UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA
KATEDRA MATEMATICKÉ ANALÝZY A APLIKACÍ MATEMATIKY

DIPLOMOVÁ PRÁCE

Genetické algoritmy pro řešení úlohy obchodního cestujícího

Vedoucí diplomové práce:
RNDr. Pavel Ženčák, Ph.D.
Rok odevzdání: 2012

Vypracovala:
Bc. Kateřina Stránská
MAP, II. ročník

Prohlášení

Prohlašuji, že jsem vytvořila tuto diplomovou práci samostatně za vedení RNDr. Pavla Ženčáka, Ph.D. a že jsem v seznamu použité literatury uvedla všechny zdroje použité při zpracování práce.

V Olomouci dne 30. března 2012

Poděkování

Ráda bych na tomto místě poděkovala zejména vedoucímu diplomové práce RNDr. Pavlu Ženčákovi, Ph.D. za obětavou spolupráci i za čas, který mi věnoval při konzultacích. Dále bych chtěla poděkovat své rodině a přátelům, kteří mě po celou dobu studia podporovali.

Obsah

Úvodní slovo

Cílem této diplomové práce je nastudovat problematiku genetických algoritmů pro problém obchodního cestujícího, programově realizovat algoritmy a provést numerická srovnání různých parametrů.

Problém obchodního cestujícího je jedním z nejznámějších a nejpopulárnějších kombinatorických problémů. Má poměrně bohatou historii, kterou lze sledovat už od roku 1759, kdy se úlohou tohoto typu zabýval švýcarský matematik Leonhard Euler, který se zajímal o vyřešení „problému jezdce". Řešení popisuje způsob, jakým jezdec při své cestě navštíví každé z 64 polí na šachovnici právě jednou. Velkým impulsem pro zájem o problém se v druhé polovině dvacátého století stal rozvoj lineárního a celočíselného programování. Problém obchodního cestujícího má obrovské množství praktických aplikací v logistice, plánování, výrobě VLSI obvodů, krystalografii a mnohých dalších oborech lidské činnosti. Proto snaha o nalezení vhodných metod pro jeho řešení nijak nepolevuje ani v současné době.

Jelikož se jedná o problém, který je NP-úplný, nepředpokládá se, že se podaří najít deterministický algoritmus, jenž by dokázal nelézt optimální řešení v polynomiálně omezeném čase. Z tohoto důvodu bylo za několik posledních desetiletí navrženo velké množství aproximativních algoritmů včetně nepřeberného množství evolučních algoritmů.

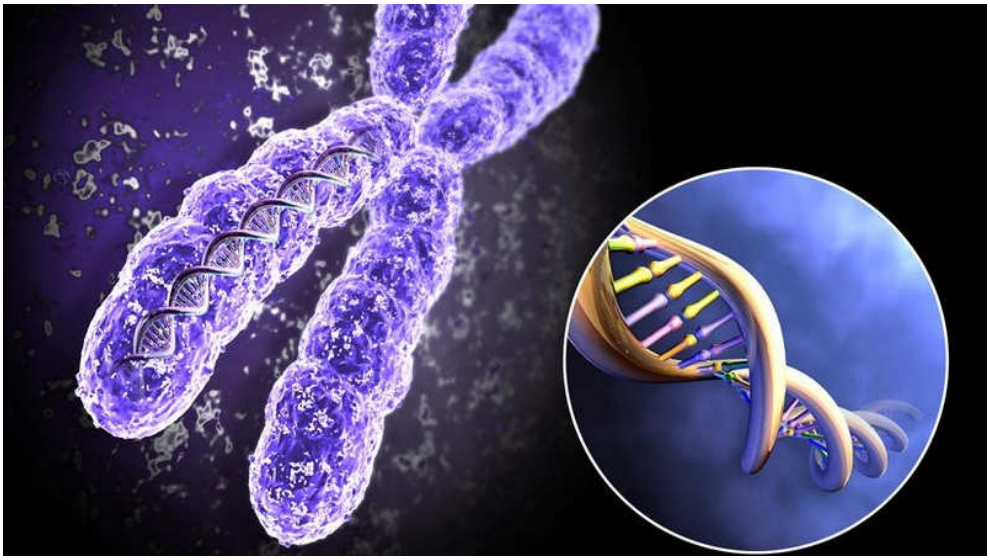
My v této práci čtenáře nejprve seznámíme v přípravné kapitole s definicí problému obchodního cestujícího a také popíšeme základní schéma genetického algoritmu.

Ve druhé kapitole jsou uvedeny některé vybrané metody řešení úlohy obchodního cestujícího, a sice konstrukční heuristiky a zlepšovací heuristiky.

Třetí kapitola je pak věnovaná samotným genetickým algoritmům. Jsou zde popsány možné způsoby reprezentace pro úlohu obchodního cestujícího, ohodnocení, metody selekce, rekombinační operátory, náhradové strategie a také zastavovací kriteria.

Ve čtvrté kapitole se věnujeme konvergenci a pátá kapitola patří popisu vlastního algoritmu a jeho parametrů. Jsou zde uvedeny výsledky testování na úloze

se 131 uzly, jejich zhodnocení a porovnání s výsledky uvedenými v literatuře. Vše budeme programovat v programovacím jazyku MATLAB.



Obrázek 0: Chromozom.[1]

[1]Zdroj: *http://udalosti.noviny.sk/zo-zahranicia/16-11-2009/klucom-k-dlhovekosti-su-chromozomy-s-dlhsimi-telomerami.html*, [citováno 29. 3. 2012].

1 Přípravná kapitola

První podkapitolu této kapitoly věnujeme problému obchodního cestujícího - definujeme tento problém, uvedeme historické souvislosti a některé z aplikací. V další podkapitole pak popíšeme základní myšlenku genetických algoritmů.

1.1 Problém obchodního cestujícího

Takže co má ten obchodní cestující vlastně za problém? Pojďme si ho definovat. Budeme vycházet z literatury [5], [7].

Definice 1.1. Je dána množina měst $\{c_1, c_2, \ldots, c_N\}$ a pro každou dvojici $\{c_i, c_j\}$ navzájem různých měst je dána vzdálenost $d(c_i, c_j)$. Úkolem obchodního cestujícího je najít uspořádání π všech měst, které minimalizuje hodnotu

$$\sum_{i=1}^{N-1} d\left(c_{\pi(i)}, c_{\pi(i+1)}\right) + d\left(c_{\pi(N)}, c_{\pi(1)}\right).$$

Tato hodnota se nazývá **délka cesty**, kterou obchodní cestující ujede, než projede všemi městy v pořadí daném permutací π a vrátí se do výchozího města.

Poznámka 1.1. V závislosti na vlastnostech matice $C = (c_{ij})$, kde c_{ij} představuje vzdálenost města i od města j $(i, j = 1, ..., N)$, můžeme problém dodatečně klasifikovat:

a) Jestliže platí $c_{ij} = c_{ji}$ pro všechna i a j, řekneme, že problém je **symetrický**. V opačném případě je **asymetrický**.

b) Pokud pro všechna i, j a k platí trojúhelníková nerovnost: $c_{ik} \leq c_{ij} + c_{jk}$, nazveme problém **metrický**.

c) Odpovídají-li hodnoty c_{ij} vzdálenostem bodů v rovině, nazveme problém **euklidovský**. Je zřejmé, že euklidovský problém je symetrický i metrický. Jeho vyřešení je snazší než vyřešení předchozích verzí.

Poznámka 1.2. Z praktických důvodů budeme délku cesty z uzlu i do uzlu j (tedy prvek c_{ij}) nazývat **cenou cesty** z uzlu i do uzlu j. Tedy i výsledná celková délka cesty bude chápána jako cena cesty. Matici C potom budeme nazývat **cenovou maticí**. Tento pojem cena můžeme chápat podle potřeby - jako skutečnou vzdálenost, jako čas potřebný k ujetí z města do města nebo jako ekonomické náklady na cestu.

Poznámka 1.3. Ve většině aplikací se používá symetrický problém. My v této práci budeme uvažovat právě pouze symetrickou úlohu.

V terminologii teorie grafů bychom problém obchodního cestujícího (Travelling Salesman Problem - TSP) popsali jako úkol nalézt nejlevnější hamiltonovský cyklus v úplném ohodnoceném grafu s N uzly.

Abychom ukázali, proč je problém obchodního cestujícího komplikovaný, definujeme třídy výpočtové složitosti. K tomu budeme potřebovat pojem **Turingův stroj**, což je teoretický model počítače popsaný matematikem Alanem Turingem. Skládá se z procesorové jednotky, tvořené konečným automatem, programu ve tvaru pravidel přechodové funkce a pravostranně nekonečné pásky pro zápis mezivýsledků. Využívá se pro modelování algoritmů v teorii vyčíslitelnosti. Pro přesnější definici můžete nahlédnout například do [11]. Definice tříd výpočtové složitosti jsme převzali z [10].

Definice 1.2. Třída složitosti P obsahuje všechny úlohy, jejichž řešení lze nalézt deterministickým Turingovým strojem v polynomiálním čase.

Definice 1.3. Třída složitosti NP je množina problémů, které lze řešit v polynomiálně omezeném čase na nedeterministickém Turingově stroji - na počítači, který umožňuje v každém kroku rozvětvit výpočet na n větví, v nichž se posléze řešení hledá současně.

Poznámka 1.4. Jako **problém P versus NP** se v teoretické informatice označuje otázka, zda platí rovnost P = NP. Považuje se za nejdůležitější otevřený problém tohoto oboru. Většina expertů věří, že P je vlastní podmnožinou NP.

Definice 1.4. NP-úplné problémy jsou takové nedeterministicky polynomiální problémy, na které jsou polynomiálně redukovatelné všechny ostatní problémy z NP.

Poznámka 1.5. Dalo by se říci, že třídu NP-úplných problémů tvoří v jistém smyslu ty nejtěžší úlohy z NP.

Jak již bylo řečeno v úvodu, TSP patří do třídy NP-úplných problémů. Jednoduchý algoritmus prohledávající všechny cesty musí projít všechny permutace množiny 1 až N, kterých je $N!$. Je zřejmé, že z hlediska času výpočtu je tento postup nepoužitelný již pro relativně malé hodnoty N. Nepředpokládá se, že se podaří najít algoritmus, který by umožňoval nalézt optimální řešení v přijatelném, tedy nejhůře polynomiálně omezeném čase (za obecně uznávaného předpokladu, že P $\neq$ NP). Proto bylo věnováno velké úsilí návrhu nejrůznějších přibližných algoritmů, z nichž alespoň některé budou v následujícím textu stručně představeny (kapitola 2).

Poznámka 1.6. Při řešení TSP nás zajímá především délka trasy popsaná nalezeným řešením. Z výše uvedeného je ale zřejmé, že dost důležitý je i čas potřebný k výpočtu. Proto se většinou musí udělat nějaký kompromis - řešení je sice horší, ale nalezneme ho v přijatelném čase. K tomu byly navrženy mnohé aproximační algoritmy.

Z historického pohledu byl v roce 1954 velký úspěch, když se podařilo najít optimální trasu pro 49 měst. V dnešní době jsou známé optimální trasy pro některé konkrétní příklady, například pro problém s 85 900 městy ([6]). Ve skutečnosti může být v tomto okamžiku počet měst, pro která je známa optimální cesta, mnohem vyšší, neboť se na tom neustále pracuje.

1.2 Genetické algoritmy

Genetické algoritmy (GA) jsou založeny na myšlence darwinovského principu evoluce ([1], [3], [4]).

Všechno to začalo v roce 1859, kdy Charles Darwin poprvé publikoval svoji proslulou knihu *O vzniku druhů přirozeným výběrem čili zachováním vhodných odrůd v boji o život.* Po uplynutí více než sta let se myšlenkou přirozeného výběru a přežití těch nejschopnějších inspirovala rozvíjející se oblast výzkumu, jehož cílem je tyto přírodní procesy napodobit.

Průkopníkem v oblasti GA se stal americký teoretický biolog John Holland, který studoval elementární procesy v populacích, jež jsou z hlediska evoluce nepostradatelné. Na základě těchto výzkumů navrhl genetický algoritmus jako abstrakci příslušných biologických procesů. V těchto algoritmech je prohledávací prostor problému reprezentován jako soubor *jedinců*. Tito jedinci jsou reprezentováni řetězcem (případně maticí) konečné délky, který bývá v analogii s obecnou genetikou nazýván *chromozom* a je v něm vhodným způsobem zakódované řešení problému. Účelem genetických algoritmů je najít jedince z prohledávaného prostoru s nejlepší „genetickou výbavou". Kvalita (ohodnocení) jedince, tzv. *fitness*, je měřena ohodnocovací funkcí.

Jako další významnou osobnost zmíníme Davida Goldberga, který je považován za jednoho z klasiků této disciplíny a který systematicky studoval GA z technického pohledu, tedy z pohledu snažícího se chápat GA jako techniku obecně aplikovatelnou na širokou třídu úloh. Dále Zbigniew Michalewicz je považován za jednoho z původců výrazných modifikací klasických GA.

Na závěr zmíníme Johna R. Kozu - zakladatele genetického programování. Genetické programování navazuje na genetické algoritmy. Využívá metod podobných biologické evoluci při vytváření počítačových programů, které co nejlépe řeší konkrétní úlohu (nebo skupinu úloh). Jedná se o metodu strojového učení, která používá evoluční algoritmy, které postupně zlepšují populaci počítačových programů. Od samotných genetických algoritmů se genetické programování odlišuje zejména ve způsobu, jakým jsou jedinci podléhající evolučnímu procesu reprezentováni, jak jsou interpretováni a ohodnocováni.

Genetický algoritmus, ve kterém t znamená vývojový čas a $G(t)$ populaci, lze zapsat zhruba následovně:

begin

$t := 0$;

inicializace $G(0)$;

vyhodnocení $G(0)$;

while (**not** zastavovací pravidlo) **do**

begin

$t := t + 1$;

selekce $G(t)$ **z** $G(t-1)$;

změna $G(t)$;

vyhodnocení $G(t)$;

end

end

Inicializace $G(0)$ znamená vytvoření počáteční populace. Ta může být sestavena z jedinců, kteří byli vytvořeni s využitím apriorních znalostí o úloze (jsou-li k dispozici), nebo může být vygenerována zcela náhodně.

V kroku **vyhodnocení** $G(t)$ je vypočtena kvalita každého jedince v populaci.

Selekce simuluje v GA proces přirozeného výběru. Znamená vybrat z populace jedince, kteří se budou podílet na vytvoření následující populace (populaci nazýváme též *generace*). Tento výběr probíhá pomocí nějakého selekčního, zpravidla pravděpodobnostního, kriteria. Cílem je, aby průměrné ohodnocení nové generace bylo lepší než průměrné ohodnocení generace předchozí.

Operace nazvaná **změna** je realizována pomocí tzv. *rekombinačních operátorů*. Zpravidla se používají operátor **křížení** a operátor **mutace**. Mutace a křížení hrají v GA různé role. Mutace je unární operace, která trochu změní konkrétního jedince. Umožňuje prozkoumání nových stavů a napomáhá tomu, abychom

neuvízli v lokálním optimu. Oproti tomu křížení generuje jednoho či více jedinců z několika (zpravidla dvou) rodičů. V případě dvou rodičů je to binární křížení. Křížení by mělo zvýšit průměrnou kvalitu populace. Výběrem vhodných operátorů křížení a mutace stoupá pravděpodobnost, že GA najde řešení blízké optimálnímu v přijatelném počtu iterací.

Postup, jak ze smíšené množiny rodičů a potomků vybrat novou populaci, se nazývá **vývojová strategie**. Zpravidla se hovoří o dvou typech vývojových strategií:

- generační (*generational evolution*), kdy je úplně celá populace nahrazena následující (což představuje analogii životního cyklu jednoletých rostlin).

- postupné (*steady state evolution*), při které do křížení vstupuje jen malá část populace a rodiče potom koexistují se svými potomky (model víceletých rostlin či vývoje déle žijících živočichů).

Pro **zastavení algoritmu** mohou být použita různá kriteria. Např. pokud je možné předem určit počet iterací, které bude třeba vykonat. Nebo zastavit výpočet, pokud po daný počet cyklů nedošlo ke zlepšení nejlepšího jedince. Zastavovací kriteria budou podrobně popsána později.

Důležitým parametrem, který ovlivňuje chování GA, je **velikost populace** N. Příliš malá populace může mít při vysokém působení změn za následek chaotické chování ve vývoji generací. Pokud by pravděpodobnost změny (křížení nebo mutace) byla příliš malá, bylo by prohledávání prostoru nedostatečné a mohli bychom uvíznout v lokálním optimu. Příliš velká populace by zase mohla zbytečně zpomalit průběh výpočtu. Předem nelze určit, jak velká populace bude pro daný problém fungovat nejlépe. Musí se určit experimentálně.

1.2.1 Aplikace genetických algoritmů

V úvodu jsme vyjmenovali celou řadu aplikací GA. Jsou to především inženýrské aplikace a všechny byly víceméně experimentálního charakteru. Skutečně praktických aplikací GA mnoho není. Uveďme ale dva významné výsledky ([1]):

Návrh motorů pro letadlo Boeing 777

O této aplikaci nemáme z pochopitelných důvodů mnoho detailů. Nicméně GA byly použity při vývoji proudového motoru pro letoun Boeing 777. Motor samotný byl navržen klasickými metodami s důrazem kladeným na minimální spotřebu paliva. Avšak pro optimalizaci a detailní doladění některých parametrů byly použity genetické algoritmy.

Dosažený výsledek umožnil snížení nákladů o téměř 2,5%, což u jednoho letounu činí úsporu asi 2 mil. dolarů ročně.

Identifikace zločinců

Zcela netechnickou aplikací genetických algoritmů je úloha rekonstrukce vzhledu osob podezřelých ze spáchání zločinu. Tato metoda je již v USA poměrně rozšířená a považuje se za velice účinnou.

Zajímavé na této aplikaci je to, že účelovou funkci v úloze subjektivně zprostředkuje svědek zločinu. GA pracuje s relativně malou populací (aby svědek bez problémů zvládl její rozsah) a použitím rekombinačních operátorů generuje různé nákresy obličejů podezřelého. Do chromozomů je zakódováno několik atributů, jako např. tvar hlavy, posazení a barva očí, tvar nosu, úst a tváří. Pomocí těchto rysů jsou též zakódována v databázovém tvaru zločinecká alba.

Na začátku algoritmu je vygenerována sada obličejů a svědek určí pořadí podobnosti s hledanou osobou. Na základě tohoto hodnocení jsou selektování nejpodobnější jedinci a jejich rekombinací vzniknou obličeje, které by měly být podobnější hledané osobě.

Podle literatury ([1]) je tato technická podpora při hledáni podezřelých dosti úspěšná. Dokonce je lepší v porovnání s „ručními" náčrty podob zločinců.

2 Některé vybrané metody pro řešení TSP

V této kapitole uvedeme některé typy algoritmů a heuristik pro řešení TSP, které sice většinou nenaleznou globální optimum, ale konstruují jistým způsobem lokálně optimální řešení. Jsou to heuristiky konstrukční a zlepšovací ([3], [5], [8]).

2.1 Konstrukční heuristiky (Construction Heuristics)

Konstrukční heuristiky postupně budují a optimalizují hamiltonovský cyklus podle určitého pravidla (například z hlediska lokálního minima). Tyto heuristiky se liší jen způsobem, jak se vyberou a vloží další vrcholy.

My si detailně popíšeme heuristiku nejbližšího souseda (Nearest Neighbour Heuristic) a heuristiku vkládání (Insertion Heuristic).

2.1.1 Heuristika nejbližšího souseda

Nalezení nejbližšího souseda je asi nejpřirozenější myšlenkou při hledání řešení TSP. Tento algoritmus napodobuje cestujícího, který vždy navštíví nejbližší dosud nenavštívené město.

Heuristika nejbližšího souseda konstruuje pořadí $c_{\pi(1)}, \ldots, c_{\pi(N)}$ měst s počátečním městem $c_{\pi(1)}$ vybraným náhodně. Obecně za $c_{\pi(i+1)}$ je vybrané to město c_k, které minimalizuje $\left\{d(c_{\pi(i)}, c_k) : k \neq \pi(j), 1 \leq j \leq i\right\}$. Korespondující cesta prochází městy ve zkonstruovaném pořadí a po navštívení města $c_{\pi(N)}$ se vrátí do $c_{\pi(1)}$.

Nevýhodou této heuristiky je fakt, že na začátku cesty volíme levné úseky, ale v závěru se úseky hodně prodlužují a tím se výsledná cesta natahuje.

2.1.2 Heuristika vkládání

Heuristika vkládání začíná s malou podmnožinou vrcholů (nejméně však se 3), které tvoří hamiltonovský cyklus, a postupně přibírá další a další vrcholy až nakonec zapojí do výsledného cyklu všechny vrcholy grafu.

Počáteční cyklus se může sestrojit například pomocí 3 případně 4 vrcholů,

které vytvoří trojúhelník nebo čtyřúhelník. Jeden ze způsobů je ten, že se vezmou krajní body, tedy bod nejvíce nalevo, bod nejvíce napravo a bod nejvíce nahoře nebo dole.

Pro způsob, jak další bod do cyklu vložit, uvedeme tři neznámější strategie:

- vkládání **nejbližšího bodu**: vloží vrchol, jehož minimální vzdálenost od vrcholů cyklu je nejmenší.
- vkládání **nejvzdálenějšího bodu**: vloží vrchol, jehož minimální vzdálenost od vrcholů cyklu je největší.
- **nejlacinější** vkládání: z nenavštívených vrcholů vybere ten, jehož vložením se délka cyklu prodlouží co nejméně.

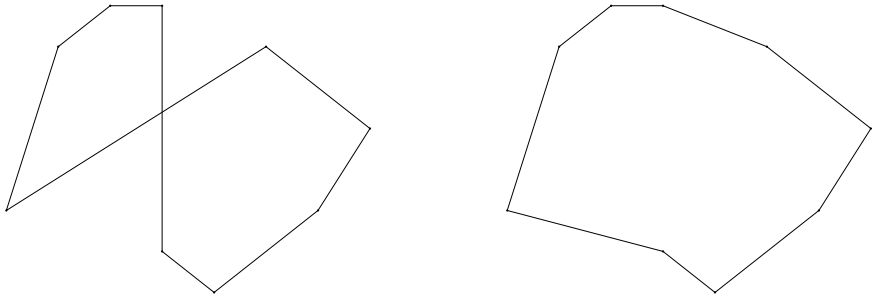
2.2 Zlepšovací heuristiky (Improving Heuristics)

V této podkapitole se budeme zabývat algoritmy založenými na lokálním prohledávání. Většinou najdou lokální optimum.

2.2.1 2-optimalizace (2-opt)

Mezi zlepšovací heuristiky patří 2-optimalizace. Tato metoda byla poprvé navržena G. A. Croesem v roce 1958. Algoritmus začíná s nějakou vhodně zvolenou počáteční permutací měst (cestou) T a snaží se ji dále vylepšovat. Pokud smazáním dvou hran a opětovným jiným spojením tak, aby vznikl opět uzavřený cyklus, vznikne kratší cyklus, hrany se vymění. To se nazývá 2-výměna. Z tohoto důvodu se definuje okolí cesty T tak, že do této množiny patří všechny cesty, které se od T liší vzájemnou výměnou dvou přímo nenavazujících hran z cesty T. Existuje-li v okolí cesty T cesta T' s menšími celkovými náklady, pak cesta T' nahradí cestu T a pokračuje se dále stejným způsobem. Pokud už žádná cesta s nižšími náklady v okolí cesty T není, algoritmus končí - cesta T je 2-optimální. Pro urychlení algoritmu se v jednotlivých iteracích obvykle negeneruje celé okolí cesty T, ale 2-výměna je provedena ihned po nalezení první kratší cesty.

Průběh 2-optimalizace je znázorněn na obrázku 1.



Obrázek 1: Vlevo cyklus před 2-optimalizací, vpravo cyklus po 2-optimalizaci.

2.2.2 k-optimalizace (k-opt)

k-optimalizace je zobecněním heuristiky 2-opt. V tomto případě se v každém kroku snažíme vylepšit trasu tak, že smažeme k existujících hran a vložíme k nových tak, aby výsledná trasa byla opět hamiltonovský cyklus. S rostoucí hodnotou parametru k se však exponenciálně zvětšuje mohutnost okolí původní cesty a stejným způsobem roste i potřebný čas. Z tohoto důvodu se v praxi používá k-opt s hodnotou maximálně $k = 3$.

3 Genetické algoritmy pro TSP

V této kapitole detailně popíšeme způsoby reprezentace jedinců, které se používají při řešení TSP. Dále se zde budeme zabývat selekcí a rekombinačními operátory (což jsou mutace a křížení) pro TSP, uvedeme náhradové strategie a zastavovací kriteria.

3.1 Reprezentace

V genetických algoritmech jsou jedinci reprezentováni pomocí řetězců S konečné délky L ve tvaru $(s_1, s_2, \ldots, s_L)$. Jak již bylo zmíněno dříve, bývají tyto řetězce nazývány chromozomy. Každá pozice v řetězci se v analogii s obecnou genetikou nazývá *alela* a konkrétní symbol s_i v chromozomu se nazývá *gen*.

Způsob, jakým jsou individua zakódována (geneticky popsána), je velmi důležitý pro úspěch či neúspěch GA na konkrétní úloze. My v této práci uvedeme několik nejpoužívanější způsobů reprezentace a zmíníme jejich výhody, popř. nevýhody. Budeme vycházet z literatury [4].

3.1.1 Binární reprezentace

Historicky nejstarší je **binární kódování**, tzn. jedinec je reprezentován chromozomem určeným dvojkovým řetězcem fixní délky. Každý gen je reprezentován řetězcem délky $\log_2 N$. Pro TSP s N městy má potom jeden chromozom délku $N \log_2 N$.

Uveďme například binární reprezentaci 6 měst pro TSP:

i	město i	i	město i
1	000	4	011
2	001	5	100
3	010	6	101

Takto definovanou reprezentací by cesta $1-2-3-4-5-6$ byla vyjádřena takto:

$$(000 \quad 001 \quad 010 \quad 011 \quad 100 \quad 101)\,.$$

Tato reprezentace není pro TSP příliš praktická. Navíc provedením rekombinace nevznikne opět uzavřený cyklus (nevznikne platná cesta) a musí být použity opravné algoritmy, které mohou být časově náročné. Ukažme si to na příkladu jednobodového křížení. Mějme dvě řešení TSP:

$$(000 \quad 001 \quad 010 \quad 011 \quad 100 \quad 101)\,,$$

$$(101 \quad 100 \quad 011 \quad 010 \quad 001 \quad 000)\,.$$

Křížení provedeme tak, že náhodně vybereme bod, kde jsou oba řetězce rozděleny do dvou částí, například takto:

$$(000 \quad 001 \quad 010 \mid 011 \quad 100 \quad 101)\,,$$

$$(101 \quad 100 \quad 011 \mid 010 \quad 001 \quad 000)\,.$$

Rekombinací těchto částí vzniknou řetězce

$$(000 \quad 001 \quad 010 \quad 010 \quad 001 \quad 000)\,,$$

$$(101 \quad 100 \quad 011 \quad 011 \quad 100 \quad 101)\,,$$

které nereprezentují platné cykly, neboť neprojdou všemi městy.

3.1.2 Sousedská reprezentace

V **sousedské reprezentaci** (*adjacency representation*) je cesta vyjádřena jako řetězec N měst tak, že město j je v seznamu na pozici i právě tehdy, když cesta vede z města i do města j. Například vektor

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

reprezentuje cestu

$$1 - 3 - 7 - 2 - 5 - 4 - 6 - 8.$$

Každá cesta má v sousedské reprezentaci unikátní vyjádření, ovšem každý řetězec nemusí reprezentovat cestu. Například

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

vyjadřuje následující množinu cyklů:

$$1-3-7,\ 2-5,\ 4-6 \text{ a } 8.$$

Snadno nahlédneme, že pro sousedskou reprezentaci opět rekombinací nevzniknou legální cesty a musí se použít opravné algoritmy. Byly ovšem navrženy a zkoumány operátory křížení přímo pro tuto reprezentaci. My se o nich nebudeme podrobně rozepisovat, neboť nebudeme tuto reprezentaci v naší práci používat.

3.1.3 Ordinální reprezentace

Ordinální reprezentace (*ordinal representation*) pochází z jednoho z historicky nejstarších pokusů o řešení TSP metodami GA. Cesta N městy je reprezentována řetězcem délky N, přičemž i-té číslo v seznamu může nabývat hodnoty v rozsahu od 1 do $n - i + 1$. Interpretace takového seznamu je pak dána tím, že máme referenční seznam (např. pro $N = 8$)

$$L = (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8)\,.$$

Potom cesta

$$1-5-3-2-8-4-7-6$$

bude reprezentována chromozomem

$$T = (1 \quad 4 \quad 2 \quad 1 \quad 4 \quad 1 \quad 2 \quad 1),$$

který je dekódován následovně: První číslo v T je 1. To znamená, že abychom dostali první město cesty, vezmeme první prvek ze seznamu L a v L ho vyškrtneme. Částečná cesta je: 1. Druhý prvek T je číslo 4. Proto jako druhé město cesty vezmeme čtvrtý prvek v seznamu L a tím je po vyškrtnutí města 1 město 5. Město 5 vyškrtneme ze seznamu L. Částečná cesta je $1-5$. Tímto způsobem pokračujeme, dokud nejsou vyškrtány všechny prvky v seznamu L a tím nalezneme cestu $1-5-3-2-8-4-7-6$.

Výhodou ordinální reprezentace je to, že použitím klasického jednobodového křížení dvou přípustných rodičů získáme vždy potomky reprezentující přípustné

řešení. Bohužel pouhou kombinací ordinální reprezentace a jednobodového křížení nelze pří řešení TSP dosáhnout dobrých výsledků. Proto byla i tato reprezentace pro TSP zavržena. Nicméně experimenty prokázaly ([1]), že je možné ordinální reprezentaci s úspěchem použít například pro plánovací úlohy z oblasti tvorby rozvrhů.

3.1.4 Přirozená reprezentace

Přirozená reprezentace nebo také **permutační kódování** (*path representation*) je jednou z nejpřirozenějších reprezentací cesty. Cesta je opět reprezentována seznamem N měst. Pokud je město i v řetězci na pozici j, znamená to, že město i bude právě j-té navštívené. Takže například cesta

$$3 - 2 - 4 - 1 - 7 - 5 - 8 - 6$$

je reprezentována takto:

$$(3 \quad 2 \quad 4 \quad 1 \quad 7 \quad 5 \quad 8 \quad 6)\,.$$

Použití klasických operátorů pro řešení TSP při přirozené reprezentaci opět není vhodné. Byly proto pro tuto reprezentaci navrženy jiné operátory křížení a mutace. Přesto se tento způsob reprezentace ukázal být nejúspěšnější ([1]). My budeme tuto reprezentaci používat v praktické části, proto se dále budeme zabývat rekombinačními operátory pouze pro tuto reprezentaci.

3.2 Ohodnocení

Každý jedinec je nějakým způsobem ohodnocen; je určena jeho kvalita (fitness). Pro problém obchodního cestujícího je tato kvalita určena cenou cesty. V našem případě je tato cena určena délkou cesty (eukleidovskou vzdáleností). Čím kratší cestu jedinec reprezentuje, tím lepší má ohodnocení.

3.3 Selekce

Pomocí selekce vybíráme z osídlení jedince, kteří nám poskytnou základ pro vytvoření následující generace. Chceme, aby průměrná kvalita populace s rostou-

cím počtem generací stoupala. Napodobujeme proces přirozeného výběru, kde je kvalitnější jedinec upřednostňován před slabšími. Schopnost selekčního mechanismu nadprůměrné jedince zvýrazňovat a podprůměrné potlačovat se nazývá *selekční tlak*.

Selekční algoritmus má velký vliv na výkon genetického algoritmu. Pokud by byl selekční tlak příliš silný, tj. selekční mechanismus by až příliš zvýhodňoval kvalitní jedince, mohlo by se stát, že nezůstane zachována dostatečná rozmanitost populace, což může mít za následek předčasnou konvergenci do lokálního extrému. Na druhou stranu pokud by výběrové kriterium bylo příliš volné, pracoval by algoritmus velmi pomalu a po mnoho generací bychom nemuseli zaznamenat jakýkoli pokrok. My si v této práci uvedeme metodu ruletového kola, metodu turnajového výběru a metodu pravděpodobnostního turnaje ([1], [2], [3]).

Jelikož selekce (a případně i rekombinace) může z populace eliminovat dosud nejlepšího jedince, zavádí se pomocná proměnná nazývaná BSF (*best-so-far*), v níž se uchovává dosud nejlepší nalezené řešení. Ztrátě nejlepšího jedince lze předejít i tím, že ho automaticky uložíme jako prvek následující generace.

3.3.1 Metoda ruletového kola

Metoda ruletového kola jistým způsobem upřednostňuje kvalitnější individua před méně kvalitními. Kvalitnějším jedincům je přiřazena větší část ruletového kola a je tedy větší pravděpodobnost, že budou vybráni. Způsobů, jak zkonstruovat „upřednostňující" ruletové kolo, je několik. My v této práci popíšeme dvě metody - metodu, kde pravděpodobnost výběru je přímo úměrná hodnotě fitness, a metodu pořadové selekce. Další známou metodou je např. postup zvaný zbytkový stochastický výběr (*remainder stochastic sampling*).

Selekce přímo úměrná hodnotě fitness

V tomto případě získá každý jedinec tak velkou část na obvodu ruletového kola, jejíž délka je přímo úměrná ohodnocení tohoto jedince. Čím má jedinec lepší ohodnocení (fitness), tím delší část obvodu kruhu získá. Pravděpodobnost výběru každého jedince je tedy přímo úměrná jeho kvalitě (*fitness-proportionate*

selection). Jedinci s nejlepším ohodnocením mají největší šanci dostat se do další generace a šířit svůj genetický materiál. Potom se kolo „roztočí" a vybere se ten jedinec, na kterém se kolo zastaví.

Matematicky můžeme tuto metodu, kde bude vybrán i-tý jedinec s pravděpodobností p_i zapsat následovně:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}, \quad i = 1, \ldots, N,$$

kde f_i je hodnota fitness i-tého jedince.

Výše uvedená metoda ruletového kola je platná pro úlohu hledání maxima. Pojďme se podívat, jak se zkonstruuje ruletové kolo v případě, kdy se hledá minimum. V tom případě pravděpodobnost, že jedinec bude vybrán, závisí na hodnotě fitness nepřímo úměrně. V naší úloze obchodního cestujícího hledáme minimum a hodnota fitness je určena délkou cesty, kterou jedinec reprezentuje.

Postupujeme tak, že nejprve normalizujeme hodnoty fitness. Určíme minimum f_{min} a maximum f_{max} ze všech hodnot fitness a spočítáme hodnotu $\tilde{f}$ každého jedince jako

$$\tilde{f}_i = \frac{f_{max} - f_i}{f_{max} - f_{min}}, \quad i = 1, \ldots, N,$$

kde f_i je hodnota fitness i-tého jedince. Poté pravděpodobnost, že i-tý jedinec bude vybrán, je

$$p_i = \frac{\tilde{f}_i}{\sum_{j=1}^{N} \tilde{f}_j}, \quad i = 1, \ldots, N.$$

Pořadová selekce

Výše uvedená metoda selekce přímo úměrná hodnotě fitness může mít několik nevýhod. Pokud by v populaci byl jeden nebo více vysoce nadprůměrných jedinců, je zřejmé, že tímto postupem by tito jedinci byli hodně zvýhodněni. Měli by příliš velký vliv, přičemž vliv podprůměrných jedinců by byl hodně malý. Nebo pokud bychom například hledali maximum nezáporné funkce, jejíž funkční hodnoty by se pohybovaly ve velmi malém intervalu, selekční tlak by naopak nebyl téměř žádný,

neboť všichni jedinci by měli skoro stejné šance být vybráni. Z toho důvodu se provádí úprava selekčního tlaku. Jednou z metod na úpravu selekčního tlaku je *pořadová selekce (rank selection).*

Postup je takový, že se nejprve jedinci v populaci uspořádají sestupně podle své kvality a podle tohoto pořadí se určí nové ohodnocení každého jedince. Nejlepší jedinec dostane transformované ohodnocení N, druhý nejlepší $N-1$, atd. až nejhorší pak ohodnocení jedna. Případně se může nejlepšímu jedinci přiřadit hodnota $N-1$, atd. až nejhoršímu jedinci připadne ohodnocení nula, čímž se nejhorší jedinec v populaci úplně zlikviduje. Vynásobením takto transformovaných ohodnocení takovou konstantou, aby součet pravděpodobností byl roven jedné, dostaneme pravděpodobnosti selekce. Celkově dostaneme pravděpodobnost p_i vybrání i-tého jedince v množině sestupně uspořádaných jedinců takto:

$$p_i = \frac{2(N+1-i)}{N(N+1)}, \quad i = 1, \ldots, N,$$

případně pro alternativu s potlačením nejhoršího

$$p_i = \frac{2(N-i)}{N(N-1)}, \quad i = 1, \ldots, N.$$

Po této úpravě pravděpodobností se „zatočí" ruletovým kolem.

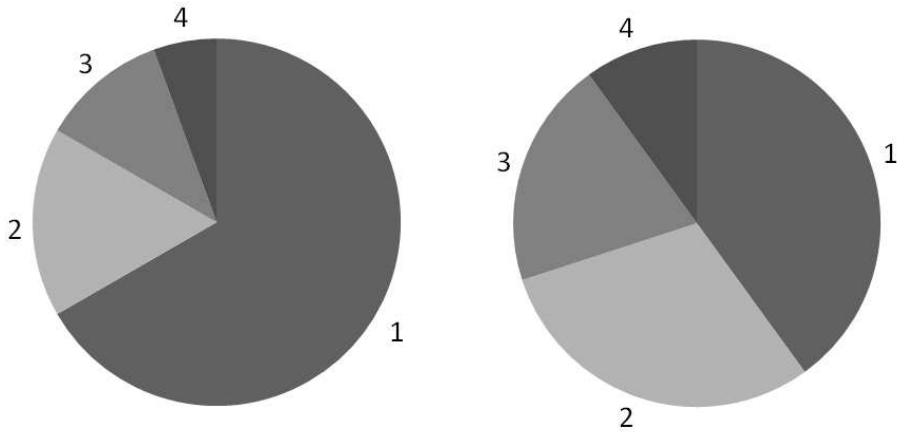
Příklad 3.1. Pro znázornění rozdílu obou výše uvedených metod uveďme příklad se čtyřmi jedinci, kde je při hledání maxima nezáporné reálné funkce hodnota fitness f_i i-tého jedince určena funkční hodnotou:

Jedinec i	f_i	p_i	Pravd. v %
1	12	$0,667$	$66,7\%$
2	3	$0,167$	$16,7\%$
3	2	$0,111$	$11,1\%$
4	1	$0,055$	$5,5\%$

Jedinec i	f_i	Ohodnocení dle pořadí	p_i	Pravd. v %.
1	12	4	0,4	40%
2	3	3	0,3	30%
3	2	2	0,2	20%
4	1	1	0,1	10%

Rozložení pravděpodobnosti, s jakou bude individuum vybráno

a) podle velikosti ohodnocení b) podle pořadí



Obrázek 2: Potlačení vlivu nadprůměrných individuí výběrem podle pořadí.

Je tedy zřejmé, že pořadová selekce potlačuje přílišný vliv nadprůměrných jedinců.

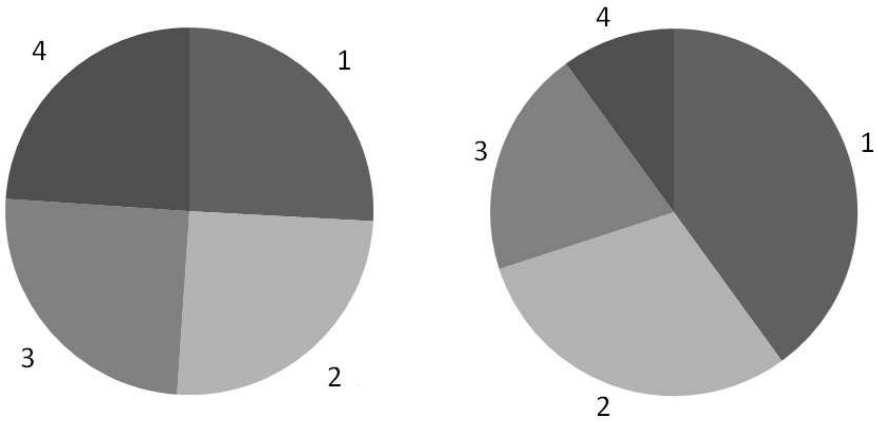
Příklad 3.2. Uvažujme maximalizaci nezáporné funkce. Mějme opět čtyři jedince, jejichž hodnota fitness je dána funkční hodnotou:

Jedinec i	f_i	p_i	Pravd. v %
1	12	$0,259$	$25,9\%$
2	$11,7$	$0,252$	$25,2\%$
3	$11,6$	$0,250$	25%
4	$11,1$	$0,239$	$23,9\%$

Jedinec i	f_i	Ohodnocení dle pořadí	p_i	Pravd. v %.
1	12	4	0,4	40%
2	$11,7$	3	0,3	30%
3	$11,6$	2	0,2	20%
4	$11,1$	1	0,1	10%

Rozložení pravděpodobnosti, s jakou bude individuum vybráno

a) podle velikosti ohodnocení b) podle pořadí



Obrázek 3: Zvýraznění vlivu lépe ohodnocených individuí výběrem podle pořadí.

Na uvedeném příkladu vidíme, že pořadovou selekcí byl zvýšen selekční tlak.

Realizace selekce metodou ruletového kola

Máme-li některou z uvedených metod zkonstruované ruletové kolo, realizujeme výběr tak, že nejprve určíme čísla

$$\overline{f_i} = \sum_{j=1}^{i} p_j, \quad i = 1, \ldots, N.$$

Následně vygenerujeme s rovnoměrným rozdělením náhodné číslo $r \in \langle 0, 1\rangle$ a i-tý jedinec je vybrán, právě když

$$\overline{f_{i-1}} < r \leq \overline{f_i}, \quad i = 1, \ldots, N.$$

3.3.2 Turnajový výběr

Metoda turnajového výběru byla inspirována přírodními procesy, kde jedinci jistých živočišných druhů často musí vzájemně fyzicky bojovat o právo účastnit se reprodukčního procesu. Tato metoda je podle ([1]) nejpoužívanější selekční metodou v reálných aplikacích GA.

Při turnajovém výběru se z populace náhodně vyberou skupinky jedinců (převážně bývají dvojčlenné), z kterých do křížení postupuje jeden nebo dva jedinci s nejlepší hodnotou fitness. V případě dvojčlenné skupinky postupuje pouze jeden - ten lepší. Postup se opakuje, dokud populace nemá potřebných N jedinců. Tím je splněna podmínka, že lepší jedinci mají větší šanci na úspěch, ale zároveň se tím zachovává rozmanitost populace, protože můžeme vybrat dva jedince, kteří představují nekvalitní řešení, ale i tak jeden z nich podstoupí křížení a tím se jeho geny dostanou do další generace. Což nemusí být na škodu, protože po křížení s jiným jedincem může vzniknout nový jedinec, který bude ještě lepší. Zároveň se tím předchází situaci, že by populace uvízla v nějakém lokálním extrému.

3.3.3 Pravděpodobnostní turnaj

Pokud zkombinujeme předešlé dvě metody, získáme metodu pravděpodobnostního turnaje. V tomto případě se pomocí ruletového kola vyberou dva jedinci, z nichž potom do křížení postoupí ten kvalitnější.

Tato výběrová strategie výrazně zvýhodňuje kvalitní jedince a méně kvalitní jedince silně potlačuje.

3.4 Rekombinace

Jak již bylo řečeno, budeme se dále zabývat pouze rekombinačními operátory, které byly navrženy speciálně pro přirozenou reprezentaci cesty (*path representation*). Detailně popíšeme ty, se kterými bylo podle literatury ([1], [4], [9]) dosaženo nejlepších výsledků a které budeme sami programovat v praktické části. A to jsou:

Operátory křížení

- Operátor křížení s částečným zobrazením (*Partially-mapped crossover* - PMX)
- Cyklický operátor křížení (*Cycle crossover operator* - CX)
- Operátor křížení se zachováním pořadí (*Order crossover operator* - OX)

- Operátor křížení s rekombinací hran (*Edge recombination crossover* - ERX)

- Operátor křížení se střídáním pozic (*Alternating-position crossover* - AP)

Operátory mutace

- Operátor mutace založený na posunutí (*Displacement mutation* - DM)

- Operátor mutace založený na výměně (*Exchange mutation* - EM)

- Operátor mutace založený na vložení (*Insertion mutation* - ISM)

- Operátor jednoduché inverzní mutace (*Simple inversion mutation* - SIM)

- Operátor zkracující inverzní mutace (ZIM)

- Operátor inverzní mutace (*Inversion mutation* - IVM)

- Operátor mutace založený na promíchání (*Scramble mutation* - SM)

3.4.1 Křížení

Operátor křížení s částečným zobrazením - PMX

Operátor křížení s částečným zobrazením byl navržen Goldbergem a Linglem (1985). Tento operátor zachovává úseky měst a částečně i pořadí měst, což je při řešení TSP výhodou. Operátor pracuje tak, že vybere dva křížící body a rozdělí tak oba rodiče na tři úseky. Potomci jsou pak konstruováni tak, že se okopírují prostřední úseky řetězců a zbývající pozice se postupně doplňují. Ukažme to na příkladu - mějme dva rodiče A a B, vytvoříme potomky A' a B'. Symbol * bude značit dosud neznámý prvek.

$$A: \quad (1 \quad 2 \quad 3 \mid 4 \quad 5 \quad 6 \quad 7 \mid 8 \quad 9 \quad 10)$$

$$B: \quad (1 \quad 9 \quad 2 \mid 4 \quad 6 \quad 3 \quad 10 \mid 7 \quad 5 \quad 8)$$

$$\Downarrow$$

$$A': \quad (* \quad * \quad * \mid 4 \quad 5 \quad 6 \quad 7 \mid * \quad * \quad *)$$

$$B': \quad (* \quad * \quad * \mid 4 \quad 6 \quad 3 \quad 10 \mid * \quad * \quad *)$$

Prostřední části řetězců implikují částečná zobrazení

$$4 \leftrightarrow 4, 5 \leftrightarrow 6, 6 \leftrightarrow 3, 7 \leftrightarrow 10,$$

které bude využito v dalším postupu. Následně se z rodičů doplní další přípustná města, tedy ta města, která nezpůsobí opakování v řetězci. Doplněna jsou z toho z rodičů, ze kterého nepochází střední část

$$A: \quad (1 \quad 2 \quad 3 \mid 4 \quad 5 \quad 6 \quad 7 \mid 8 \quad 9 \quad 10)$$

$$B: \quad (1 \quad 9 \quad 2 \mid 4 \quad 6 \quad 3 \quad 10 \mid 7 \quad 5 \quad 8)$$

$$\Downarrow$$

$$A': \quad (1 \quad 9 \quad 2 \mid 4 \quad 5 \quad 6 \quad 7 \mid * \quad * \quad 8)$$

$$B': \quad (1 \quad 2 \quad * \mid 4 \quad 6 \quad 3 \quad 10 \mid 8 \quad 9 \quad *)$$

K doplnění chybějících symbolů se pak využije částečných zobrazení, a pokud zbudou neznámé symboly, doplníme je náhodně tak, aby byl splněn požadavek bezkonfliktnosti

$$A': \quad (1 \quad 9 \quad 2 \mid 4 \quad 5 \quad 6 \quad 7 \mid 7 \leftrightarrow 10 \quad * \quad 8)$$

$$B': \quad (1 \quad 2 \quad * \mid 4 \quad 6 \quad 3 \quad 10 \mid 8 \quad 9 \quad 7 \leftrightarrow 10)$$

$$\Downarrow$$

$$A': \quad (1 \quad 9 \quad 2 \mid 4 \quad 5 \quad 6 \quad 7 \mid 10 \quad 3 \quad 8)$$

$$B': \quad (1 \quad 2 \quad 5 \mid 4 \quad 6 \quad 3 \quad 10 \mid 8 \quad 9 \quad 7)$$

Cyklický operátor křížení - CX

Cyklický operátor křížení byl navržen Oliverem (1987). Tento operátor zachovává asi polovinu absolutních pozic měst v chromozomu z každého rodiče. Oliver z teoretických a praktických výsledků vyvodil ([4]), že operátor CX dává pro TSP lepší výsledky než operátor PMX. Potomek je tvořen tak, že každá pozice je

obsazena prvkem, který se v prvním nebo druhém rodiči vyskytuje na stejné pozici. Mějme například rodiče

$$A: \quad (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10),$$

$$B: \quad (3 \quad 6 \quad 1 \quad 10 \quad 8 \quad 4 \quad 9 \quad 7 \quad 2 \quad 5).$$

Rodič, ze kterého je vybrán první prvek a zkopírován do potomka, je vybrán náhodně. Pozice, ze které se prvek zkopíruje do potomka, je též vybrána náhodně. Nechť je například vybrán rodič A a nechť volba padla na prvek, který je na čtvrté pozici. V našem příkladě je to prvek 4:

$$C: \quad (* \quad * \quad * \quad 4 \quad * \quad * \quad * \quad * \quad * \quad *).$$

Prvek, který jsme nyní zkopírovali z vybraného rodiče, už nemůže být zkopírován na tu pozici, na které se nachází u druhého (nevybraného) rodiče. Další prvek potomka tedy určíme tak, že na tu pozici, na které má nevybraný rodič (v našem případě B) už použitý prvek, tedy 4, zkopírujeme prvek opět z vybraného rodiče, tedy z rodiče A. Prvek 4 se v rodiči B nachází na šesté pozici, tudíž šestý prvek v potomku je obsazen stejným číslem jako v rodiči A:

$$C: \quad (* \quad * \quad * \quad 4 \quad * \quad 6 \quad * \quad * \quad * \quad *).$$

Stejným způsobem určíme, že druhá pozice potomka bude obsazena stejným prvkem, jaký je na druhé pozici ve vybraném rodiči - v rodiči A:

$$C: \quad (* \quad 2 \quad * \quad 4 \quad * \quad 6 \quad * \quad * \quad * \quad *).$$

Analogicky i devátý, sedmý, osmý, pátý a desátý prvek musí být do potomka zkopírován z rodiče A:

$$C: \quad (* \quad 2 \quad * \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10).$$

Pozice prvků, které byly doposud vybrány, se nazývají cyklus. Všechny zbylé prvky jsou nyní doplněny z rodiče B. Tím nalezneme potomka

$$C: \quad (3 \quad 2 \quad 1 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10).$$

Chceme-li z tohoto křížení mít dva potomky, vybereme první prvek do prvního potomka z rodiče A a první prvek do druhého potomka vybereme z rodiče B.

Operátor křížení se zachováním pořadí - OX

Operátor křížení se zachováním pořadí byl navržen Davisem (1985). Tento operátor zachovává relativní pořadí měst z rodičů. Operátor pracuje tak, že vybere dva křížící body a rozdělí tak oba rodiče na tři úseky. Potomci jsou konstruováni tak, že se okopírují prostřední úseky řetězců a zbývající pozice jsou pak postupně doplněny tak, že od druhého řezu kopírujeme města z druhého rodiče, přičemž vynecháme ta města, která již na cestě existují. Po dosažení posledního prvku řetězce pokračujeme od prvního. Mějme například rodiče

$$A: \quad (1 \quad 2 \mid 3 \quad 4 \quad 5 \mid 6 \quad 7 \quad 8)$$

$$B: \quad (2 \quad 4 \mid 6 \quad 8 \quad 7 \mid 5 \quad 3 \quad 1)$$

$$\Downarrow$$

$$A': \quad (* \quad * \mid 3 \quad 4 \quad 5 \mid * \quad * \quad *)$$

$$B': \quad (* \quad * \mid 6 \quad 8 \quad 7 \mid * \quad * \quad *)$$

Nyní začneme od druhého řezu kopírovat města z druhého z rodičů. Do potomka A' bychom zkopírovali z rodiče B prvek 5, ale jelikož ten už je v prostřední části, přejdeme na další prvek a tím je 3. Ta už je ale v potomku A' také obsažena. Zapíšeme tedy až další prvek, kterým je 1. Tím jsme došli na konec rodiče B. Začneme tedy kopírovat od prvního prvku rodiče B a do potomka A' zapíšeme 2. Další prvek v rodiči B, kterým je 4, je už v potomku A' obsažen. Postoupíme tedy na další prvek a tím je 6. Zapíšeme ji do potomka A'. Tím jsme došli na konec potomka A' a další prvky budeme do potomka zapisovat od začátku. Na první místo tedy zkopírujeme z rodiče B prvek 8, dále prvek 7. Získáme potomky

$$A': \quad (8 \quad 7 \mid 3 \quad 4 \quad 5 \mid 1 \quad 2 \quad 6)$$

$$B': \quad (4 \quad 5 \mid 6 \quad 8 \quad 7 \mid 1 \quad 2 \quad 3)$$

Operátor křížení s rekombinací hran - ERX

Operátor křížení s rekombinací hran byl navržen Whitleym (1989, 1991). Tento operátor je vhodný pro symetrickou úlohu obchodního cestujícího. Předpokládá, že jsou důležité pouze hodnoty hran, ne jejich směr. V souladu s tímto předpokladem se můžeme na hrany dívat jako na přenašeče dědičné informace. ERX se pokouší zachovat hrany rodičů, aby maximum informací přešlo do potomka. Zlomení hrany je chápáno jako nechtěná mutace. Podle [1] je to nejúspěšnější známý operátor křížení pro TSP. V kapitole 5 uvidíme, zda můžeme toto tvrzení na základě vlastních testů potvrdit nebo ne.

Tento operátor je zvláštní tím, že ze dvou rodičů vytvoří jen jednoho potomka. Aby byl tedy kompatibilní s naším naprogramovaným algoritmem, který počítá s tím, že zkřížením dvou rodičů dostaneme dva potomky, udělali jsme tu úpravu, že jsme ze dvou stejných rodičů vytvořili dva potomky.

Každý úsek v řetězci pochází z některého z rodičů. K tomu se používá tzv. *hranová tabulka*. Ta každému městu přiřadí seznam sousedů, kteří se vyskytují v obou rodičích.

Postupujeme tak, že z měst, které mají minimální počet sousedů, náhodně vybereme jedno město a vyškrtneme ho ze všech seznamů sousedů v hranové tabulce. Důvod, proč vybíráme mezi těmi městy, která mají minimální počet sousedů, je, abychom předešli možné izolaci města, která je nežádoucí a mohla by nastat v dalším postupu. Dále pak vybíráme mezi sousedy naposledy vybraného města a opět náhodně volíme mezi těmi z nich, která mají nejméně sousedů. Mějme například rodiče

$$A: \quad (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10),$$

$$B: \quad (1 \quad 9 \quad 2 \quad 4 \quad 6 \quad 3 \quad 10 \quad 7 \quad 5 \quad 8).$$

Hranová tabulka:

Město	Sousedé	Město	Sousedé
1	2, 8, 9, 10	6	3, 4, 5, 7
2	1, 3, 4, 9	7	5, 6, 8, 10
3	2, 4, 6, 10	8	1, 5, 7, 9
4	2, 3, 5, 6	9	1, 2, 8, 10
5	4, 6, 7, 8	10	1, 3, 7, 9

V našem příkladě mají všechna města 4 sousedy, proto na začátku můžeme vybírat ze všech deseti. Předpokládejme například, že bylo náhodně vybráno město 1:

$$C: \quad (1 \quad * \quad * \quad * \quad * \quad * \quad * \quad * \quad * \quad *)$$

Hranová tabulka:

Město	Sousedé	Město	Sousedé
2	3, 4, 9	7	5, 6, 8, 10
3	2, 4, 6, 10	8	5, 7, 9
4	2, 3, 5, 6	9	2, 8, 10
5	4, 6, 7, 8	10	3, 7, 9
6	3, 4, 5, 7		

Nyní mezi sousedy města 1 náhodně zvolíme další město, přičemž opět upřednostňujeme města s minimálním počtem sousedů. Nechť volba padne na 8:

$$C: \quad (1 \quad 8 \quad * \quad * \quad * \quad * \quad * \quad * \quad * \quad *)$$

Hranová tabulka:

Město	Sousedé	Město	Sousedé
2	3, 4, 9	6	3, 4, 5, 7
3	2, 4, 6, 10	7	5, 6, 10
4	2, 3, 5, 6	9	2, 10
5	4, 6, 7	10	3, 7, 9

Dále mezi sousedy města 8 zvolíme město 9, které má pouze dva sousedy:

$$C: \quad (1 \quad 8 \quad 9 \quad * \quad * \quad * \quad * \quad * \quad * \quad *)$$

Hranová tabulka:

Město	Sousedé	Město	Sousedé
2	3, 4	6	3, 4, 5, 7
3	2, 4, 6, 10	7	5, 6, 10
4	2, 3, 5, 6	10	3, 7
5	4, 6, 7		

Nechť je dále ze sousedů města 9, kterými jsou města 2 a 10, a která mají obě dva sousedy, náhodně vybráno například město 10:

$$C: \quad (1 \quad 8 \quad 9 \quad 10 \quad * \quad * \quad * \quad * \quad * \quad *)$$

Hranová tabulka:

Město	Sousedé	Město	Sousedé
2	3, 4	5	4, 6, 7
3	2, 4, 6,	6	3, 4, 5, 7
4	2, 3, 5, 6	7	5, 6

Sousedem města 10 s minimálním počtem sousedů je město 7:

$$C: \quad (1 \quad 8 \quad 9 \quad 10 \quad 7 \quad * \quad * \quad * \quad * \quad *)$$

Hranová tabulka:

Město	Sousedé	Město	Sousedé
2	3, 4	5	4, 6
3	2, 4, 6	6	3, 4, 5
4	2, 3, 5, 6		

Dalším vybraným městem bude město 5, neboť ze všech sousedů města 7 má město 5 nejméně sousedů. Dostáváme:

$$C: \quad (1 \quad 8 \quad 9 \quad 10 \quad 7 \quad 5 \quad * \quad * \quad * \quad *)$$

Hranová tabulka:

Město	Sousedé	Město	Sousedé
2	3, 4	4	2, 3, 6
3	2, 4, 6	6	3, 4

Dále bude vybráno město 6:

$$C: \quad (1 \quad 8 \quad 9 \quad 10 \quad 7 \quad 5 \quad 6 \quad * \quad * \quad *)$$

Hranová tabulka:

Město	Sousedé	Město	Sousedé
2	3, 4	4	2, 3
3	2, 4		

Náhodným zvolením města 3 mezi městy 3 a 4 dostáváme:

$$C: \quad (1 \quad 8 \quad 9 \quad 10 \quad 7 \quad 5 \quad 6 \quad 3 \quad * \quad *)$$

Hranová tabulka:

Město	Sousedé	Město	Sousedé
2	4	4	2

Nechť je nyní ze zbývajících náhodně zvoleno město 4:

$$C: \quad (1 \quad 8 \quad 9 \quad 10 \quad 7 \quad 5 \quad 6 \quad 3 \quad 4 \quad *)$$

Na poslední pozici pak zbývá doplnit město 2:

$$C: \quad (1 \quad 8 \quad 9 \quad 10 \quad 7 \quad 5 \quad 6 \quad 3 \quad 4 \quad 2)\,.$$

Operátor křížení se střídáním pozic - AP

Tento operátor vytvoří potomka tak, že od začátku vybírá prvky střídavě z jednoho rodiče a další prvek z druhého rodiče, přičemž vynechává prvky, které už potomek obsahuje. Když u jednoho rodiče narazí na prvek, který už byl použit, nezapíše nic a další prvek vybírá z druhého rodiče. Mějme například rodiče

$$A: \quad (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8)$$

$$B: \quad (3 \quad 7 \quad 5 \quad 1 \quad 6 \quad 8 \quad 2 \quad 4)$$

Potomka A' podle výše popsaného postupu tedy konstruujeme tak, že začneme kopírovat střídavě první prvek z rodiče A, kterým je 1, a první prvek z rodiče

B, kterým je 3. Dále druhý prvek z rodiče A, což je 2, druhý prvek z rodiče B, kterým je 7. Další by byl třetí prvek rodiče A, tedy 3, ale jelikož prvek 3 už je v potomku A' obsažen, vynecháme ho a zapíšeme třetí prvek z rodiče B, kterým je 5. Pokračováním tohoto postupu dostaneme potomky:

$$A': \quad (1 \quad 3 \quad 2 \quad 7 \quad 5 \quad 4 \quad 6 \quad 8)$$

$$B': \quad (3 \quad 1 \quad 7 \quad 2 \quad 5 \quad 4 \quad 6 \quad 8)$$

3.4.2 Mutace

Operátor mutace založený na posunutí - DM

Tento operátor pracuje tak, že vybere nějaký náhodný úsek cesty a tento úsek přesune na jiné náhodně vybrané místo. Uvažujme například cestu reprezentovanou jedincem

$$A: \quad (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8)$$

a předpokládejme, že byl vybrán úsek (3 4 5) a náhodně vybráno město 7, za které má být tento úsek vložen. Výsledný jedinec:

$$A^*: \quad (1 \quad 2 \quad 6 \quad 7 \quad 3 \quad 4 \quad 5 \quad 8)$$

Operátor mutace založený na výměně - EM

Tento operátor náhodně vybere dvě města a vymění je. Mějme například cestu

$$A: \quad (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8)$$

a předpokládejme, že bylo náhodně vybráno třetí a páté město. Výsledný jedinec:

$$A^*: \quad (1 \quad 2 \quad 5 \quad 4 \quad 3 \quad 6 \quad 7 \quad 8)$$

Operátor mutace založený na vložení - ISM

Tento operátor vybere náhodně nějaké město z cesty a vloží ho na jiné náhodně vybrané místo. Uvažujme opět cestu

$$A: \quad (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8)$$

a nechť je vyjmuto město 4 a vloženo za město 7. Výsledný jedinec:

$$A^* : \qquad (1 \quad 2 \quad 3 \quad 5 \quad 6 \quad 7 \quad 4 \quad 8)$$

Operátor jednoduché inverzní mutace - SIM

Tento operátor náhodně vybere dvě pozice, kde řetězec rozdělí a invertuje pořadí měst mezi dělícími body. Například cestu

$$A : \qquad (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8)$$

rozdělíme mezi městy 2 a 3 a mezi městy 5 a 6:

$$A : \qquad (1 \quad 2 \mid 3 \quad 4 \quad 5 \mid 6 \quad 7 \quad 8)\,.$$

Výsledek:

$$A^* : \qquad (1 \quad 2 \quad 5 \quad 4 \quad 3 \quad 6 \quad 7 \quad 8)$$

Operátor zkracující inverzní mutace - ZIM

Tento operátor je variací předchozího operátoru jednoduché inverzní mutace. Od mutace SIM se liší v tom, že pořadí měst není invertováno v náhodně vybraném úseku, ale v takovém úseku, aby inverzí došlo ke zkrácení výsledné cesty. Jinými slovy dojde k výměně dvou uzlů a inverzi měst mezi nimi, což je 1 výměna z 2-optimalizace.

Operátor zkracující inverzní mutace je náš vlastní návrh. Vznikl na základě inspirace článkem [15], který se zabývá rychlostí konvergence při použití 2-opt a 3-opt zlepšovacího algoritmu jako mutace.

Operátor inverzní mutace - IVM

Tento operátor vyjme z řetězce náhodný úsek, invertuje pořadí měst v tomto úseku a vloží ho na jiné náhodně vybrané místo. Z cesty

$$A : \qquad (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8)$$

se tak vybráním úseku (3 4 5) a vložením po inverzi za město 7 stane cesta

$$A^* : \qquad (1 \quad 2 \quad 6 \quad 7 \quad 5 \quad 4 \quad 3 \quad 8)$$

Operátor mutace založený na promíchání - SM

Tento operátor vyjme z řetězce náhodný úsek a náhodně promíchá města v tomto úseku. Mějme například cestu

$$A: \qquad (1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8)$$

a předpokládejme, že byl vybrán úsek (4 5 6 7), který byl náhodně promíchán na (5 6 7 4). Výsledný jedinec pak vypadá takto:

$$A^*: \qquad (1 \quad 2 \quad 3 \quad 5 \quad 6 \quad 7 \quad 4 \quad 8)$$

3.5 Náhradové strategie

Náhradová strategie má za úkol vybrat ze smíšené množiny rodičů, potomků a mutantů novou populaci.

Uvažujme například populaci o velikosti $N = 100$. Nechť pravděpodobnost křížení $P_x = 0,9$ a pravděpodobnost mutace $P_m = 0,2$. Po selekci 100 prvků pak přibližně 90 z nich (45 párů) podstoupí křížení a vygenerují 90 potomků. Mutací vznikne přibližně 20 mutantů. Nebo pokud bychom mutovali původní prvky i potomky vzniklé křížením, může být mutantů 38. Máme tedy 100 původních jedinců, 90 potomků a 20 (nebo 38) mutantů. Dohromady tito jedinci tvoří rozšířenou množinu velikosti $N' = 210$ (nebo dokonce 228) prvků[2]. Náhradová strategie má za úkol vybrat z této množiny novou populaci o velikosti $N = 100$.

Náhradových strategií existuje celá řada. My v této práci popíšeme ty nejdůležitější, budeme vycházet z [1].

Strategie zvaná *elitismus* pracuje tak, že ze všech jedinců vybere jednoho nebo několik nejlepších (nejkvalitnějších) a zbytek do N se vhodně doplní - zpravidla upřednostněním potomků a mutantů. Vybrání nejlepších N jedinců z celé rozšířené množiny je pak extrémním případem elitismu. Experimentálně bylo zjištěno ([1]), že takovýto extrém způsobuje předčasnou konvergenci a homogenizaci populace.

[2]Všechna uvedená čísla jsou pouze přibližná a vycházejí z předpokladu platnosti zákona velkých čísel.

Dalším extrémním případem náhradové strategie je plně *náhodný výběr*. V tomto případě je potřebných N prvků z rozšířené množiny vybráno zcela náhodně bez ohledu na kvalitu jedinců. Modifikací tohoto postupu je pak *prázdná* strategie, kdy je náhrada spojena se selekcí v následujícím generačním cyklu. Selekce ruletovým kolem se pak provede tak, že ruleta je zkonstruována pro všech N' jedinců, ale ruletovým kolem se točí pouze N-krát. Při turnajové selekci se uspořádá N soutěží s jedním vítězem, avšak soutěžící se vybírají z celkové množiny N'.

3.6 Ukončovací kriteria

K zastavení algoritmu mohou být použita různá kriteria. Uvedeme si několik nejčastěji používaných ([13], [14]).

Počet generací

Toto kriterium zastaví výpočet po provedení maximálního počtu iterací (zadaných uživatelem). Toto ukončovací kriterium budeme používat v praktické části.

Výpočetní čas

Toto kriterium zastaví běh algoritmu v okamžiku, kdy je dosažena maximální (uživatelem zadaná) doba trvání výpočtu. Algoritmus lze ukončit na konci cyklu nebo i během výpočtu aktuální generace.

Dosažení požadované hodnoty fitness

Při použití tohoto kriteria je výpočet zastaven, pokud hodnota fitness nejlepšího jedince v aktuální populaci je menší než uživatelem zadaná hodnota (v případě minimalizace) nebo větší než zadaná hodnota (v případě maximalizace). Použití tohoto kriteria vyžaduje znalost problému.

Konvergence populace

Dalším kriteriem je konvergence populace. Kdy považujeme populaci za zkonvergovanou? Je to v případě, že v populaci již nedochází ke změnám (např. když

průměr už jen osciluje). Algoritmus je potom ukončen, například pokud se průměrné ohodnocení aktuální populace liší od nejlepšího jedince v dané populaci o zadaný počet procent.

Konvergence genů

Toto je ukončovací metoda, která zastaví výpočet, pokud zkonvergovalo zadané procento genů, které tvoří chromozom. Konvergence genu k určité alele je určena dosaženým počtem procent (které stanoví uživatel). Například když 90% populace v genetickém algoritmu má v daném genu 1, řekneme, že gen zkonvergoval k alele 1. Potom pokud zkonvergovalo například 80% genů v populaci, je genetický algoritmus ukončen.

Toto ukončovací kriterium není vhodné pro TSP, neboť v úloze TSP je důležité pořadí ale ne přesná pozice.

4 Konvergence

Tato práce je zaměřena na praktickou realizaci genetického algoritmu. Proto se konvergencí nebudeme zabývat detailněji a pouze uvedeme přehled různých přístupů používaných při zkoumání konvergence evolučních algoritmů či k vysvětlení jejich chování. Budeme vycházet z článku [12]. *Evoluční algoritmus* je společný název pro všechny pravděpodobnostní optimalizační algoritmy, které jsou inspirovány principy biologické evoluce. Ačkoli by se mohlo zdát, že konkrétní evoluční algoritmy se od sebe dost liší, mají ve skutečnosti hodně společného, proto je snaha o společnou teorii konvergence.

Teorie schémat

Teorie schémat reprezentuje jeden z raných pokusů vysvětlit chování genetického algoritmu. Tato teorie, která byla poprvé publikována v roce 1975, byla považována za zásadní pro porozumění genetickým algoritmům až do devadesátých let. Důvodů, proč se od této teorie pak upustilo, bylo několik. Především tato teorie vychází z předpokladu nekonečně velkých (nebo aspoň obrovských) populací. Při „rozumné" velikosti populace závěry velmi rychle přestávají platit ([1]). Dalším důvodem byl příchod teorie Markovových řetězců na pole evolučního počítání.

Teorie Markovových řetězců

Markovovy řetězce jsou k modelování a analyzování evolučních algoritmů vhodné, neboť populace v evolučních algoritmech závisí pouze na stavu předchozí populace pravděpodobnostním způsobem. První teoretické výsledky založené na kvalitativních modelech zabývajících se limitním chováním evolučních algoritmů vznikly v roce 1991. Zhruba ve stejnou dobu se objevily první práce popisující přesné přechodové matice Markovových řetězců spojené s některými evolučními algoritmy. I přes znalost těchto matic přechodu je však příliš obtížné získat tímto způsobem odhad rychlosti konvergence pro obecný evoluční algoritmus. Z toho důvodu byly doposud tímto způsobem vyšetřovány pouze jednoduché verze evolučních algoritmů.

Dimenzionální analýza

Skutečnost, že určitý Markovův model je izomorfní se souvisejícím evolučním algoritmem, vedla k nápadu přiblížení se evolučním algoritmům pomocí dimenzionální analýzy, která se pokouší identifikovat důležité vlastnosti a klíčové znaky celého systému a zjistit jejich závislost. Při aplikaci na evoluční algoritmy jsou odhadnuté závislosti ověřovány simulacemi. Úspěšnost tedy závisí na dobrém odhadu závislostí. Tento přístup je užitečný hlavně pro získání podkladů a podnětů pro podrobnější teoretická zkoumání.

Kvantitativní genetika

Tato metoda vychází z předpokladu, že analýza biologicky inspirovaných dynamických systémů by měla využít výsledků dosažených v teoretické biologii. Problém je ale v tom, že teoretické otázky, které se objeví v evolučním počítání, jsou obvykle odlišné od těch, které se řeší v teoretické genetice. Výjimka byla objevena Mühlenbeinem a Schlierkamp-Voosenem, kteří představili specifické evoluční algoritmy, které mohou být analyzovány pomocí teorie přímo vyvinuté pro kvantitativní genetiku. Ačkoli je tento přístup omezen pouze na aditivně separovatelné fitness funkce a nekonečně velké populace, má své místo v teorii evolučního počítání.

Analýza ortogonálních funkcí

Ortogonální funkce, jako jsou Fourierovy, Walshovy a Haarovy funkce, jsou používány jako nástroj pro vytváření některých fitness funkcí. Walshovy transformace byly občas použity při modelování evolučních algoritmů pomocí kvadratických dynamických systémů.

Kvadratické dynamické systémy

Dalším tradičním přístupem používaným k modelování různých přírodních fenoménů ve fyzice a biologii jsou modely kvadratických dynamických systémů (KDV). Jednoduchý genetický algoritmus může být za předpokladu nekonečně velké populace převeden na KDV. Tento přístup také není příliš efektivní, neboť

simulace KDV je náročná a navíc předpovědi KDV jsou dostatečně přesné pouze pro extrémně velké populace.

Statistická fyzika

Fyzikové vyvinuli různé způsoby, jak pracovat se stochastickými systémy, na které narazí ve statistické fyzice. Někteří autoři proto hledají inspiraci ve statistické fyzice a snaží se o začlenění biologických modelů a evolučních algoritmů do jejich rámce.

5 Numerické testování algoritmu

Praktickou část této diplomové práce tvoří programová realizace algoritmů ve výpočetním systému MATLAB. Všechny M-soubory jsou přiloženy na CD.

V této kapitole uvedeme všechny naprogramované M-soubory, popíšeme parametry hlavního M-souboru a ukážeme na příkladě, jak se projevují změny a různé kombinace parametrů. Potom se pokusíme vyřešit problém větších rozměrů.

5.1 Naprogramované M-soubory

Na přiloženém CD čtenář nalezne tyto M-soubory:

`ga.m` - tento M-soubor obsahuje hlavní genetický algoritmus

`VLSI131.m` - spouštěcí skript pro úlohu VLSI se 131 uzly, ve kterém se nastavují všechny parametry

`VLSI237.m` - spouštěcí skript pro úlohu VLSI s 237 uzly, ve kterém se nastavují všechny parametry

`VLSI1083.m` - spouštěcí skript pro úlohu VLSI s 1083 uzly, ve kterém se nastavují všechny parametry

`cenovam.m` - vytvoří cenovou matici pro dané uzly

`init.m` - vytvoří počáteční populaci

`evaluate.m` - ohodnotí každého jedince v populaci

`change.m` - provede křížení mezi vybranými jedinci

`change2.m` - provede mutaci na vybraných jedincích

`turnajselect.m` - turnajová selekce

`rankselect.m` - pořadová selekce

`fpselect.m` - selekce založená na hodnotě fitness

`krizPMX.m` - křížení s částečným zobrazením

`krizCX.m` - cyklický operátor křížení

`krizOX.m` - křížení se zachováním pořadí

`krizERX.m` - křížení s rekombinací hran

`krizAP.m` - křížení se střídáním pozic

`mutDM.m` - mutace založená na posunutí

`mutEM.m` - mutace založená na výměně

`mutISM.m` - mutace založená na vložení

`mutSIM.m` - jednoduchá inverzní mutace

`mutIVM.m` - inverzní mutace

`mutSM.m` - mutace založená na promíchání

`mutZIM.m` - zkracující inverzní mutace

`elitismus.m` - náhradová strategie elitismus

`graf.m` - grafické vykreslení nalezené cesty

`opt2.m` - heuristika 2-opt

`opt2C.c` - heuristika 2-opt v jazyce C

`heurNN.m` - heuristika nejbližšího souseda

`heur1.m` - heuristika vkládání nejbližšího bodu

`heur2.m` - heuristika vkládání nejvzdálenějšího bodu

`heur3.m` - heuristika nejlacinějšího vkládání

Dále pak na CD nalezneme datové soubory:

`data131` - obsahuje data pro 131 uzlů

`data237` - obsahuje data pro 237 uzlů

`data1083` - obsahuje data pro 1083 uzlů

5.2 Hlavní algotritmus „ga"

Hlavní M-soubor, který obsahuje celý algoritmus, se jmenuje `ga.m`. Jeho vstupními parametry jsou `uzly` a `options`. `Uzly` jsou vstupní data v podobě matice tvaru $n \times 2$, kde n je velikost problému. Veškerá testování i další příklady budeme provádět na datech pro VLSI úlohy. Jsou to úlohy, ve kterých jsou dané uzly v rovině, a jejich vzdálenost tedy můžeme měřit klasickou eukleidovskou normou. Každý řádek vstupní matice `uzly` potom tedy vyjadřuje souřadnice x, y v rovině. `Options` je struktura obsahující všechny vstupní parametry.

Parametr `popsize`, pomocí kterého určíme velikost populace, např. 500, 1000 apod.

Parametr `pop`, který znamená počet jedinců, kteří budou vybráni selekcí. V případě náhradové strategie elitismu se tyto dva parametry mohou lišit. Pokud použijeme prázdnou strategii, musí tyto dva parametry být nastaveny na stejnou hodnotu.

Dalším parametrem je metoda křížení, kterou najdeme pod názvem `krizeni`. Můžeme zvolit jedno z pěti křížení: `@krizPMX`, `@krizCX`, `@krizOX`, `@krizERX`, `@krizAP`.

Dále nastavíme parametr s názvem `mutace`. Zde vybereme jednu ze sedmi mutací: `@mutDM`, `@mutEM`, `@mutISM`, `@mutSIM`, `@mutIVM`, `@mutSM`, `@mutZIM`.

V předchozím textu byly podrobně popsány tři metody selekce. Mezi těmito třemi si vybereme a nastavíme je parametrem `selekce`. Zadáním `@turnajselect` zvolíme turnajovou selekci. Pro nastavení pořadové selekce zadáme `@rankselect` a pro selekci založenou na hodnotě fitness zadáme `@fpselect`.

Dalšími vstupními parametry jsou `PX` a `PM`. Jsou to čísla v rozmezí mezi 0 a 1 a vyjadřují pravděpodobnost, s jakou jedinec vybraný selekcí podstoupí rekombinaci (tj. křížení a mutaci). Pravděpodobnost křížení `PX` jsme pro všechny výpočty měli nastavenou na hodnotu 0,95 a pravděpodobnost mutace `PM` téměř vždy na 0,1 (případ, kdy jsme tuto hodnotu změnili, bude popsán dále). Podle některých zdrojů ([1], [3]) se mutace při řešení TSP používá zřídka. Pokud tedy budeme chtít mutaci úplně vyloučit, jednoduše nastavíme `PM` na hodnotu 0.

Dále zadáváme parametr `vyvoj`. Tento parametr určuje náhradovou strategii. Na výběr máme elitismus a prázdnou strategii. Pro elitismus nastavíme tento parametr na hodnotu 1, pro prázdnou strategii na hodnotu 2. S tímto parametrem se pojí i další parametr `elita`. Ten určuje, kolik nejlepších jedinců má být do další generace vybráno v případě zvolení náhradové strategie elitismu.

Dalším parametrem je `max_iter`, které vyjadřuje maximální počet iterací, které chceme vykonat.

Následuje parametr `PO`, což je pravděpodobnost, s jakou bude daný jedinec vylepšen pomocí 2-optimalizace.

Posledním parametrem je `initpop`, kterým zvolíme způsob vytvoření počáteční populace. Nastavíme-li tento parametr na hodnotu 1, bude počáteční populace vytvořena zcela náhodně. Pokud zadáme hodnotu 2, bude 8 jedinců vytvořeno pomocí konstrukčních a zlepšovacích heuristik, zbytek zcela náhodně. Zadáním hodnoty 3 zajistíme, že 8 jedinců v počáteční populaci bude opět vytvořeno pomocí konstrukčních a zlepšovacích heuristik, zbylí jedinci budou vytvořeni náhodně, ale s pravděpodobností `PO` budou vylepšeni pomocí 2-opt.

5.3 Realizace genetického algoritmu

Z důvodu urychlení výpočtu jsme zlepšovací algoritmus 2-opt naprogramovali v jazyce C. Soubor `opt2C.c` je tedy nutné nejprve přeložit. Mohlo by se stát, že na některých počítačích se to nezkompiluje. Potom je nutné dát překlad v M-souboru `ga.m` do poznámky.

Pro použití mutace jsou dvě možnosti - zda mutovat pouze potomky nebo

potomky i rodiče. My jsme algoritmus sestavili tak, že mutaci s danou pravděpodobností podstoupí jak potomci, tak i rodiče.

V případě prázdné náhradové strategie dostáváme v každé iteraci algoritmu populaci o velikosti `popsize`+1. To proto, že pokud bychom novou generaci získali z té předchozí vybráním (selekční metodou) počtu `popsize` jedinců ze smíšené množiny potomků a rodičů, mohli bychom tak ztratit nejlepšího jedince. Přidali jsme tedy toho nejlepšího jedince do následující populace, čímž jsme dostali počet `popsize`+1.

Do výstupní proměnné nazvané `output` jsme uložili vstupní parametry a dále jsme pak nechali v každé iteraci ukládat nejlepšího jedince, desátého nejlepšího jedince, padesátého nejlepšího jedince, průměrnou délku, maximální délku a nakonec nalezené řešení a jeho délku.

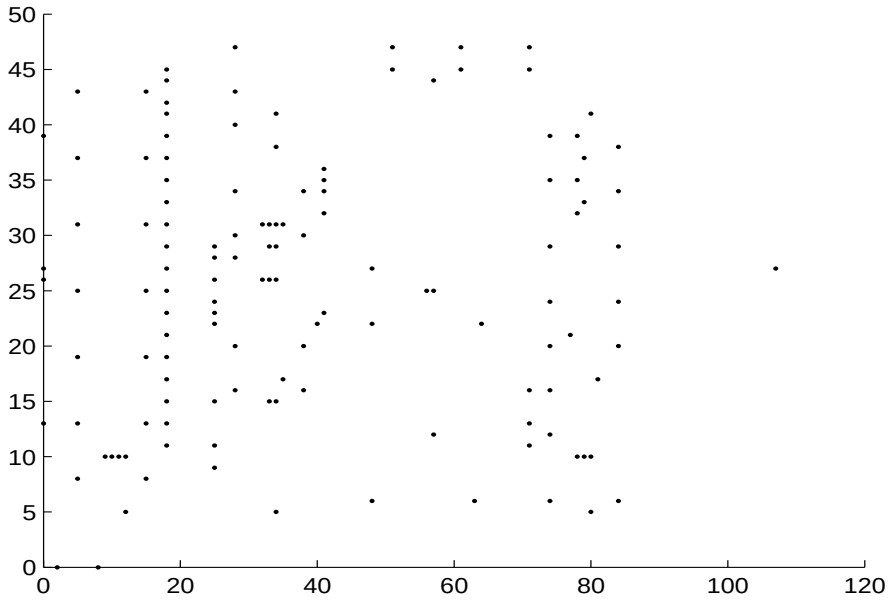
Je zřejmé, že kombinací, jak zvolit vstupní parametry, je hodně. Proto jsme testovali jen některé (ty nejvýznamnější) z nich. Testování jsme prováděli na problému 131 měst, jehož výsledky rozebereme v následující podkapitole.

Veškeré výpočty byly prováděny ve výpočetním systému MATLAB 7.0 na notebooku s procesorem Intel Core 2 Duo, 1,66 GHz, 1 GB RAM.

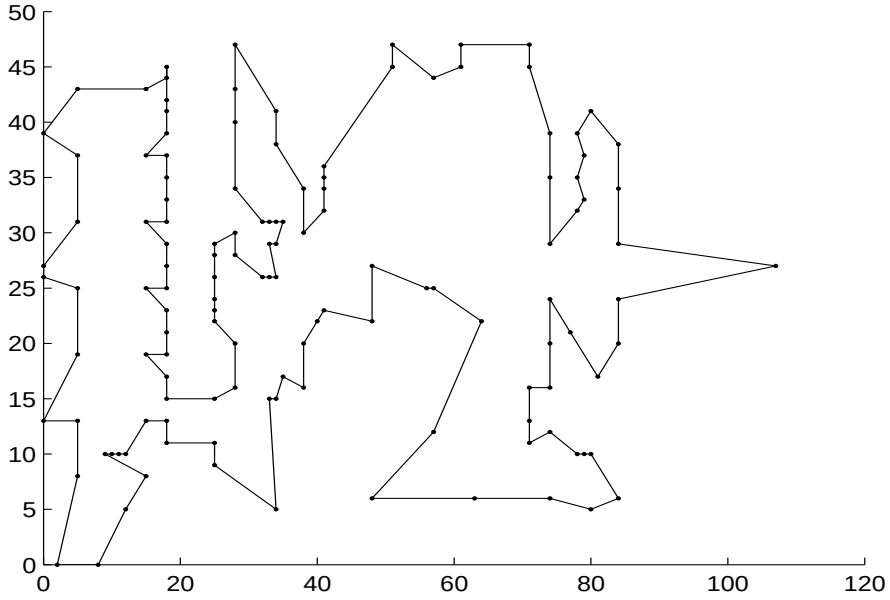
5.4 VLSI pro 131 uzlů

Data pro tento příklad jsme čerpali z [6]. Důvod proč testujeme právě na úloze VLSI je jednoduchost počítání vzdáleností bodů, neboť jsme v rovině a používáme eukleidovskou normu. Podobně jako v [6] také vzdálenosti bodů zaokrouhlujeme, což nám umožňuje srovnávat výsledky s referenčními.

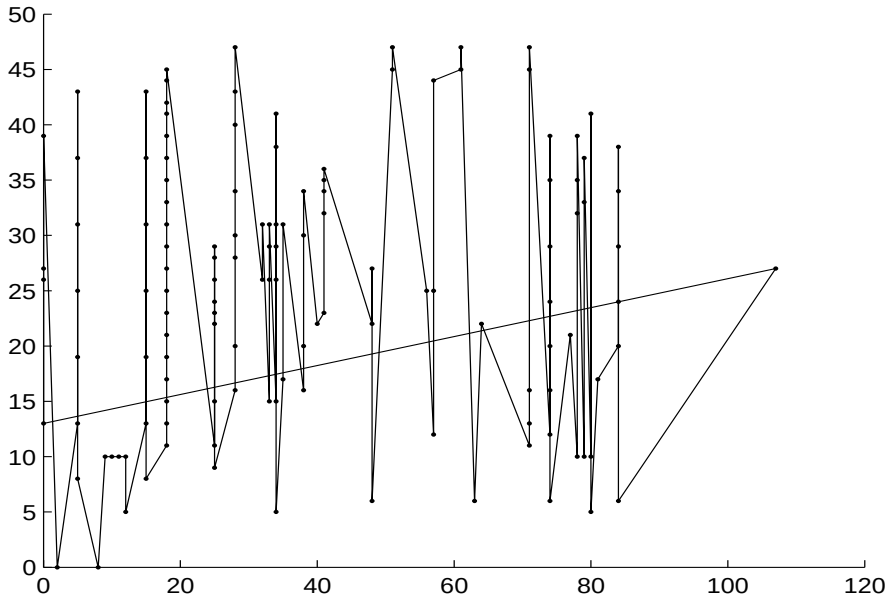
Máme 131 bodů v rovině, optimální trasa je známá - je 564. Takto vypadá rozložení uzlů, optimální trasa a trasy nalezené heuristikami:



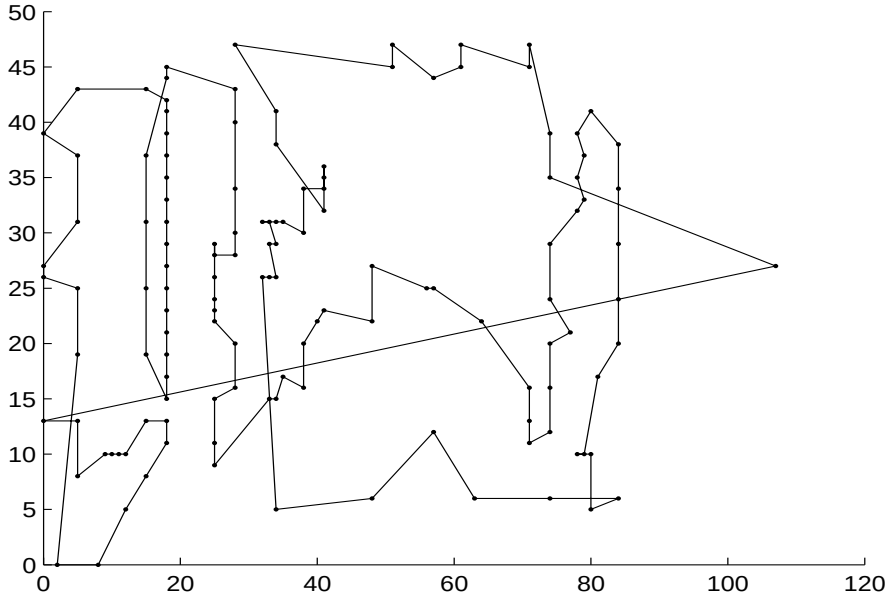
Obrázek 4: Rozložení 131 uzlů.



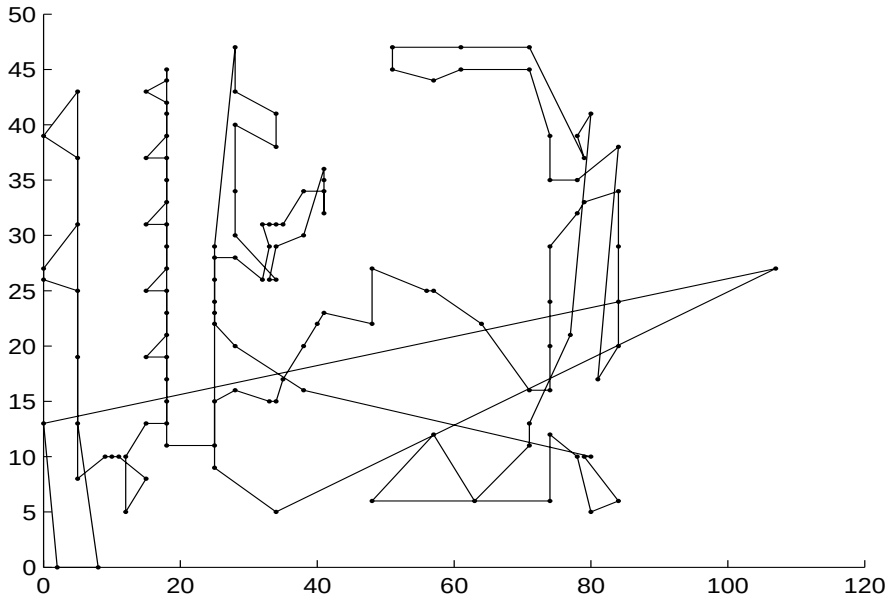
Obrázek 5: Optimální trasa pro 131 uzlů délky 564.



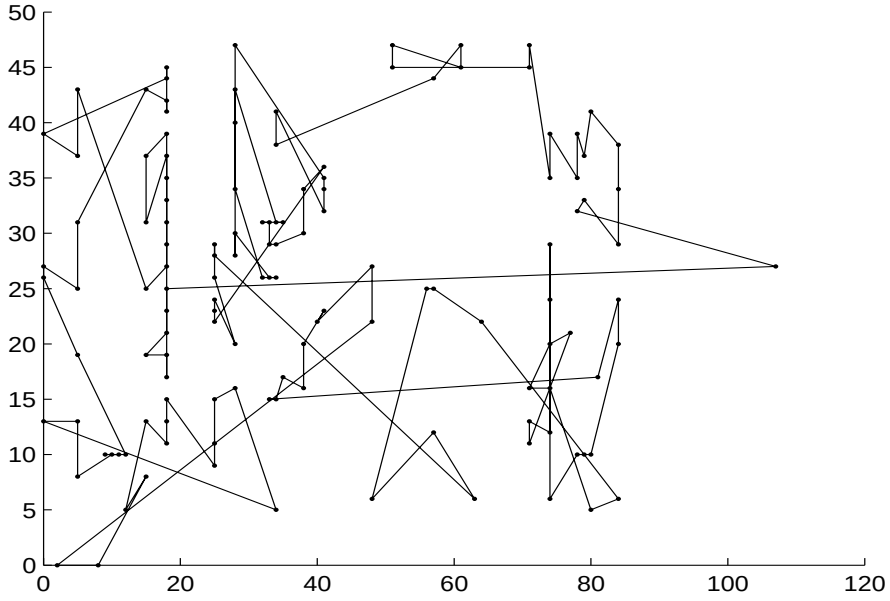
Obrázek 6: Cesta délky 1275 daná permutací 1 až 131.



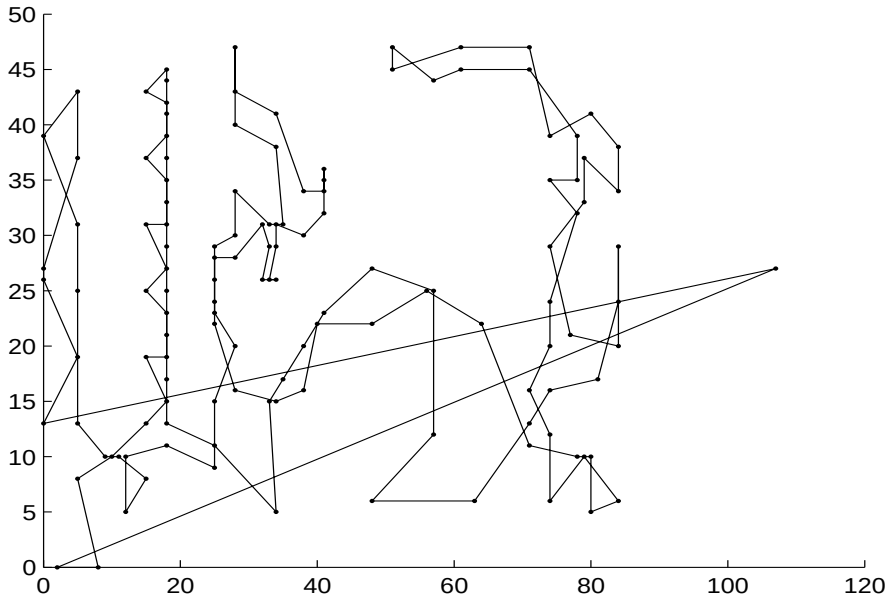
Obrázek 7: Řešení délky 596 získané heuristikou nejbližšího souseda.



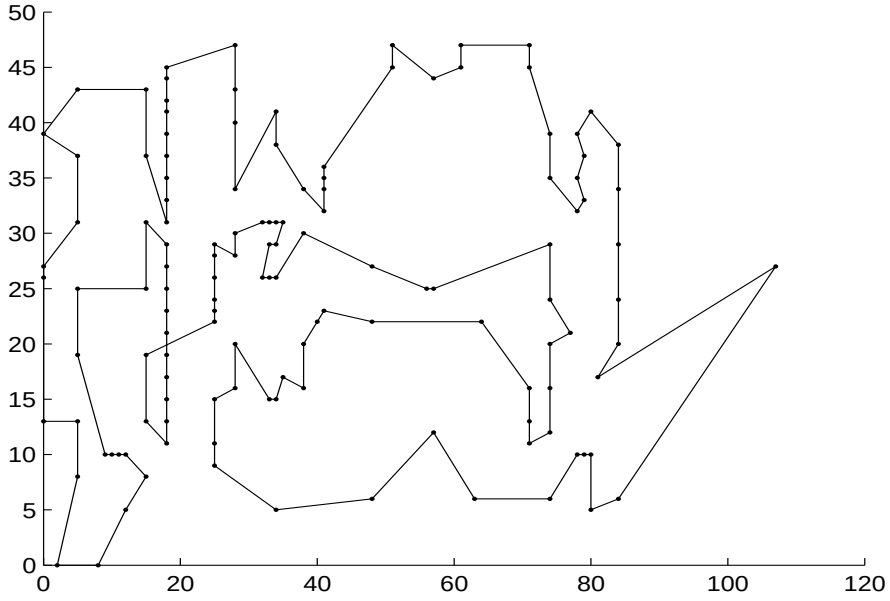
Obrázek 8: Řešení délky 913 získané heuristikou vkládání nejbližšího bodu.



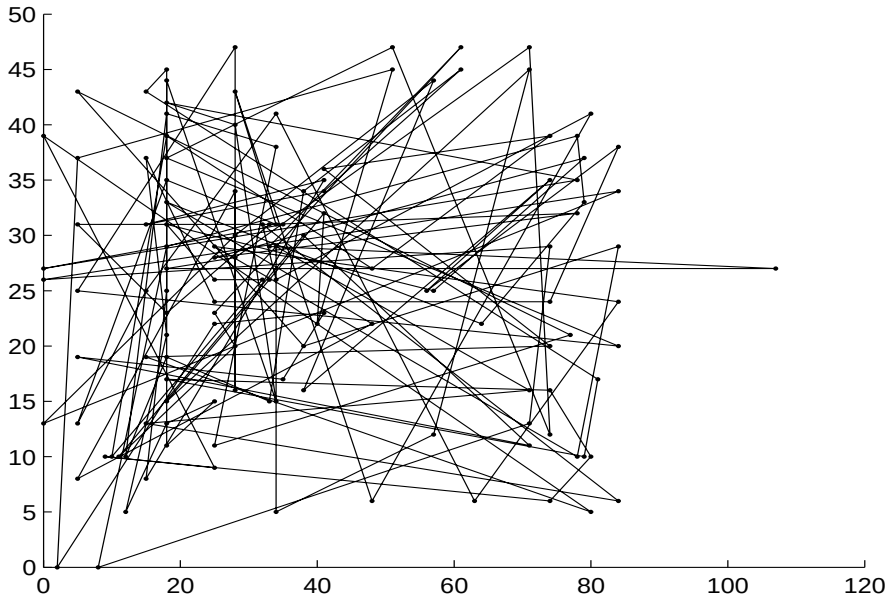
Obrázek 9: Řešení délky 1018 získané heuristikou vkládání nejvzdálenějšího bodu.



Obrázek 10: Řešení délky 921 získané heuristikou nejlacinějšího vkládání.



Obrázek 11: Řešení získané použitím 2-optimalizace na permutaci 1 až 131 má délku 612.



Obrázek 12: Řešení délky 4560 získané náhodnou permutací.

5.4.1 Test vlivu náhradové strategie

Nechali jsme proběhnout 1000 iterací pro každé křížení - jednou s náhradovou strategií elitismus a podruhé s prázdnou strategií. S parametry:

```
popsize = 1000
pop = 1000
mutace = @mutDM
selekce = @turnajselect
PX = 0,95
PM = 0,1
initpop = 1
PO = 0
```

Budeme měnit náhradové strategie a pro každou z nich otestujeme všechna křížení.

Pro každé křížení jsme udělali deset běhů a pro každý výsledek spočítali relativní chybu ρ definovanou jako

$$\rho = \frac{D - D_{opt}}{D_{opt}},$$

kde D je délka nalezeného řešení a D_{opt} je optimální délka.

Dříve než však spustíme testování, musíme si položit otázku, kolik nejlepších jedinců v případě náhradové strategie elitismu by mělo postoupit do další generace? V literatuře jsme o tom nenalezli žádné detaily, proto nejdříve provedeme test pro každé křížení, ve kterém se pokusíme určit optimální hodnotu tohoto parametru, označme ho n. Výsledky jsou následující:

ELITISMUS pro různá n

	PMX		CX		OX		ERX		AP	
n	D	ρ	D	ρ	D	ρ	D	ρ	D	ρ
1	2677	3,75	1264	1,24	2376	3,21	2969	4,26	2643	3,69
2	2459	3,36	1487	1,64	2060	2,65	2973	4,27	2615	3,64
5	1927	2,42	1219	1,16	1709	2,03	2703	3,79	2626	3,66
10	1700	2,01	1280	1,27	1311	1,32	2635	3,67	2589	3,59
20	884	0,57	1129	1,00	1215	1,15	2527	3,48	2381	3,22
30	833	0,48	938	0,66	885	0,57	2384	3,23	2315	3,10
50	718	0,27	836	0,48	766	0,36	2413	3,28	2178	2,86
100	743	0,32	818	0,45	635	0,13	2373	3,21	1610	1,85
200	687	0,22	828	0,47	663	0,18	2358	3,18	1291	1,29
300	701	0,24	751	0,33	631	0,12	2321	3,12	928	0,65
500	702	0,24	775	0,37	672	0,19	2313	3,10	1016	0,80

Vidíme, že u všech křížení dostáváme nejlepší výsledek přibližně pro $n = 300$, pro vyšší hodnoty už se výsledek nelepší. Otestujeme tedy všechna křížení s hodnotou $n = 300$. Délky řešení nalezených po 1000 iteracích jsou v následujících tabulkách (ve sloupci k uvádíme číslo výpočtu se stejným nastavením, v řádku P pak průměrné hodnoty za 10 výpočtů):

ELITISMTUS pro $n = 300$

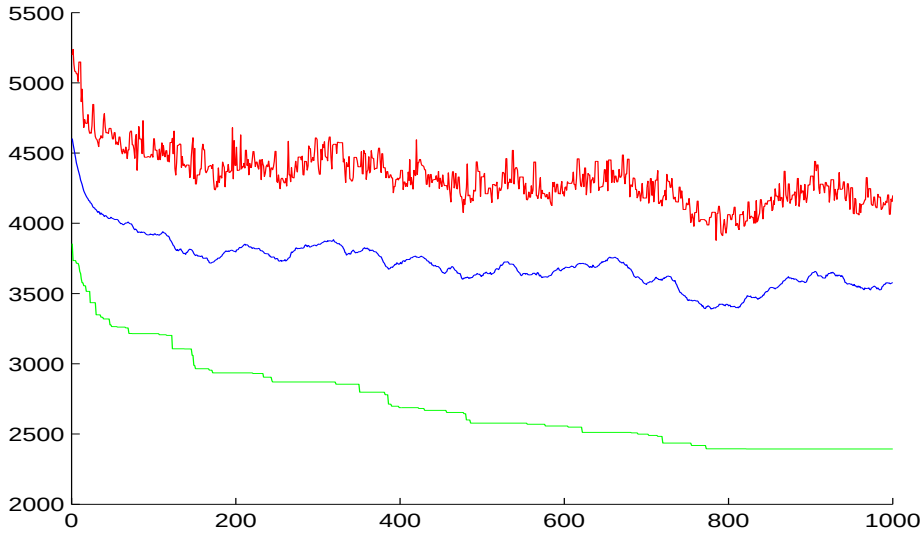
	PMX		CX		OX		ERX		AP	
k	D	ρ	D	ρ	D	ρ	D	ρ	D	ρ
1	701	0,24	751	0,33	631	0,12	2321	3,12	928	0,65
2	699	0,24	803	0,42	625	0,11	2237	2,97	952	0,69
3	682	0,21	786	0,39	607	0,08	2234	2,96	917	0,63
4	692	0,23	799	0,42	623	0,10	2319	3,11	981	0,74
5	644	0,14	819	0,45	646	0,15	2288	3,06	929	0,65
6	726	0,29	800	0,42	624	0,11	2296	3,07	919	0,63
7	707	0,25	784	0,39	621	0,10	2250	2,99	1043	0,85
8	689	0,22	790	0,40	616	0,09	2246	2,98	1022	0,81
9	655	0,16	746	0,32	622	0,10	2302	3,08	1117	0,98
10	663	0,18	788	0,40	639	0,13	2333	3,14	980	0,74
P	686	0,22	787	0,39	625	0,11	2283	3,05	979	0,74

PRÁZDNÁ STRATEGIE

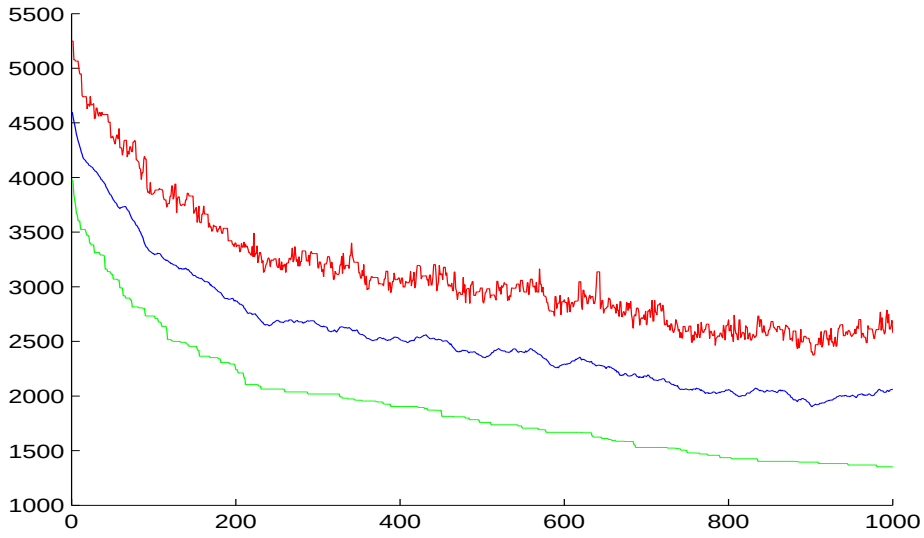
	PMX		CX		OX		ERX		AP	
k	D	ρ	D	ρ	D	ρ	D	ρ	D	ρ
1	714	0,27	830	0,47	729	0,29	919	0,63	914	0,62
2	695	0,23	866	0,54	733	0,30	936	0,66	878	0,56
3	719	0,27	844	0,50	748	0,33	969	0,72	917	0,63
4	761	0,35	855	0,52	703	0,25	941	0,67	932	0,65
5	712	0,26	870	0,54	722	0,28	982	0,74	892	0,58
6	726	0,29	889	0,58	699	0,24	966	0,71	897	0,59
7	697	0,24	896	0,59	711	0,26	973	0,73	864	0,53
8	723	0,28	845	0,50	725	0,29	942	0,67	854	0,51
9	662	0,17	813	0,44	719	0,27	934	0,66	845	0,50
10	755	0,34	871	0,54	673	0,19	943	0,67	877	0,55
P	716	0,27	858	0,52	716	0,27	951	0,69	887	0,57

Z uvedených výsledků můžeme vyvodit, že obě strategie dávají téměř srovnatelné výsledky při vhodném nastavení parametru n u elitismu. Výjimkou je křížení ERX, které v případě prázdné strategie dává mnohem lepší výsledky než v případě strategie elitismu.

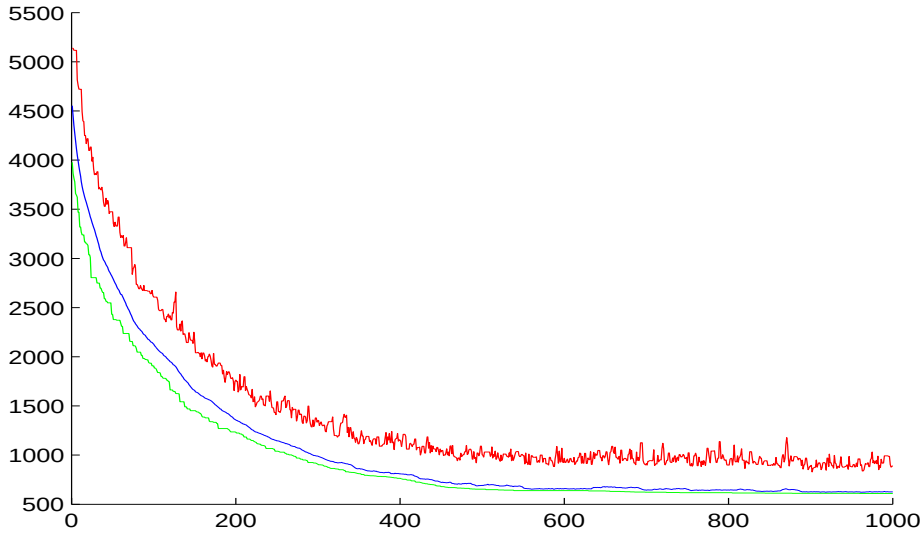
Vyšší hodnota parametru n způsobuje větší selekční tlak a tím rychlejší klesání délek nejlepšího jedince, průměrných délek i nejhoršího jedince. To vyjadřují i následující grafy selekčního tlaku.



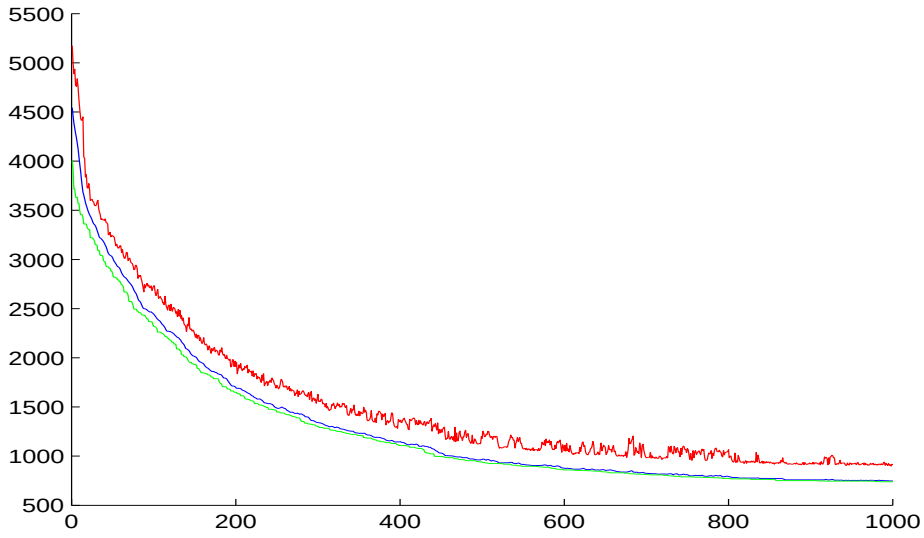
Obrázek 13: Selekční tlak u elitismu pro křížení OX, $n = 1$. Červená - nejhorší jedinec, modrá - průměr, zelená - nejlepší jedinec.



Obrázek 14: Selekční tlak u elitismu pro křížení OX, $n = 10$. Červená - nejhorší jedinec, modrá - průměr, zelená - nejlepší jedinec.

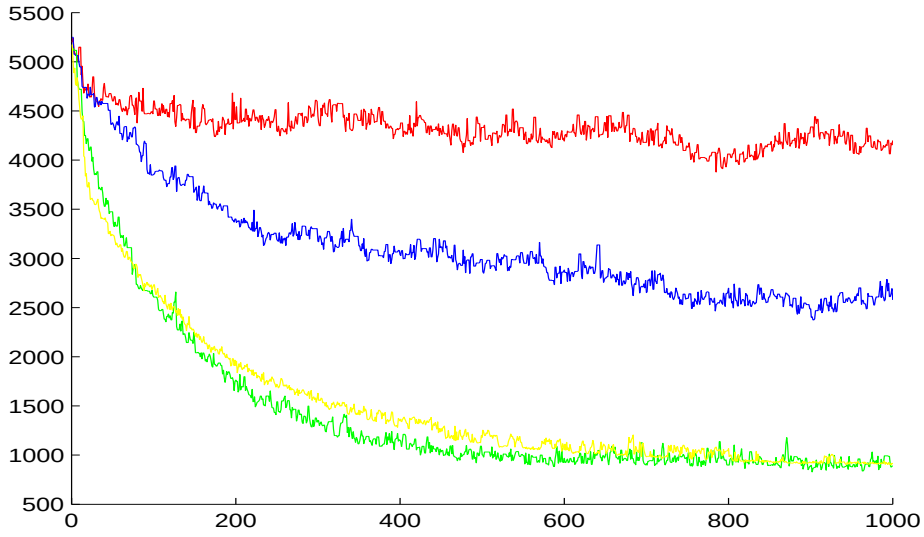


Obrázek 15: Selekční tlak u elitismu pro křížení OX, $n = 300$. Červená - nejhorší jedinec, modrá - průměr, zelená - nejlepší jedinec.

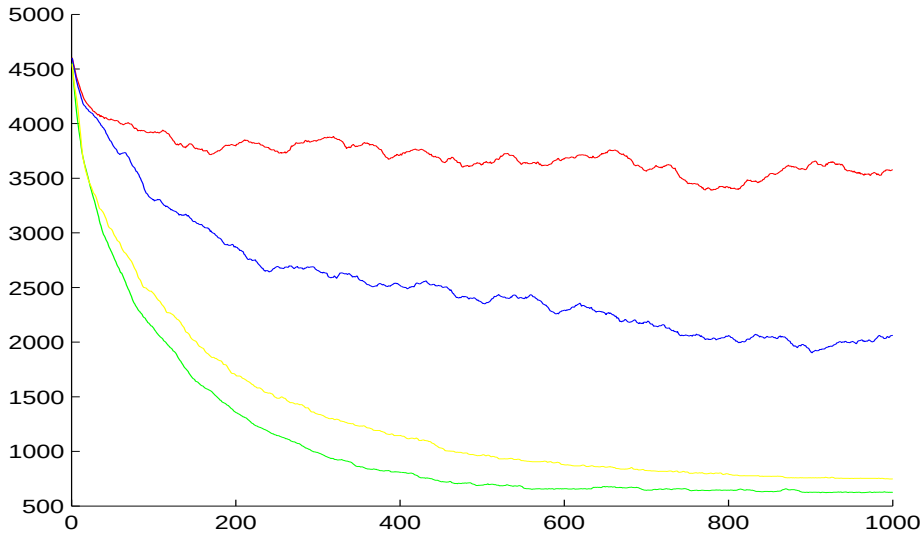


Obrázek 16: Selekční tlak u prázdné strategie pro křížení OX. Červená - nejhorší jedinec, modrá - průměr, zelená - nejlepší jedinec.

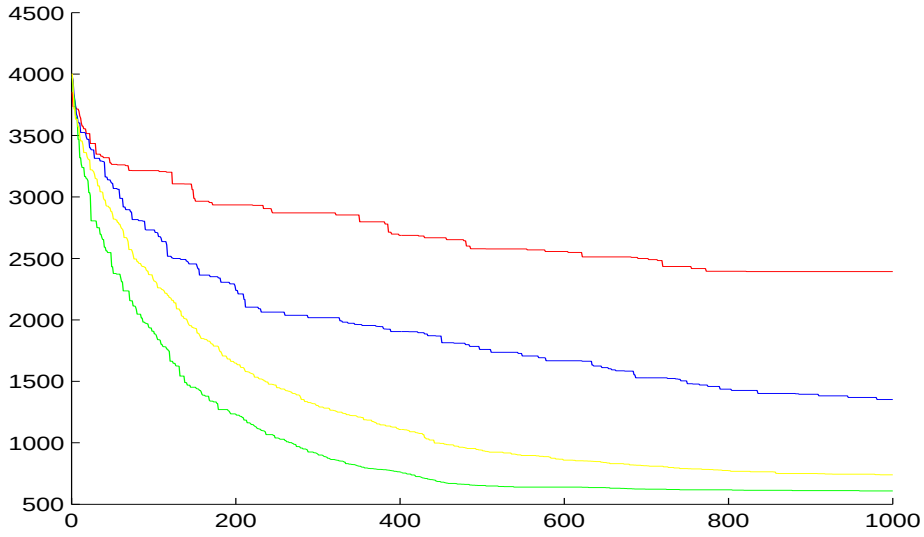
Pomocí následujících obrázků můžeme porovnat strategie - vidíme na nich, jak se u každého případu mění nejlepší jedinec, průměr a nejhorší jedinec.



Obrázek 17: Porovnání nejhorších jedinců u jednotlivých strategií pro OX. Červená - elitismus pro $n = 1$, modrá - elitismus pro $n = 10$, zelená - elitismus pro $n = 300$, žlutá - prázdná strategie.



Obrázek 18: Porovnání průměru u jednotlivých strategií pro OX. Červená - elitismus pro $n = 1$, modrá - elitismus pro $n = 10$, zelená - elitismus pro $n = 300$, žlutá - prázdná strategie.



Obrázek 19: Porovnání nejlepších jedinců u jednotlivých strategií pro OX.
Červená - elitismus pro $n = 1$, modrá - elitismus pro $n = 10$,
zelená - elitismus pro $n = 300$, žlutá - prázdná strategie.

Podle uvedených grafů bychom mohli usoudit, že náhradová strategie elitismus s parametrem $n = 300$ je lepší než prázdná strategie, nicméně nemusí to platit pro všechna křížení.

5.4.2 Test vlivu selekce

Zajímá nás, jak běh celého algoritmu ovlivní změna selekce. To budeme testovat pro tyto parametry:

```
popsize = 1000
pop = 1000
mutace = @mutDM
PX = 0,95
PM = 0,1
initpop = 1
PO = 0
vyvoj = 2
```

Pro jednotlivá křížení jsme po 1000 iteracích dostali tyty výsledky:

Křížení PMX

	turnaj		pořadová		fitness	
k	D	ρ	D	ρ	D	ρ
1	714	0,27	669	0,19	1187	1,10
2	695	0,23	703	0,25	1194	1,12
3	719	0,27	665	0,18	1207	1,14
4	761	0,35	673	0,19	1070	0,90
5	712	0,26	630	0,12	1146	1,03
6	726	0,29	706	0,25	1147	1,03
7	697	0,24	722	0,28	1210	1,15
8	723	0,28	674	0,20	1108	0,96
9	662	0,17	651	0,15	1232	1,18
10	755	0,34	680	0,21	1100	0,95
P	716	0,27	677	0,20	1160	1,06

Křížení CX

	turnaj		pořadová		fitness	
k	D	ρ	D	ρ	D	ρ
1	830	0,47	742	0,32	1642	1,91
2	866	0,54	781	0,38	1529	1,71
3	844	0,50	746	0,32	1555	1,76
4	855	0,52	735	0,30	1514	1,68
5	870	0,54	787	0,40	1528	1,71
6	889	0,58	784	0,39	1533	1,72
7	896	0,59	775	0,37	1531	1,71
8	845	0,50	746	0,32	1387	1,46
9	813	0,44	751	0,33	1492	1,65
10	871	0,54	751	0,33	1502	1,66
P	858	0,52	760	0,35	1521	1,70

Křížení OX

k	turnaj		pořadová		fitness	
	D	ρ	D	ρ	D	ρ
1	729	0,29	649	0,15	971	0,72
2	733	0,30	663	0,18	921	0,63
3	748	0,33	676	0,20	950	0,68
4	703	0,25	660	0,17	968	0,72
5	722	0,28	705	0,25	996	0,77
6	699	0,24	671	0,19	953	0,69
7	711	0,26	650	0,15	1013	0,80
8	725	0,29	665	0,18	1018	0,80
9	719	0,27	672	0,19	949	0,68
10	673	0,19	636	0,13	1095	0,94
P	716	0,27	665	0,18	983	0,74

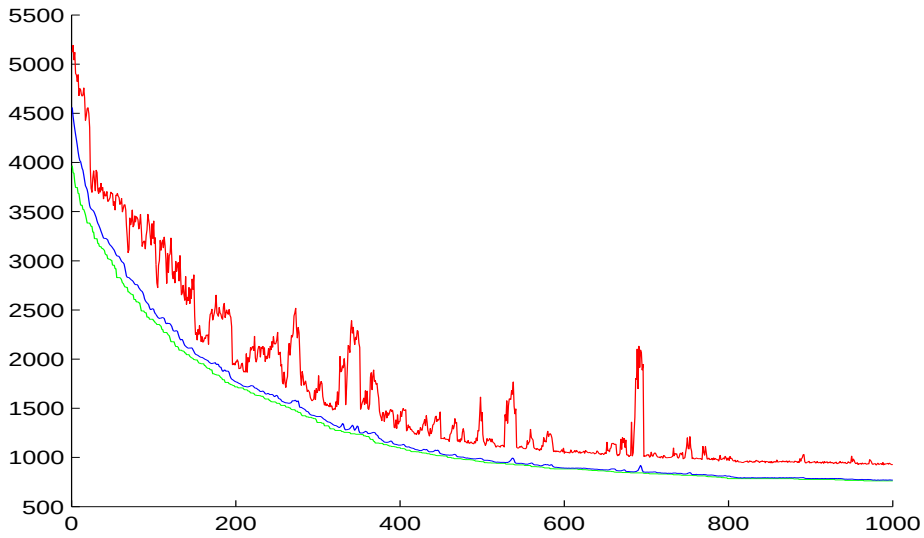
Křížení ERX

k	turnaj		pořadová		fitness	
	D	ρ	D	ρ	D	ρ
1	919	0,63	936	0,66	1780	2,16
2	936	0,66	886	0,57	1625	1,88
3	969	0,72	921	0,63	1614	1,86
4	941	0,67	925	0,64	1658	1,94
5	982	0,74	938	0,66	1588	1,82
6	966	0,71	905	0,60	1682	1,98
7	973	0,73	919	0,63	1676	1,97
8	942	0,67	935	0,66	1759	2,12
9	934	0,66	901	0,60	1642	1,91
10	943	0,67	936	0,66	1638	1,90
P	951	0,69	920	0,63	1666	1,95

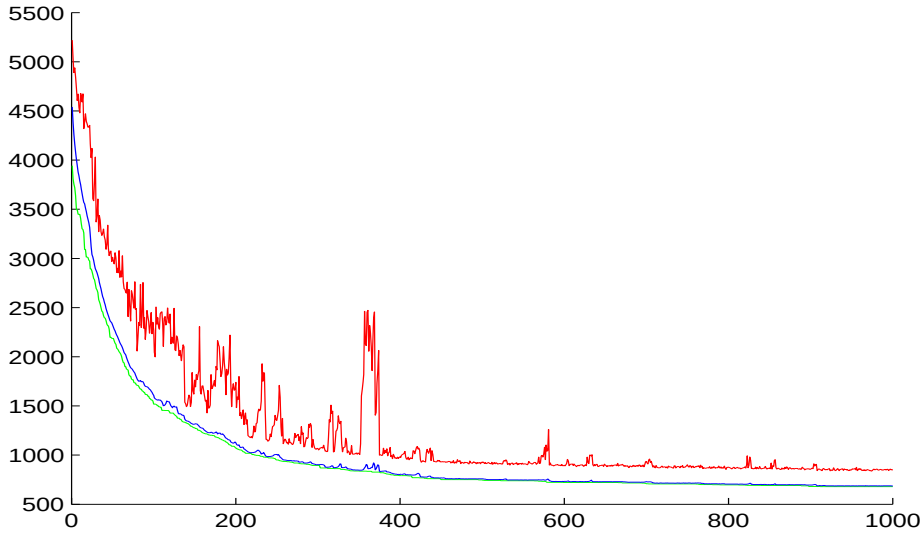
Křížení AP

k	turnaj		pořadová		fitness	
	D	ρ	D	ρ	D	ρ
1	914	0,62	764	0,35	1549	1,75
2	878	0,56	746	0,32	1611	1,86
3	917	0,63	793	0,41	1758	2,12
4	932	0,65	782	0,39	1546	1,74
5	892	0,58	762	0,35	1460	1,59
6	897	0,59	828	0,47	1666	1,95
7	864	0,53	807	0,43	1677	1,97
8	854	0,51	842	0,49	1615	1,86
9	845	0,50	787	0,40	1624	1,88
10	877	0,55	798	0,41	1665	1,95
P	887	0,57	791	0,40	1617	1,87

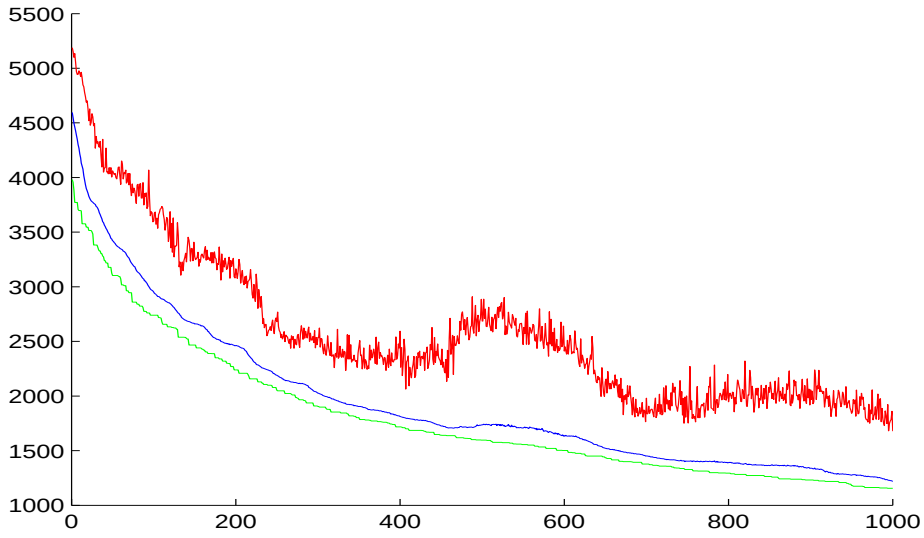
Porovnáním těchto výsledků bychom mohli udělat závěr, že turnajová a pořadová selekce jsou pro úlohu obchodního cestujícího vhodnější než selekce založená na hodnotě fitness. Selekční tlak potom vyjadřují následující grafy.



Obrázek 20: Selekční tlak u turnajové selekce pro křížení PMX. Červená - nejhorší jedinec, modrá - průměr, zelená - nejlepší jedinec.

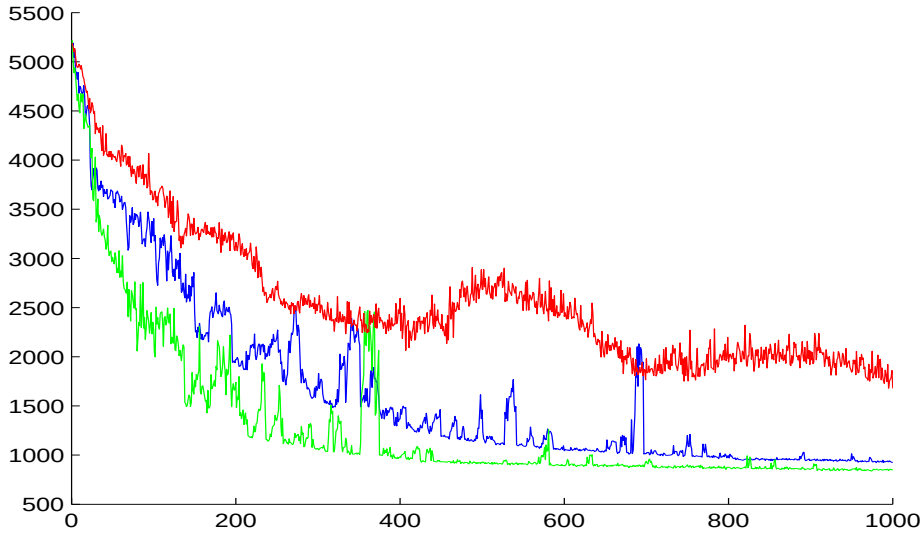


Obrázek 21: Selekční tlak u pořadové selekce pro křížení PMX. Červená - nejhorší jedinec, modrá - průměr, zelená - nejlepší jedinec.

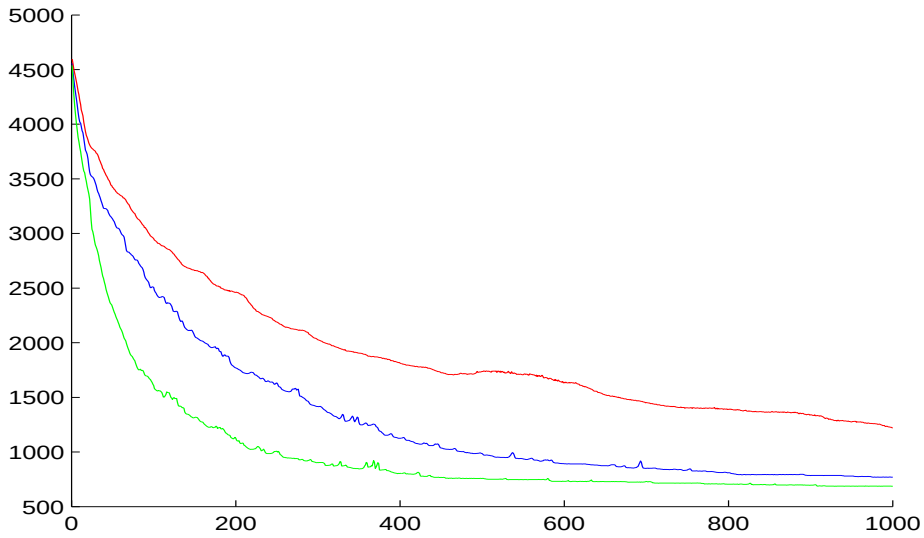


Obrázek 22: Selekční tlak u selekce založené na hodnotě fitness pro křížení PMX. Červená - nejhorší jedinec, modrá - průměr, zelená - nejlepší jedinec.

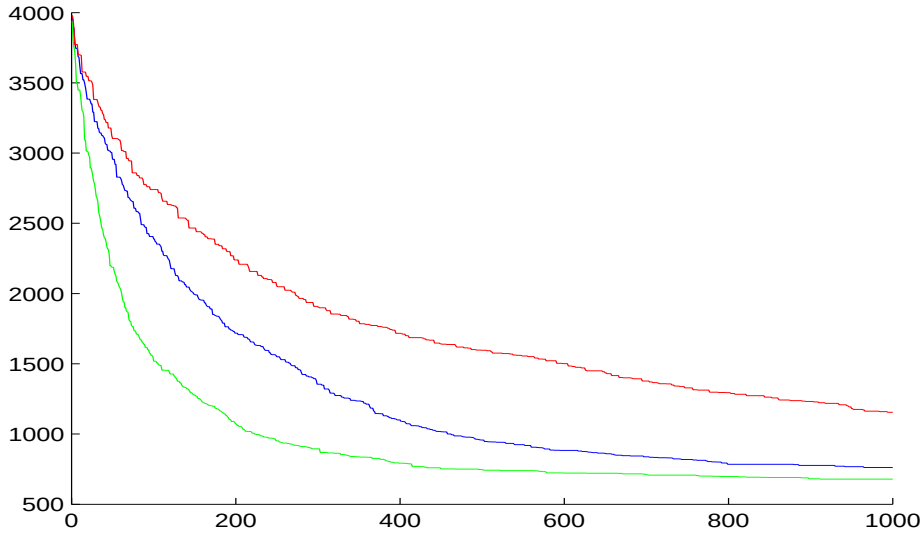
Pomocí následujících obrázků můžeme typy selekce porovnat - vidíme na nich, jak se u každého případu mění nejlepší jedinec, průměr a nejhorší jedinec.



Obrázek 23: Porovnání nejhorších jedinců u jednotlivých selekcí pro PMX. Modrá - turnajová selekce, zelená - pořadová selekce, červená - selekce založená na hodnotě fitness.



Obrázek 24: Porovnání průměru u jednotlivých selekcí pro PMX. Modrá - turnajová selekce, zelená - pořadová selekce, červená - selekce založená na hodnotě fitness.



Obrázek 25: Porovnání nejlepších jedinců u jednotlivých selekcí pro PMX.
Modrá - turnajová selekce, zelená - pořadová selekce,
červená - selekce založená na hodnotě fitness.

5.4.3 Test vlivu velikosti populace

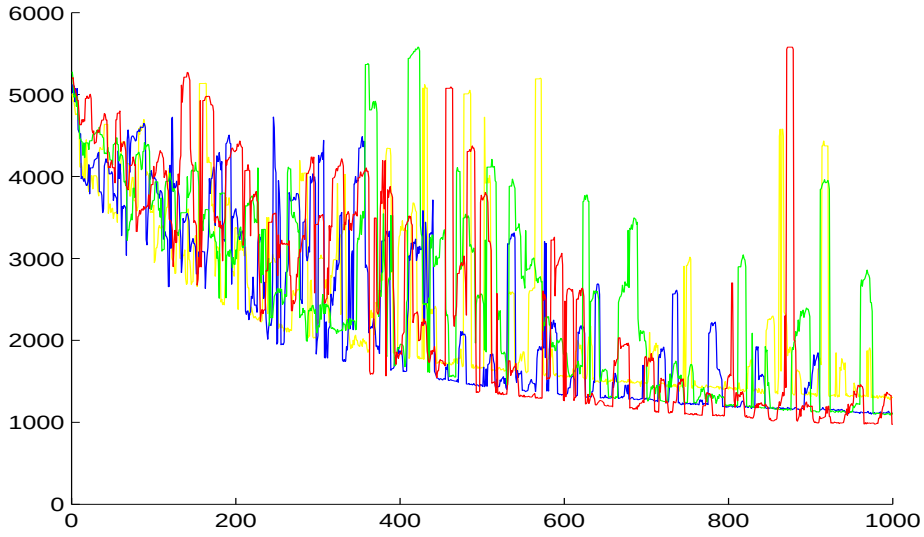
Důležitým parametrem, jak již bylo zmíněno dříve, je volba velikosti populace. Pro většinu testování na našem příkladě 131 měst jsme používali popsize = 1000. Pojďme se podívat, jaké výsledky dostaneme po 1000 iteracích, pokud zvětšíme popsize na 2000 a naopak zmenšíme na 500 či na 100. Toto testování jsme prováděli s parametry:

```
krizeni = @krizCX
mutace = @mutDM
selekce = @turnajselect
PX = 0,95
PM = 0,1
initpop = 1
PO = 0
vyvoj = 2
```

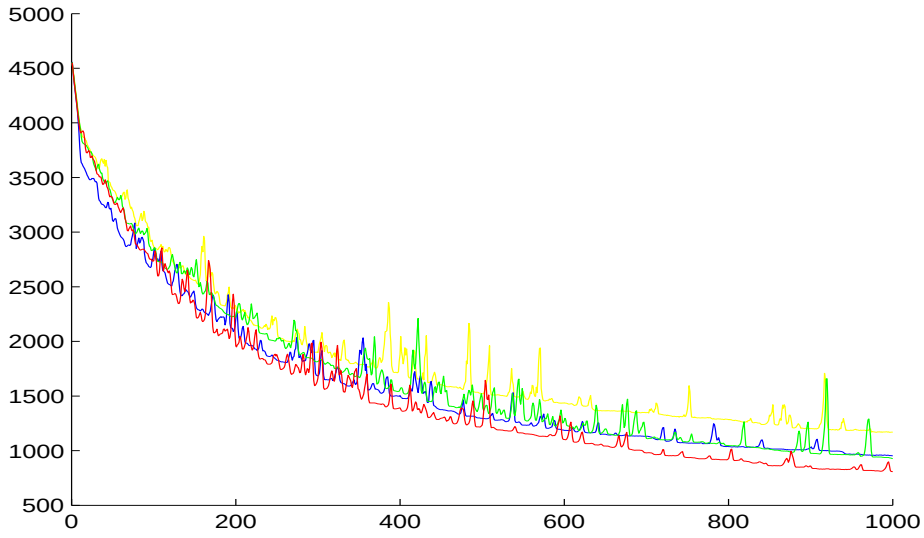
VLIV VELIKOSTI POPULACE

	100		500		1000		2000	
k	D	ρ	D	ρ	D	ρ	D	ρ
1	1151	1,04	900	0,60	830	0,47	799	0,42
2	1107	0,96	906	0,61	866	0,54	783	0,39
3	1223	1,17	960	0,70	844	0,50	829	0,47
4	1135	1,01	872	0,55	855	0,52	794	0,41
5	1133	1,01	877	0,55	870	0,54	804	0,43
6	1110	0,97	913	0,62	889	0,58	798	0,41
7	1147	1,03	932	0,65	896	0,59	850	0,51
8	1264	1,24	910	0,61	845	0,50	778	0,38
9	1174	1,08	950	0,68	813	0,44	805	0,43
10	1096	0,94	932	0,65	871	0,54	783	0,39
P	1154	1,05	915	0,62	858	0,52	802	0,42

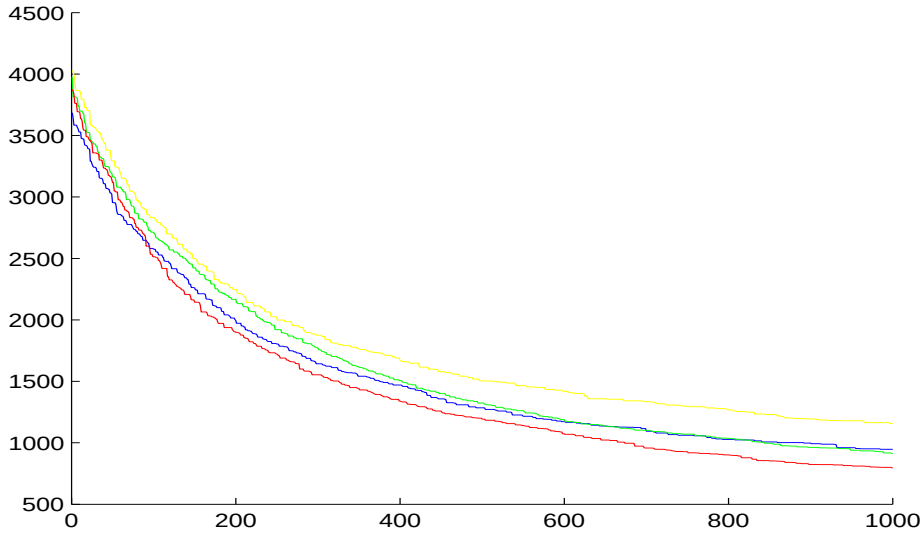
Vidíme, že zdvojnásobením velikosti populace došlo k mírnému zlepšení. V případě zmenšení velikosti populace na polovinu je též patrný rozdíl způsobený pravděpodobně nedostatečnou rozmanitostí v populaci. Populace velikosti 100 je už pro tuto úlohu příliš malá, dává výrazně horší výsledky než populace velikosti 1000. Pro porovnání změny nejhorších jedinců, průměru a nejlepších jedinců při různých velikostech počáteční populace slouží následující obrázky.



Obrázek 26: Porovnání nejhorších jedinců. Žlutá - popsize = 100, modrá - popsize = 500, zelená - popsize = 1000, červená - popsize = 2000.



Obrázek 27: Porovnání průměru. Žlutá - popsize = 100, modrá - popsize = 500, zelená - popsize = 1000, červená - popsize = 2000.



Obrázek 28: Porovnání nejlepších jedinců. Žlutá - popsize = 100, modrá - popsize = 500, zelená - popsize = 1000, červená - popsize = 2000.

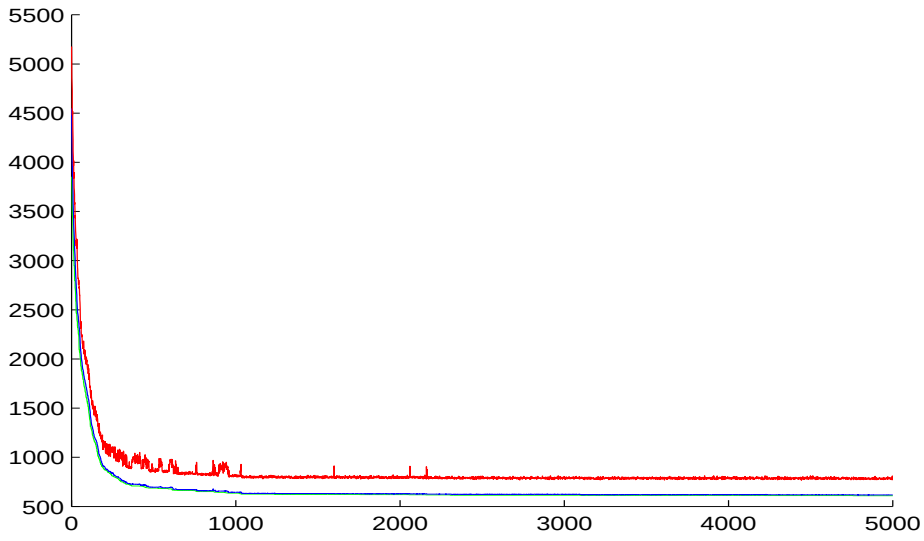
5.4.4 Test vlivu počtu iterací

Doposud jsme všechna testování nechali běžet 1000 iterací. Pojďme se nyní podívat, jak blízko optimálnímu řešení se dostaneme, pokud necháme algoritmus běžet více či méně iterací. Mějme parametry:

```
popsize = 1000
pop = 1000
krizeni = @krizOX
mutace = @mutDM
selekce = @rankselect
PX = 0,95
PM = 0,1
initpop = 1
PO = 0
vyvoj = 2
```

VLIV POČTU ITERACÍ

k	100		500		1000		3000		5000	
	D	ρ	D	ρ	D	ρ	D	ρ	D	ρ
1	1341	1,38	703	0,25	649	0,15	619	0,10	609	0,08
2	1435	1,54	701	0,24	663	0,18	638	0,13	620	0,10
3	1359	1,41	678	0,20	676	0,20	642	0,14	602	0,07
4	1500	1,66	686	0,22	660	0,17	617	0,09	614	0,09
5	1429	1,53	720	0,28	705	0,25	633	0,12	631	0,12
6	1498	1,66	726	0,29	671	0,19	631	0,12	604	0,07
7	1410	1,50	691	0,23	650	0,15	643	0,14	625	0,11
8	1387	1,46	701	0,24	665	0,18	618	0,10	604	0,07
9	1502	1,66	650	0,15	672	0,19	616	0,09	625	0,11
10	1332	1,36	722	0,28	636	0,13	636	0,13	604	0,07
P	1419	1,52	698	0,24	665	0,18	629	0,12	614	0,09



Obrázek 29: Změna nejlepšího jedince (zelená), průměru (modrá) a nejhoršího jedince (červená) během 5000 iterací.

Zvýšením počtu iterací se pochopitelně nalezené řešení zlepšilo. Ovšem v porovnání s výsledky dosaženými po 1000 iteracích můžeme říci, že se během posledních 4000 iterací zlepšuje poměrně pomalu.

5.4.5 Test vlivu mutace

Podle literatury ([1]) je mutace až druhotnou rekombinační operací. Doposud jsme tento parametr v žádném testování neměnili a používali mutaci DM. Otestujme nyní i ostatní mutace. Parametry:

```
popsize = 1000
pop = 1000
krizeni = @krizPMX
selekce = @turnajselect
PX = 0,95
PM = 0,1
initpop = 1
PO = 0
vyvoj = 2
```

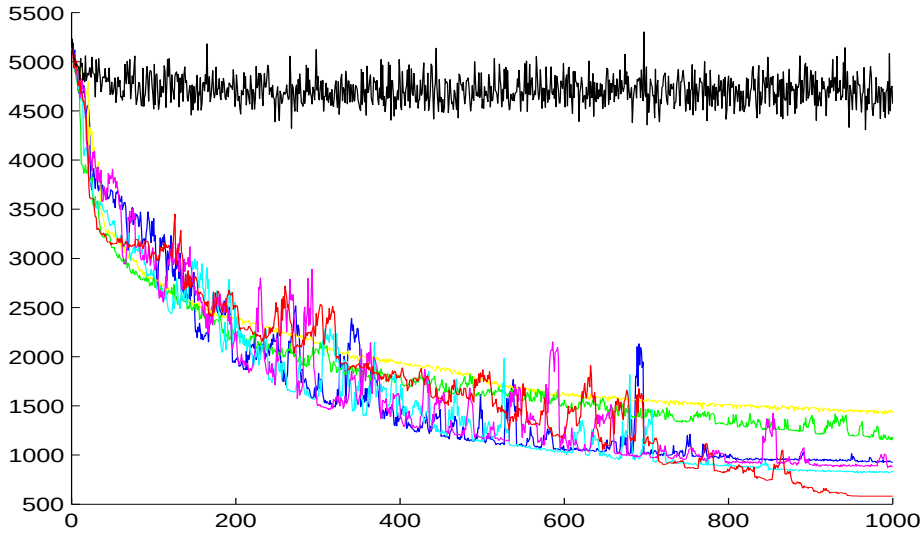
Po 1000 iteracích bylo dosaženo těchto výsledků:

VLIV MUTACE

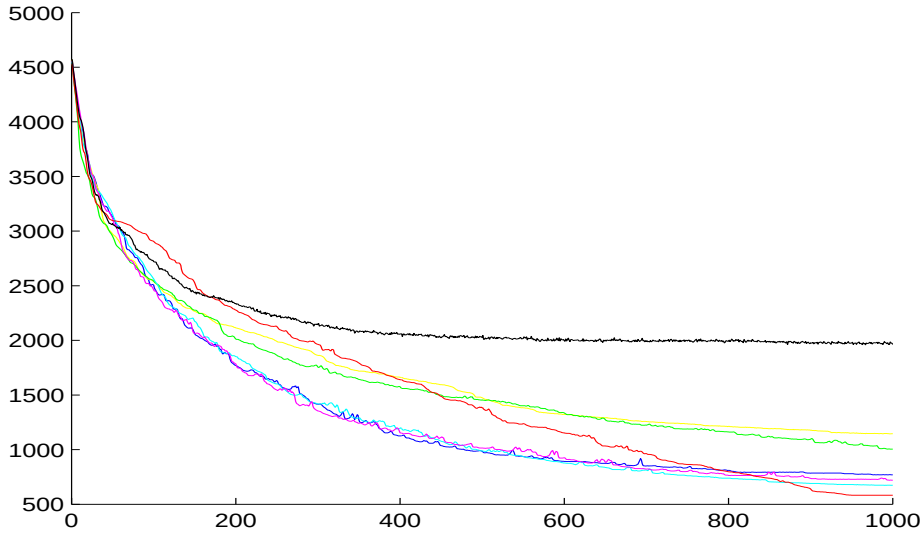
	DM		EM		ISM		SIM	
k	D	ρ	D	ρ	D	ρ	D	ρ
1	714	0,27	1301	1,31	973	0,73	634	0,12
2	695	0,23	1079	0,91	959	0,70	654	0,16
3	719	0,27	1265	1,24	989	0,75	648	0,15
4	761	0,35	1148	1,04	1004	0,78	604	0,07
5	712	0,26	1027	0,82	1008	0,79	663	0,18
6	726	0,29	1270	1,25	1002	0,78	634	0,12
7	697	0,24	1204	1,13	1017	0,80	615	0,09
8	723	0,28	1353	1,40	1031	0,83	617	0,09
9	662	0,17	1241	1,20	970	0,72	643	0,14
10	755	0,34	1396	1,48	972	0,72	623	0,10
P	716	0,27	1228	1,18	993	0,76	634	0,12

k	ZIM		IVM		SM	
	D	ρ	D	ρ	D	ρ
1	602	0,07	754	0,34	1874	2,32
2	597	0,06	769	0,36	1800	2,19
3	609	0,08	728	0,29	1895	2,36
4	581	0,03	707	0,25	1839	2,26
5	651	0,15	705	0,25	1795	2,18
6	637	0,13	736	0,30	1872	2,32
7	626	0,11	743	0,32	1829	2,24
8	616	0,09	714	0,27	1759	2,12
9	587	0,04	722	0,28	1591	1,82
10	644	0,14	735	0,30	1819	2,23
P	615	0,09	731	0,30	1807	2,20

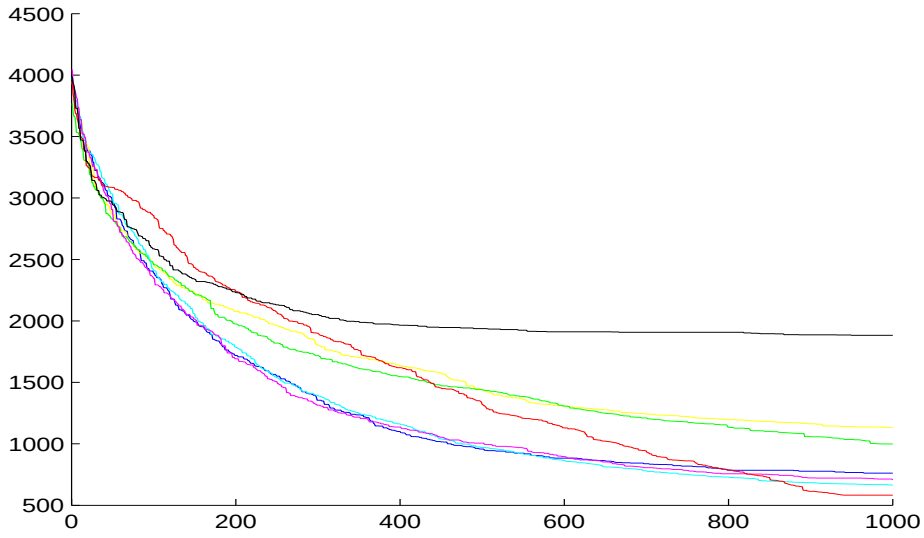
Grafické znázornění rozdílů mezi jednotlivými mutacemi:



Obrázek 30: Porovnání nejhorších jedinců. Modrá - DM, žlutá - EM, zelená - ISM, modrozelená - SIM, červená - ZIM, fialová - IVM, černá - SM.



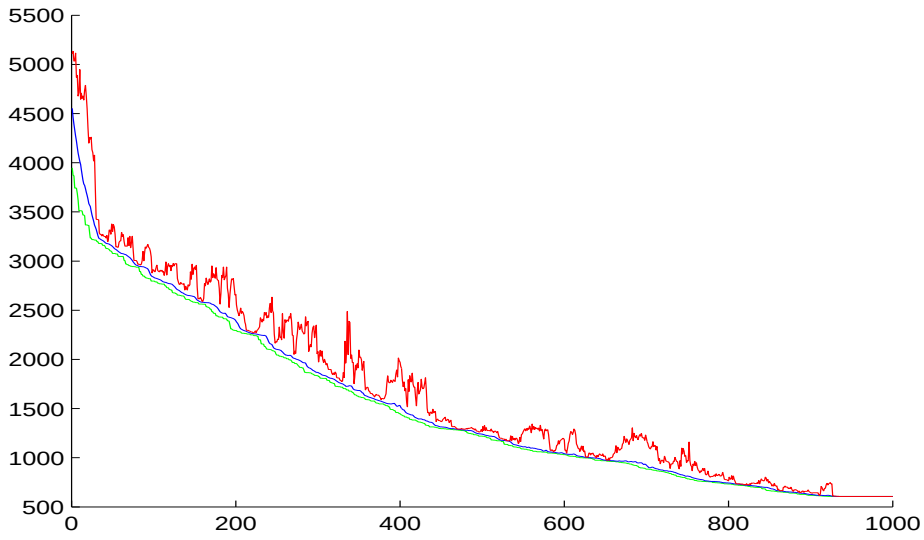
Obrázek 31: Porovnání průměru. Modrá - DM, žlutá - EM, zelená - ISM, modrozelená - SIM, červená - ZIM, fialová - IVM, černá - SM.



Obrázek 32: Porovnání nejlepších jedinců. Modrá - DM, žlutá - EM, zelená - ISM, modrozelená - SIM, červená - ZIM, fialová - IVM, černá - SM.

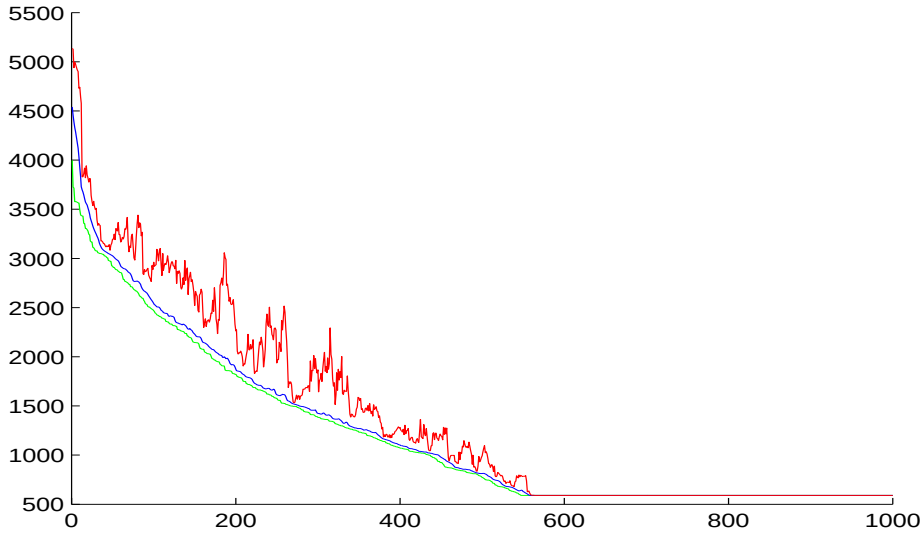
Vidíme, že mutace ZIM, jejímž použitím vždy dojde ke zkrácení cesty, dává lepší výsledky než ostatní mutace. Použitím této mutace ale dochází ke značné

homogenizaci populace, což ilustruje i následující obrázek, ve kterém je vidět, že zhruba od 930. iterace je už populace konstantní.

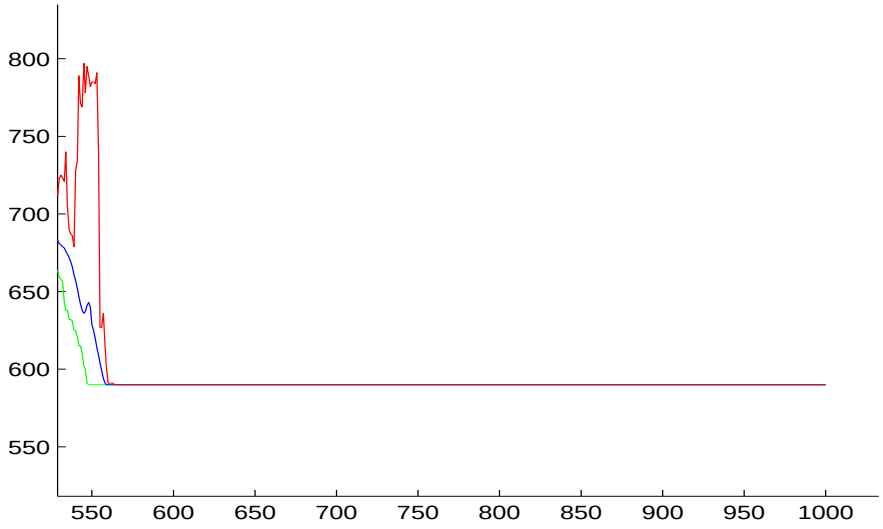


Obrázek 33: Změna nejlepšího jedince (zelená), průměru (modrá) a nejhoršího jedince (červená) při použití mutace ZIM s PM = 0,1.

Zvýšením parametru PM z hodnoty 0,1 na 0,5 dojde ještě k rychlejší homogenizaci. Na následujícím obrázku vidíme průběh, kdy se populace stala konstantní již zhruba v 550. generaci.

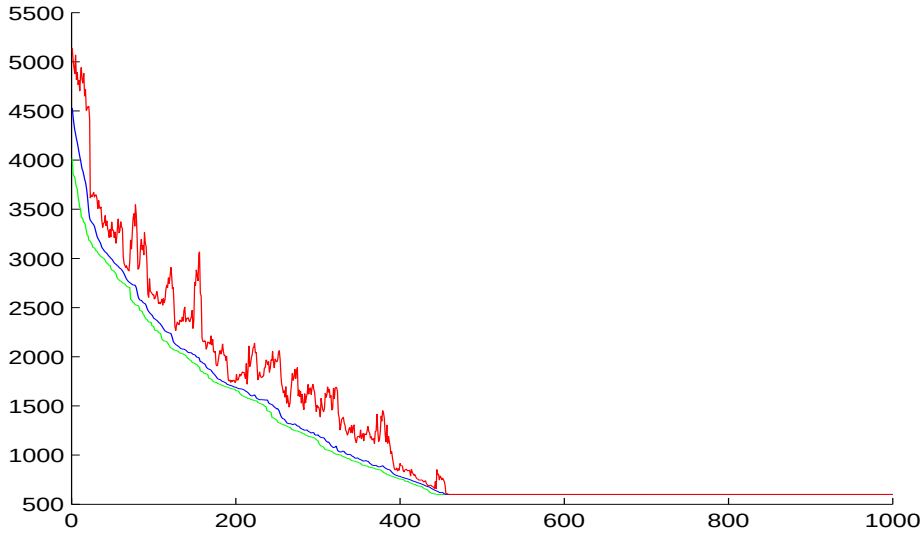


Obrázek 34: Změna nejlepšího jedince (zelená), průměru (modrá) a nejhoršího jedince (červená) při použití mutace ZIM s PM = 0,5.



Obrázek 35: Detail změny od 550. generace při použití ZIM s PM = 0,5. Nejlepší jedinec - zelená, průměr - modrá, nejhorší jedinec - červená.

Podívejme se, co se změní, pokud položíme PM = 1.



Obrázek 36: Změna nejlepšího jedince (zelená), průměru (modrá) a nejhoršího jedince (červená) při použití mutace ZIM s PM = 1.

Mutace by měla v genetických algoritmech vnést do populace větší rozmanitost a tím předcházet homogenizaci. Na příkladě mutace ZIM jsme viděli, že tato mutace sice urychluje hledání lepšího jedince, ale na rozdíl od ostatních mutací způsobuje homogenizaci populace. Proto by možná bylo výhodné tuto mutaci provádět pouze například v prvních 100 iteracích pro urychlení konvergence a poté používat jinou mutaci pro zvýšení rozmanitosti.

5.4.6 Vliv použití 2-opt

Dále jsme zkoušeli použít 2-optimalizaci v každém kroku algoritmu na celou populaci. Parametry byly:

```
popsize = 1000
pop = 1000
mutace = @mutDM
selekce = @turnajselect
PX = 0,95
PM = 0,1
```

```
initpop = 3
PO = 1
vyvoj = 2
max_iter = 100
```

Výsledky testování ukazuje následující tabulka. První ze dvou sloupečků vždy popisuje délku nalezeného řešení, druhý sloupeček počet iterací, po kterých toto řešení bylo nalezeno:

VÝPOČET S 2-OPT

	PMX		CX		OX		ERX		AP	
k	D	$iter$	D	$iter$	D	$iter$	D	$iter$	D	$iter$
1	565	100	565	100	565	100	564	27	565	100
2	564	28	564	39	565	100	564	50	564	17
3	564	9	565	100	564	15	564	38	564	41
4	564	14	564	39	565	100	564	40	564	27
5	564	27	564	77	564	21	565	100	564	55
6	564	21	565	100	564	17	564	100	564	18
7	564	22	564	78	564	26	564	40	564	16
8	564	11	565	100	564	31	565	100	564	33
9	564	18	565	100	565	100	565	100	564	34
10	564	14	564	18	564	18	564	27	565	100

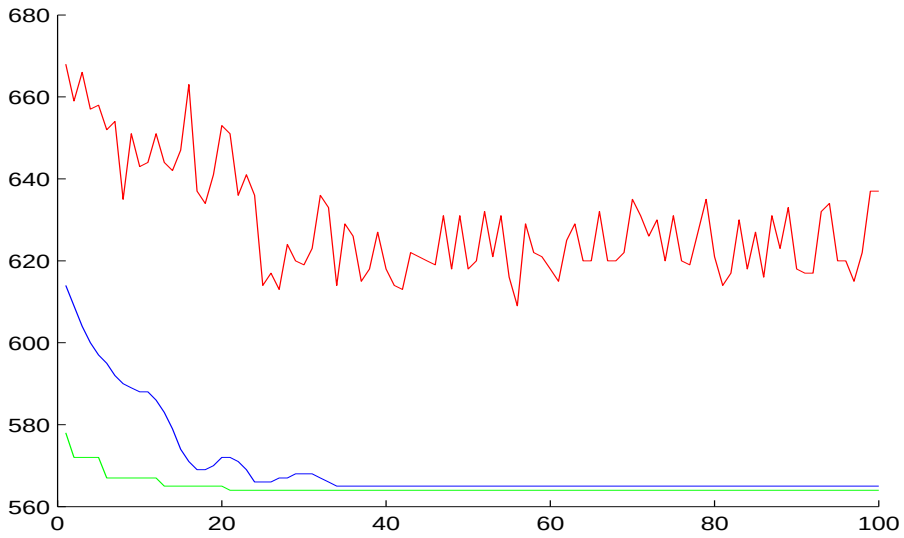
Pro srovnání uvádíme i tabulku s výsledky získanými bez použití 2-opt. Parametry:

```
popsize = 1000
pop = 1000
mutace = @mutDM
selekce = @turnajselect
PX = 0,95
PM = 0,1
initpop = 1
PO = 0
vyvoj = 2
max_iter = 1000
```

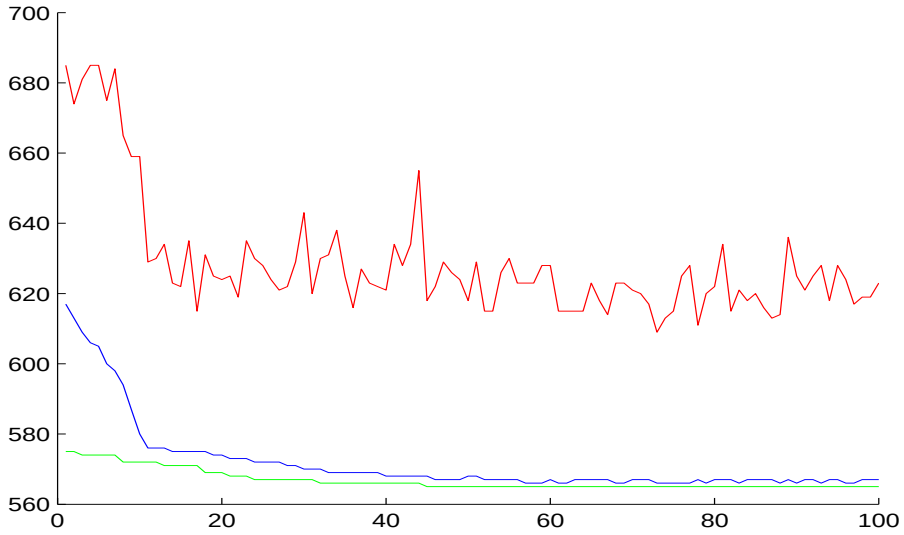
VÝPOČET BEZ 2-OPT

k	PMX		CX		OX		ERX		AP	
	D	ρ	D	ρ	D	ρ	D	ρ	D	ρ
1	714	0,27	830	0,47	729	0,29	919	0,63	914	0,62
2	695	0,23	866	0,54	733	0,30	936	0,66	878	0,56
3	719	0,27	844	0,50	748	0,33	969	0,72	917	0,63
4	761	0,35	855	0,52	703	0,25	941	0,67	932	0,65
5	712	0,26	870	0,54	722	0,28	982	0,74	892	0,58
6	726	0,29	889	0,58	699	0,24	966	0,71	897	0,59
7	697	0,24	896	0,59	711	0,26	973	0,73	864	0,53
8	723	0,28	845	0,50	725	0,29	942	0,67	854	0,51
9	662	0,17	813	0,44	719	0,27	934	0,66	845	0,50
10	755	0,34	871	0,54	673	0,19	943	0,67	877	0,55
P	716	0,27	858	0,52	716	0,27	951	0,69	887	0,57

Vidíme, že u všech křížení došlo k nalezení optimálního řešení délky 564 zhruba v polovině nebo ve většině případů a to i přesto, že maximální počet iterací byl 100 a nikoliv 1000 jako u výpočtu bez 2-opt.



Obrázek 37: Změna nejlepšího jedince (zelená), průměru (modrá) a nejhoršího jedince (červená) pro křížení PMX s použitím 2-opt.
Optimální řešení nalezeno v 21. iteraci.



Obrázek 38: Změna nejlepšího jedince (zelená), průměru (modrá) a nejhoršího jedince (červená) pro křížení CX s použitím 2-opt.
Nalezené řešení má délku 565.

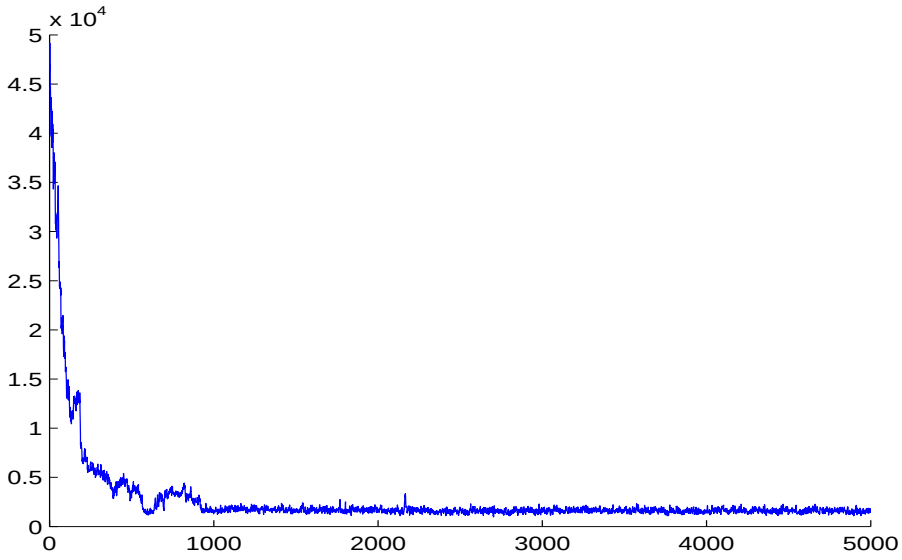
5.4.7 Základní statistické charakteristiky populace

My jsme pro všechna uvedená testování používali jako zastavovací kriterium maximální počet iterací.

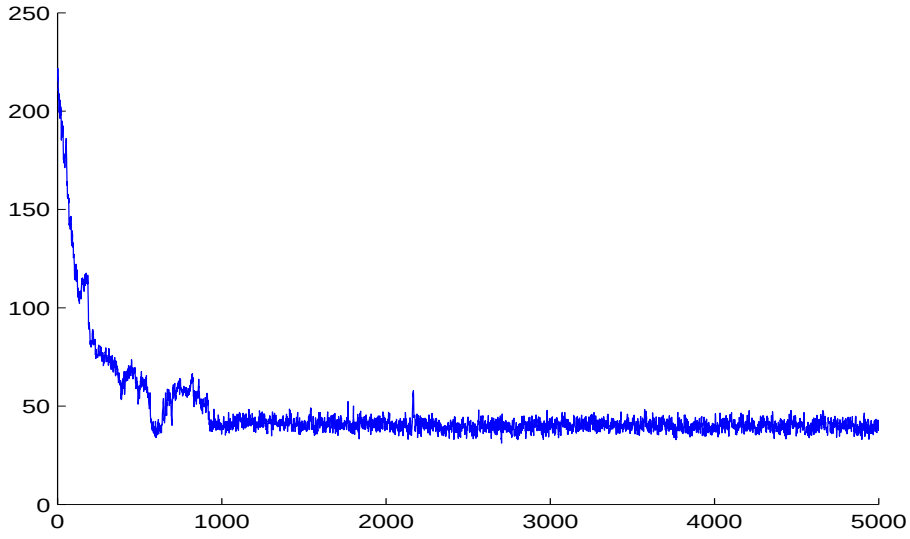
Kromě zastavovacích kriterií popsaných v kapitole 3.6 lze k zastavení výpočtu použít také toleranci založenou například na relativní změně směrodatné odchylky nebo rozptylu. Pro bližší představu, jak se tyto hodnoty během výpočtu mění, uvádíme následující grafy. Byly použity parametry:

```
popsize = 1000
pop = 1000
krizeni = @krizOX
mutace = @mutDM
selekce = @turnajselect
PX = 0,95
PM = 0,1
vyvoj = 1
```

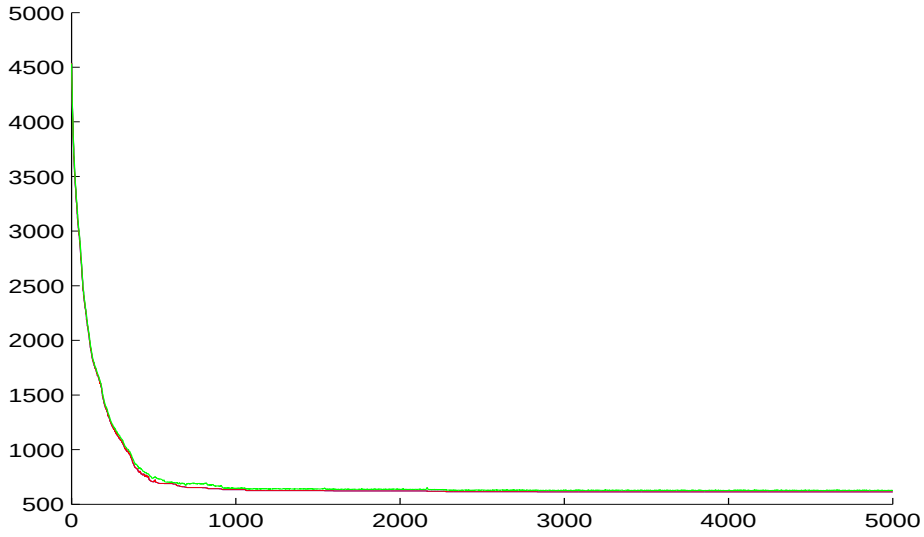
```
elita = 300
initpop = 1
PO = 0
max_iter = 5000
```



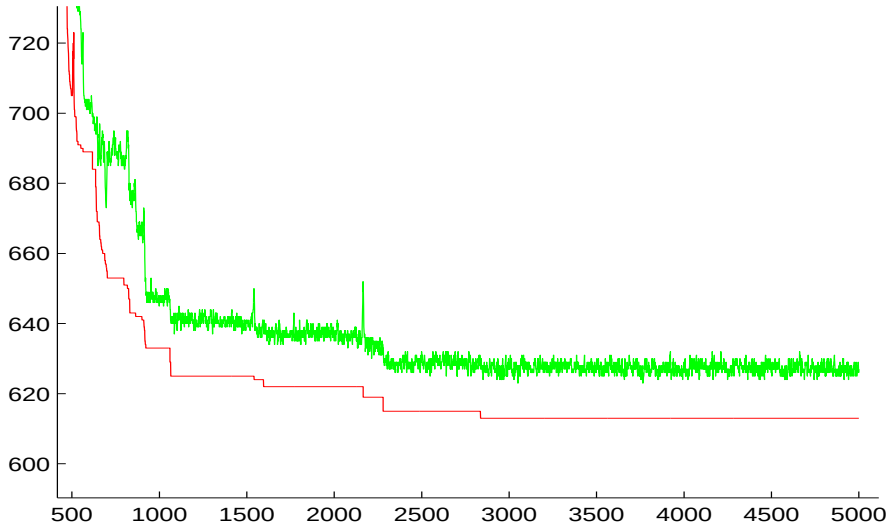
Obrázek 39: Změna rozptylu populace během výpočtu.



Obrázek 40: Změna směrodatné odchylky populace během výpočtu.



Obrázek 41: Změna mediánu (červená) a průměru (zelená) během výpočtu.



Obrázek 42: Detail změny mediánu (červená) a průměru (zelená) od 500. generace.

Z obrázku 42 je zřejmé, že medián (jeho relativní změnu) jako ukončovací kriterium použít nelze.

5.4.8 Zhodnocení

Provedli jsme testování genetického algoritmu s různými parametry a uvedli jejich výsledky, některé jsme znázornili graficky. Pokusíme se nyní výsledky zhodnotit, případně dát nějaké doporučení na volbu parametrů.

Jako první jsme se zabývali náhradovou strategií. Ukázalo se, že oběma námi zkoumanými metodami, což byly elitismus a prázdná strategie, lze dosáhnout téměř srovnatelných výsledků při vhodné volbě parametru určujícího počet nejlepších jedinců, kteří postoupí do další generace v případě použití elitismu. Experimentálně jsme zjistili, že pro naši úlohu 131 uzlů je vhodné tento parametr nastavit na hodnotu 300. Dalo by se ale očekávat, že pro úlohu větších rozměrů by byla vhodná jiná hodnota než tato.

Dále jsme pro každé křížení otestovali vliv selekce. Zkoumali jsme tři typy selekce - turnajovou, pořadovou a selekci založenou na hodnotě fitness. Ačkoli je v literatuře uváděno ([1]), že pro TSP se používá téměř výhradně turnajová selekce, zjistili jsme, že i pořadovou selekcí lze dosáhnout dobrých výsledků. V některých případech dokonce lepších výsledků než turnajovou selekcí. Selekce založená na hodnotě fitness se pro TSP neosvědčila vůbec.

Další otázkou, kterou se budeme snažit zodpovědět, je velikost populace. Pro naši úlohu 131 uzlů se ukázalo, že není příliš velký rozdíl, zda máme populaci velikost 500, 1000 nebo 2000. V případě menší populace (500) bychom po více iteracích (více než 1000) došli ke stejným výsledkům jako s populací velikosti 1000 (za 1000 iterací). Příliš malá populace ale postrádá dostatečnou rozmanitost a po 1000 iteracích dojde ke značně horším výsledkům než populace velikosti 1000, jak ukázal náš test s populací velikosti 100. Ale jak velká je příliš malá? To bude opět pro úlohu jiných rozměrů jiné.

Při zkoumání vlivu počtu iterací nebylo překvapením, že po větším počtu iterací bylo dosaženo lepších výsledků. Jak ale ukazuje obrázek 29, na kterém je znázorněn vývoj nejlepšího jedince, průměru a nejhoršího jedince během 5000 iterací, dochází k největšímu zlepšení během zhruba 1500 prvních iterací. Dále dochází už jen k velmi mírnému zlepšování nalezeného řešení.

Dále jsme zkoumali a porovnávali jednotlivé mutace. Přestože je mutace podle literatury považována za druhotnou rekombinační operaci, ukázalo se v našich výpočtech, že dosti výrazně ovlivní běh výpočtu. Ze všech mutací, které jsou uvedeny v teoretické části této práce, se mutace založená na posunutí (DM), jednoduchá inverzní mutace (SIM) a inverzní mutace (IVM) ukázaly být nejefektivnější.

Jako poslední jsme zkusili použít zlepšovací algoritmus 2-opt na všechny jedince v každé generaci. Tím došlo k velmi rychlému vylepšení a ve většině případů bylo během 100 iterací nalezeno optimální řešení. Při použití tohoto zlepšování na větších úlohách je ale nevýhoda v tom, že 2-opt výrazně zpomalí běh algoritmu. Proto by bylo vhodné použít 2-opt na jedince s nějakou menší pravděpodobností, například 0,5.

Na závěr zhodnotíme všechna použitá křížení. Na základě uvedených výsledků testování by se dalo říci, že křížení PMX, CX a OX dávají o něco lepší výsledky než ostatní. Bohužel nemůžeme potvrdit, že by křížení ERX bylo nejlepší ze všech, jak se píše v literatuře [1]. Nicméně se můžeme domnívat, že vhodnou volbou a kombinací parametrů lze u všech křížení dosáhnout výsledků srovnatelných. Na testování všech možných kombinací však v této práci není prostor, proto zůstaneme u tohoto závěru.

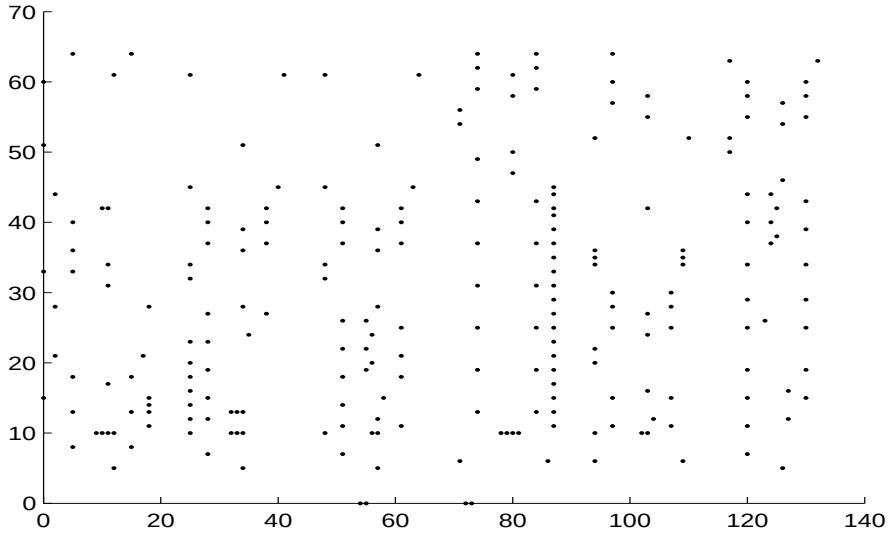
Za zmínku také stojí použití heuristik. Zajímavým výsledkem je například to, že velmi jednoduchý (a rychlý) algoritmus nejbližšího souseda dává mnohdy lepší výsledek než celý genetický algoritmus. Mohli bychom si položit otázku, proč genetickému algoritmu trvá tak dlouho (tolik generací), než se přiblíží optimálnímu řešení. Při náhodném generování počáteční populace (nepoužití zlepšovací heuristiky 2-opt) začíná algoritmus s populací jedinců vypadajících asi jako na obrázku 12. Je tedy zřejmé, že nějakou dobu trvá, než se jedinci výrazně zlepší. Je to způsobeno tím, že genetický algoritmus negeneruje lokálně optimální jedince a zaměřuje se hlavně na prohledávání celého prostoru.

U žádných výpočtů jsme neuváděli výpočetní čas, ani jsme ho nikdy nevolili jako zastavovací kriterium. A to z toho důvodu, že by tyto hodnoty mohly být

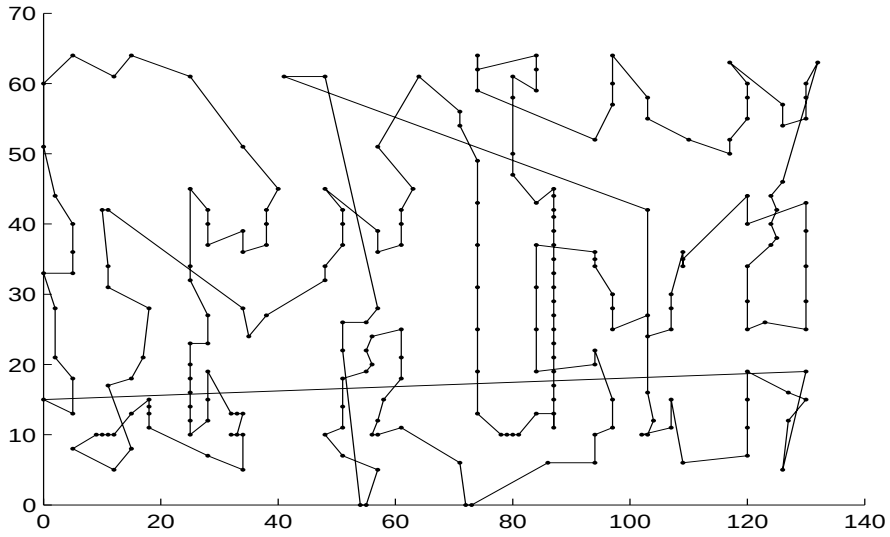
příliš zavádějící, neboť v Matlabu čas závisí do značné míry na tom, jak lze či nelze výpočet vektorizovat.

5.5 VLSI pro 237 uzlů

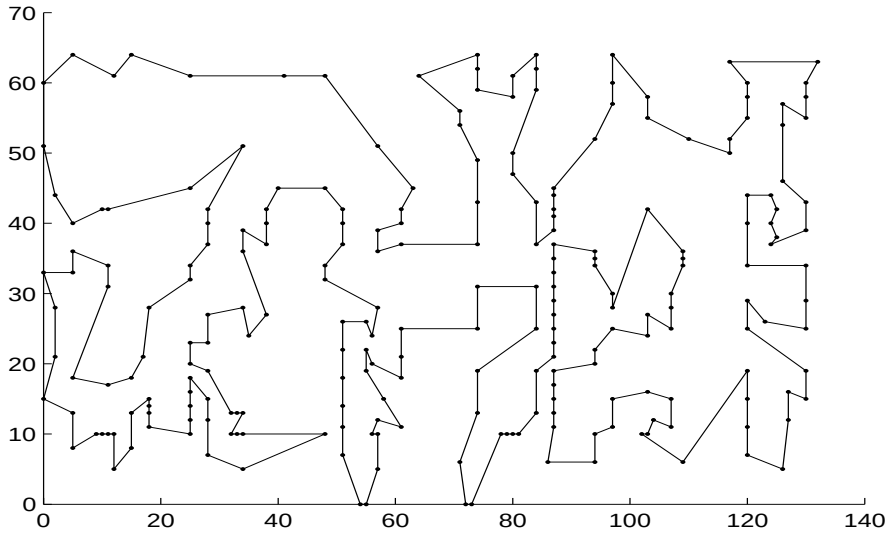
Mějme úlohu s 237 uzly (data viz [6]). Optimální trasa má délku 1019.



Obrázek 43: Rozložení 237 uzlů.



Obrázek 44: Cesta délky 1213 nalezená heuristikou nejbližšího souseda.

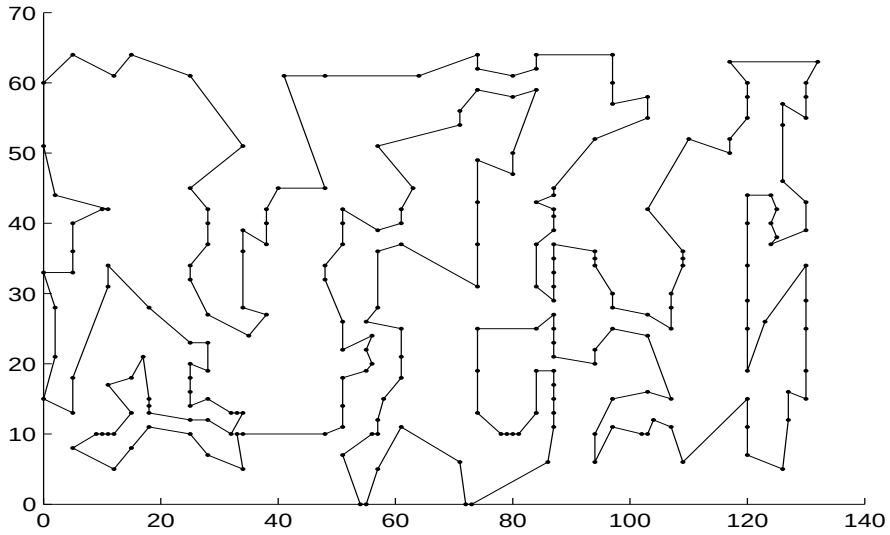


Obrázek 45: Cesta nalezená pomocí zlepšovacího algoritmu 2-opt na permutaci 1 až 237 má délku 1116.

Budeme vhodně volit (na základě předchozích závěrů) parametry a snažit se najít optimální řešení. Zvolme:

```
popsize = 1000
pop = 1000
krizeni = @krizOX
mutace = @mutSIM
selekce = @rankselect
PX = 0,95
PM = 0,1
initpop = 1
PO = 0
vyvoj = 2
```

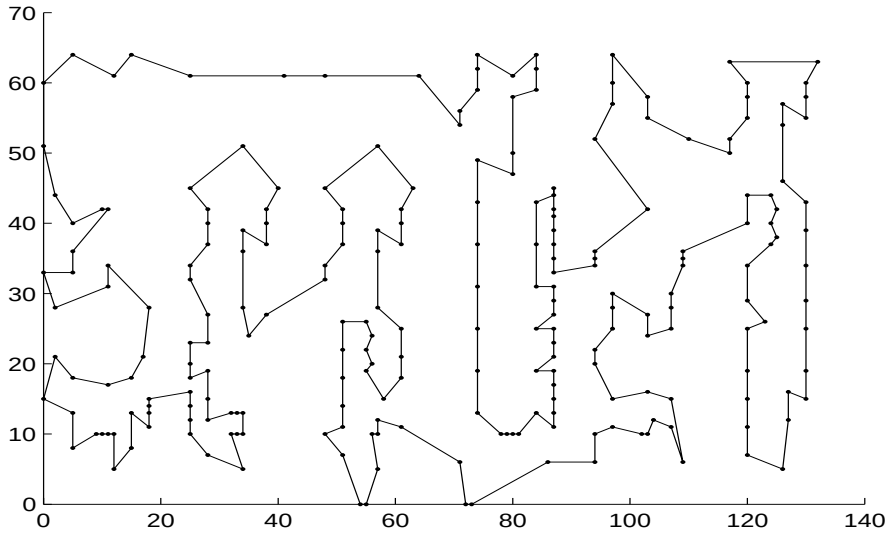
Po 1000 iteracích bylo nalezeno řešení délky 1191. Po 5000 iteracích jsme došli k řešení délky 1135, které je na následujícím obrázku.



Obrázek 46: Cesta délky 1135 nalezená po 5000 iteracích.

Dále použijeme zlepšovací algoritmus 2-opt. Parametry:

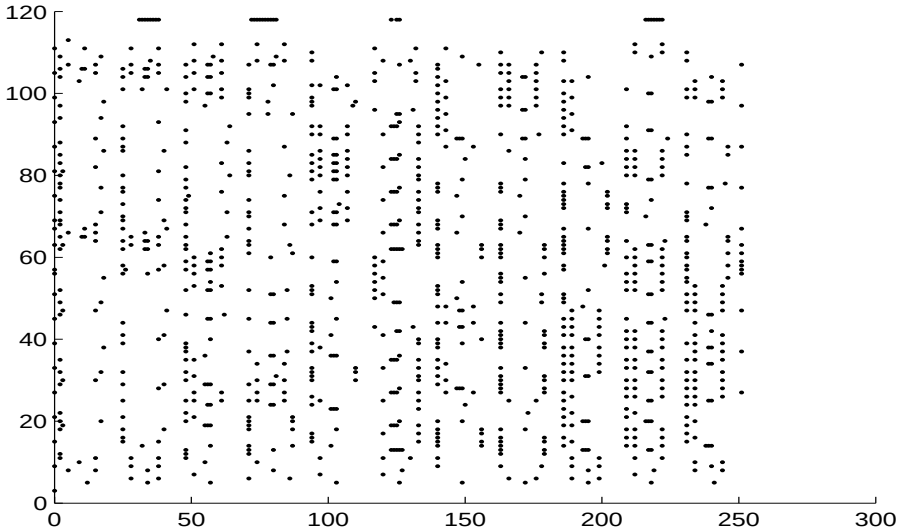
```
popsize = 1000
pop = 1000
krizeni = @krizPMX
mutace = @mutDM
selekce = @turnajselect
PX = 0,95
PM = 0,1
initpop = 3
PO = 1
vyvoj = 2
max_iter = 100
```



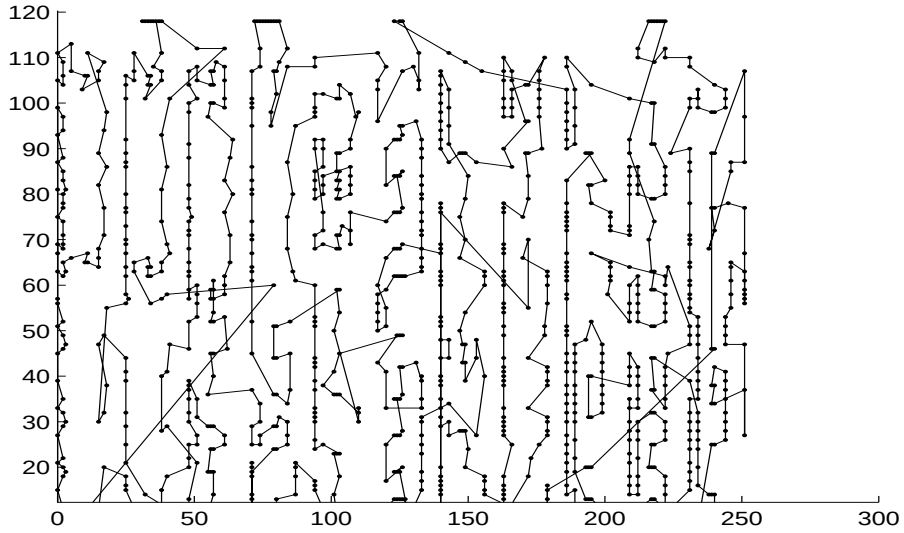
Obrázek 47: Cesta délky 1025 nalezená s využitím 2-opt zlepšovacího algoritmu v každé iteraci.

5.6 VLSI pro 1083 uzlů

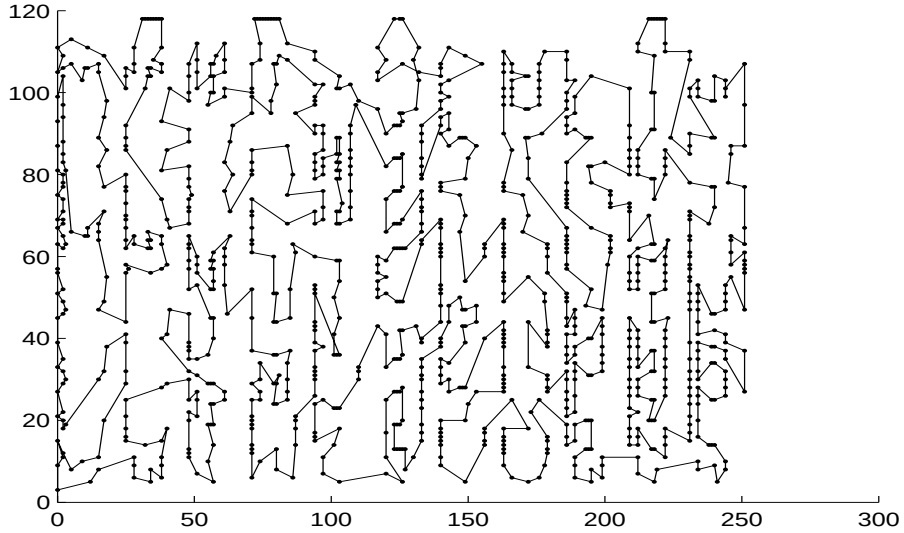
Mějme úlohu s 1083 uzly (data viz [6]). Optimální trasa má délku 3558.



Obrázek 48: Rozložení 1083 uzlů.



Obrázek 49: Cesta nalezená heuristikou nejbližšího souseda délky 4461.



Obrázek 50: Cesta nalezená pomocí zlepšovacího algoritmu 2-opt na permutaci 1 až 1083 má délku 4093.

Na základě předchozích testů se pokusíme zvolit vhodné parametry a řešit tuto úlohu. Zvolme:

```
popsize = 1000
```

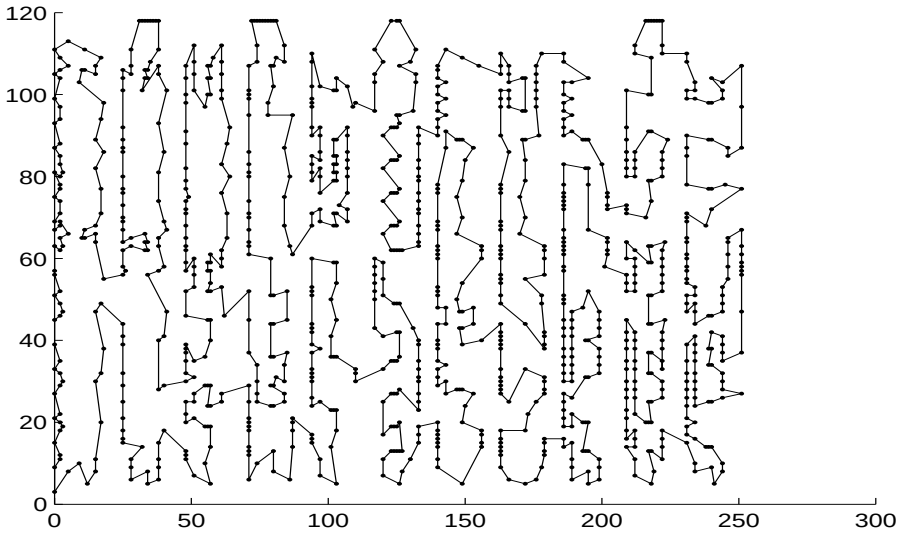
```
pop = 1000
krizeni = @krizOX
mutace = @mutDM
selekce = @rankselect
PX = 0,95
PM = 0,1
initpop = 1
PO = 0
vyvoj = 2
```

Po 1000 iteracích bylo nalezeno řešení délky 27186. 1000 iterací je tedy zřejmě pro tuto úlohu málo. Po 2000 iteracích jsme se dostali na hodnotu 16519. Po 5000 iteracích bylo nalezeno řešení délky 9269.

Dále jsme zkusili zvětšit popsize na 2000. V tomto případě bylo po 1000 iteracích nalezeno řešení délky 24888, po 2000 iteracích řešení délky 14484 a po 5000 iteracích jsme se dostali ha hodnotu 7664. Od optimálního řešení jsme pořád ještě docela daleko. Zkusíme tedy použít zlepšovací algoritmus 2-opt. Parametry:

```
popsize = 1000
pop = 1000
krizeni = @krizOX
mutace = @mutDM
selekce = @rankselect
PX = 0,95
PM = 0,1
initpop = 3
PO = 0,5
vyvoj = 2
```

Po 100 iteracích dostáváme řešení délky 3584.



Obrázek 51: Cesta délky 3584 nalezená genetickým algoritmem s využitím zlepšování populace algoritmem 2-opt .

Závěr

V teoretické části této práce jsme se nejprve seznámili s problémem obchodního cestujícího, popsali jsme základní schéma genetického algoritmu a uvedli jsme některé oblasti aplikace genetických algoritmů. Dále jsme popsali některé jednoduché heuristiky pro řešení úlohy obchodního cestujícího. Seznámili jsme čtenáře s možnými způsoby reprezentace, selekce, s metodami křížení a mutace. Vysvětlili jsme pojem náhradové strategie a uvedli nejpoužívanější zastavovací kriteria.

V části praktické jsme uvedli výsledky testování vlastního algoritmu na datech s různými parametry. Zhodnotili jsme, které parametry fungují lépe, a podali něco jako doporučení pro jejich volbu a vzájemnou kombinaci. Vše jsme doplnili grafickými ilustracemi.

Dále jsme spustili algoritmus na větších úlohách (s 237 a 1083 uzly). Při řešení těchto úloh jsme došli k docela uspokojivým výsledkům.

V této práci nebyly zdaleka prozkoumány všechny možnosti algoritmu. Dalo by se zajisté pracovat na zlepšení algoritmu i na jeho zrychlení.

Věříme, že se nám povedlo vytvořit ucelený přehled o problému obchodního cestujícího a genetických algoritmech. Taktéž v části praktické jsme naprogramovali genetický algoritmus a podali přibližný návod, jak nastavit jednotlivé parametry. Tím se nám podařilo naplnit cíl této diplomové práce.

Díky této práci jsem se seznámila s genetickými algoritmy a hlavně zlepšila své znalosti ohledně používání matematického softwaru MATLAB.

Literatura

[1] V. Mařík, O. Štěpánková, J. Lažanský a kolektiv: *Umělá inteligence (3)*, Academia, Praha, 2001.

[2] T. Weise: *Global Optimization Algorithms - Theory and Application* [online], dostupné z: http://www.it-weise.de/projects/book.pdf, [citováno 16. 2. 2012].

[3] J. Hynek: *Genetické algoritmy a genetické programování*, Grada Publishing, a.s., Praha, 2008.

[4] P. Larrañaga, C.M.H. Kuijpers, R.H. Murga, I. Inza, S. Dizdarevic: *Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators*, Artificial Intelligence Review 13: 129-170, 1999, 1999 Kluwer Academic Publisher.

[5] D.S. Johnson, L.A. McGeoch: *The Travelling Salesman Problem: A Case Study in Local Optimization*, November 20, 1995.

[6] *The Traveling Salesman Problem* [online], dostupné z: http://www.tsp.gatech.edu/, [citováno 16. 2. 2012].

[7] J. Tuháček: *Problém obchodního cestujícího* [online], dostupné z: http://www.volny.cz/jtuhacek/school/paa_tsp/index.htm,

[citováno 16. 2. 2012].

[8] S. Maslák: *Problém obchodného cestujúceho*, Bakalářská práce, Univerzita Palackého v Olomouci, 2008.

[9] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, C. Whitley: *A Comparison of Genetic Sequencing Operators* [online], dostupné z: http://citeseerx.ist.psu.edu/viewdoc/versions?doi=10.1.1.18.4329, [citováno 16. 2. 2012].

[10] *Turingův stroj* [online], dostupné z:
http://cs.wikipedia.org/wiki/Turingův_stroj, [citováno 16. 2. 2012].

[11] *Třídy složitosti* [online], dostupné z:
http://cs.wikipedia.org/wiki/Kategorie:Třídy_složitosti,
[citováno 16. 2. 2012].

[12] A. E. Eiben, G. Rudolph: *Theory of evolutionary algorithms: a bird's eye view*, Theoretical Computer Science 229 (1999) 3-9.

[13] *Genetic Server/Library* [online], dostupné z:
http://www.nd.com/products/genetic/termination.htm,
[citováno 29. 3. 2012].

[14] M. Safe, J. Carballido, I. Ponzoni, N. Brignole: *On Stopping Criteria for Genetic Algorithms*, dostupné z:
http://www.springerlink.com/content/cgt7635e3lhu3pdn/fulltext.pdf,
[citováno 29. 3. 2012].

[15] Y. S. Zhang, Z. F. Hao: *Runtime Analysis of (1+1) Evolutionary Algorithm for a TSP Instance*, dostupné z:
http://www.springerlink.com/content/0701587827191u43/fulltext.pdf,
[citováno 29. 3. 2012].