VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

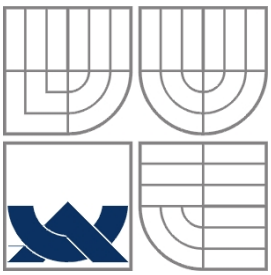
PROBLÉM OBCHODNÍHO CESTUJÍCÍHO

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

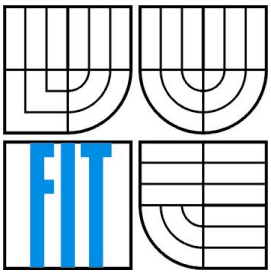
AUTOR PRÁCE
AUTHOR

MARTIN ŠŮSTEK

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PROBLÉM OBCHODNÍHO CESTUJÍCÍHO

TRAVELLING SALESMAN PROBLEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN ŠŮSTEK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. František V. Zbořil, Csc.

BRNO 2015

Abstrakt

Práce se zaměřuje na úpravu známých postupů ACO a GA s ohledem na zvyšování efektivity nalézaných řešení. Jsou zde prezentovány dva nové přístupy pro řešení TSP. Pomocí jednoho z nich lze také vytvořit počáteční populaci pro GA. Je uveden konkrétní návrh programu a v příloze pak i jeho implementace v jazyce Java. Aby se zlepšila efektivita řešení, jsou navržené a implementované lokální optimalizace. Po uplynutí předem stanoveného strojového času jsou mezi sebou porovnány minimální vzdálenosti dosažené zvolenými metodami. Experimenty jsou provedeny na sadách s různými počty míst, konkrétně od 101 až po 3891.

Abstract

This thesis is focused on modification of known principles ACO and GA to increase their performance. Thesis includes two new principles to solve TSP. One of them can be used as an initial population generator. The appendix contains the implementation of the application in Java. The description of this application is also part of the thesis. One part is devoted to optimization in order to make methods more efficient and produce shorter paths. In the end of the thesis are described experiments and their results with different number of places from 101 up to 3891.

Klíčová slova

ACO, GA, Optimalizace mravenčí kolonií, Genetický algoritmus, TSP, Problém obchodního cestujícího, Optimalizace GA, Počáteční populace pro GA, Deterministické řešení pro TSP

Keywords

ACO, GA, Ant Colony Optimization, Genetic Algorithm, TSP, Travelling Salesman Problem, GA Optimization, Initial Population for GA, Deterministic solution for TSP

Citace

Martin Šůstek: Problém obchodního cestujícího, bakalářská práce, Brno, FIT VUT v Brně, 2015

Problém obchodního cestujícího

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. Ing. Františka V. Zbořila, Csc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Šůstek
18. května 2015

Poděkování

Chtěl bych poděkovat vedoucímu práce doc. Ing. Františkovi V. Zbořilovi, CSc. za vedení a směrování této práce, poskytnutí informací a vstřícný přístup.

Obsah

1 Úvod

Problém obchodního cestujícího je obtížný diskrétní optimalizační problém. Pro tuto úlohu neexistují postupy, jak jednoduše nalézt řešení. Genetický algoritmus, simulované žíhání či optimalizace mravenčí kolonií jsou často využívané metody, pomocí nichž lze dosáhnout reálně použitelného výsledku s ohledem na obtížnost úlohy.

Kapitola 2 popisuje, co vlastně tato úloha vyjadřuje. V kapitole 3 jsou prezentovány vybrané metody. Konkrétně se části 3.1 a 3.2 zaměřují na optimalizaci mravenčí kolonií a genetický algoritmus s důrazem na jejich obecný popis a vysvětlení, z čeho vycházejí. Kapitola 3.3 má za úkol přiblížit netradiční přístup, který může být taktéž použit pro řešení TSP, včetně objasnění jeho principu. Dále (kapitola 4) následuje stručné přiblížení hlavních částí výsledného programu tak, aby se čtenář dokázal ve výsledném kódu orientovat. Pro lepší výsledky jsou do aplikace zahrnuty optimalizace, jejichž princip včetně ukázek konkrétních částí, které se za pomocí optimalizací dají zlepšit, je vysvětlen v kapitole 5. V předposlední (6.) kapitole jsou pak uvedeny experimenty s různými počty míst, které mají sloužit pro srovnání navržených metod. První experiment poukazuje na zlepšení plynoucí z optimalizací. Část 6.8 je pak věnována zhodnocení dosažených výsledků experimentů. Naznačuji zde i možná zlepšení, která by mohla přispět k efektivnější implementaci pozměněním prezentovaných metod. Závěr pak slouží jako zhodnocení celé práce. Navrhuji zde další možné úpravy programu či zvolených algoritmů.

2 Problém obchodního cestujícího

Tato kapitola je věnována popisu problému obchodního cestujícího a přibližuje některé pojmy spjaté s touto tematikou. Nechť v tomto textu pojem **cesta** značí posloupnost mezi místy a **trasa** se váže pouze k cestě o délce 1 (odpovídá hraně v názvosloví teorie grafů).

2.1 Popis úlohy

Problém obchodního cestujícího, v angličtině známý jako Traveling Salesman Problem (**TSP**) je úloha, o které byla zmínka již v první polovině 20. století. Její řešení spočívá v nalezení nejkratší cesty mezi všemi zadanými místy tak, aby poslední navštívené místo bylo zároveň počátečním. Pokud uvažujeme, že mezi libovolnými dvěma místy existuje trasa a toto přímé spojení je nejkratší možnost, jak se z místa A do místa B dostat (tedy neexistuje místo C takové, že |AC| + |CB| < |AB|, pak lze na problém nahlížet i jako na hledání hamiltonovské kružnice, protože v teorii grafů tato podmínka koresponduje s trojúhelníkovou nerovností. Tato podmínka je nutná, protože každé místo musí být navštíveno **právě jednou**, a tak je zaručeno, že cestu s nejkratší možnou vzdáleností je tímto způsobem možno vytvořit.

2.2 Spojitost s hamiltonovským grafem

Mějme sadu uzlů v grafu, hamiltonovským grafem je nazván právě tehdy, obsahuje-li hamiltonovskou kružnici, která vzniká spojením všech těchto uzlů, přičemž každý uzel je v hamiltonovském cyklu obsažen právě jednou. Každý uzel má tedy 2 sousední uzly. Představme si nyní místa jako uzly. Sousední uzly pak představují předchozí a následující místo v uspořádaném seznamu navštívených míst.

Vstupními informacemi je pouze sada bodů, přičemž se předpokládá, že trasa implicitně existuje mezi všemi místy. K dopočítání délky hrany mezi místy se dojde na základě Pythagorovy věty a dodaných souřadnic. Důsledkem je symetričnost vzdálenosti míst, tedy trasa z bodu A do bodu B má stejnou délku jako z B do A. Nalezení hamiltonovské kružnice je triviální, stačí vytvořit libovolnou posloupnost neopakujících se míst, cílem je ovšem vytvořit takovou posloupnost, kde je celkový součet vzdáleností mezi sousedními místy **minimální**.

2.3 Možná řešení

Počítač je velmi výkonný stroj, ovšem složitost a s tím související délka výpočtu prozkoumáním celého stavového prostoru je neúnosná. Tento způsob by totiž zahrnoval vytvoření všech možných cest a následný výběr nejkratší. Avšak ze vztahu 1 (***S*** představuje počet možných řešení a ***n*** počet míst) plyne, že třída složitosti je pro tento způsob faktoriálová. Pro 50 míst by tak bylo nutné prozkoumat asi $3*10^{62}$ cest, což je i pro superpočítač nemožné.

$$S = \frac{(n-1)!}{2} \quad \text{pro } n \geq 3 \tag{1}$$

Kvůli této náročnosti se od analytického přístupu prozkoumáním celého stavového prostoru upouští a nahrazují ho jiné způsoby. Možností řešení je celá řada, často se využívá genetických algoritmů, optimalizace mravenční kolonií, simulovaného žíhání či jiných optimalizačných metod a

postupů pro zlepšení výsledků. Všechny využívají náhodná (pseudonáhodná) čísla, pomocí nichž může v každém běhu programu vznikat odlišná posloupnost řešení, což z těchto metod dělá nedeterministické algoritmy (abychom mohli získat přesný výsledek programu, museli bychom dopředu znát tyto náhodná čísla, která slouží pro rozhodování s nějakou pravděpodobností). Metody taktéž nedokáží určit, zda-li už nalezly ideální řešení. Proto je potřeba zvolit podmínku pro zastavení výpočtu (často buď ručně, pomocí počtu iterací nebo po uplynutí předem stanoveného strojového času).

Jiným přístupem by byl algoritmus, který by při každém spuštění provedl **stejné kroky** a získal **totožný výsledek**. Takový algoritmus by nemělo smysl spouštět vícekrát.

2.4 Řešení člověkem

Pokud je problém obchodního cestujícího zadán graficky, pak jsou lidé pro **malý počet míst** schopni nalézt velmi dobré řešení blížící se optimálnímu. Pro větší počet míst již intuitivnost není zaručena, ale člověk je schopen vyhodnotit na první pohled křížení dvou tras nebo může vidět možná vylepšení aktuálního řešení. Tato práce bere v úvahu zmíněná fakta, aby se na základě uvedených předpokladů pomocí optimalizací v kapitole 5 simuloval intuitivní postup lidí.

3 Zvolené metody pro řešení TSP

Řešení prohledáním stavového prostoru nemá z uvedených důvodů smysl. Chtěl bych se zaměřit na metodu řešení problému obchodního cestujícího pomocí genetického algoritmu a umělých mravenců, jelikož tyto metody se zásadně liší v přístupu a obě mají potenciál produkovat přijatelné výsledky, které bych následně chtěl porovnat. V literatuře jsem nenarazil na deterministické řešení, které by nezahrnovalo prozkoumání celého stavového prostoru a zároveň by produkovalo zajímavé výsledky. Proto jsem navrhl jednoduchou metodu, kterou pro účely této práce označím jako deterministický algoritmus.

3.1 Optimalizace mravenčí kolonií

Mravenci jsou velmi zajímavý hmyz, který žije v koloniích. Na základě jejich metody vyhledávání potravy je založen způsob pro řešení TSP, kterému se říká optimalizace mravenčí kolonií. V angličtině se lze pro základní verzi algoritmu setkat s názvem Ant Colony Optimization (**ACO**). Mravenci se při hledání cesty za potravou neorientují podle zraku, ale podle feromonové stopy, ta se sama časem vypařuje. Síla této feromonové stopy určuje výhodnost volby cesty (nejsilnější stopa by měla být na nejvýhodnější cestě).

3.1.1 Přechod na umělé mravence

Aby se dalo chování mravenců aplikovat na řešení TSP, jsou provedeny následující změny oproti živým mravencům:

- Jelikož se jedná o diskrétní problém, mravenci se v prostoru nepohybují spojitě, ale nejkratší možný pohyb, který mohou provést, je z jednoho místa do druhého.
- Živí mravenci nejprve náhodně prohledávají okolí za účelem lokalizace potravy, následně tento zdroj vyhodnotí na základě dostupnosti a objemu. U umělých mravenců je cíl jen jeden a je známý již na počátku (nalezení cesty), navíc neprobíhá žádná úvodní fáze prohledávání, na počátku je již dostupná informace o vzdálenostech mezi místy, což je pro každého mravence důležité pro tvorbu cesty (umělý zrak).
- Disponují znalostmi o vzdálenostech mezi místy, které jsou využity pro rozhodovací proces. Nejsou tedy ovlivňováni pouze pachovou stopou.
- Množství uvolňovaného feromonu se spočítá až na základě vyhodnocení celé cesty, živí jedinci feromon uvolňují hned.
- Pohyb všech umělých mravenců je synchronizován, jejich počáteční místo je stejné jako konečné. Pro každého mravence v každém cyklu se může mraveniště (počáteční místo) lišit.

3.1.2 Algoritmus

V algoritmu figuruje několik mravenců, ti se rozhodují do jisté míry náhodně, přičemž je snaha o úpravu pravděpodobností pro volbu následujícího místa tak, aby se nacházely cesty s lepším ohodnocením, tedy kratší. V prvním cyklu se řídí rozhodování pouze podle vzdálenosti míst, přičemž zvyšování parametru β způsobí větší pravděpodobnost návštěvy bližšího místa. Zvýšení parametru α způsobí větší vliv feromonu na výslednou pravděpodobnost a také zapříčiní, že rozdíly mezi hodnotami feromonu mají citelnější vliv na rozhodovací proces. Algoritmus má mnoho podob a možných vylepšení, popíši zde základní princip:

1. Inicializace času t na hodnotu 0. Spočítání vzdálenosti mezi místy a inicializace hodnot feromonu na každé hraně (trase mezi místy) na počáteční hodnotu a.

2. Každý mravenec si náhodně volí počáteční místo.
3. Mravenec se na základě vypočítané pravděpodobnosti pohybu ze vztahu č. 2 rozhodne, do kterého místa se vydá. Tato pravděpodobnost se váže k aktuálnímu místu ***i*** a místům ***j***, které dosud v tomto cyklu ***k***-tý mravenec ještě **nenavštívil**, pro navštívená místa je pravděpodobnost rovna **nule**. Symbol τ označuje intenzitu feromonů (v čase nula má právě hodnotu ***a***), η vyjadřuje viditelnost míst a d_{ij} vzdálenost mezi místy ***i*** a ***j***. Pokud ještě mravenec nenavštívil všechna místa, následuje opakování tohoto bodu.
4. Pokud vznikla kratší cesta než dosud nejkratší, aktualizace nejkratší cesty.
5. Vypaření feromonů, které je realizováno vynásobením aktuální hodnoty feromonu číslem v rozmezí 0 až 1. Toho je dosaženo aplikací vztahu č. 3. Parametr ρ představuje koeficient vypařování.
6. Úprava feromonové stopy. Každý mravenec zvýší feromonovou stopu na všech trasách (hranách v grafu), kterými prošel, o hodnotu ze vztahu č. 4. Parametr ***Q*** představuje množství feromonu, které je každým mravencem uvolňováno, L_k značí délku cesty.
7. Pokud je splněna ukončovací podmínka (nejčastěji pevný počet cyklů), konec algoritmu. Jinak inkrementace času a návrat na bod 2.

$$p_{ij}^{k}(t)=\frac{[\tau_{ij}(t)]^{\alpha}.[\eta_{ij}]^{\beta}}{\sum_{l=1}^{n}([\tau_{ij}(t)]^{\alpha}.[\eta_{ij}]^{\beta})} \qquad \text{kde } \eta_{ij}=1/d_{ij} \tag{2}$$

$$\tau_{ij}(t+1)=\tau_{ij}(t).(1-\rho)+\Delta\tau_{ij}(t) \qquad \text{kde } \rho\in\langle 0,1) \tag{3}$$

$$\Delta\tau_{ij}(t)=Q/L_k \qquad \text{pro navštívené hrany mravencem } k \text{ v čase } t \tag{4}$$

3.1.3 Volba parametrů a implementační detaily

Nikde v literatuře se nevyskytují přesné ani přibližné hodnoty, které **musí** být algoritmu zadány. Určité zdroje však doporučují jisté rozmezí parametrů. Např. podle [3] by měl mít parametr β hodnotu v rozmezí 0 až 15 a počáteční hodnota feromonu ***a*** být v rozmezí 0 až 1.

V programu lze zadat počet mravenců, počet cyklů, parametry α, β, ρ (koeficient vypařování), ***a*** (počáteční množství feromonu) a ***Q*** (množství feromonu uvolňované mravencem). Pro popsaný algoritmus platí, že feromon vypouští všichni mravenci, ale existuje i varianta (MAX-MIN Ant System), ve které v každém cyklu uvolní feromon pouze mravenec, který nalezl nejkratší cestu. Na základě tohoto principu byla do programu zahrnuta možnost zvolit, jaký počet mravenců feromon uvolní (např. pokud se zadá hodnota 3, tak v každém cyklu dojde k výběru 3 nejkratších cest a zde dojde ke zvýšení hodnoty feromonou, ostatním mravencům není úprava dovolena.

Aplikace zahrnuje i optimalizace, které budou popsány až v kapitole 5. Optimalizace jsou provedeny po vytvoření cesty, vzniká tak problém, kdy by mělo dojít k uvolnění feromonu. Z variant uvolnění feromonu před optimalizací, po optimalizaci a před i po optimalizaci se experimentálně ukázal (pro většinu případů) jako nejvhodnější způsob ten, kdy je feromon uvolněn před optimalizací, pak dojde k úpravě cest a feromon je opět uvolněn pro zvolený počet nejlepších řešení.

Náročnost výpočtu

Pro ACO je nutné provádět velké množství výpočtů, proto jsem se snažil o minimalizaci výpočtů. Jednou z možností bylo předpočítaní celkové hodnoty jmenovatele pro každé místo ze vztahu č. 2. Tuto sumu je totiž nutné počítat pro každého mravence při každé volbě následujícího místa. Bylo by

tak přínosné předpočítat si sumu[1] pro všechna (navštívená i nenavštívená) místa a od ní pouze odečíst hodnoty navštívených míst. Po implementaci takto matematicky korektní modifikace bylo však zjištěno, že pokud se parametr β zvolí jako velké číslo (např. 40), může dojít k situaci, kdy všechna blízká místa již byla navštívena a dojde k zaokrouhlovací chybě. Výsledky získané těmito způsoby se však lišily až o několik řádů, což způsobí chybu (v tomto programu potenciálně nekonečný cyklus). Tato konkrétní modifikace, která citelným způsobem zkracovala výpočet, byla zavržena.

3.2 Genetický algoritmus

Ve všech lidských buňkách s výjimkou pohlavních je obsaženo 23 párů chromozomů. Jeden z každého páru pochází od matky a druhý od otce. Každý chromozom obsahuje geny, přičemž konkrétní formy se nazývají alely. Tato fakta jsou základem pro genetický algoritmus (**GA**). V lidské populaci jsou osoby, rodí se děti a staří lidé umírají. V umělé populaci jsou jedinci, kteří se nemusí dál rozmnožovat, zánik jedinců je podmíněn pouze rozumným využitím paměti a výpočetních prostředků. Každý jedinec představuje řešení úlohy, je tak možné, aby v populaci zůstávala nejlépe ohodnocená řešení. Funkci, která tomuto jedinci ohodnocení přiřadí, se říká fitness funkce. Z ohodnocených řešení se provede výběr vhodných rodičů a jejich reprodukcí vznikají dva potomci. Tyto obvykle obsahují geny obou rodičů tak, že jeden potomek má gen na určitém místě od matky a druhý od otce. Je možné, že populace obsahuje velké množství málo se lišících jedinců a tak reprodukcí vznikají jedinci opět podobní. To může spět ke ztrátě schopnosti generovat jedince mimo ty, které již populace obsahuje (malá rozmanitost genomu). Z tohoto důvodu může dojít stejně jako např. v lidské populaci k mutacím, které jsou obvykle realizovány náhodnou změnou jednoho či více genů.

3.2.1 Aplikace genetického algoritmu na TSP

Jedinec v populaci bude představovat jedno řešení – tedy jednu cestu. Genů bude obsahovat stejně jako míst, přičemž každý může být libovolné místo. Ze zadání úlohy ale plyne pro všechna řešení povinnost navštívit každé místo **právě jednou**. To znamená, že jedinec nesmí obsahovat stejný gen (místo) dvakrát a zároveň mu žádný nesmí scházet. Tento požadavek by mohl být porušen při tvorbě nové populace výměnou genů na stejných pozicích u obou rodičů (z **A-|B|-C** a **A-|C|-B** by vzniklo **A-C-C** a **A-B-B**). Aby se takové situaci předešlo, je nutno přistoupit k jinému způsobu křížení. Mutace náhodnou změnou genu taktéž není ze stejného důvodu možná.

3.2.2 Popis upraveného algoritmu

Uvedu zde pouze obecný popis genetického algoritmu s respektováním zmíněných úprav pro TSP:

1. Vytvoření počáteční generace, inicializace čítače generací na hodnotu 0.
2. Zhodnocení populace, ukončení výpočtu při splnění zvolených podmínek (počet generací, minimální změna průměrného ohodnocení populace, nalezené řešení je lepší než zvolená mez, nezlepšení nejlepšího po určitý počet cyklů, …).
3. Výběr rodičů a reprodukce. Reprodukce je realizována pomocí křížení, z každých 2 rodičů vznikají 2 děti.
4. Pokud vzniklo lepší řešení, aktualizace nejlepšího. Možná mutace jedinců, zařazení nových potomků do stávající populace, opakování bodu 2.

Každá z fází může být provedena různými způsoby, následuje tedy konkrétnější popis vybraných částí.

1 zůstává v každém cyklu stejná

Výběr rodičů

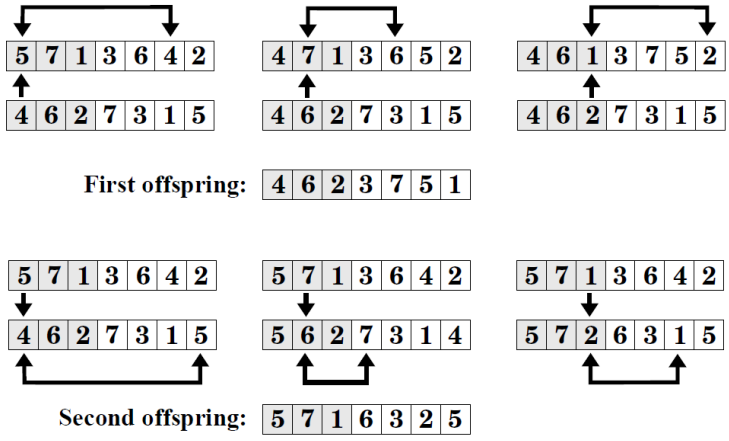
Pro výběr rodičů existuje několik metod, nejtriviálnější je náhodný výběr ze všech možných jedinců, jsou však možné i jiné metody, které se snaží zvýhodnit lépe ohodnocené jedince. Vychází se z předpokladu, že dobře ohodnocený jedinec má větší šanci vyprodukovat křížením obstojné řešení. Následují některé modifikace výběru:

- **Ruleta** – rozložení pravděpodobnosti není uniformní, ale závisí na hodnotě fitness funkce, lepší jedinci mají větší šanci být vybráni.
- **Elita** – jedinci jsou seřazeni podle fitness funkce, šanci stát se rodiči má pouze jistý počet nejlépe ohodnocených.
- **Turnaj** – Vybere se několik jedinců (typicky 2) a nejlépe ohodnocený z nich je vybrán jako rodič, tímto způsobem mohou produkovat děti všichni jedinci kromě nejhůře ohodnoceného (za předpokladu, že nelze vybrat stejného potomka dvakrát).

Vytváření potomků

Obecně existuje celá řada křížení, avšak pro TSP je nutno volit z již zmíněných důvodů speciální, např. tyto:

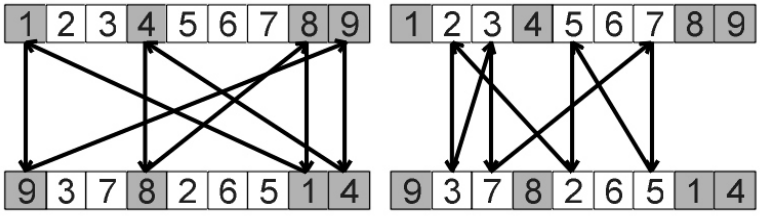
- **Jednobodové křížení s použitím indexové tabulky** – místa jsou seřazena tak, aby se ke každému dalo přistupovat pomocí indexu. Předpokládejme místa **A**, **B**, **C** a **D**. Vytvoří se indexový chromozom, který pro tyto místa může mít tvar „**1-2-0-0**“[2]. Index na první pozici je index místa (1 → 2. místo → místo **B**), pro každé další platí, že pokud se vybere místo s nižším indexem, jejich index se sníží o 1. Předpokládejme druhého rodiče „**3-0-1-0**“. V prvním případě se jedná o cestu „**B-D-A-C**“ a v druhém „**D-A-C-B**“. Nyní se náhodně zvolí bod křížení, předpokládejme pouze za prvním místem, pak vzniknou potomci „**1-0-1-0**“ a „**3-2-0-0**“ představují řešení „**B-A-D-C**“ a „**D-C-A-B**“. Vzniklé cesty nezachovávají části, a tak tento způsob není v souladu s myšlenkou přejímání části genetické informace od otce a části od matky.
- **Křížení PMX (partially matched crossover)** – tento operátor využívá dva body křížení, které se volí náhodně. Předpokládejme místa **A**, **B**, **C**, **D**, **E** a **F**. Mějme body křížení 2 a 4 a cesty „**E-D-|A-B|-C-F**“ a „**D-A-|F-E|-B-C**„, pak dojde v obou řešení k výměnám chromozomů na stejných pozicích, v tomto případě **A** za **F** a **E** za **B**, výsledkem jsou tedy „**B-D-F-E-C-A**“ a „**D-F-A-B-E-C**„, tento způsob tak do jisté míry může zachovávat části od rodičů. Jiný příklad je uveden na obrázku 3.1, který je přejatý z [1].



Obrázek 3.1: Ilustrace operátoru PMX

2 číslo představuje index v uspořádané množině nevybraných míst (indexace od 0), výběrem místa se z množiny odstraní

- **Křížení CX (cycle crossover)**[2] – předpokládejme sadu míst **A**, **B**, **C**, **D** a **E**. Cesty rodičů jsou „**D-A-B-C-E**“ a „**A-B-D-E-C**“. Následuje volba počátečního indexu, nechť je zvolena druhá pozice. V prvním rodiči je na druhé pozici místo **A**, nyní zjistíme, na jaké pozici se nachází místo **A** v druhém rodiči (na první pozici). Na tuto pozici se podíváme a zjistíme, co za místo se nachází v prvním řešení (místo **D**). Postup takto opakujeme, dokud nenarazíme na počáteční místo. (Místo **D** se v druhém jedinci vyskytuje na pozici 3, na této pozici v 1. je místo **B**, které se vyskytuje na pozici 2). Po nalezení cyklu – v tomto případě „**A-D-B**“ se vymění tato část mezi rodiči, vzniknou tak potomci „**A-B-D-C-E**“ a „**D-A-B-E-C**“. Tento operátor je také slibný a může produkovat zajímavé výsledky s respektováním částí rodičovských cest. Jiný příklad na hledání cyklů je na obr. 3.2, který je převzatý z [10].



Obrázek 3.2: Ilustrace operátoru CX

Mutace a zařazení do populace

Před zařazením nových jedinců je ještě možná mutace, nejjednodušším legálním způsobem mutace je prohození dvou míst. Dalším způsobem může být určení dvou hranic. Místa nejblíže k hranici si vymění svou pozici, pak jejich sousedi směrem od hranice. Tento postup následuje dokud se nedosáhne středu hranice. Např. z „**A-|B-D-C-F|-E**“ vznikne „**A-|F-C-D-B|-E**“.

Způsobů zařazení do populace je mnoho, noví potomci mohou nahradit své rodiče, náhodné jedince v populaci či nejhůře ohodnocené jedince. Lze přidat pouze nejlépe ohodnoceného potomka, všechny potomky či určitý počet. Tento počet se také může dynamicky měnit v závislosti na určité podmínce.

3.2.3 Změny pro implementaci

Implementace GA se v určitých částech liší od popsaného teoretického principu. Experimentováním s různými modifikacemi a zkoušením nových nápadů za účelem zlepšení efektivity algoritmu jsem věnoval nejvíce času. V programu je umožněno nastavení velikosti populace a počtu generací.

Pro výběr rodičů bylo zvoleno křížení PMX. S účinností tohoto křížení jsem nejprve **nebyl spokojen**, protože téměř nikdy nedocházelo při rekombinaci ke vzniku lepšího řešení, než dosud nalezeného. Avšak po začlenění optimalizací (kapitola 5) dochází velmi často ke vzniku nového nejlepšího řešení.

Výběr rodičů a zařazení do populace

Zvolil jsem kombinaci mezi **turnajem** a **elitou**. Populace je vždy seřazena podle kvality řešení, na nižším indexu se vyskytují kvalitnější řešení. Nejdříve je na základě parametru ***Elite percentage*** určeno, kolik procent nejlépe ohodnocených má možnost stát se rodičem. Následně se vygenerují dvě odpovídající náhodná kladná celá čísla a menší z nich představuje index rodiče v poli s populací. Takto jsou vybrání dva rodiče pro každé křížení. V programu je zamezeno, aby se rodič křížil se sebou samým, protože by se mohl hromadit počet stejných jedinců, což by byl jev nežádoucí.

Počet křížení určuje parametr ***Number of reproductions***, přičemž z každého křížení vznikají 2 potomci. Rozhodl jsem se ignorovat mutace z algoritmu a nahradit je jinou formou. Jejich účel je nahrazen zahrnutím ***Random per cycle*** nových náhodných jedinců v každém cyklu mezi potomky.

Celkově tak vzniká **2 * Number of Reproductions + Random per cycle** nových jedinců v každé generaci. Tyto jsou seřazeny od nejlepšího po nejhorší

Původní algoritmus zařazení do populace byl navržen tak, aby garantoval neustálé zlepšování fitness funkce. Algoritmus spočíval v následujících krocích:

1. Vyberte nejlepšího jedince z potomků a nejhoršího jedince z populace.
2. Pokud vybraný jedinec z populace představuje horší řešení než vybraný potomek, nahraďte ho. Jinak konec algoritmu.
3. Vyberte dalšího (hůře ohodnoceného) potomka. Z populace taktéž vyberte dalšího (lépe ohodnoceného) jedince. Návrat k bodu 2.

Tento princip fungoval obstojně, avšak v rámci možného experimentování jsem provedl úpravu v druhém bodě (přidání parametru ***Comparison constant***). Tato konstanta říká, kolikrát horší (větší délka cesty) může být nový potomek, aby stále došlo k nahrazení. Přestože není garantováno zvyšování (nebo alespoň neklesání) kvality populace na základě fitness funkce, algoritmus má lepší vlastnosti. Oproti přechozímu řešení dojde k úpravě druhého bodu:

2. Pokud vybraný jedinec z populace představuje horší řešení **(Jako horší je označen v případě, že platí vztah č. 5,** kde je ***len*** délkou řešení, ***old*** aktuální v populaci, ***new*** potomek a již zmíněná konstanta je označena jako ***const***) než aktuální potomek, nahraďte ho. Jinak konec algoritmu.

Tato změna může být potlačena nastavením parametru ***Comparison constant*** na hodnotu 1.

$$old_{len} * const > new_{len} \tag{5}$$

Vytváření nových náhodných jedinců

Nejprve musí být vytvořena počáteční generace. Jedinci by měli být náhodní, avšak ne zcela. Pokud je totiž vytvoření jakéhokoli jedince stejně pravděpodobné, průměrné řešení je velmi špatné. Vytvořil jsem tedy následující algoritmus pro generování nových náhodných jedinců, uvedené pravděpodobnosti byly zvoleny experimentálně:

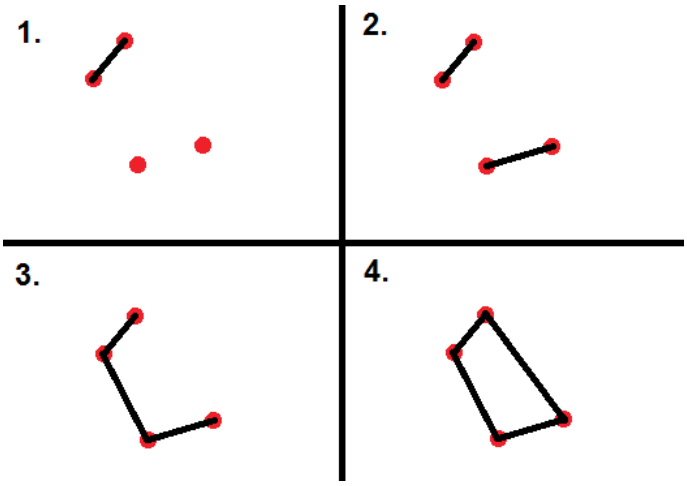
1. Vytvořte seznam vzdáleností mezi místy pro každé místo. Náhodně zvolte první počáteční místo.
2. Pokud byla navštívena všechna místa kromě posledního, zvolte toto místo a ukončete algoritmus.
3. S pravděpodobností 90 % zvolte jako další navštívené místo v pořadí **nejbližší možné** a vraťte se k bodu 2. Jinak označte jako **potenciální** místo pro navštívení další (druhé) nejbližší možné místo, které lze navštívit (nebylo ještě navštíveno).
4. Pokud neexistuje žádné vzdálenější místo než **potenciální**, zvolte ho jako další navštívené místo a vraťte se k bodu 2.
5. S pravděpodobností 50 % zvolte potenciální místo jako další navštívené místo a vraťte se k bodu 2. Jinak označte **potenciální** místo jako další nejbližší možné místo k navštívení a vraťte se k bodu 4.

Pomocí tohoto algoritmu se dají generovat přijatelná řešení, ale stále je zde prostor pro zlepšení. Tento postup předcházel nápadu na algoritmus nejkratších cest (část 3.3.2), který vytváří ještě lépe ohodnocené jedince. Uvažoval jsem tak o jeho nahrazení, ale nakonec je největší efektivita dosažena kombinací obou způsobů. Při tvorbě náhodných jedinců se tak vždy **střídá** vytváření jedinců podle **zmíněného postupu** a podle **algoritmu nejkratších cest**.

3.3 Deterministické řešení

Jak již bylo zmíněno v kapitole 2, ACO ani GA neposkytují při opakovaném spuštění stejné výsledky. Chtěl jsem využít nějakého postupu, který by vytvořil přijatelné řešení za krátký čas.

3.3.1 Popis algoritmu



Obrázek 3.3: Simulace deterministického algoritmu

Pokud by se problém obchodního cestujícího skládal pouze z míst na kružnici, řešením by bylo spojit vždy sousední body, pro všechny body by tak byla splněna podmínka, že jejich dvě nejkratší trasy jsou součástí řešení, a pak je zřejmé, že i toto řešení bude ideální. Z tohoto jednoduchého principu vycházím, přestože jsem si vědom, že tento případ rozložení bodů je ojedinělý. Algoritmus postupně vybírá trasy a připojuje je do výsledné cesty (může vznikat více nespojených částí). Odpovídá to postupu, kdy by člověk měl graficky zobrazenou sadu míst, vzal si tužku a začal spojovat vždy nejkratší trasy takovým způsobem, aby stále mohl vzniknout hamiltonovský graf. Tento postup je znázorněn na obrázku 3.3. Následuje zjednodušený popis:

1. Vytvořte seřazený seznam všech možných tras od nejkratší po nejdelší, nastavte proměnnou *aktuální index* na prvního (index 0 nebo 1 v závislosti na indexování).
2. Vyberte trasu ze seřazeného seznamu na pozici *aktuálního indexu*. Pokud by přidáním (nakreslením) této trasy nevzniknul cyklus mezi místy a pokud jsou obě místa spojena s méně než 2 místy (nemají ještě oba sousedy), přidejte tuto trasu.
3. Pokud není *aktuální index* roven poslednímu, inkrementujte ho a návrat k bodu 2.
4. Sestrojte cestu z dostupných tras a zobrazte výsledek.

3.3.2 Úprava na nedeterministický algoritmus

Jak již bylo řečeno, aktuální podoba algoritmu vytvoří pro konkrétní sadu míst vždy stejné řešení. Navrhl jsem změny tak, aby myšlenka zůstala stejná, ale bylo možné vytvořit více různých řešení:

1. Pro každé místo vytvořte seřazený seznam od nejkratších po nejdelší trasy vedoucí z tohoto místa.
2. Náhodně vyberte místo, které ještě nemá zvolené 2 sousední místa.
3. Procházejte seznam seřazených tras pro vybrané místo od nejkratších. Pokud by připojením trasy mezi těmito místy nevzniknul cyklus a pokud je místo spojeno s méně než 2 sousedy, přidejte tuto trasu. Pokud ještě nemají všechna místa 2 sousedy, návrat na bod 2.
4. Sestrojte cestu z dostupných tras.

Pokud by bylo požadováno vytvoření více řešení, nemusí se již znovu provádět bod 1. Algoritmus díky své jednoduchosti může vytvářet řešení i pro velký počet míst v krátkém čase. Na popsaný algoritmus se pro účely této práce budu odkazovat jako na algoritmus **nejkratších cest** (SP).

Algoritmus nejkratších cest využívá na rozdíl od ACO a GA stejné informace pro sestrojení každé cesty, s postupem času se tak nezvyšuje ani potenciální šance na nalezení lepšího řešení. U ACO se předpokládá, že feromonová stopa by k lepšímu řešení mohla být nápomocná, stejně jako lepší kvalita populace u GA. Algoritmus nejkratších cest je tak vhodný i jako generátor náhodných cest blízkých optimálnímu řešení pro genetický algoritmus.

3.3.3 Implementace a řazení míst

U deterministického algoritmu může činit problém řazení. Závislost mezi počtem tras k seřazení ***C*** a počtem míst ***n*** vyjadřuje vztah č. 6. Pro 1000 míst se řadí téměř 500 000 tras. S ohledem na tento fakt by bylo možné provést úpravy tak, aby se řazení uskutečnilo víckrát, avšak vždy maximálně pro ***n*** tras (tento přístup je uplatněn při nedeterministické modifikaci). V programu lze nedeterministické verzi nastavit počet iterací.

$$C = \frac{n.(n-1)}{2} \tag{6}$$

4 Struktura programu

Tato kapitola stručně popisuje vytvořený program[3] a jeho ovládání. Aplikace je implementována v jazyce Java a obsahuje 4 balíky. Tyto nebudou popsány detailně, neboť každá metoda je okomentována přímo ve zdrojovém kódu. Aplikace má vnitřně nastavené jisté rozmezí většiny parametrů, ku příkladu je zamezeno zadání velikosti populace GA větší než 5000 (bude automaticky nastavena na tuto hodnotu). Jiná omezení jsou nutná pro správný běh programu. Rozsáhlé testování aplikace nebylo předmětem této práce, takže je možné, že určitá neplatná kombinace parametrů způsobí chybu, avšak velká část by měla být ověřena.

4.1 Balík main

Obsahuje pouze jednu třídu **main** s hlavní metodou, pomocí které se spouští celá aplikace, což je její jediná zodpovědnost.

4.2 Balík model

Obsahuje třídy **city** a **cities** představující místa (původně města), ke kterým se má zjišťovat nejkratší cesta. Tyto třídy mají mimo jiné metody pro přidávání míst. Algoritmy předpokládají, že na konkrétních souřadnicích bude maximálně 1 místo, tedy že pro každé místo bude právě jedna vzdálenost rovna nule a to vzdálenost k sobě samému (pokud by to nebylo zajištěno, algoritmy mohou přestat fungovat). Při přidávání míst se ověřuje jedinečnost, ověřování lze pro načítání dat ze souboru zakázat, uživatel ale musí garantovat, že dvě místa nebudou mít stejné souřadnice.

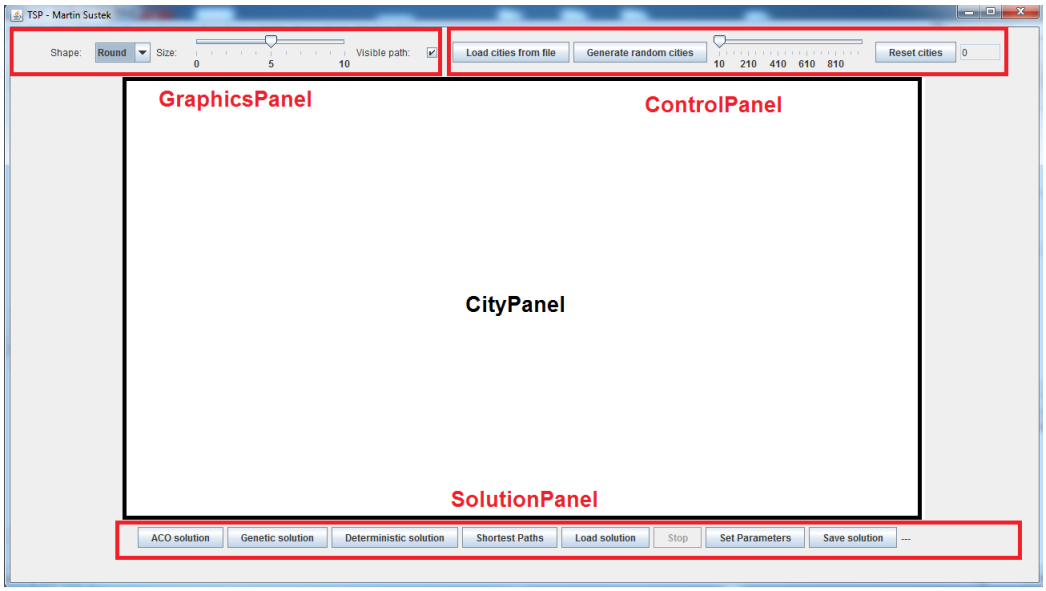
4.3 Balík solving

Součástí tohoto balíku jsou třídy pro konkrétní způsob řešení, ale i obecné, které jsou společné pro více řešení. Aby bylo možné do určité míry ke všem řešením přistupovat stejným způsobem (kvůli různému počtu různých parametrů nelze přistupovat zcela stejně), byla vytvořena abstraktní třída **AbstractSolution**, balík celkově obsahuje 10 tříd. AbstractSolution poskytuje kromě základních společných atributů i optimalizace, které jsou popsány v kapitole 5.

Třída **IndividualPath** představuje řešení (1 cestu), obsahuje posloupnost míst a jejich celkovou vzdálenost. **CityRoute** má význam trasy mezi místy, předpokládá se pole těchto hodnot (všechny se váží k **jednomu konkrétnímu místu**), které je následně seřazeno, vznikne tak pro konkrétní místo posloupnost vzdáleností od nejbližšího po nejvzdálenější místo. Podobný význam má i třída **DetTour**, kterou využívá deterministický algoritmus. Instance této třídy obsahuje informace o obou místech, která spojuje (nevztahuje se k jednomu konkrétnímu místu, ale vznikne pole všech tras, každá trasa tak musí uchovávat informace o **obou místech**, která spojuje). **InfoClusterOptim** je pomocná třída nesoucí informace o optimalizaci shluků (tato optimalizace je popsána v části 5.3).

Od abstraktní třídy **AbstractSolution** jsou odvozené třídy **Ants** (optimalizace mravenčí kolonií), **Genetic** (genetický algoritmus), **Deterministic** (deterministické řešení), **ShortestPath** (algoritmus nejkratších cest) a **LoadSolution** (řešení nahrané ze souboru).

3 Program je součástí přílohy

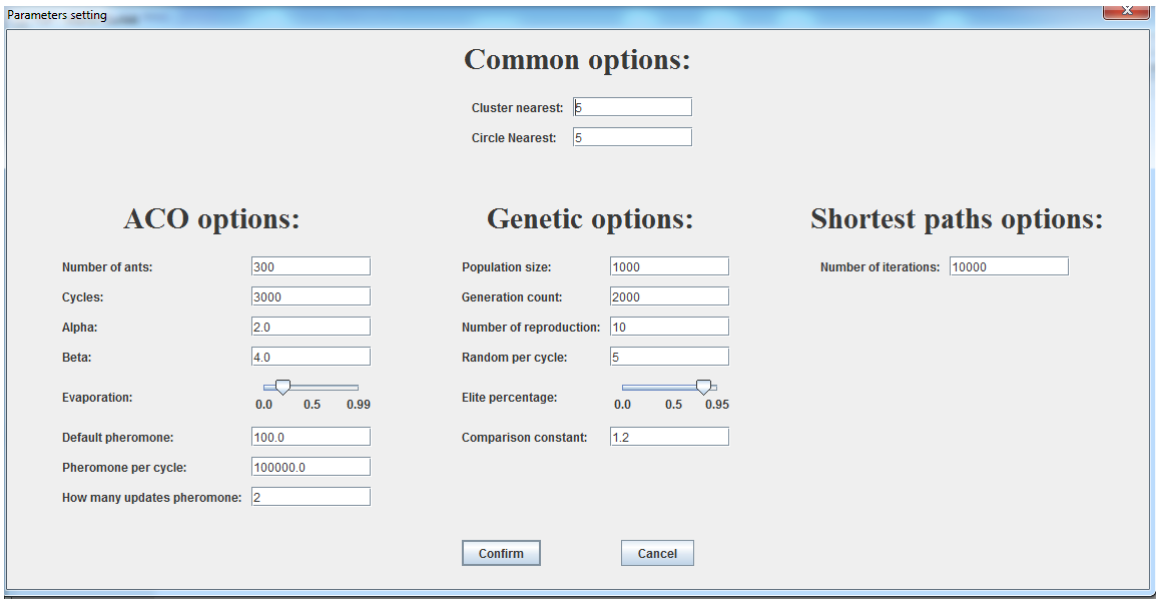


Obrázek 4.1: Náhled aplikace

4.4 Balík gui

Balík představuje grafické uživatelské rozhraní, jeho součástí jsou třídy **CityPanel, ControlPanel, Frame, GraphicsPanel, Parameters** a **SolutionPanel**.

Frame slouží jako hlavní okno, ve kterém je plátno (**CityPanel**) s místy a případně nalezenou cestou, grafický panel (**GraphicsPanel**) umožňující editaci vzhledu plátna, řídící panel (**ControlPanel**) a panel s řešením (**SolutionPanel**). Grafický panel dokáže změnit velikost či tvar vykreslovaných míst. Pomocí řídícího panelu lze generovat náhodná místa, načíst vytvořené sady míst ze souboru a poskytuje informaci o aktuálním počtu míst. V panelu s řešením jsou tlačítka pro volbu konkrétní metody, zastavení výpočtu a editaci parametrů. Po stisku tlačítka *Stop* nedojde



Obrázek 4.2: Modální okno pro nastavení parametrů

k okamžitému ukončení výpočtu, ale dojde pouze k notifikaci probíhajícího algoritmu, aby nepokračoval další iterací (tato akce může trvat pro velký počet míst i několik desítek sekund). Po stisku *Set Parameters* se otevře modální okno **Parameters**. Kromě řešení TSP pomocí zmíněných 4 způsobů (tlačítka *ACO Solution*, *Genetic Solution*, *Deterministic Solution* a *Shortest path*) lze nahrát řešení ze souboru (*Load Solution*) a aktuálně zobrazené také do souboru uložit (*Save Solution*). Je však nutné dodržet striktní formát (posloupnost indexů míst oddělených novým řádkem). Popsaný balík reprezentuje vzhled programu, který je viditelný na obrázku 4.1, podoba modálního okna určeného pro zadání parametrů je zobrazeno na obrázku 4.2.

4.5 Ovládání

Nejprve je nutné pomocí řídícího panelu zvolit místa. Lze buď vygenerovat zvolený počet náhodně nebo nahrát místa ze souboru. Místa lze také přidat ručně (kliknutím do plátna). Úprava míst je zakázána po dobu řešení TSP (po kliknutí do plátna nebude místo přidáno). Panel s řešením poskytuje možnost změny parametrů. Pokud jsou v mapě alespoň 3 body, je možné spustit řešení pomocí zvoleného algoritmu. Při potřebě znovupoužití daného řešení umožnuje aplikace uložení do souboru po ukončení výpočtu. Nevyhovuje-li zobrazení míst, v jakémkoliv okamžiku je možno změnit velikost či tvar vykreslovaných míst a případně zakázat vykreslování nejlepší dosažené cesty. V pravém horním rohu se vyskytuje počítadlo míst, v pravém spodním rohu délka nejkratší nalezené cesty. Program neumožnuje uložit nastavené parametry, při opětovném spuštění programu dojde k jejich reinicializaci na výchozí hodnoty.

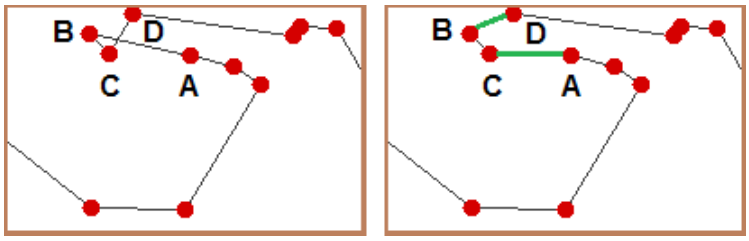
5 Optimalizace

Optimalizace byly navrhnuty kvůli neuspokojivým vysledkům získaných pomocí GA, následně však byly začleněny i do ostatních metod (ACO, SP, deterministický algoritmus). V programu lze zadat dva parametry, které se k optimalizacím vztahují.

Jak bylo zmíněno v podkapitole 2.4, inspiroval jsem se faktem, že lidé jsou schopni intuitivně řešit TSP, zanalyzoval jsem tedy, co mi na dosažených výsledcích nepřijde kvalitní (proč člověk považuje dané řešení za špatné) a pokusil jsem se vymyslet postup, jak tyto artefakty vymýtit. Můj úsudek vyústil v následující 3 optimalizační postupy.

5.1 Výměna sousedních míst

Pokud nastane situace, která je viditelná na obr. 5.1, každý člověk ji zaregistruje a intuitivně ví, jak ji odstranit. Konkrétně mám na mysli situaci, kdy má část cesty tvar „**A-B-C-D**“, avšak trasy „**A-B**“ a „**C-D**“ se kříží. Přestože člověk tento jev vidí na první pohled, pro algoritmické ověření je nutno znát vzdálenosti mezi místy, což v našem případě není problém, jelikož tyto hodnoty využívají samotné algoritmy.

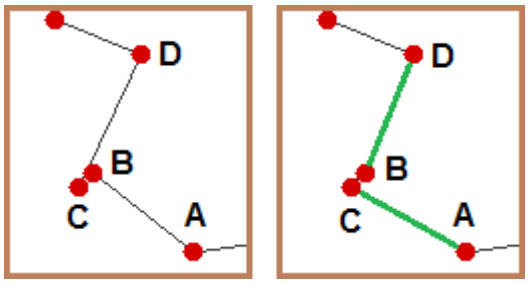


Obrázek 5.1: Odstranění křížení výměnou sousedů

K odstranění této vady stačí jednou projít seznam všech hran (předpokládejme, že pro místa „**A-B-C-D**“ je aktuální hrana „**B-C**“). Uvedený algoritmus tak má lineární složitost:

1. Procházejte postupně všechny hrany v kruhovém seznamu představující řešení. Po iteraci přes všechny hrany ukončete algoritmus.
2. Označte místa, které spojuje aktuální hrana jako v pořadí **B** a **C**, místo před **B** jako **A** a místo za **C** jako **D**.
3. Pokud platí, že vzdálenost |**AC**| + |**BD**| < |**AB**| + |**CD**|, vyměňte pořadí míst spojující aktuální hranu (tedy **B** a **C**), aktualizujte celkovou vzdálenost cesty. Návrat k bodu 1.

Nemusí být zřejmé, že tento algoritmus neodstraňuje pouze křížení, ale celkově provádí jakoukoli výměnu dvou sousedních míst, dojde tedy i k opravě, kterou lze pozorovat na obr. 5.2.

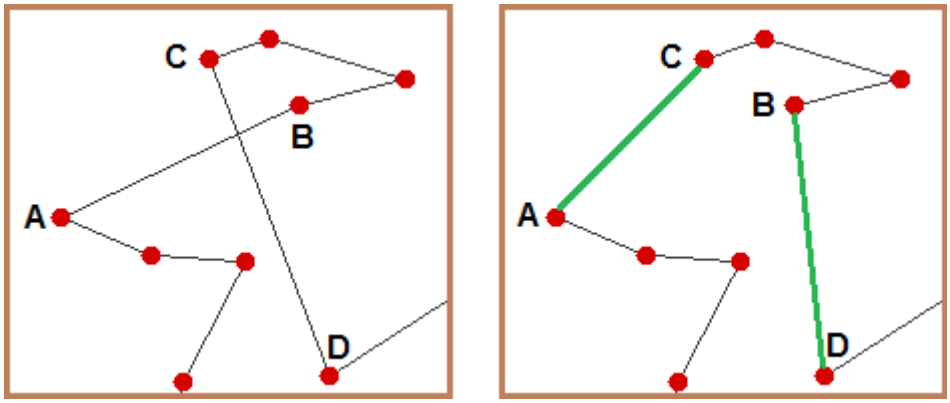


Obrázek 5.2: Zkrácení výměnou sousedů

5.2 Křížení tras

Člověk je opět schopný na první pohled rozpoznat křížení tras (obr. 5.3), ovšem pro počítač už to tak jednoduché není. Třída složitosti počtu ověření křížení všech tras v závislosti na počtu tras je **kvadratická**. Při ověřování křížení tímto způsobem je však doba výpočtu velmi velká. Pokusil jsem se rychlost (i efektivitu) zvýšit, avšak třída složitosti zůstává stejná. Vycházel jsem z předpokladu, že není nutné odstranit všechna možná křížení, může se tak s určitou malou pravděpodobností stát, že algoritmus křížení neodstraní. To ovšem nevadí, protože počet vznikajících řešení je velký. Program umožňuje zadat parametr ***Circle nearest***.

1. Procházejte postupně všechny hrany v kruhovém seznamu představující řešení. Po iteraci přes všechny ukončete algoritmus.
2. Označte místa, které spojuje aktuální hrana jako v pořadí **A** a **B**. Pokud je místo **B** v seřazeném seznamu vzdáleností pro místo **A** nebo místo **A** v seřazeném seznamu vzdáleností pro místo **B**[4] na lepší pozici než ***Circle nearest***, návrat k bodu 1. Jinak nastavte, že v tomto cyklu ještě nebylo nalezeno „lepší řešení".
3. V cyklu procházejte postupně další hrany, nezačínejte sousední hranou, ale až následující, stejně tak zamezte, aby poslední hranou byla sousední hrana (z druhé strany) či hrana samotná. Místa, která tato hrana spojuje, označte v pořadí jako **C** a **D**.
4. Pokud platí, že vzdálenost **(|AC| + |BD|) - (|AB| + |CD|) > 0**:
   - Zaznamenejte, že došlo k nalezení „lepšího řešení".
   - Platí-li, že tato vzdálenost je větší než doposud nejlepší „lepší řešení", pak označte právě nalezené jako **nejlepší**.
   - Přejděte na bod 6.
5. Pokud je zaznamenáno, že v aktuálním cyklu již došlo k nalezení „lepšího řešení", přechod na bod 7 (snaha o urychlení).
6. Je-li možno vybrat další hranu tak, aby nebyly porušeny podmínky v bodě 3, přejděte na bod 3, jinak na bod 7.
7. Pokud je nalezeno „lepší řešení", pak z **nejlepší** nalezené hrany použijte místa **C** a **D** a pro tyto proveďte následující změny v cestě::
   - Vyměňte pořadí všech míst v ceste mezi místy **B** a **C** včetně těchto. Posloupnost míst od místa **B** (včetně) po místo **C** (včetně) se objeví ve výsledné cestě v opačném pořadí.
   - Přepočítejte délku cesty.
8. Návrat na bod 1.



Obrázek 5.3: Ilustrace odstranění křížení tras

4 V programu se při zadání parametru např. na hodnotu 5 uvažuje 5 nejbližších vzdáleností včetně implicitní vzdálenosti k sobě samému, tedy pokud je pro místo **A** místo **B** maximálně 4. nebližší a naopak.

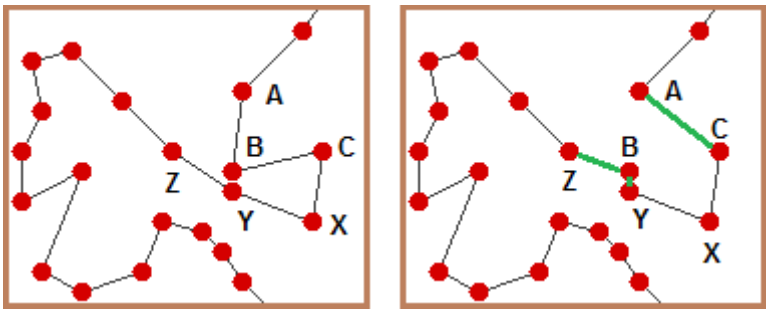
Aby byly reflektovány všechny skutečnosti implementovaného algoritmu, je zapsán komplikovaně, uvádím tedy i zjednodušený slovní popis:

- Ke každé hraně, která má předpoklad (podle ***Circle nearest***) býti hranou nevyhovující (mohla by být provedena změna), zkuste najít křížení s jinou hranou.
- Pokud nějaké křížení naleznete, zkuste pokračovat v hledání dalších hran tak dlouho, dokud platí, že všechny tyto hrany nějak zlepšují výsledek (pokus o nalezení lokálního extrému).

5.3 Optimalizace shluků

Poslední návrh na opravu (zkrácení) cesty jsem nazval optimalizací shluků. Všiml jsem si, že se občas vytvoří řešení, ve kterém by přesunem místa na jinou pozici v kruhovém seznamu míst došlo ke zkrácení výsledné délky, konkrétně provést akci viditelnou na obr. 5.4. Tím se vlastně přiřadí místo ke správnému shluku. Teoreticky by třída složitosti tohoto algoritmu (pro každé místo ověření, zda-li dojde umístěním na lepší pozici k vylepšení cesty) byla kvadratická. Úpravami však vzniká algoritmus, který má třídu složitosti lineární (samozřejmě bez garance všech oprav).

1. Procházejte postupně všechna místa (aktuální místo označte jako místo **B**, sousední místa v cestě označte jako **A** a **C**) ve vytvořené cestě. Po iteraci přes všechna ukončete algoritmus.
2. Inicializujte čítač nalezených **sousedů na hodnotu 0**. Procházejte v seřazeném seznamu vzdáleností k ostatním místům pro místo **B** od nejkratších[5]. Neprocházejte však všechna, při dosažení indexu ***Cluster nearest*** (omezení hloubky) přejděte na bod 1. Místo, která tato trasa spojuje (s místem **B**), označte jako **Y**, jeho sousedy jako **X** a **Z**.
3. Pokud je místo **Y** zároveň místem **A** nebo **C**, pak[6]:
   - Inkrementujte čítač sousedů o jedna
   - Pokud má nyní hodnotu **2** (nalezeni oba sousedi), přechod na bod 1.
   - Přechod na bod 2.
4. Pokud platí, že **(K + |BY|) < (|AB| + |BC| - |AC|)**, kde **K** je menší ze vzdáleností **|BX| - |XY|** a **|BZ| - |YZ|** (princip spočívá ve volbě, zda-li je místo **B** výhodnější zařadit před nebo za místo **Y**):
   - Vyjměte místo **B** ze seznamu míst a vložte ho vedle místa **Y**. Podle toho, která vzdálenost byla menší, ho buď vložte mezi místa **X** a **Y** nebo **Y** a **Z**.
   - Aktualizujte délku cesty.
   - Návrat na bod 1.
5. Návrat na bod 2.



Obrázek 5.4: Přesun místa za účelem zkrácení délky cesty

5 V programu se při zadání parametru např. na hodnotu 5 provádí ověření pro 4 místa (na indexu 0 je implicitně vzdálenost k sobě samému).

6 Místo **Y** je jedno ze sousedů místa **B**, už je s ním spojeno, nemá smysl nic ověřovat

5.4 Provádění optimalizací

Pořadí a četnost optimalizací se v průběhu evoluce programu různily. Nakonec mají optimalizace stejnou podobu pro všechny metody (ACO, GA, deterministický algoritmus i algoritmus nejkratších cest). Provádějí se v pořadí, ve kterém byly prezentovány (výměna sousedů, křížení tras, optimalizace shluků). Pro jedno řešení se ale optimalizuje v cyklu do té doby, dokud alespoň jedna z optimalizací zkracuje cestu.

Při optimalizacích nastával problém s porovnáváním délek cest, docházelo totiž často k drobnému zlepšování cest, jelikož se desetinné číslo (datový typ double) reprezentující délku cesty mohlo při výpočtu stejné cesty různými optimalizačními způsoby lišit o zanedbatelné číslo (zaokrouhlovací chyba). Aby se této zbytečné aktualizaci délky předešlo, je při porovnávání použita malá konstanta (0,0001). Pokud by se měla provést změna, musí se řešení zlepšit alespoň o tuto hodnotu.

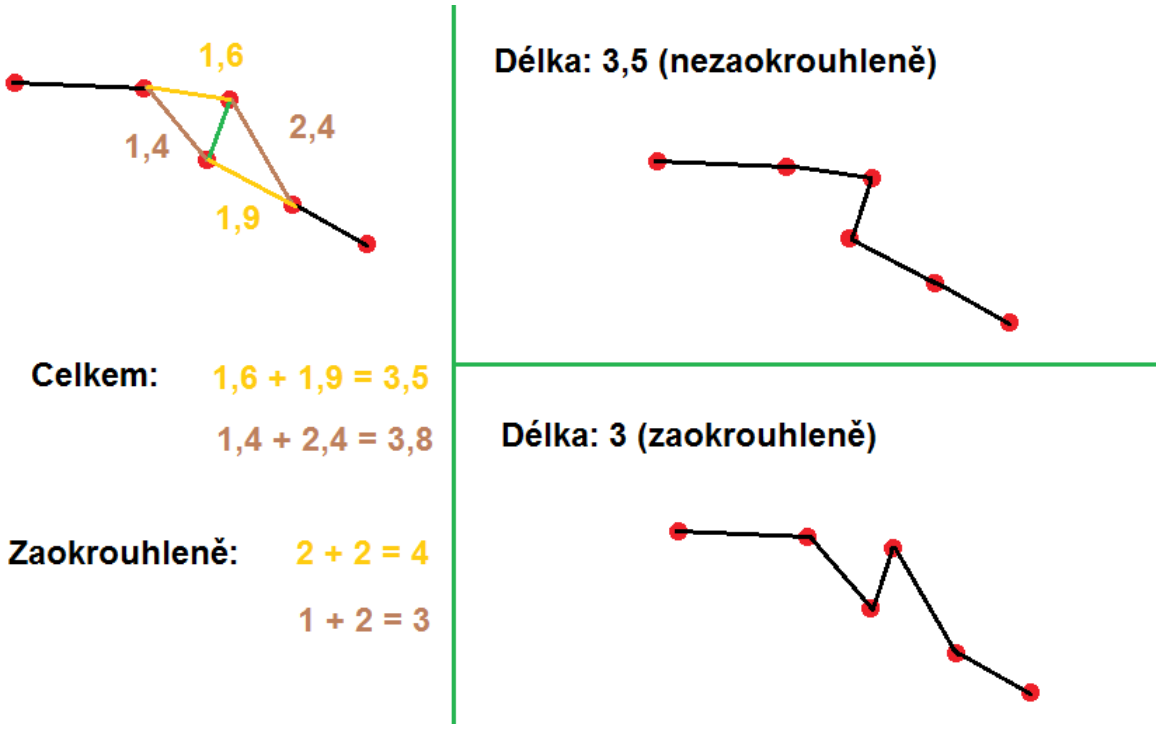
Optimalizace nelze v programu zakázat, jsou implicitně zahrnuty do algoritmů (po vytvoření každého řešení dojde k jejich provedení). Lze však potlačit opravu křížení tras nastavením parametru ***Circle nearest*** na větší hodnotu, než je počet míst a optimalizaci shluků nastavením parametru ***Cluster nearest*** na hodnotu 1. Naopak si lze povšimnout, že při zadání ***Circle nearest*** rovno jedné a ***Cluster nearest*** na hodnotu větší, než je počet míst, je rychlost výpočtu velmi pomalá. Doporučuji volit oba parametry blízko hodnotě **5**.

6 Provádění experimentů

V této kapitole popíši experimentování s různými sady míst (široké spektrum počtu míst). Z demonstračních důvodů nejprve uvedu i některé příklady se zakázanými optimalizacemi (přímo v programu). Aplikace je spouštěna na 64 bitovém operačním systému Microsoft Windows 7, s procesorem *AMD A6-3420M APU with Radeon(tm) HD Graphics 1.50 GHz* a 5,48 GB op. paměti.

Experimenty jsou prováděny se sadami dostupnými z [9]. Ovšem na tomto webu jsou uvedeny vzdálenosti, které nejsou přesné. Využívají knihovnu TSPLIB, která pracuje s celočíselnými hodnotami délek tras (dojde k zaokrouhlení). To ve výsledku znamená, že např. pro sadu míst *eil101* uvádějí celkovou délku nejlepšího řešení 629, zatímco mnou získaná hodnota, která je téměř jistě (dále budu považovat fakt) optimální, má hodnotu přibližně 640,2 (obr. 6.2). Jejich cesty nejenom, že mají kratší vzdálenost, ale také nemusí být spojeny ideálně, příklad je uveden na obrázku 6.1.

Soubory (skupiny míst) použité pro experimenty jsou přiloženy k výsledné aplikaci.



Obrázek 6.1: Změna cest při použití zaokrouhlování délek tras

6.1 Bez optimalizací

Bez lokálních optimalizací popsaných v kapitole 5 je možné efektivně řešit TSP pouze pro malý počet míst. Zobrazené výsledky jsou pro již zmíněnou ***eil101***, která obsahuje **101** míst. V tabulkách 1 a 2 jsou uvedeny průměrné vzdálenosti pro 10 pokusů s časovým omezením 1 a 5 sekund (jedná se o dva různé běhy programu). V prvních dvou sloupcích jsou výsledky pro ACO s rozdílným parametrem ***β***, v následujících 3 sloupcích je GA s odlišnou velikostí populace, v posledním z GA se navíc v každém cyklu nevytváří noví náhodní jedinci (***Random per cycle*** = 0). Předposlední sloupec tvoří algoritmus nejkratších cest a poslední pro doplnění deterministický algoritmus, který má mnohem kratší dobu výpočtu.

Metoda	ACO, $\beta = 4$	ACO, $\beta = 40$	GA, p1000	GA, p100	GA, p100-	SHORT	DETER
Délka	872,1	699,2	**668,7**	678,1	720,4	676,3	794,6

Tabulka 1: Tabulka s výsledky metod pro eil101 bez optimalizace po 1 sekundě výpočtu.

Metoda	ACO, $\beta = 4$	ACO, $\beta = 40$	GA, p1000	GA, p100	GA, p100-	SHORT	DETER
Délka	868,7	695,2	**668,8**	670,2	717,6	669,6	794,6

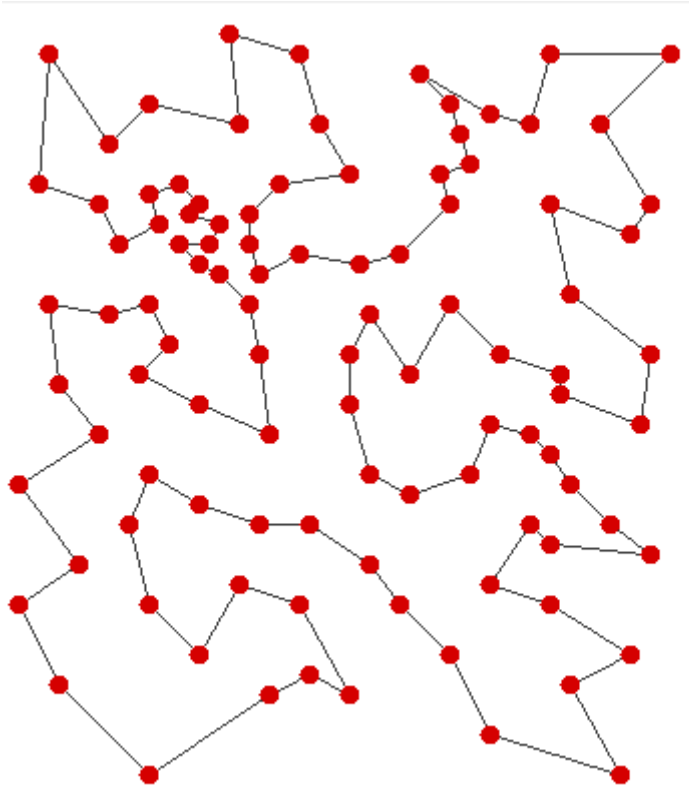
Tabulka 2: Tabulka s výsledky metod pro eil101 bez optimalizace po 5 sekundách výpočtu.

Z tabulek je patrné, že razantní rozdíl mezi délkami cest po 1 sekundě a 5 sekundách není. Přijatelné řešení pro ACO je pouze s vysokou hodnotou parametru β.

Sada **20** náhodných míst byla vyřešena optimálně pro ACO s parametrem $\beta = 40$ a taktéž pro GA i pro metodu nejkratších cest za méně než 1 sekundu. Dále se nemá cenu zabývat programem bez použití optimalizací.

6.2 S optimalizacemi

Pro demonstrační účely nejprve poukáži na rozdíly s optimalizacemi, takže budou prezentovány výsledky pro *eil101*. Později se již nebudu zaměřovat na různou volbu parametrů, ale na porovnání samotných metod. Budu zvyšovat počet míst (i čas poskytnutý pro získání řešení) a uvádět přibližnou procentuální odchylku od (předpokládané) ideální cesty.



Obrázek 6.2: Optimální řešení pro eil101

Metoda	ACO, $\beta = 4$	ACO, $\beta = 40$	GA, p1000	GA, p100	GA, p100-	SP[7]	DETER
Délka	646,9	643,1	640,7	**640,2**	**640,2**	640,8	665,3

Tabulka 3: Tabulka s výsledky metod pro eil101 s optimalizacemi po 1 sekundě výpočtu.

Metoda	ACO, $\beta = 4$	ACO, $\beta = 40$	GA, p1000	GA, p100	GA, p100-	SP	DETER
Délka	644,3	641,2	**640,2**	**640,2**	**640,2**	**640,2**	665,3

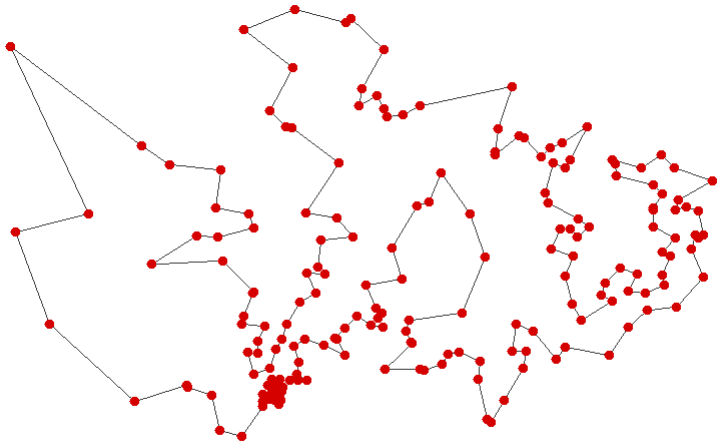
Tabulka 4: Tabulka s výsledky metod pro eil101 s optimalizacemi po 5 sekundách výpočtu.

Z porovnání tabulek 1, 2, 3 a 4 je patrné, jak moc se zlepší řešení s optimalizacemi, nejviditelnější je rozdíl mezi ACO s malým parametrem β. Při velikosti populace 100 bylo GA již po 1 sekundě nalezeno optimální řešení ve všech případech. Lze pozorovat, že GA funguje lépe, než pouhé generování náhodných řešení (algoritmus nejkratších cest). Při velikosti populace 1000 je nutno vytvořit 1000k počátečních řešení (polovinu podle SP), což způsobí, že se za 1 sekundu nemusí stihnout nalézt optimální řešení.

Pro všechny následující metody budou využity parametry implicitně zadané v programu. Odchylka bude zobrazena v procentech (pro průměrnou hodnotu) oproti předpokládanému ideálnímu řešení (je možné, že nějaké řešení označím jako ideální, přestože může existovat o velmi málou vzdálenost lepší řešení). Dále bude zobrazena minimální, maximální a průměrná délka. Všechny experimenty jsou prováděny s 10 pokusy. Deterministické řešení je zde jen pro doplnění, protože není možné ho srovnávat s ostatními metodami, protože výpočet trvá mnohem kratší čas. Deterministické řešení ve většině případů produkuje kratší cestu než algoritmus nejkratších cest s jednou iterací.

6.3 Města v Kataru

Jedná se o soubor s názvem *qa194*, který obsahuje 194 míst. Ideální řešení má délku 9353,5 (obr. 6.3). V tabulce 5 jsou zobrazeny statistiky provedených experimentů. Genetickému algoritmu ve většině případů stačí velmi malý čas k tomu, aby nalezlo ideální řešení, ostatní metody (kromě deterministického způsobu) nemají špatné výsledky, průměrná hodnota se od ideálního řešení neliší ani o jedno procento.



Obrázek 6.3: Optimální řešení pro qa194

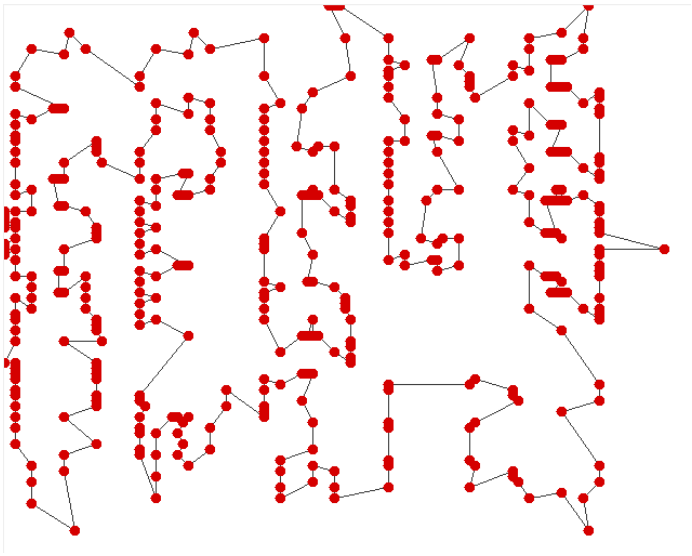
7 Shortest path – algoritmus nejkratších cest

Čas	5 sekund			15 sekund			-
Metoda	**ACO**	**GA**	**SP**	**ACO**	**GA**	**SP**	**DETER**
Min	9390,5	9353,5[8]	9371,9	9360,8	9353,5	9361,6	9768,8
Max	9454,7	9382,3	9425,3	9442,9	9353,5	9394,7	9768,8
Průměr	9425,5	9356,4	9396,7	9410,6	9353,5	9377,3	9768,8
Odchylka [%]	0,77	0,03	0,46	0,61	0	0,25	4,44

Tabulka 5: Tabulka s výsledky metod pro qa194.

6.4 Testovací sada PBK411

Tato množina **411** míst byla uměle vytvořena. Nejkratší uvedená cesta má po přepočtu vzdálenost 1365,4 (celočíselná délka 1343). Ovšem při běhu programu jsem nalezl řešení, které má délku 1359,0. Na obr. 6.4 je zobrazena cesta, která má vzdálenost 1359,5 (cesta s délkou 1359,0 nebyla uložena a tak ji nemohu graficky zobrazit). K nalezené hodnotě 1359,0 se budu chovat jako k ideální. Výsledky jsou poskytnuty v tabulce 6. Za zmínku stojí maximální hodnoty po provedení 10 experimentů po dobu 60 sekund. GA získal v nejhorším případě délku 1364,2. V tomto případě se odchylka od předpokládaného ideálního řešení neliší ani o **půl procenta**. Po 30 i po 60 sekundách výpočtu je opravdu **razantní** rozdíl odchylek mezi GA a ACO.



Obrázek 6.4: Řešení pbk411 s délkou 1359,5

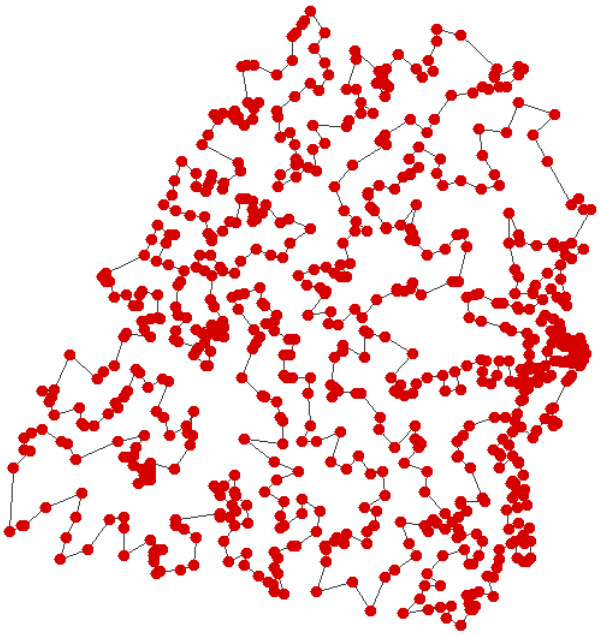
Čas	30 sekund			60 sekund			-
Metoda	**ACO**	**GA**	**SP**	**ACO**	**GA**	**SP**	**DETER**
Min	1379,8	1359,5	1379,4	1377,3	1359,0	1376,5	1427,3
Max	1395,2	1370,3	1389,1	1387,8	1364,2	1385,5	1427,3
Průměr	1389,9	1363,9	1383,2	1382,4	1360,5	1381,9	1427,3
Odchylka [%]	2,25	0,34	1,76	1,7	0,09	1,66	5

Tabulka 6: Tabulka s výsledky metod pro pbk411.

8 Tento výsledek (ideální řešení) byl získán v 9 z 10 případů

6.5 Města v Uruguayi

Jak vyplývá z názvu *uy734*, tato mapa Uruguaye obsahuje **734** míst. Tyto jsem použil, abych vypočetl skutečnou vzdálenost 79134,6 (uvedená celočíselná délka je 79114). Tato cesta je zobrazena na obrázku 6.5. Z tabulky 7 už je pozorovatelný patrný rozdíl mezi genetickým algoritmem a optimalizací mravenčí kolonií. Po 100 sekundách výpočtu je nejhorší vzdálenost GA kratší, než nejlepší poskytnutá ACO.



Obrázek 6.5: Nejlepší nalezená cesta pro uy734

V tabulkách 7, 8 a 9 se nově vyskytují další dva řádky oproti předcházejícím tabulkách. V těchto řádcích budou uvedeny výsledky a odchylka od ideálního řešení, ale pouze po provedení 1 experimentu (pro ostatní časy se jich provádí vždy 10) s větším časovým kvantem. Z tabulky 7 lze také vypozorovat, že maximální a minimální hodnota se pro algoritmus nejkratších cest po 100 sekundách liší velmi málo (81135,8 a 81241,3) v porovnání s ostatními metodami. Po 300 sekundách lze pozorovat, že genetický algoritmus se opravdu s plynoucím časem může dále zlepšovat, na rozdíl od ostatních přístupů. Průměrná délka cesty získaná pomocí SP není po 30 sekundách výpočtu o mnoho delší, než délka získaná pomocí GA.

Čas	**30 sekund**			**100 sekund**			**-**
Metoda	**ACO**	**GA**	**SP**	**ACO**	**GA**	**SP**	**DETER**
Min	81373,7	80664,2	80967,9	81107,7	79821,8	81135,8	83184,0
Max	81795,3	82510,9	81715,6	81604,3	80665,1	81385,9	83184,0
Průměr	81577,3	81224,3	81368,2	81322,4	80230,8	81241,3	83184,0
Odchylka [%]	3,09	2,64	2,82	2,76	1,39	2,66	5,12
300 sekund	81285,3	79684,1	81194,4	-	-	-	-
Odchylka [%]	2,72	0,69	2,6	-	-	-	-

Tabulka 7: Tabulka s výsledky metod pro uy734.

6.6 Testovací sada DJA1436

Tato uměle vytvořená testovací skupina obsahuje **1436** míst. Vyhodnocení výpočtů uvádím v tabulce 8. Je nutné si uvědomit, že pro takto velkou množinu míst trvá úvodní fáze, kdy se musí spočítat délka všech tras (kterých je přibližně milion) a další potřebné hodnoty, velmi dlouho. Proto je 50 sekund opravdu minimální čas, aby bylo možné získat nějaké řešení. Za ideální délku cesty považuji vzdálenost 5337,5 (uvedena celočíselná hodnota 5257). ACO má po 50 i 120 sekundách lepší výsledky než GA a SP, avšak po 500 sekundách (nemusí být absolutně vypovídající, pouze 1

test) se již pořadí změnilo. Při spuštění měl parametr β hodnotu 40, což může být další příčina, proč ACO dokázalo sestrojit lepší řešení. Pravděpodobnost výběru nejbližšího místa tak byla velmi vysoká (pokud se nastaví $\beta = 4$, pak ACO po 120 sekundách vytváří v průměru asi o 100 jednotek delší řešení než při nastavení parametru β na hodnotu 40).

Čas	**50 sekund**			**120 sekund**			**-**
Metoda	**ACO**	**GA**	**SP**	**ACO**	**GA**	**SP**	**DETER**
Min	5497,4	5510,7	5514,5	5463,1	5465,4	5502,4	5551,4
Max	5538,7	5562,3	5555,8	5526,1	5541,9	5542,0	5551,4
Průměr	5522,9	5536,6	5542,8	5507,3	5509,1	5527,7	5551,4
Odchylka [%]	3,47	3,73	3,85	3,18	3,21	3,56	4,01
500 sekund	5503,8	5438,0	5480,4	-	-	-	-
Odchylka [%]	3,12	1,88	2,68	-	-	-	-

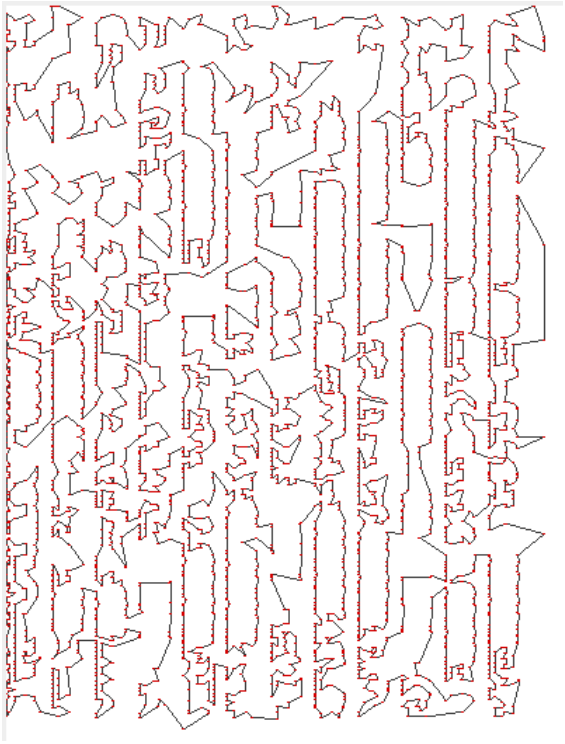
Tabulka 8: Tabulka s výsledky metod dja1436 .

6.7 Testovací sada XQE3891

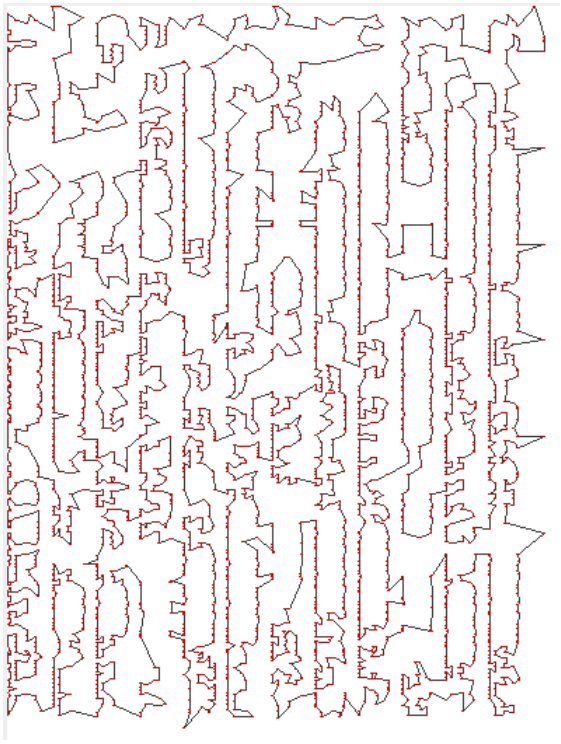
Tato testovací sada obsahuje **3891** míst. Nemá tak smysl provádět experimenty s časovým limitem v rámci desítek sekund. Rozhodl jsem se proto provést pouze **jeden** experiment pro každou metodu. Z tabulky 9 vyplývá zajímavá skutečnost. Deterministický algoritmus totiž získal kratší řešení než všechny ostatní metody za 20 minut, ten přitom netrval ani **30 sekund**. Domnívám se, že je to způsobeno konkrétní sadou a počtem míst. Pokud by někdo chtěl prozkoumat všechny možné cesty, podle vztahu 1 by jejich počet byl asi $8{,}7*10^{12600}$. Pro tento příklad považuji za ideální hodnotu vzdálenost 12216,0. Cesta získaná deterministickým algoritmem a výsledek považovaný za ideální jsou zobrazeny na obrázcích 6.6 a 6.7.

Metoda	**ACO**	**GA**	**SP**	**DETER**
20 minut	12751,4	12733,4	12746,7	12707,6
Odchylka [%]	4,38	4,24	4,34	4,02

Tabulka 9: Tabulka s výsledky metod xqe3891 .



Obrázek 6.6: Výsledek xqe3891 získaný deterministickým algoritmem



Obrázek 6.7: Výsledek xqe3891 považovaný za optimální

6.8 Zhodnocení výsledků

Průběžně jsem u každé sady míst upozorňoval na zajímavé jevy, pokusím se je ve stručnosti shrnout. Program byl spouštěn pro různý počet míst, přičemž byly využity množiny uměle vygenerovaných míst i reálně existujících. Díky poměrně dobrým vlastnostem se **nazaměřuji na menší počty míst**. Podle různých zdrojů a aplikací, které byly implementovány se většina prací zaměřuje na maximální počet okolo 100 míst, což podle mě nemá pro tuto aplikaci velký smysl. Např. v pracích [4] ani [5] nebylo nalezeno optimální řešení ani po několika minutách výpočtu, přičemž s optimalizacemi můj program poskytuje lepší výsledky pro **všechny metody již po 1 sekundě**, navíc za tuto dobu nalézá **GA** téměř vždy **optimální řešení**.

Pokud tato aplikace počítá TSP pro méně než 200 míst, pak má genetický algoritmus velkou šanci najít **velmi rychle optimální řešení**. Pokud je míst méně než 80, ACO také nalézá optimální řešení velmi rychle a to i při zadání parametru β jako malou hodnotu (např. 4). Přesto se domnívám, že má ACO dobré výsledky i pro více míst. Při vývoji jsem se zaměřoval na počet míst okolo 200 a snažil jsem se upravovat primárně GA tak, aby efektivně nalézalo optimální řešení, proto GA zvládá tento počet míst velmi dobře. S rostoucím počtem míst prokazuje genetický přístup stále dobré výsledky, jelikož je schopno s přibývajícím časem vylepšovat výsledek s poměrně obstojnou efektivitou, což ACO ani algoritmus nejkratších cest z experimentů neprokazují.

Z poskytnutých dat nemusí vyplývat, že GA nalézá pro zmíněný počet 200 míst velmi rychle optimální řešení, protože existuje víc různých velmi kvalitních řešení, ze kterých kombinací může vzniknout ideální. Tento způsob ovšem nefunguje u ACO, které buď potřebuje velmi dlouhou dobu k ustálení feromonových stop nebo lze nastavením parametru β na velkou hodnotu (např. 40) zaručit vysokou pravděpodobnost přechodu k **blízkým místům**, což způsobí nalezení kratších řešení. Ovšem nalézt právě to **optimální** trvá déle než genetickým algoritmem, protože se vytváří velice **podobná řešení**. Pro 200 míst je nalezení právě optimálního řešení způsobem ACO velmi nepravděpodobné,

algoritmus je totiž zaměřen na jednu konkrétní cestu (vychází ze skutečných mravenců, kteří se snaží nalézt a zlepšit jen jednu cestu). Naproti tomu genetický algoritmus vychází z velké rozmanitosti populace, kdy se spoléhá na určité zlepšení plynoucí z kombinace jedinců.

Jelikož se navrhnuté optimalizace zaměřují na lokální zlepšení, lépe fungují v kombinaci s GA (kvůli rozmanitosti je možné zlepšit velký počet cest, zatímco ACO má velmi podobné cesty a optimalizace vytvoří opět velmi podobné či stejné cesty). Pro 1000 a více míst nevěřím, že by bylo pomocí ACO reálné získat právě **ideální** řešení i za předpokladu dlouhého běhu (právě kvůli absenci rozmanitosti). Na druhou stranu věřím, že GA má předpoklady toho dosáhnout. Domnívám se ale, že pro větší efektivitu by bylo nutné upravit způsob křížení. V aktuální podobě se vždy náhodně vyberou 2 indexy a rekombinace probíhá od menšího po větší index. Pokud by bylo PMX křížení nahrazeno jiným způsobem nebo by pro něj byla povolena výměna i od vyššího indexu po nižší (lze provést, protože řešení je reprezentováno kruhovým seznamem) a zároveň byl nějakým způsobem **omezen maximální počet míst (genů), která se budou vyměňovat** (např. 100 nebo 200)[9], pak věřím, že by se dosahovalo lepších výsledků i pro větší počty míst.

Algoritmus nejkratších cest spolu s optimalizacemi funguje do jisté míry jako hrubá síla. Doporučil bych ovšem vytvořit asi 1 000 až 10 000 řešení pro dobrý výsledek (např. pro 100 míst postačuje ve většině případů 10 000 jedinců k nalezení optimálního řešení) a 10 000 až 100 000 pro další zlepšení. Se zvětšováním počtu se ovšem ve většině případu bude nejlepší nalezené řešení jen zřídkakdy měnit. **Velmi dobře a rychle** tak lze získat „slušné“ řešení. Jako algoritmus samotný bych ho doporučil pouze pokud by byly omezené výpočetní prostředky, slouží však výborně jako generátor nových řešení pro prezentovaný GA, kde tímto způsobem vzniká polovina všech nových řešení (pouze polovina, protože SP neumožňuje vytvořit libovolné řešení, ale druhý způsob popsaný v části 3.2.2 má s určitou malou pravděpodobností možnost vytvořit libovolného jedince). Z porovnání GA a SP tak lze usoudit, jak velké zlepšení přináší populační přístup do této problematiky. Nejvíce se tento rozdíl prohlubuje s dostatečným výpočetním časem, jelikož GA využívá výskyt velmi dobrých jedinců ve své populaci k jejich křížení a vzniku ještě lepších.

Deterministický způsob neposkytuje velmi kvalitní řešení (jeho odchylka od ideální cesty se pohybuje kolem 5 %), ale trvání samotného výpočtu je velmi krátké. Nejdelší částí je vytvoření seřazeného seznamu cest. To do jisté míry zlepšuje jeho deterministická verze SP. Pro velký počet míst se tak pro rychlé řešení vyplatí nejen získat deterministické řešení, ale vytvořit i malý počet řešení pomocí SP a vybrat nejlepšího z nich (tento způsob je taktéž velmi rychlý).

Z rozdílů mezi obrázky 6.6 a 6.7 se naskýtá příležitost modifikovat deterministický algoritmus. Algoritmus v každém cyklu pro každé místo zjišťuje, jakou nejkratší možnou vzdálenost musí urazit pro spojení s jiným místem. Nyní probíhá výběr té nejkratší z nejkratších možných (pro každé místo), avšak řešení založené na výběru **nejdelší z nejkratších** by mohlo produkovat taktéž slibné výsledky. Případně výběr jednou na základě nejdelší a jednou nejkratší vzdálenosti či náhodné střídání, čímž by opět vznikl námět na jiný již nedeterministický způsob.

9 Například pokud je prováděno křížení pro mapu obsahující 3000 míst a má se provést křížení mezi indexy 100 a 2500, pak nejspíše tyto potomci nebudou vůbec připomínat svoje rodiče, touto úpravou by se povolil maximální rozdíl indexů pouze 100 nebo 200.

7 Závěr

Hlavním cílem této práce bylo vytvořit aplikaci, která bude schopna řešit TSP. Domnívám se, že kvalita dosažených výsledků v kapitole 6 není vůbec špatná. Dále se práce zaměřuje na porovnání různých metod, přičemž GA a ACO byly modfikacemi změněny z obecně známých metod tak, aby poskytovaly lepší výsledky. Přidal jsem experimentálně dvě nové metody, jedna nebyla pojmenována, ale poskytuje vždy stejný výsledek pro konkrétní sadu míst, takže pokud se na ni referuji, pak používám název deterministický algoritmus. Návrh na další úpravy je uveden v posledním odstavci části 6.8. Druhou metodu (úprava první) jsem nazval pro účely této práce jako metodu nejkratších cest (SP).

Aplikace byla při vývoji neustále spouštěna a byly analyzovány dosažené výsledky včetně použití funkce **profiler**, která pomohla odhalit pomalé části kódu, aby se modifikacemi dosahovalo ještě lepších výsledků. Pomocí prvního řádku v souboru s místy lze změnit vizualizaci míst tak, aby se všechna zobrazila na plátno. Implicitně se zobrazují souřadnice pouze od [0, 0] do [1000, 550].

Výsledná aplikace není dokonalá, existuje mnoho možností pro její vylepšení, některé zmíním. V aktuální podobě není možné uložit mapu, což by se mohlo hodit obzvláště při vytvoření náhodné sady míst, která má předpoklad být znovupoužita. Dále by bylo možné poskytnout uživateli možnost pro zakázání optimalizací, uložení a nahrátí aktuální kombinaci parametrů. Co se týče rychlosti a struktury programu, byl vytvořen jako **jednovláknový** (mimo částí komunikujících s uživatelem). Některé části však **mohou probíhat paralelně** a tak pokud by došlo k modifikaci částí kódu, výpočet na vícejádrových procesorech by mohl být mnohem rychlejší. Aplikace nyní počítá úvodní fázi vzdáleností mezi místy pro každé řešení znovu, bylo by možné do programu přidat podmínku tak, aby se při více kontinuálních řešení různými či stejnými přístupy využívalo již získaných vzdáleností.

Jako metoda s **nejlepšími výsledky** se jeví **GA**, přičemž v kapitole 6.8 nastiňuji, jak by se dala ještě více zlepšit efektivita pro velké počty míst pomocí změny způsobu rekombinace jedinců. Dalšími možnostmi pro zlepšení výkonnosti programu by byly možné modifikace principu ACO a hlavně podrobné experimentování s velikostmi všech parametrů tak, aby se získaly nejvýhodnější hodnoty. Úpravami pořadí prováděných optimalizací taktéž vzniká prostor pro experimentální zjišťování lepší efektivity metod.

Literatura

[1] ÜÇOLUK, Göktürk. Genetic Algorithm Solution of the TSP Avoiding Special Crossover and Mutation. Dostupné z: https://www.ceng.metu.edu.tr/~ucoluk/research/publications/tspnew.pdf

[2] Class CycleCrossover. [online]. [cit. 2015-05-04]. Dostupné z: http://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/genetics/CycleCrossover.html

[3] GAERTNER, Dorian a Keith CLARK. *On Optimal Parameters for Ant Colony Optimization algorithms* [online]. [cit. 2015-05-04]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.6751&rep=rep1&type=pdf

[4] RAMADAN, Saleem Zeyad. Reducing Premature Convergence Problem in Genetic Algorithm: Application on Travel Salesman Problem. *Computer and Information Science*. 2013. Dostupné z: http://www.ccsenet.org/journal/index.php/cis/article/viewFile/18406/14953

[5] KARÁSEK, Štěpán. *SFC – Praktická aplikace GA: Problém obchodního cestujícího*. Projekt do předmětu SFC. FIT VUT v Brně.

[6] Cong., S., Jia, Y., Deng., K.: *Particle Swarm and Ant Colony Algorithms and Their Applications in Chinese Traveling Salesman Problem*, in New Achievements in Evolutionary Computation, INTECH, February 2010

[7] ZBOŘIL, František a František ZBOŘIL jr. *Základy umělé inteligence*. FIT VUT v Brně, 2012. Skripta do předmětu IZU.

[8] ZBOŘIL, František. *Soft Computing*. FIT VUT v Brně, 2015. Studijní pora k předmětu SFC.

[9] *The Traveling Salesman Problem* [online]. [cit. 2015-05-06]. Dostupné z: http://www.math.uwaterloo.ca/tsp/

[10] *Touring Singer Problem (TSP)* [online]. [cit. 2015-05-06]. Dostupné z: https://github.com/dideler/intro-to-genetic-algorithms/wiki

Seznam příloh

Příloha 1: CD se zdrojovými soubory, složkou obsahující instance míst a tímto textem ve formátech pdf a odt.