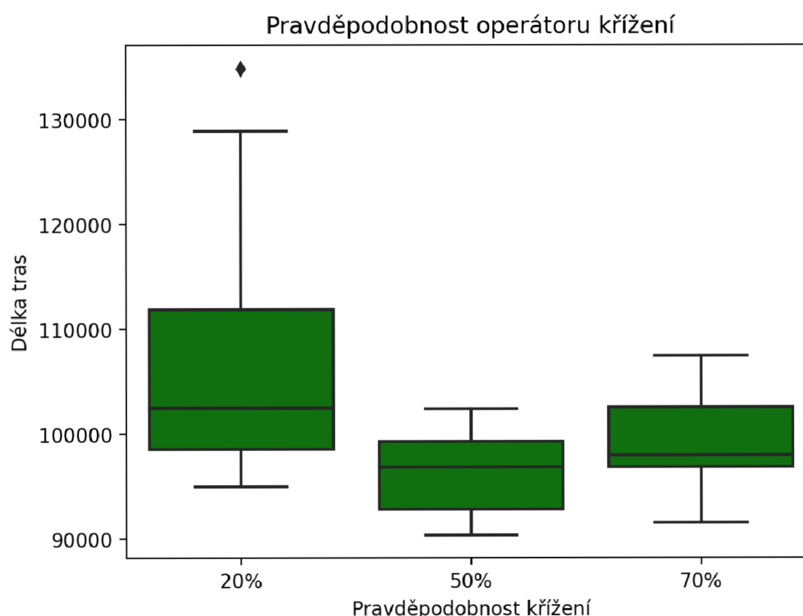
Zvolili jsme 1 double-bridge mutaci jako mutaci v našem genetickém algoritmu, ovšem vykreslil jsem i zbytek výsledků, pro lepší orientaci. Výsledky ukazují, čím menší populace, tím lepších dosahujeme výsledků. Samozřejmě při menším počtu jedinců, se prodlužuje počet iterací algoritmu. Ovšem menší počet jedinců poté následně mohl způsobit nízkou různorodost řešení a algoritmus by poté nebyl tak kvalitní. Tento aspekt by se dále mohl projevit při zavedení operátorů křížení, kdy by nízká různorodost mohla být potencionální problém. Z výsledků nám tedy vyplývá, že nejvhodnější nastavení, co se týče populace je 10 jedinců v populaci. V dalších experimentech budou tyto parametry zafixované.

Pravděpodobnost křížení

V tomto experimentu se zaměříme na nalezení nejlepší pravděpodobnosti křížení. V literatuře je doporučená 70% pravděpodobnost křížení, tudíž zahrneme tuto možnost, a ještě dvě další. Pro experiment jsme určili nejsilnější operátor křížení, který je heuristický, a to operátor VGX. Limit iterací je mnohem menší, než v předchozích iteracích, ale operátor je velice rychlý v konvergenci, tudíž nám bude stačit menší počet iterací. Nastavení tohoto experimentu je následující:

- Velikost TSP: 10 150 (XMC10150),
- Limit iterací: 1 000,
- Velikost populace: 10,
- Počet double-bridge mutací: 1,
- Pravděpodobnost mutace: 30%,
- Velikost matice nejbližších měst: 5,
- Počet nezávislých běhů: 20,
- Procento vylepšených genů: 10 %,
- Operátor křížení: VGX,
- Pravděpodobnost křížení: 20%, 50%, 70%.

Výsledky experimentu vidíme na grafu 6.7.



Obrázek 6.7: Na grafu vidíme výsledky experimentu. Graf ukazuje vliv pravděpodobnosti řešení na výsledky TSP.

Nejlepšího výsledku dosahujeme při 50% pravděpodobnosti křížení, 20% je příliš malá pravděpodobnost a v některých případech se vůbec nepřiblížíme hranici výkonnosti křížení operátoru VGX a při 70% se opět příliš rychle zasekneme v lokálním optimu, ze kterého se potom už nedostaneme. Tudíž nejvhodnější nastavení pravděpodobnosti křížení je 50%.

6.2 Úplné výsledky experimentů

Po předchozích experimentech, kde jsme si určili všechny parametry programu budeme testovat výkonnost programu na různě velkých TSP. Tyto experimenty budou 4.

- Experiment s genetických algoritmem
- Experiment s genetickým algoritmem + zkrácený 2-opt algoritmus v průběhu
- Experiment s genetickým algoritmem + úplný 2-opt algoritmus
- Experiment s genetickým algoritmem + zkrácený 2-opt v průběhu + 3-opt na konci evoluce

Tyto experimenty nám řeknou sílu samotných genetických operátorů, a zejména operátorů křížení. Dále se dozvíme, jak velký vliv má kombinace 2-opt a 3-opt algoritmů na výsledky řešení TSP. Nastavení následujících experimentů je následující:

- Velikost TSP: 29 až 25 tisíc (jedná o se TSP měst v různých státech),
- Limit iterací: 10 000,
- Velikost populace: 10,
- Počet double-bridge mutací: 1,
- Pravděpodobnost mutace: 30%,
- Velikost matice nejbližších měst: 5,
- Počet nezávislých běhů: 20,
- Procento vylepšených genů: 10 %,
- Pravděpodobnost křížení: 50%,
- Operátor křížení: VGX, PMX, OX, CX, VGXG, LGX, LGXG, PMX2-Opt, OX2-opt. Výsledky experimentů vidíme v tabulkách 6.1, 6.2, 6.3 a 6.4.

Genetický algoritmus								
	29	194	929	4663	7146	9847	16862	24978
VGX	7.21	34.46	90.34	390.51	393.05	646.32	595.23	741.73
LGX	3.49	40.32	90.75	379.75	397.04	597.33	615.08	750.22
PMX	27.59	213.31	649.13	-	-	-	-	-
CX	29.09	216.80	649.35	-	-	-	-	-
OX	29.69	225.64	664.01	-	-	-	-	-
VGXG	16.36	41.94	127.10	658.44	656.93	1072.52	1051.04	1084.72
LGXG	14.48	44.24	138.37	623.81	656.45	1073.53	1069.46	1111.77
PMX2-Opt	8.68	41.33	43.80	52.85	45.83	40.43	46.59	41.01
OX2-Opt	8.65	28.45	33.88	36.09	44.42	41.88	48.37	44.47

Tabulka 6.1: V tabulce vidíme vyhodnocení experimentu. Můžeme pozorovat vliv velikosti TSP na výsledky pouze genetického algoritmu. Na levé straně tabulky máme operátory křížení. V horním řádku máme velikost problému obchodního cestujícího. Dále každé číslo je procentuální rozdíl průměru dvaceti běhů od známého optima. Uvedu zde příklad. Optimum je 100 a průměr 20 běhu je 120, tudíž procentuální rozdíl je 20 %. Vzorec pro výpočet je ((AVG([výsledky 20 běhů])/(známé optimum))-1)*100. V místě pomlček byly výsledky tak nekvalitní, že jsem je ani neuváděl (až 3000 a více procent od optima).

Při použití pouze genetického algoritmu nedosahujeme kvalitních výsledků. Některé operátory křížení dosáhnout výsledku do deseti procent od známého optima, ovšem pouze u nejmenšího TSP. Už pro TSP se 194 městy začíná být problémové najít řešení v rozumné délce oproti známému optimu. Od tisíců a více měst začne být opravdu problémové najít vůbec nějaké blízkosti řešení TSP. Heuristické operátory křížení dávají lepší výsledky, než stochastické operátory křížení. Stochastické operátory křížení jsou zde opravdu špatné, kdy od tisíců měst mají vzdálenost od optima přes 3 tisíce procent, z tohoto důvodu jsem je zde ani nezahrnoval. Nejlepší výsledky pouze genetického algoritmu jsou ovšem operátory s lokálním prohledáváním 2-opt. Drží se v okolí 40 % od nejlepšího známého optima. Je to dáno tím, že se může stát, že například dva náhodné řezy budou od sebe vzdálené téměř celý genom, tudíž proběhne jeden cyklus 2-opt algoritmu. Z tohoto experimentu se dá vyvodit, že pouze genetický algoritmus zde není schopen řešit kvalitně TSP.

Genetický algoritmus + zkrácený 2-opt algoritmus v průběhu								
	29	194	929	4663	7146	9847	16862	24978
VGX	3.42	20.95	24.66	39.32	37.00	38.52	46.41	45.36
LGX	1.70	13.57	25.11	39.64	39.64	44.29	42.97	42.51
PMX	8.08	42.42	60.02	76.90	74.83	68.50	70.91	73.97
CX	6.69	43.11	58.35	74.43	72.42	67.67	78.84	77.40
OX	3.13	39.39	63.06	71.29	74.81	69.71	83.41	77.98
VGXG	3.26	20.44	25.70	40.16	43.04	43.06	45.59	44.89
LGXG	3.74	15.10	25.14	34.37	41.31	41.57	42.62	45.66
PMX2-Opt	4.09	19.13	20.43	26.66	24.41	24.32	23.24	25.15
OX2-Opt	5.80	12.72	14.36	21.66	19.36	20.24	18.39	24.05

Tabulka 6.2: V tabulce vidíme vyhodnocení experimentu. Můžeme pozorovat vliv velikosti TSP na výsledky genetického algoritmu s jedním během 2-opt algoritmu. Systém tabulky zůstává stejný jako v tabulce 6.1.

V experimentech, jsme použili s genetickým algoritmem i lokální prohledávací algoritmus 2-opt, ovšem pouze jeden cyklus 2-opt v průběhu, kdy jsme se poté vrátili ke genetickému algoritmu. V tomto případě jsme posunuli výsledky u všech velikostí TSP. Už se nevyskytuje řešení TSP, které by bylo nad 80 % od nejlepšího známého optima. Tohoto efektu jsme dosáhli pouze jedním cyklem lokálního prohledávání 2-opt. V tomto případě využíváme rychlé konvergence řešení TSP k nějakému lokálnímu minimu genetického algoritmu. Následně dále vylepšíme řešení pomocí jednoho cyklu 2-opt, kdy následně spustíme opět genetický algoritmus. V tomto experimentu dosahují nejlepších výsledků operátory křížení s lokálním prohledáváním a to PMX-2opt a OX-2opt. Dochází zde více k použití lokálního prohledávacího algoritmu 2-opt a výsledky se zlepšují.

Genetický algoritmus + úplný 2-opt algoritmus								
	29	194	929	4663	7146	9847	16862	24978
VGX	1.92	5.97	8.10	10.00	11.90	11.22	11.91	12.20
LGX	3.81	5.96	9.25	9.13	11.98	13.16	13.21	13.65
PMX	1.01	11.17	9.33	10.21	12.36	12.71	16.71	15.43
CX	2.85	8.88	7.77	11.86	13.18	12.68	14.27	15.35
OX	2.06	7.34	8.30	11.99	11.75	12.76	15.35	15.50
VGXG	1.84	5.25	8.54	10.00	13.43	11.57	12.07	12.43
LGXG	3.09	6.95	8.37	10.89	11.19	11.22	12.95	14.01
PMX2-Opt	1.97	8.35	9.00	11.02	11.75	12.45	13.89	12.85
OX2-Opt	2.46	8.52	9.74	10.56	11.77	12.18	12.80	11.12

Tabulka 6.3: V tabulce vidíme vyhodnocení experimentu. Můžeme pozorovat vliv velikosti TSP na výsledky genetického algoritmu s 2-opt algoritmem. Systém tabulky zůstává stejný jako v tabulce 6.1.

Při experimentu, kdy průběh je stejný jako v předchozím, ovšem po skončení genetického algoritmu, se ještě provede lokální algoritmus 2-opt v plné verzi. Zde dosahujeme výsledků pod 10 % od známého optima do velikosti 1000 měst, které jsem určil za ještě triviální. Od 1000 měst se jedná o netriviální TSP a zde jsme lehce nad 10 % od optima. Tato varianta se jeví jako nejoptimálnější, kdy časová náročnost a výsledky, které přináší jsou nejlepší kombinací. V dalším experimentu ukážeme částečné výsledky experimentu s 3-opt variantu.

Genetický algoritmus + zkrácený 2-opt v průběhu + 3-opt na konci evoluce								
	29	194	929	4663	7146	9847	16862	24978
VGX	0.29	4.29	3.04	6.61	X	X	X	X
LGX	0.90	7.22	5.80	7.04	X	X	X	X
PMX	2.42	3.76	4.88	8.71	X	X	X	X
CX	1.35	4.75	5.18	7.03	X	X	X	X
OX	0.94	5.20	8.00	7.83	X	X	X	X
VGXG	6.94	9.66	5.82	6.22	X	X	X	X
LGXG	4.63	5.40	7.21	9.51	X	X	X	X
PMX2-Opt	1.47	6.35	6.79	8.06	X	X	X	X
OX2-Opt	1.21	3.52	7.41	7.37	X	X	X	X

Tabulka 6.4: V tabulce vidíme vyhodnocení experimentu. Můžeme pozorovat vliv velikosti TSP na výsledky genetického algoritmu s jedním během 3-opt algoritmem. Systém tabulky zůstává stejný jako v tabulce 6.1.

Lokální algoritmus 3-opt je velice časově náročný, z tohoto důvodu nebylo možné dokončit všechny TSP. Dosažené výsledky jsou všechny pod 10 % od optima, což už je velice blízko známému optimu. Ovšem časová náročnost je mnohonásobně větší než u 2-opt. Porovnání časové náročnosti se budeme věnovat v další sekci.

6.3 Vývoj hodnot fitness funkcí a časové náročnosti

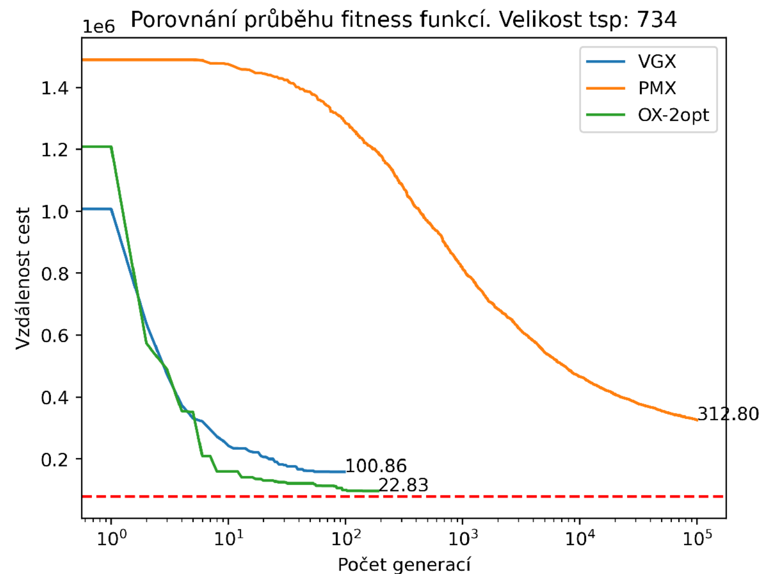
Porovnávali jsme zde různé druhy operátorů křížení. Máme 3 druhy. Jedna varianta je heuristická, kdy tvoříme jedince nějakým rozhodováním nebo na lokální úrovni (VGX, LGX, VGXG, LGXG). Druhá varianta je s lokálním prohledávacím algoritmem 2-opt, kdy na část řešení použijeme jeden cyklus 2-opt algoritmu (PMX-2opt, OX-2opt). Třetí varianta je stochastická, kdy pouze jednoduchým operátorem křížení vytváříme nová řešení TSP (PMX, OX, CX). Tyto různé druhy dosahují podobných výsledku ovšem různými způsoby. Dále jsme zkoumali jakých výsledků dosahují různé varianty genetických algoritmů a jejich kombinaci s nějakými lokálními prohledávacími algoritmy. Tyto varianty dosahují různých kvalit řešení, ovšem i zde si představíme vývoj hodnot jejich fitness a časové náročnosti, jelikož i ta je určující.

Porovnání variant operátorů křížení

Máme 3 druhy operátorů křížení, které se chovají jinak. Nyní si porovnáme tyto různé druhy. Nastavení experimentů je následující:

- Velikost TSP: 734,
- Limit iterací: 100 000,
- Velikost populace: 10,
- Počet double-bridge mutací: 1,
- Pravděpodobnost mutace: 30%,
- Velikost matice nejbližších měst: 5,
- Procento vylepšených genů: 10 %,
- Pravděpodobnost křížení: 50%,
- Operátor křížení: VGX, PMX a OX-2opt.

Výsledek experimentu můžeme vidět na grafu 6.8.



Obrázek 6.8: Na grafu vidíme výsledky experimentu. Osa X, která znázorňuje počet iterací, je logaritmická. Vidíme zde průběh fitness funkcí různých operátorů křížení.

Jak vidíme, operátory VGX a OX-2opt, tedy ty heuristické nebo které obsahují lokální algoritmus mají křivku, která je spíše klasická pro genetické algoritmy. Nejdříve strmě klesá, kdy následně dojde k vyrovnání a zmírnění klesání, kdy nakonec hodnota fitness neklesá vůbec a genetický operátor je ukončen. Oproti tomu operátor PMX, který je operátor křížení, který je stochastický, klesá velice pozvolna a udržuje si toto tempo klesání. Zastavení genetického algoritmu při použití operátoru křížení PMX nastane až v samotném limitu iterací.

Z experimentu je nám tedy známý vývoj hodnot fitness funkcí a nyní si porovnáme časovou náročnost různých druhů operátorů. Uvedeme si, jak dlouho trval vývoj těchto křivek z experimentu výše a výsledky vidíme zde 6.5.

Časová náročnost různých operátorů křížení + vzdálenost od optima		
VGX	PMX	OX-2opt
15 sec (100.86 % od optima)	2613 sec (312.80 % od optima)	104 sec (22.83 % od optima)

Tabulka 6.5: V tabulce vidíme vyhodnocení experimentu. Můžeme pozorovat vliv operátora křížení na čas řešení TSP.

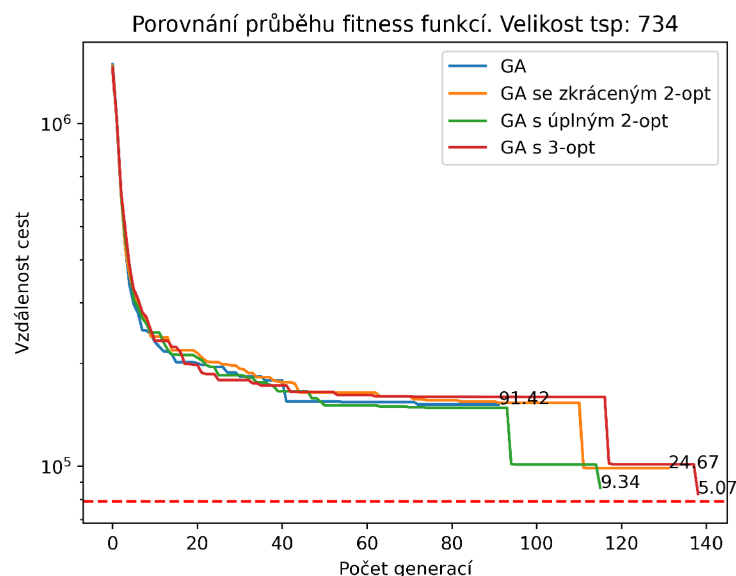
Z časové náročnosti vidíme, že operátor křížení PMX trvá mnohonásobně větší dobu než ostatní operátory křížení a dosahuje i nejhorších výsledků. Z toho vychází, že prosté operátory křížení, nejsou dostatečné pro řešení TSP v porovnání s heuristickými nebo stochastických s lokálním algoritmem. Nyní můžeme porovnávat heuristické a stochastické s lokálním algoritmem. Výhoda heuristického operátoru křížení je rychlost, oproti stochastickým operátorům s lokálním algoritmem. Za sedminu času jsme získali přibližně o 75 % horší řešení. Musíme brát v úvahu, že se jedná pouze o genetický algoritmus na menším TSP. V případě větších TSP je zcela jasné, že pouhý genetický algoritmus nestačí a je nutné použití nějakou kombinace genetického algoritmu a dalších algoritmů.

Porovnání variant genetického algoritmu a hybridů genetického operátoru

Jelikož jsme prováděli experimenty na 4 variantách genetického algoritmu, ukážeme si opět vývoj hodnot fitness funkcí a porovnáme časovou náročnost. Nastavení experimentů je následující:

- Velikost TSP: 734,
- Limit iterací: 10 000,
- Velikost populace: 10,
- Počet double-bridge mutací: 1,
- Pravděpodobnost mutace: 30%,
- Velikost matice nejbližších měst: 5,
- Procento vylepšených genů: 10 %,
- Pravděpodobnost křížení: 50%,
- Operátor křížení: VGX,
- Druhy GA: GA, GA se zkráceným 2-opt, GA s úplným 2-opt a GA s 3-opt.

Výsledek experimentu můžeme vidět na grafu 6.9.



Obrázek 6.9: Na grafu vidíme výsledky experimentu. Můžeme pozorovat vliv výběru varianty hybridního genetického algoritmu na vývoj křivky historie fitness funkcí.

Vidíme, že prostý genetický algoritmus se zasekne v lokálním optimu a dále nepokračuje. Vylepšené varianty genetického algoritmu pokračují dále a propady fitness křivek na nižší hodnoty fitness jsou kroky, kdy se použije nějaká varianta lokálního algoritmu. Vidíme, že zkrácená verze končí okolo 25 % od optima, oproti úplné variantě 2-opt, která končí okolo 10 % anebo dokonce 3-opt variantě, která končí pouze 5 % od optima. Z experimentu je nám tedy známý vývoj hodnot fitness funkcí a nyní si porovnáme časovou náročnost různých variant genetických algoritmů. Uvedeme si, jak dlouho trval vývoj těchto křivek z experimentu výše, výsledky můžeme vidět 6.6.

Časová náročnost různých variant genetických algoritmů			
GA	GA se zkráceným 2-opt	GA s úplným 2-opt	GA s 3-opt
15 sec (91.42 % od optima)	24 sec (24.67 % od optima)	49 sec (9.34 % od optima)	381 sec (5.07 % od optima)

Tabulka 6.6: V tabulce vidíme vyhodnocení experimentu. Můžeme pozorovat vliv varianty genetického algoritmu na čas řešení TSP.

Časová náročnost pouze genetického algoritmu je nejmenší ovšem získané výsledky jsou nejhorší. Za dvojnásobný čas genetického algoritmu se zkráceným 2-opt jsme získali téměř 4krát lepší výsledky. Tyto výsledky jsme schopni ještě více vylepšit úplnou verzí 2-opt algoritmu, kdy získáme při dvojnásobném času opět okolo 3krát lepší výsledky, kdy je cena za vylepšení přijatelná. Zlom nastává u 3-opt varianta. Zde sice získáme 2krát lepší výsledky než u úplného 2-opt algoritmu za cenu 8 násobného času. Tato varianta je tudíž příliš časově náročná, kdy ještě na TSP do velikost 1000 měst ji lze použít ovšem pro větší velikosti při časové náročnosti $\mathcal{O}(n^3)$, za pouze 2krát lepší výsledky, kdy se s genetickým algoritmem

a úplnou variantou 2-opt pohybujeme okolo 10 % od optima. Z tohoto důvodu mi přijde jako nejlepší varianta genetický algoritmus + úplný 2-opt algoritmus.

Zhodnocení experimentů

V provedených experimentech jsme zjistili jakých výsledků je schopen dosáhnout každý operátor křížení, který je u genetických algoritmů zásadní. Dále jsme porovnali časové náročnosti různých druhů genetických algoritmů a různých variant operátorů křížení. Pokud bychom pracovali pouze s genetickým algoritmem, jsou operátory PMX-2opt a OX-2opt, které jsem vytvořil v této práci, nejúspěšnější ze všech, kdy dosahují výsledků do 50 % od optima. V další variantě genetického algoritmu se zkrácenou verzí 2-opt během evoluce jsou opět operátory PMX-2opt a OX-2opt nejúspěšnější, kdy dosahují výsledků do 25 % od optima. Pomocí varianty s úplnou verzí 2-opt algoritmu získáváme výsledky do 13 % od optima. Nyní je i důležité časové srovnání. V tomto případě je genetický algoritmus s úplnou verzí 2-opt při velikosti 734 měst 2krát pomalejší než varianta se zkrácenou verzí 2-opt ovšem získáváme přibližně 1,5krát lepší výsledky. Nyní už záleží na případu užití, jestli dbáme na rychlost nebo na nalezení co nejnižší délky trasy. Od tohoto se odvýjí, jakou variantu bychom preferovali. Varianta genetického operátoru s 3-opt je už opravdu časově náročná. Výsledky jsou lepší, ale také časová náročnost $\mathcal{O}(n^3)$ je mnohonásobě větší, tudíž tato varianta není nejvhodnější.

Při porovnání křivek hodnot získané fitness funkcí jsme zjistili, že stochastické operátory křížení, se nechovají klasicky jak jsme zvyklí u genetických algoritmů a jejich klesání je opravdu pozvolné. Tento druh operátorů bych nedoporučoval, jelikož jsou i časově náročnější oproti heuristickým anebo s lokálním prohledáváním.

Výkonnost operátorů křížení jsme získali v první variantě genetického algoritmu. Ke srovnání pouhého genetického algoritmu v jiné práci nedošlo, jelikož ostatní práce pracují většinou s TSP do velikost 1000 měst. Jiné práce, které by kombinovali lokální prohledávací algoritmus s genetickým algoritmem jsem nenašel, a tudíž je neporovnávám. V další sekci představím další možné varianty, úprav či přístupů k řešení TSP, které by pomohli získat ještě lepší výsledky za menší časovou náročnost.

Potencionální přístupy a úpravy

Jedním z prvních vylepšení by byla nějaká akcelerovaná nebo vylepšená verze lokálního prohledávání. V práci [19] je představena varianta kombinací 2 až 4 opt s k-swaps, což jsou obecně double-bridge mutace. Dále tam představují možnosti zrychlení samotného algoritmu seznamem kandidátů, či greedy startem. Tato práce by mohla vylepšit lokální algoritmus v mém genetickém přístupu a získat lepší výsledky.

Další možností vylepšení by byla clusterizace či nějaké rozdělení na oblasti zadání TSP. V práci [8] je prezentováno rozdělení zadání TSP do clusterů, která následně řešíme odděleně. V tomto případě se zde i nabízí nějaká paralelizace takto rozděleného řešení, které bychom následně spojili. Zde je také použitá Lin-Kernighan heuritiska, kde se jedná o adaptivní k-opt, kdy k je proměnlivé. LKH se také nabízí jako potencionální vylepšení. Dále je také možnost rozdělit problém TSP pomocí bloků do mřížky, kdy následně můžeme každý blok řešit odděleně a následně spojit výsledek do konečného výsledku.

Kapitola 7

Závěr

V práci jsme popsali problém obchodního cestujícího, reprezentaci řešení obchodního cestujícího a náročnost problému obchodního cestujícího. Dále pak jsme popsali evoluční algoritmy, příklady evolučních algoritmu, a jak řeší TSP. Dále jsme se konktrétněji zaměřili na genetické algoritmy, které bude použity pro řešení TSP v této diplomové práci. Popsali jsme reprezentaci genotypů, co znamená fenotyp, dále pak procesy z evoluce jako křížení, mutace a selekce. Všechny tyto operace jsou popsány na příkladech a ilustrovány. Uvedli jsme si existující operátory křížení, které se používají pro problémy obchodního cestujícího

Dále jsme si představili mnou navrhnutá vylepšení, která měla pomoct řešení TSP. Matice nejbližších měst je zásadní, jelikož nám umožnuje použít heuristické operátory křížení a také nám umožňuje vylepšit náhodně prvotně vygenerovaná řešení TSP, kdy urychlíme konvergenci. Dále jsme si představili nové operátory křížení, která vycházejí z operátorů křížení z literatury. Přínosy těchto navrhnutých řešení byly představeny nebo otestovány.

Další fáze byla experimentální, kde jsme si krátce představili schéma programu a popsali si vstupní data. Dále jsme si v první části nejdříve hledali pomocí experimentů nejvhodnější nastavení parametrů, kdy jsme i pozorovali, jestli mají parametry vliv na výsledky. Všechno bylo řádně zdokumentováno a výsledky byly vykresleny pomocí boxplotů. Po nastavení všech parametrů jsme otestovali plnou verzi genetického algoritmu a jeho různých vylepšených variant na sadě problémů obchodního cestujícího. Velikost TSP byla od 29 do 25 tisíc měst. Netriviální problémy obchodního cestujícího byli od 1000 měst a výše, kdy jsme se soustředili hlavně na tyto. Všechny výsledky jsme si okomentovali. Dále jsme se věnovali časové náročnosti a vývoji hodnot fitness funkcí. V této časti jsme porovnali vývoj hodnot fitness funkcí u různých druhů operátorů křížení, a také u různých variant genetického operátoru. Došlo i na zhodnocení časových rozdílů u těchto variant.

V poslední části jsme si představili další možná vylepšení současného stavu řešení. Dle mého se limit výsledků genetického algoritmu už vyčerpal, kdy následné vylepšení se bude muset týkat vylepšení lokálního prohledávacího algoritmu, kde je ocitovaný článek, který se tomuto věnuje. Další varianty jsou clusterizace zadání TSP, kdy je možnost lépe vyřešit menší dílčí části TSP a ty následně spojit do finálního řešení. Zde se i nabízí možnosti nějaké paralelizace.

Problém obchodního cestujícího jsem vyřešil kombinací rychlé konvergence k přibližnému řešení pomocí genetického algoritmu, kdy jsem následně změnil přístup na lokální prohledávací algoritmus a vyřešil jsem problémy TSP do lokálního minima okolo 10% od známého optima.

Literatura

[1] Applegate, D. L., Bixby, R. E., Chvatal, V. a Cook, W. J. *The traveling salesman problem*. Princeton, NJ: Princeton University Press, leden 2007. Princeton Series in Applied Mathematics.

[2] Blum, C. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*. 2005, sv. 2, č. 4, s. 353–373. DOI: https://doi.org/10.1016/j.plrev.2005.10.001. ISSN 1571-0645. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1571064505000333.

[3] Brabazon, A., O'Neill, M. a McGarraghy, S. *Natural Computing Algorithms*. Springer Berlin Heidelberg, 2015. Natural Computing Series. ISBN 9783662436318. Dostupné z: https://books.google.ie/books?id=wAi0CgAAQBAJ.

[4] Croes, G. A. A Method for Solving Traveling-Salesman Problems. *Operations Research*. INFORMS. 1958, sv. 6, č. 6, s. 791–812. ISSN 0030364X, 15265463. Dostupné z: http://www.jstor.org/stable/167074.

[5] Darwin, C. *On the Origin of Species by Means of Natural Selection*. London: Murray, 1859. Or the Preservation of Favored Races in the Struggle for Life.

[6] Davis, L. Handbook of genetic algorithms. CumInCAD. 1991.

[7] Held, M. a Karp, R. M. A Dynamic Programming Approach to Sequencing Problems. *Journal of the Society for Industrial and Applied Mathematics*. Society for Industrial & Applied Mathematics (SIAM). březen 1962, sv. 10, č. 1, s. 196–210. DOI: 10.1137/0110015. Dostupné z: https://doi.org/10.1137/0110015.

[8] Helsgaun, K. Solving the Clustered Traveling Salesman Problem Using the Lin-Kernighan-Helsgaun Algorithm. Květen 2014.

[9] Holland, J. H. *Adaptation in natural and artificial systems*. Cambridge, MA: Bradford Books, červen 2019. Complex Adaptive Systems.

[10] Hussain, A., Muhammad, Y. S., Sajid, M. N., Hussain, I., Shoukry, A. M. et al. Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator. *Computational Intelligence and Neuroscience*. Hindawi Limited. 2017, sv. 2017, s. 1–7. DOI: 10.1155/2017/7430125. Dostupné z: https://doi.org/10.1155/2017/7430125.

[11] Ismkhan, H. Accelerating the ANT Colony Optimization By Smart ANTs, Using Genetic Operator. *International Journal on Computational Science & Applications*. Listopad 2014, sv. 4. DOI: 10.5121/ijcsa.2014.4208.

[12] Ismkhan, H. a Zamanifar, K. Study of Some Recent Crossovers Effects on Speed and Accuracy of Genetic Algorithm, Using Symmetric Travelling Salesman Problem. *International Journal of Computer Applications*. Říjen 2013, sv. 80, s. 1–6. DOI: 10.5120/13862-1716.

[13] Krasnogor, N., Nicosia, V., Pavone, M. a Pelta, D. A., ed. *Nature inspired cooperative strategies for optimization (NICSO 2007)*. 2008. vyd. Berlin, Germany: Springer, květen 2008. Studies in Computational Intelligence.

[14] Laarhoven, P. J. M. van a Aarts, E. H. L. Simulated annealing. In: *Simulated Annealing: Theory and Applications*. Dordrecht: Springer Netherlands, 1987, s. 7–15.

[15] Laporte, G. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*. Elsevier BV. červen 1992, sv. 59, č. 2, s. 231–247. DOI: 10.1016/0377-2217(92)90138-y. Dostupné z: https://doi.org/10.1016/0377-2217(92)90138-y.

[16] Lawler, E. *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, 1985. Wiley-Interscience series in discrete mathematics and optimization. Dostupné z: https://books.google.ie/books?id=qbFlMwEACAAJ.

[17] Lin, B., Sun, X. a Salous, S. Solving Travelling Salesman Problem with an Improved Hybrid Genetic Algorithm. *Journal of Computer and Communications*. Leden 2016, sv. 04, s. 98–106. DOI: 10.4236/jcc.2016.415009.

[18] Lin, B., Sun, X. a Salous, S. Solving Travelling Salesman Problem with an Improved Hybrid Genetic Algorithm. *Journal of Computer and Communications*. Scientific Research Publishing, Inc. 2016, sv. 04, č. 15, s. 98–106. DOI: 10.4236/jcc.2016.415009. Dostupné z: https://doi.org/10.4236/jcc.2016.415009.

[19] Misevicius, A. Combining 2-OPT, 3-OPT and 4-OPT with K-SWAP-KICK perturbations for the traveling salesman problem. In:. Leden 2011.

[20] Peake, J., Amos, M., Yiapanis, P. a Lloyd, H. Scaling Techniques for Parallel Ant Colony Optimization on Large Problem Instances. In:. červenec 2019. DOI: 10.1145/3321707.3321832.

[21] Randerson, J. *Double mutant gene eliminates malaria risk*. New Scientist, Nov 2001. Dostupné z: https://www.newscientist.com/article/dn1566-double-mutant-gene-eliminates-malaria-risk/.

[22] Skinderowicz, R. Improving Ant Colony Optimization efficiency for solving large TSP instances. *Applied Soft Computing*. 2022, sv. 120, s. 108653. DOI: https://doi.org/10.1016/j.asoc.2022.108653. ISSN 1568-4946. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1568494622001302.

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

[24] Talbi, E.-G. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

[25] Umbarkar, Walchand College of Engineering, Sheth a Government College of Engineering, Karad. Crossover operators in genetic algorithms: A review. *ICTACT J. Soft Comput.* ICT Academy. říjen 2015, sv. 06, č. 01, s. 1083–1092.

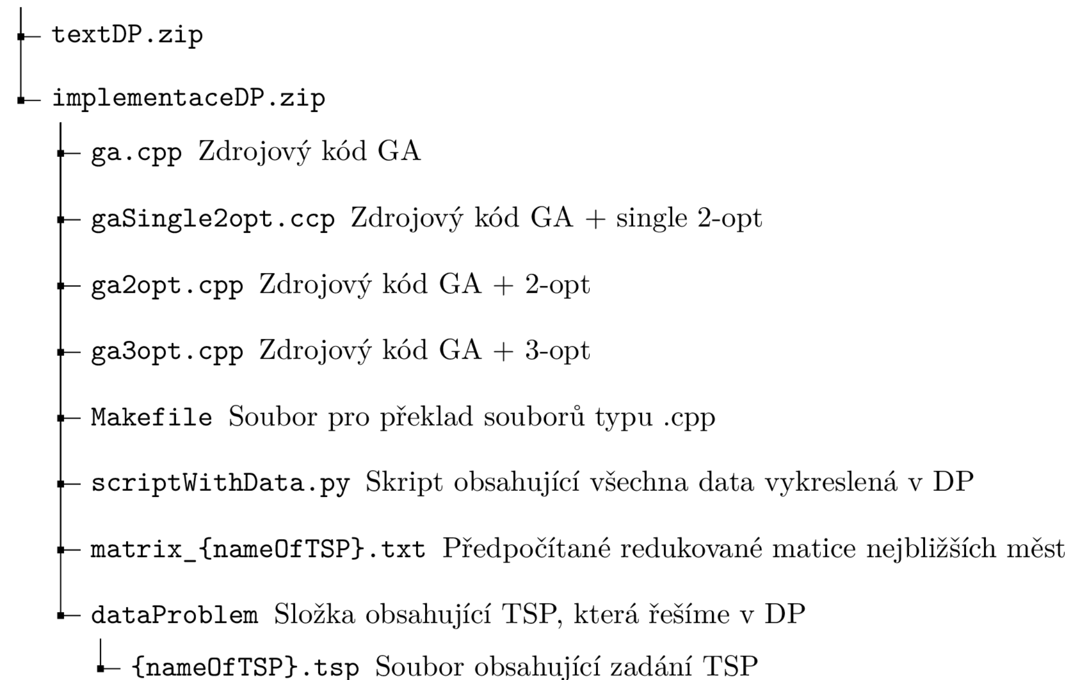
[26] Yang, X., Dong, L. a Su, C. Solving TPS by SA Based on Probabilistic Double Crossover Operator. In: *2021 18th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP).* 2021, s. 372–378. DOI: 10.1109/ICCWAMTIP53232.2021.9674103.

[27] Zelinka, I. *Evoluční výpočetní techniky: principy a aplikace.* 2009.

Příloha A

Struktura odevzdaného adresáře

Odevzdal jsme jeden zip s názvem `DPxhlady01.zip`. V tomto adresáři jsou dva zip soubory. V `textDP.zip` se nachází zdrojový tvar písemné zprávy a PDF diplomové práce. V souboru `implementaceDP.zip` se nachází úplná dokumentace a všechny zdrojové texty programů. Struktura adresáře je:



Adresář se nachází na SD kartě přiložené k diplomové práci i na nextcloudu odkazu, který je napsaný u odevzdané práce v VUT IS.

Příloha B

Návod na spuštění programu

Zde poskytnu návod na spuštění mého programu, uvedeme si jaké potřebujeme prostředí, argumenty programu a případné chybové hlášky. Já jsem používal pro překlad `g++ (GCC) 9.5.0` na edesign serveru. Po unzipnutí `implementaceDP.zip` stačí napsat příkaz `make`. Vzniknout 4 programy:

`ga` Tento program je pouze genetický algoritmus

`gaSingle2opt` Tento program je genetický algoritmus + zkrácený 2-opt algoritmus v průběhu

`ga2opt` Tento program je genetický algoritmus + jeden běh 2-opt algoritmu během evoluce + úplný 2-opt algoritmus

`ga3opt` Genetický algoritmus + zkrácený 2-opt v průběhu + 3-opt na konci evoluce

Jedná se o různé varianty genetických algoritmů. Příklad spuštění programu je
`./{název_programu} {limit_iterací} {druh_operátoru_křížení}`.

Název programu `ga, gaSingle2opt, ga2opt, ga3opt`

Limit iterací Zde zadáme limit iterací genetického algoritmu

Druh operátoru křížení Zde vybere jaký chceme používat operátor křížení. Máme 1-9.

Druhy operátoru křížení jsou 1-VGX, 2-LGX, 3-VGXG, 4-LGXG, 5-PMX, 6-OX, 7-CX, 8-PMX2opt a 9-OX2opt. Jméno TSP, které chceme řešit dáme do proměnné `tsp` ve funkci `main`. Problémy, které chceme řešit se musejí nacházet ve složce `dataProblem` ve formátu `.tsp`. Program také potřebuje soubor `matrix_{nameOfTSP}.txt`. Pokud tento soubor ke konkrétnímu tsp neexistuje, program ho vytvoří a uloží. Výstupem programu je vytvoření nebo zápis do souboru `{jméno_operátoru_křížení}.txt` kde zapíše jméno tsp, délku trasy tsp a čas řešení tsp. Dalším výstupem programu je soubor
`{jméno_tsp}_{jméno_operátoru_křížení}_{délka_trasy}.txt` kde je trasa vyřešeného TSP.

Při špatném počtu argumentů nám vrátí program `exit code 1`, při problému s nějakým otevřením souborů `exit code 2`, funkce pro kontrolu validity řešených tsp při chybě vrátí `exit code 3` a při špatně zvoleném druhu operátoru křížení vrátí `exit code 4`. V případě, že vše proběhlo v pořádku nám program vrátí `exit code 0`.