

Evoluční algoritmy a NP problémy

Bc. Zdeněk Coufal

Diplomová práce
2002



Univerzita Tomáše Bati ve Zlíně
Fakulta technologická

Vložit oficiální zadání diplomové/bakalářské práce

Děkuji vedoucímu diplomové práce Doc. Ing. Ivanu Zelinkovi Ph.D. za odborné vedení, rady a připomínky, které mi poskytoval během mé diplomové práce. Také bych chtěl touto formou poděkovat svým spolužákům, rodině a přítelkyni za podporu při studiu.

Prohlašuji, že jsem na celé diplomové práci pracoval samostatně a použitou literaturu jsem citoval.

Ve Zlíně, 02. 01. 2002

.....

Zdeněk Coufal

Abstrakt

Tato diplomová práce se zabývá problematikou Evolučních algoritmů a NP problémů. Je složena z několika částí. Náplní první části diplomové práce je přehled problematiky NP problémů. Druhá část se týká popisu již řešených příkladů užití Evolučních algoritmů na úlohu obchodního cestujícího. Ve třetí části diplomové práce je již samotné naprogramování vybraných evolučních algoritmů pro užití na úlohu obchodního cestujícího. Čtvrtá část obsahuje výpočty vybraných evolučních algoritmů a diskusi výsledků.

The diploma work deals with Evolutional algorithms and NP probléms. There are few parts in these diploma work. Objectives of the first part are NP problems. In the socond part is deskription of Evolutionary algorithms and their aplication on Travel salesman problem. There are description of programming modules and enviroment in third part. Last, fourth part deals with results evalution.

OBSAH

ÚVOD.....	6
1 TEORETICKÁ ČÁST.....	7
1.1 TRÍDY ALGORITMICKY ŘEŠITELNÝCH ÚLOH PODLE SLOŽITOSTI	7
1.1.1 Řešitelnost úloh	7
1.1.2 P-úlohy a NP-úlohy	8
1.2 OPTIMALIZAČNÍ ALGORITMY	14
1.2.1 Současný stav	15
1.2.2 Nástin principu vybraných algoritmů	16
1.2.3 Diferenciální evoluce.....	21
1.2.4 SOMA– SamoOrganizující se Migrační Algoritmus	28
1.3 JIŽ ŘEŠENÉ PŘÍKLADY ÚLOHY OBCHODNÍHO CESTUJÍCÍHO.....	33
1.3.1 Vytvořené modifikace genetických algoritmů pro úlohu TSP	33
2 EXPERIMENTÁLNÍ ČÁST.....	40
2.1 ZVOLENÉ ZADÁNÍ ÚLOHY OBCHODNÍHO CESTUJÍCÍHO.....	40
2.2 PROGRAMOVÉ PROSTŘEDKY	41
2.2.1 Vytvořené funkce	41
2.2.2 Grafické rozhraní.....	45
2.3 TESTOVACÍ ČÁST.....	48
2.3.1 Volba parametrů Diferenciální Evoluce.....	48
2.3.2 Volba Parametrů SOMA	49
2.4 VÝSLEDKY VÝPOČTŮ	51
2.4.1 Hodnocení výsledků	60
ZÁVĚR.....	62
SEZNAM POUŽITÝCH ZDROJŮ	63
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	64
SEZNAM OBRÁZKŮ	65
SEZNAM TABULEK	66
SEZNAM PŘÍLOH.....	67

ÚVOD

Tato práce se zabývá problematikou na poli NP problémů a snahou o řešení těchto mnohdy velmi složitých úloh pomocí evolučních výpočetních technik. Jako příklad typického NP problémů byla zvolena tzv. úloha obchodního cestujícího. Stručný popis úlohy: „Je dáno m měst. Některá města jsou propojena silnicemi, délka silnic je známa. Úkolem obchodního cestujícího je navštívit všechna města (v tomto případě právě jednou). Nalezněte takovou túru (permutaci měst), aby délka uražené cesty byla co nejmenší (menší než K).“

Pro tuto úlohu dosud není znám polynomiální algoritmus, který by ji řešil. Z tohoto důvodu vyplívá potřeba řešit takto koncipované úlohy jinou cestou. Jako vhodné řešení se jeví aplikace Evolučních algoritmů na tyto úlohy.

Cílem této diplomové práce je tedy najít vhodné řešení mezi algoritmy evolučního typu a dále pak provést alternativní přístup k úloze obchodního cestujícího pomocí zvolených algoritmů. V první části práce bude pozornost věnována právě vybraným algoritmům, popisu jejich principu a chování. V druhé části budou vyvíjeny programové prostředky tak, aby bylo možno zvolené algoritmy aplikovat na úlohu obchodního cestujícího. Ve třetí části již budou prováděny příslušné výpočty, na jejichž základech bude provedeno srovnání vybraných metod.

Veškeré vstupní, výsledné hodnoty i průběhy výpočtů jsou zaznamenávány do souborů a celá práce je tedy koncipována tak aby na jejich základech bylo možno v budoucnu pokračovat v hledání ideálních Evolučních metod pro úlohu obchodního cestujícího.

1 TEORETICKÁ ČÁST

1.1 Třídy algoritmicky řešitelných úloh podle složitosti

1.1.1 Řešitelnost úloh

Řada úloh „ze života“ jsou tak zvané optimalizační úlohy. To znamená úlohy, kdy hledáme řešení, které je podle nějakého kritéria nejlepší (nejlevnější, nejdražší, nejkratší apod.). Teorie složitosti, a zejména pojmy P a NP, je studována pro tak zvané rozhodovací úlohy. Rozhodovací úloha (decision task) je úloha, jejímž řešením je odpověď „ano“ nebo odpověď „ne“. Ke každé optimalizační úloze existuje její rozhodovací verze. Příkladem může být úloha obchodního cestujícího. Úloha je abstrakcí tohoto problému. Obchodní cestující má navštívit všechna města $M_1, \dots, M_n$ tak, aby ujel nejkratší možnou vzdálenost. Řešením je pořadí měst, ve kterém je obchodní cestující má navštívit.

Úloha obchodního cestujícího. Je dána množina měst $\{M_1, \dots, M_n\}$. Pro každou dvojici měst M_i, M_j je dána přímá vzdálenost $d(M_i, M_j)$ měst M_i, M_j . Trasa (tour) je posloupnost měst $M_{\pi(1)}, M_{\pi(2)}, \dots, M_{\pi(n)}$, kde π je permutace čísel $1, \dots, n$. Délka této trasy je:

$$\sum_{i=1}^{n-1} d(M_{\pi(i)}, M_{\pi(i+1)}) + d(M_{\pi(n)}, M_{\pi(1)}) \quad (1.1)$$

Optimalizační verze: Najděte trasu nejmenší možné délky.

Rozhodovací verze: Je dáno navíc číslo K . Rozhodněte, zda existuje trasa délky menší nebo rovné číslu K .

Optimalizační úloha hledá řešení, které je „nejlepší“, kdežto u rozhodovací verze je součástí instance ještě číslo K a otázka zní, zda existuje řešení, jež má hodnotu „ne horší“ než je dané číslo K .

Je zřejmé, že optimalizační verze úlohy není lehčí než její odpovídající rozhodovací verze. Najdeme-li „nejlepší“ řešení, víme, zda je aspoň tak dobré jako K . Na druhou stranu lze dokázat, že je-li rozhodovací verze řešitelná polynomiálně, pak je řešitelná polynomiálně i optimalizační verze. I zde hraje roli fakt, že jsme si vybrali třídu všech polynomů. V důkazu předchozího tvrzení je důležité, že skládáním, násobením a sčítáním polynomů dostaneme opět polynomy. [7]

1.1.2 P-úlohy a NP-úlohy

Definice: Třída P je třída všech rozhodovacích úloh U , pro něž existuje polynomiální algoritmus, který řeší U tj. algoritmus, který má složitost $O(p(n))$ pro nějaký polynom $p(n)$.

Ne všechny rozhodovací úlohy jsou ve třídě P. Například pro rozhodovací verzi obchodního cestujícího není dosud znám polynomiální algoritmus, který by řešil tuto úlohu. Existuje algoritmus, dal by se pojmenovat „algoritmus hrubé síly“, který úlohu řeší: Stačí prohlédnout všechny trasy, tj. všechny permutace čísel $\{1, \dots, n\}$ a pro každou spočítat její délku. Tím zjistíme, zda existuje trasa délky nejvýše K . Ovšem tento postup vyžaduje v nejhorším případě (totiž v případě, kdy jediná trasa kratší než K je ta poslední) čas $O(n!)$, Tím se tento algoritmus stává v praxi nepoužitelný již pro $n = 15$.

Tato úloha má však jednu zajímavou vlastnost: Jestliže pro instanci obchodního cestujícího existuje trasa délky nejvýše K a my ji náhodou odhadneme, pak na základě tohoto odhadu jsme velmi rychle schopni zdůvodnit, že správná odpověď pro danou instanci je „ano“. Neexistuje-li však pro tuto instanci trasa délky nejvýše K , žádné „rychlé zdůvodnění“ nemusí existovat. NP-úlohy jsou právě úlohy, kdy odpověď „ano“ je možné (máme-li ještě nějakou dodatečnou informaci) rychle ověřit, ale odpověď „ne“ může být složitější.

Mějme rozhodovací úlohu U . Její instance rozdělíme do dvou tříd. ANO-instance jsou ty instance, na které je odpověď „ano“; NE-instance jsou ty instance, na které je odpověď „ne“. Zhruba řečeno, NP-úloha je taková rozhodovací úloha, pro níž platí: pro každou ANO-instanci existuje objekt vhodného tvaru (nabízené řešení), pro který je v polynomiálním čase možné ověřit správnost odpovědi „ano“. Pro žádnou NE-instanci takový objekt neexistuje. Vlastnímu objektu se často říká „certifikát“ a procedura, která objekt generuje, „oraculunn“. [7]

V případě úlohy obchodního cestujícího je tímto objektem trasa délky nejvýše K .

1.1.2.1 Nedeterministický algoritmus

Existuje několik formálních definic NP-úloh. Formálně nejpřesnější je definice pomocí nedeterministického Turingova stroje. Pro účely této práce však plně postačí definice nedeterministického algoritmu. Nedeterministický algoritmus není reálný, v praxi

použitelný algoritmus; zde poslouží pouze ke klasifikaci obtížnosti rozhodovacích úloh.

Nedeterministický algoritmus (nondeterministic algorithm) má dvě fáze:

Nedeterministická fáze. Algoritmus náhodně vygeneruje řetězec symbolů s a napíše jej na předem dané místo v paměti.

Deterministická fáze. Deterministický algoritmus má na vstupu jak instanci úlohy, tak řetězec s . Na základě vstupu po konečném počtu kroků dá odpověď „ano” nebo „ne”.

Počet kroků potřebný k práci nedeterministického algoritmu je součet počtu kroků obou fází. To je počet kroků potřebných k vygenerování řetězce s plus počet kroků deterministického algoritmu deterministické fáze.

Pustí-li se deterministický algoritmus několikrát se stejným vstupem, bude algoritmus pracovat vždy stejně a dostane vždy stejný výstup. To neplatí pro nedeterministický algoritmus. Pro vstup I_u se práce algoritmu i jeho výstup může lišit; výsledek záleží na náhodně vygenerovaném řetězci s . Nedeterministický algoritmus A dá odpověď „ano” pro instanci I_u , existuje-li alespoň jeden výpočet s s výstupem „ano”.

Zhruba řečeno, na řetězec s se lze dívat jako na navržené řešení a deterministická část algoritmu ověří, že opravdu jde o řešení odpovídajících vlastností. V případě úlohy obchodního cestujícího je takovým řešením trasa délky nejvýše K . Jestliže je I_u NE-instancí, pak taková trasa neexistuje a algoritmus nemůže odpovědět „ano”. Jestliže I_u je ANO-instancí, pak taková trasa existuje a když budeme mít velké štěstí, můžeme ji uhodnout.

Definice: Třída NP je třída rozhodovacích úloh, pro něž existuje nedeterministický algoritmus pracující v polynomiálním čase.

Poznámka: Do třídy NP patří ty úlohy, pro něž je lehké ověřit správnost řešení (v případě obchodního cestujícího to je trasa odpovídající délky). Navíc toto řešení musí mít nejvýše polynomiální velikost vzhledem k velikosti vstupu. Odpověď ne, tedy zjištění, že žádné takové řešení neexistuje, může být daleko obtížnější.

Tvrzení 1

Platí $P \subseteq NP$.

Důkaz: Každý deterministický algoritmus A lze považovat za nedeterministický algoritmus. První fázi buď vynecháme, nebo vygenerujeme prázdný řetězec. Druhá fáze je pak vlastní deterministický algoritmus, který s řetězcem s vůbec nepracuje.

Otázka, zda $P = NP$ je nejdůležitější dosud nezodpovězená otázka v teoretické informatice. Vzhledem k podstatě úloh, které jsou NP a pro něž není znám polynomiální algoritmus, se má za to, že rovnost neplatí. Ovšem zatím se to nikomu nepodařilo dokázat.[7]

1.1.2.2 Redukce a polynomiální redukce úloh

Obtížnost rozhodovacích úloh se dá srovnávat. Pojem redukce zpřesňuje sdělení, že rozhodovací úloha U není těžší než rozhodovací úloha V .

Definice: Řekneme, že rozhodovací úloha U se redukuje na rozhodovací úlohu V (U is reduced to V), jestliže existuje algoritmus A , který pro každou instanci I úlohy U zkonstruuje instanci $A(I)$ úlohy V a to tak, že:

I je ANO-instancí úlohy U právě tehdy, když $A(I)$ je ANO-instancí úlohy V .

Je-li navíc algoritmus A polynomiální, mluvíme o polynomiální redukci a píšeme

$$U \propto_p V.$$

Výrok „úloha U se redukuje na úlohu V “ (označovaný $U \propto V$) říká, že úloha U není těžší než úloha V . Ano, je-li algoritmus B , který vyřeší úlohu V pak zkonstruujeme i algoritmus, který řeší úlohu U . Lze vytvořit např. takto:

Mějme instanci I úlohy U .

1. Na I aplikujeme algoritmus A , který zkonstruuje instanci $A(I)$ úlohy V .
2. Na instanci $A(I)$ aplikujeme algoritmus B , který odpoví „ano“ právě tehdy, když $A(I)$ je ANO-instancí úlohy B a to je právě tehdy, když I je ANO-instancí úlohy U .

Pokud jde o polynomiální redukci, tj. jestliže platí $U \propto_p V$ a je-li úloha V polynomiálně řešitelná, pak i úloha U je polynomiálně řešitelná. Oba kroky předchozího algoritmu vyžadují polynomiální čas. Krok 1 trvá polynomiálně dlouho, a proto velikost instance $A(I)$ je rovna $q(n)$, kde q je polynom a n je velikost instance I . Navíc i algoritmus V je polynomiální. Protože složení dvou polynomů je polynom, trvá krok 2, a tedy i celý algoritmus, polynomiálně dlouho (vzhledem k velikosti instance I).[7]

1.1.2.3 NP-úplné úlohy

Definice: Rozhodovací úloha U je NP-úplná (NP-complete), jestliže je NP-úlohou a každá NP-úloha se na U polynomiálně redukuje. Třída všech NP-úplných úloh se označuje NPC.

Jinými slovy, NP-úplné jsou ty úlohy, které jsou nejtěžší mezi všemi NP-úlohami.

Tvrzení 2

Mějme rozhodovací úlohy U , V a W . Jestliže úloha U se polynomiálně redukuje na úlohu V a úloha V se polynomiálně redukuje na úlohu W pak se U polynomiálně redukuje na W , (tj. relace $\propto_p$ je tranzitivní).

Důkaz: Toto tvrzení vyplývá z faktu, že složení dvou polynomů je opět polynom. Přesněji: protože $U \propto_p V$ existuje polynomiální algoritmus A , který pro každou instanci I velikosti n zkonstruuje instanci $A(I)$ úlohy V velikosti $q(n)$ pro nějaký polynom $q(n)$. Dále platí $V \propto_p W$, tj. existuje polynomiální algoritmus B , který pro instanci $A(I)$ úlohy V zkonstruuje instanci $B(A(I))$ úlohy W čase $p(q(n))$ pro nějaký polynom $p(m)$. Navíc I je ANO-instance úlohy U právě tehdy, když $B(A(I))$ je ANO-instance úlohy W .

Důsledek: Mějme NP-úlohy U a V takové, že $U \propto_p V$. Pak platí:

1. jestliže U je NP-úplná úloha, pak i V je NP-úplná úloha;
2. jestliže V je P-úloha, pak i U je P-úloha.

Do dnešní doby je známo více než dva tisíce NP-úplných úloh.

Nejznámější příklady NP-Úplných úloh:

Splnitelnost Booleovských formulí (satisfiability). Je dána formule v CNF (v konjunktivním normálním tvaru). Rozhodněte, zda je tato formule splnitelná, tj. rozhodněte, zda existuje pravdivostní ohodnocení logických proměnných tak, aby v něm byla daná formule pravdivá.

Fakt, že splnitelnost Booleovských formulí je NP-úplným problémem, dokázal Cook v roce 1971. Je to obsah známé Cookovy věty.

Barevnost - chromatické číslo (chromatic number). Je dán neorientovaný graf L s množinou vrcholů V a množinou hran E . Obarvení (coloring) grafu L je zobrazení $b: V \rightarrow B$, kde B je konečná množina tzv. barev, takové, že pro každou hranu $e \in E$ jsou oba její krajní vrcholy obarveny různými barvami. (Jinými slovy $b(u) \neq b(v)$, kdykoli u a v jsou krajní vrcholy nějaké hrany $e \in E$.)

Řekneme, že graf L je k -barevný (k -colorable), jestliže jej lze obarvit k barvami (tj. množina B z definice má k prvků).

Optimalizační verze. Je dán neorientovaný graf L . Určete nejmenší počet barev, kterými jej lze obarvit (tj. určete jeho barevnost).

Rozhodovací verze. Je dán neorientovaný graf L a číslo $k \geq 3$. Rozhodněte, zda L je k -barevný.

Úloha zůstane NP-úplná, i když se omezíme jen na rovinné grafy. Problém, zda graf je dvoubarevný, je ovšem polynomiálně řešitelná úloha.

Nezávislé množiny. Mějme neorientovaný graf L s množinou vrcholů V a množinou hran E . Množina vrcholů $N \subseteq V$ je nezávislou množinou (independent set), jestliže žádná hrana z E nemá oba krajní vrcholy v množině N . Jinými slovy, jestliže úplný podgraf indukovaný množinou N neobsahuje žádnou hranu.

Optimalizační verze. Je dán neorientovaný graf L . Najděte nejpočetnější nezávislou množinu grafu L .

Rozhodovací verze. Je dán neorientovaný graf L a číslo $k \geq 3$. Rozhodněte, zda v L existuje nezávislá množina o alespoň k vrcholech.

Nejkratší cesty v grafech se zápornými cykly. Je dán orientovaný graf L s vrcholy V a hranami E , dále je dáno ohodnocení c hran celými čísly (tj. $c: E \rightarrow \mathbb{Z}$).

Optimalizační verze. Najděte nejkratší cesty mezi všemi dvojicemi vrcholů (délka cesty je rovna součtu ohodnocení všech hran, ze kterých se cesta skládá).

Rozhodovací verze. Je navíc dáno číslo k . Rozhodněte, zda mezi každými dvěma vrcholy existuje cesta délky nejvýše k .

Kdybychom se omezili pouze na grafy, ve kterých neexistují cykly záporné délky, úloha by byla polynomiálně řešitelná.

Nejdelší cesty v grafech s cykly kladné délky. Úloha je obdobná jako v minulém případě, pouze hledáme cesty co největší délky.

Existence hamiltonovské cesty, kružnice. Je dán neorientovaný graf L s vrcholy V a hranami E . Kružnice (cesta) v L se nazývá hamiltonovská, jestliže obsahuje všechny vrcholy (tj. všechny vrcholy právě jednou). Je dán neorientovaný graf L . Rozhodněte, zda v něm existuje hamiltonovská kružnice (hamiltonovská cesta).

Obdobně se mohl formulovat problém existence orientované hamiltonovské cesty, nebo hamiltonovského cyklu pro orientované grafy. Také tyto problémy jsou NP-úplné.

Problém batohu (knapsack problém). Je dáno n předmětů $p_1, \dots, p_n$, každý předmět má váhu $w(p_i)$ a cenu $c(p_i)$, pro $i = 1, \dots, n$. Dále je dáno přirozené číslo K .

Optimalizační verze: Najděte podmnožinu předmětů tak, aby jejich váha nepřesáhla K a přitom jejich cena byla největší možná. (Jinými slovy, najděte podmnožinu $I \subseteq \{1, \dots, n\}$ takovou, že

$$\sum_{i \in I} w(p_i) \leq K \quad \text{a} \quad \sum_{i \in I} c(p_i) \quad (1.2)$$

je největší možná.)

Rozhodovací verze: Navíc je dáno číslo C . Rozhodněte, zda lze vybrat předměty tak, aby součet jejich vah nepřesáhl K a součet jejich cen byl alespoň C . (Jinými slovy, rozhodněte, zda existuje podmnožina $I \subseteq \{1, \dots, n\}$ taková, že

$$\sum_{i \in I} w(p_i) \leq K \quad \text{a} \quad \sum_{i \in I} c(p_i) \geq C \quad (1.3)$$

1.1.2.4 NP-těžké úlohy

Úloha U je NP-úplná, jestliže je sama NP-úlohou, a každá NP-úloha se na U polynomiálně redukuje. Vynecháme-li požadavek, aby úloha U byla ve třídě NP, dostáváme tzv. NP-těžké úlohy. Zhruba řečeno, úloha je NP-těžká, je-li aspoň tak těžká jako NP-úplné úlohy (možná i těžší).

Definice: Řekneme, že rozhodovací úloha U je NP-těžká (NP-hard), jestliže se na ni polynomiálně redukuje každá NP-úloha.

Z tranzitivity redukce úloh dostáváme, že NP-těžké jsou ty úlohy, na které se polynomiálně redukuje některá NP-úplná úloha.

V některé literatuře, se termín NP-těžká úloha používá pro optimalizační verzi NP-úplné úlohy. To v jistém smyslu odpovídá představě, že NP-těžká úloha je „aspoň tak těžká jako všechny NP-úplné úlohy“. Kdybychom totiž uměli polynomiálně rychle vyřešit optimalizační verzi některé NP-úplné úlohy, uměli bychom polynomiálně vyřešit všechny

NP úlohy. Ovšem pojem NP-těžké úlohy je širší, obsahuje i rozhodovací úlohy, které nejsou ve třídě NP. Příklad NP-těžké úlohy, která není ve třídě NP.

př.

Je dán graf L a číslo G . Rozhodněte, zda existuje přesně G různých obarvení grafu L barvami $\{1, \dots, k\}$, pro $k \geq 3$.

Je zřejmé, že pro tuto úlohu není možnost jednoduše ověřit odpověď ano. I kdyby někdo vypsál G obarvení grafu L barvami $\{1, \dots, k\}$, neměli bychom jistotu, že jiná obarvení neexistují. Na druhé straně, kdybychom měli polynomiální algoritmus pro tuto úlohu, uměli bychom polynomiálně řešit i úlohu, zda daný graf je trojbarevný. To by znamenalo, že $P = NP$. [7]

1.2 Optimalizační algoritmy

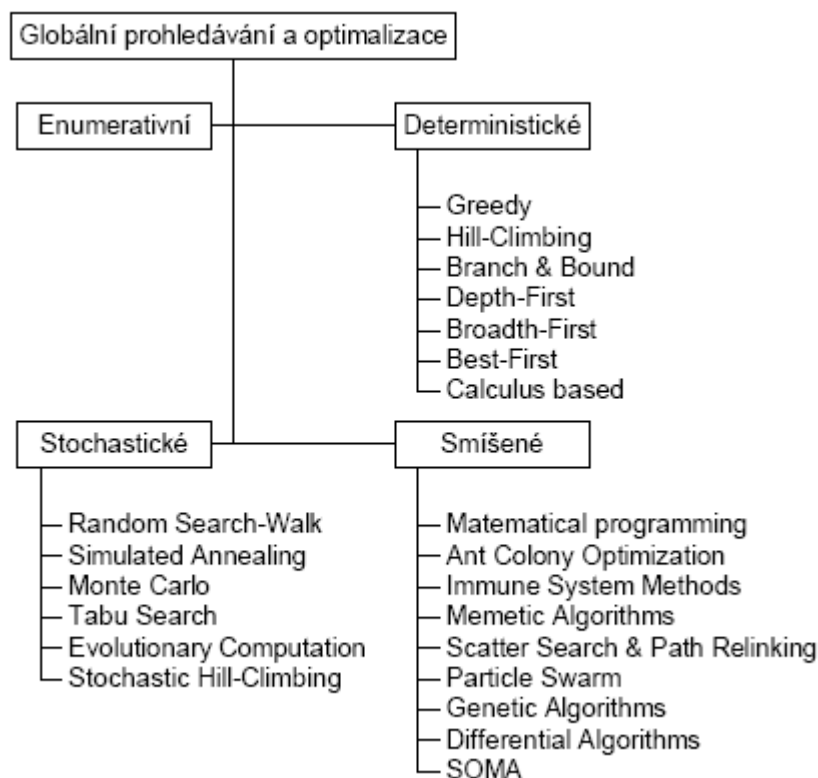
Optimalizační algoritmy jsou mocným nástrojem pro řešení mnoha problémů inženýrské praxe. Obvykle se používají tam, kde je řešení daného problému analytickou cestou silně nevhodné či nereálné. Při vhodné implementaci mohou být použity tak, že dokonce není potřeba častého uživatelského zásahu v činnosti příslušného zařízení, kde jsou použity. Většina problémů inženýrské praxe může být definována jako optimalizační problém jako např. nalezení optimální trajektorie robota, optimální tloušťky stěny tlakové nádoby, optimální nastavení parametrů regulátoru, optimální relace mezi fuzzy množinami atd. Jinými slovy, řešený problém lze převést na matematický problém daný vhodným funkčním předpisem, jehož optimalizace vede k nalezení argumentů tzv. účelové funkce, což je cílem optimalizace. Příkladů lze nalézt nespočetně. Řešení takových problémů obvykle vyžaduje práci s argumenty optimalizovaných funkcí, přičemž definiční obor těchto argumentů může být různorodého charakteru jako např. obor celočíselný, reálný, komplexní, diskrétní apod. Navíc se může stát, že pro určité subintervaly z povoleného intervalu hodnot může příslušný argument optimalizované funkce nabývat různých typů hodnot (opět celočíselný, reálný, komplexní, diskrétní apod.). Navíc v rámci optimalizace mohou být uplatněny různé penalizace a omezení nejen na dané argumenty, ale také na funkční hodnotu optimalizované funkce. Řešení takového optimalizačního problému analytickou cestou je mnohdy možné, nicméně značně komplikované a zdlouhavé. Pro úspěšné řešení takových problémů byla v posledních dvou desetiletích vyvinuta množina

velmi výkonných algoritmů, které umožňují řešit velmi složité problémy efektivním způsobem. Tato třída algoritmů má svůj specifický název a to „evoluční algoritmy“. Tyto algoritmy jsou schopny řešit velmi složité problémy tak elegantně, že se staly velmi oblíbené a používané v mnoha inženýrských oborech.

Typickým rysem pro evoluční algoritmy je, že pracují s tzv. populacemi možných řešení, jimž se říká jedinci. Tito jedinci navzájem ovlivňují svou kvalitu na základě určitých evolučních principů v cyklech, které obvykle nesou jméno „Generace“. Cílem celého evolučního procesu je nalézt nejlepší řešení.

1.2.1 Současný stav

Optimalizační algoritmy slouží k nalezení minima dané účelové funkce tak, že hledají optimální numerickou kombinaci jejich argumentů. Tyto algoritmy lze rozdělit podle principů jejich činnosti tak, jak je to naznačeno v Tab. 1 Toto rozdělení není samozřejmě jediné možné, nicméně vzhledem k tomu, že vcelku dobře vystihuje současný stav, lze jej brát jako jeden z možných pohledů na klasické, ale i moderní optimalizační metody.

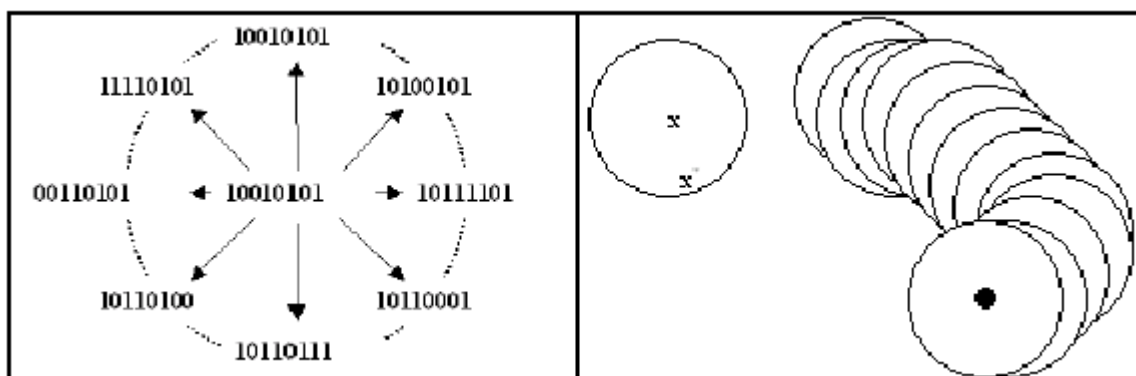


Obr. 1 Možné uspořádání optimalizačních algoritmů

Jednotlivé třídy algoritmů představují obecně způsob řešení daného problému metodami s různým stupněm efektivity a složitosti.[1]

1.2.2 Nástin principu vybraných algoritmů

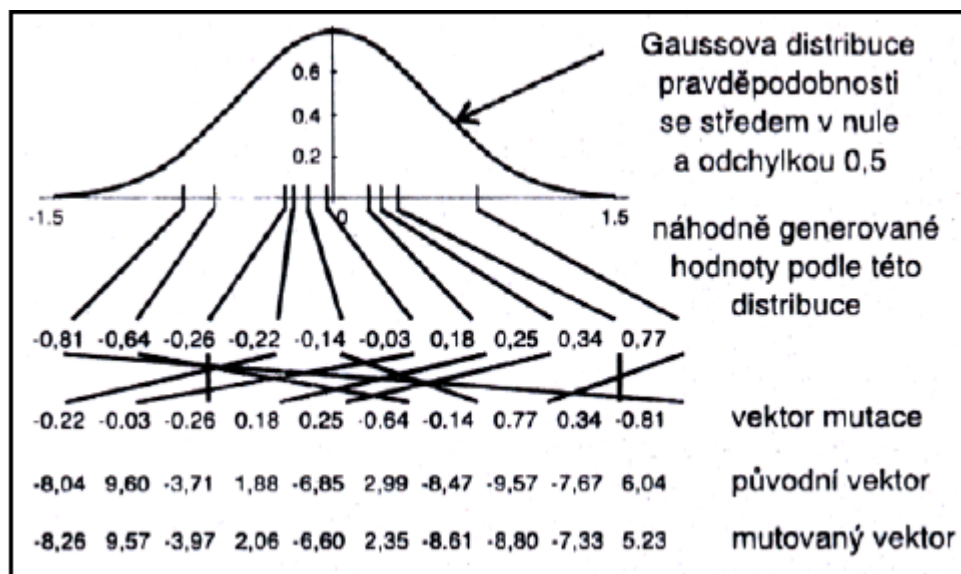
Stochastic Hill-Climbing (stochastický horolezecký algoritmus) je verze tzv. horolezeckého algoritmu obohacena o stochastickou složku. Patří mezi gradientní metody tzn., že prohledává prostor možných řešení ve směru největšího spádu. Díky své gradientní povaze velmi často uvázne v lokálním extrému. Základní verze funguje tak, že se vždy vychází z náhodného bodu v prostoru možných řešení. Pro momentálně navržené řešení se pomocí konečného souboru transformací navrhne určité okolí (Obr. 2) a daná funkce se minimalizuje jen v tomto okolí. Získané lokální řešení se pak použije jako střed pro výpočet nového okolí.



Obr. 2 Princip horolezeckého algoritmu

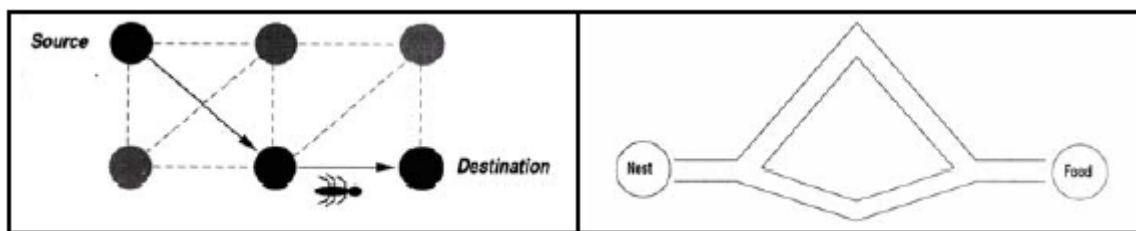
Celý proces se pak iterativně opakuje. Během procesu se zaznamenává nejlepší nalezené řešení, které po ukončení slouží jako nalezené optimum. Stochastický horolezecký algoritmus je v podstatě jen mnohonásobné zopakování standardního horolezeckého algoritmu, pokaždé z jiné, náhodně zvolené pozice. Nevýhodou tohoto algoritmu je to, že v něm může za určitých podmínek dojít k zacyklení a řešení tak uvízne v lokálním extrému .

Tabu Search (zakázané prohledávání) je vylepšená verze horolezeckého algoritmu. Jejím tvůrcem je Prof. Fred Glover z Univerzity of Colorado. Vylepšení spočívá v tom, že do horolezeckého algoritmu je zavedena tzv. krátkodobá paměť, jejímž úkolem je pamatovat si ty transformace, pomocí nichž byl vypočítán aktuální střed. To má v konečném důsledku ten efekt, že nedochází k zacyklení díky zakázanému použití těchto transformací. Odtud také plyne název „TabuSearch“. Tato metoda byla vylepšena ještě o tzv. dlouhodobou paměť, která obsahuje transformace, jenž nejsou v paměti krátkodobé, ale byly často použity. Jejich použití je pak penalizováno, což snižuje četnost jejich použití. Na rozdíl od algoritmu horolezeckého Tabu Search tak snadno neuvízne v lokálních extrémech.[8]



Obr. 3 Princip evoluční strategie

Ant Colony Optimization (ACO, „optimalizace mravenčí kolonií“). Je to algoritmus, jehož činnost je založena na chování mravenců v kolonii. Princip je následující. Necht' existuje zdroj mravenců (mraveniště) a cíl jejich snažení (potrava, viz Obr. 3). Když jsou vypuštěni, tak po nějaké době dojde k tomu, že všichni mravenci se pohybují po kratší (optimální) cestě mezi zdrojem a cílem. Tento efekt, kdy mravenci naleznou optimální cestu je dán faktem, že mravenci si svou cestu značkují feromonem. Jeho intenzita pak ovlivňuje rozhodnutí mravence. Pokud dorazí k rozcestí dvou cest vedoucích ke stejnému cíli první mravenci, pak jejich rozhodnutí, po které cestě se vydají, je náhodné. Ti, kteří zvolí kratší cestu ji označkují a při návratu jsou díky těmto značkám při rozhodování ovlivněni ve prospěch kratší cesty. Při návratu ji označkují podruhé, což opět zvyšuje pravděpodobnost rozhodnutí dalších mravenců v její prospěch. Tyto principy jsou použity v ACO algoritmu. Feromon je zde zastoupen vahou, která je přiřazena dané cestě vedoucí k cíli. Tato váha je aditivní, což umožňuje přidávat další „feromony“ od dalších „mravenců“. V ACO algoritmu je zohledněn i fakt vypařování feromonů tak, že váhy u jednotlivých spojů s časem slábnou. To zvyšuje robustnost algoritmu z hlediska nalezení globálního extrému. ACO byl použit s úspěchem na optimalizační problémy jako je problém obchodního cestujícího či návrh telekomunikačních sítí.



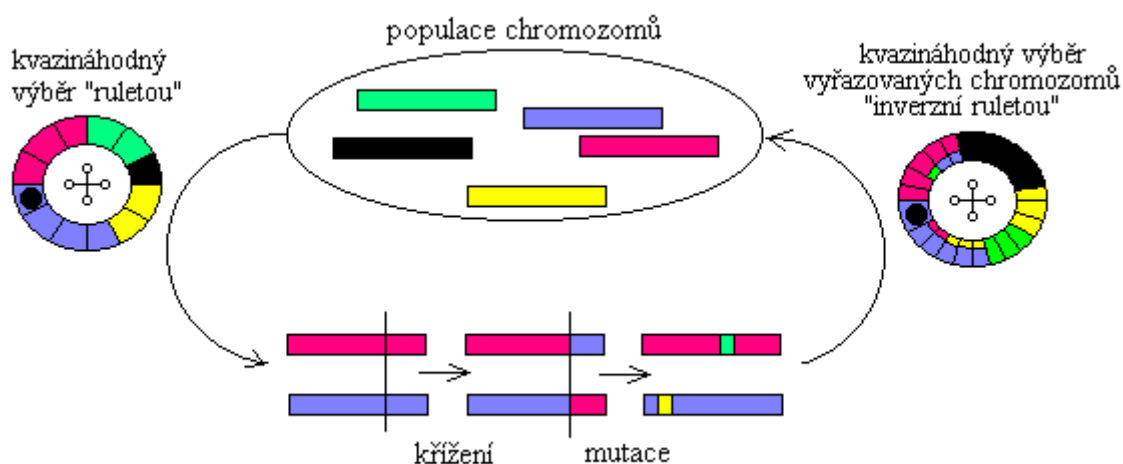
Obr. 4 ACO algoritmus, princip

Genetické algoritmy (GA = Genetic Algorithms)

Základní myšlenkou genetických algoritmů je snaha napodobit vývoj a učení nějakého živočišného druhu a takto vzniklý algoritmus použít při řešení úloh, které se vyskytují ve složitém, případně i měnícím se prostředí, kde programátor není schopen dopředu nadefinovat všechny vzniklé případy a správné reakce na ně.

V přírodě a tedy i v genetických algoritmech platí, že kvalitnější jedinci se častěji rozmnožují a také déle přežívají, proto zanechávají více potomků, kteří nesou dál část jejich genetické informace. Přesto je tento výběr ovlivněn náhodou, neboť se i kvalitní řešení vybírají k dalšímu přežití sice úměrně své kvalitě, avšak náhodně. V informatice pro tento tzv. kvazináhodný výběr podle kvality používáme zařízení nazývané "ruleta".

Situaci, kdy se většina jedinců natolik zkříží, že by další křížení produkovalo stejné jedince a tudíž nemělo smysl, brání mutace. Mutace také nabízejí možnost adaptovat populaci na měnící se prostředí, ale musí se provádět jen u malého procenta populace,



Obr. 5 GA, princip činnosti

Obr.5 : Z populace se kvazináhodně vyberou dva chromozómy, které si křížením vymění část řetězců. Výsledné chromozómy se pak ještě podrobí mutaci, která překlápí náhodně zvolený bit. Tato dvojice se vrací do populace, kde vytěsní dvojici kvazináhodně

vybraných chromozómů s malou silou. Existuje více strategií GA, základ je však ve výše uvedeném. Pro účely této práce se jeví jako nejlepší strategie: **Strategie operátorem křížení s rekombinací hran**. Operátor křížení s rekombinací hran je zpravidla označován jako **ERX** (edge recombination crossover). Je to neúspěšnější známý operátor křížení pro úlohu obchodního cestujícího. Zvláštností operátoru ERX je, že ze dvou rodičů buduje jen jednoho potomka. Celý GA se pak musí přizpůsobit tomuto faktu, neboť dva rodiče vytvoří jediného potomka a do algoritmu je tedy nutné zahrnout vhodnou náhradovou strategii. Každý úsek cesty ve vytvářeném potomku pochází z některého z rodičů. Operátor ERX k tomu využívá tzv. hranovou tabulku, která je použita, jakmile jsou zvoleni rodiče pro křížení. Tato tabulka přisuzuje ke každému městu seznam jeho sousedů vyskytujících se v obou rodičích.

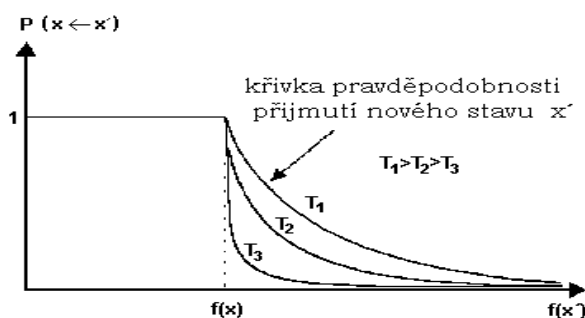
Simulované žíhání(SA)

Počátkem 80. let dostali Kirkpatrick, Gelatt, Vecchi a Černý nápad, že problém hledání globálního minima funkce může být realizovatelný podobným způsobem jako žíhání tuhého tělesa, a nazvali ho metodou simulovaného žíhání.

Uvažujme hypotetický systém popsany n-rozměrným stavem $x = (x_1, x_2, \dots, x_n)$, kde x_i patří do nějakého intervalu. Stavy tohoto systému jsou ohodnoceny funkcí f , která přiřadí každému hodnotu $y = f(x)$. Naší úlohou je najít takový stav $x_{\min}$, ve kterém nabývá funkce f globálního minima. Nechť x je přípustný stav systému, ten přejde do nového stavu x' , formálně $x' = \text{Opert}(x)$, kde Opert je "operátor" poruchy - perturbace. Otázka, zda bude nový stav akceptován do dalšího procesu, se řeší tzv. Metropolisovým kritériem, resp. Metropolisovým algoritmem, který určuje pravděpodobnost nahrazení starého stavu novým. Celý tento vztah lze zapsat:

$$P(x \leftarrow x') = \begin{cases} 1 & \text{pro } f(x') \leq f(x) \\ \exp(-|f(x') - f(x)|/T) & \text{pro } f(x') > f(x) \end{cases} \quad (1.3)$$

kde T je parametr interpretovaný jako "teplota".



Obr. 6 křivka pravděpodobnosti přijetí nového stavu

V případě, že nový stav x' má menší nebo stejnou funkční hodnotu jako původní stav x , potom provedeme záměnu stavu x stavem x' . V opačném případě je nový stav x' akceptován s pravděpodobností $0 < P(x \leftarrow x') < 1$. Necht' $random$ je náhodné číslo z intervalu $(0,1)$; potom nový stav je akceptován, pokud $random < P(x \leftarrow x')$, jinak se proces opakuje s původním stavem x .

Hodnota parametru T podstatně ovlivňuje hodnotu pravděpodobnosti $P(x \leftarrow x')$ pro případ $f(x') > f(x)$. Pro velké hodnoty T je tato pravděpodobnost blízká jedné (tj. přijmou se téměř všechny nové stavy), ale blíží-li se T k nule, potom pravděpodobnost přijetí je velmi nízká, téměř nulová.

Při vhodném nastavení počáteční "teploty" T_{max} (větší hodnota) metoda simulovaného žíhání prohledává nejprve prostor řešení D silně stochasticky, přijímá i stavy s horším ohodnocením než má současné řešení. Tato vlastnost simulovaného žíhání je velmi důležitým rysem této metody, protože poskytuje možnost snadno se dostat z lokálního extrému a prohledat i jiné oblasti v prostoru řešení. Tahle možnost se pak postupně snižuje úměrně se zmenšováním hodnoty T . Pro malé T připustí Metropolisovo kritérium akceptování už jen lepšího řešení než je současné.

K podrobnějším studiím byly po dohodě s vedoucím Diplomové práce zvoleny algoritmy Diferenciální evoluce a SOMA. Oba tyto algoritmy byly v praktické části programovány a přizpůsobeny pro řešení tzv. Úlohy obchodního cestujícího.[8]

1.2.3 Diferenciální evoluce

Diferenciální evoluce je poměrně nový typ evolučního algoritmu, který poprvé použili Ken Price a Rainer Storm při řešení problému s Čebyševovými polynomy. Jeho schéma je dost podobné algoritmům genetickým, s nimiž má několik společných rysů. Má však i několik rysů, kterými se od nich odlišuje a které tento algoritmus zvyhodňují a to:

- 1) **Jednoduchost.** Tento algoritmus lze naprogramovat velmi jednoduše např. i v takových prostředcích, jakými jsou tabulkové procesory.
- 2) **Hybridnost** čísel, se kterými diferenciální evoluce pracuje. Lze dohromady bez jakéhokoliv problému kombinovat čísla typu Integer, Real nebo i jen vybranou množinu čísel, jako např. $\{-5, 2, 8, 55, 3, 100\}$.

- 3) **Používání dekadických čísel.** Dané decimální hodnoty se nemusí převádět do binárního kódu, který je běžně používán u genetických algoritmů. Převodem do binárního kódu vzniká totiž zkreslení daného čísla (binární řetězec má omezenou délku). Navíc u binárního zápisu mohou mutace způsobit skokovou změnu čísla, což nemusí mít dobrý dopad na průběh evoluce. Např. čísla 15, 16 a 17 jsou reprezentována 01111, 10000 a 10001. Přejít z 15 na 16 znamená inverzi všech pěti bitů, tj. 100% úspěšnou mutaci. Přejít z 16 na 17 však vyžaduje už jen mutaci jednoho jediného bitu. Tyto "nerovnoměrnosti" se sice dají odstranit tzv. Grayovým kódováním, nicméně práce s reálnými čísly je stále výhodnější.
- 4) **Rychlost.** Díky své jednoduchosti a práci přímo s dekadickými hodnotami je velmi rychlý.
- 5) **Nezávislost křížení na kvalitě rodičů.** U genetických algoritmů se kříží na základě kvality rodičů, což může vést ke konvergenci do lokálního minima. Diferenciální evoluce vybírá "rodiče" náhodně. Díky tomu téměř vždy nalezne globální extrém. Někdy se této schopnosti - efektu říká "diverzibilita".
- 6) **Schopnost nalézt extrém i u funkcí, které jsou v grafickém slova smyslu ploché a extrém je jen "dírou" v této rovině.**
- 7) **Schopnost dát vícenásobné řešení.** Výsledkem evoluce je nejlepší jedinec - jedno řešení. Pokud jsou však z poslední populace vybráni např. tři nejlepší jedinci, pak představují tři různá řešení problému. S odstupňovanou kvalitou ovšem.
- 8) **Účinnost při řešení nelineárních problémů s ohraničením** (lze spojit s penalizační funkcí).

1.2.3.1 Princip činnosti

Celá generace (cyklus) pracující s populací a jejími jedinci se člení do následujících aktivit (viz Tab. 1):

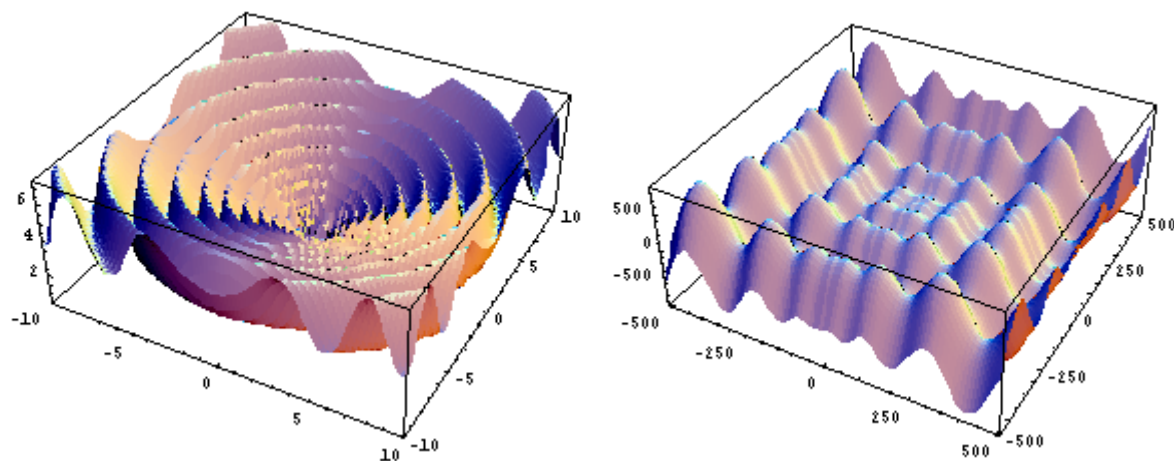
1. **Stanovení parametrů** - jde o parametry, které určují chod celé evoluce. Jsou to parametry **F** - mutační konstanta (0-2), **CR** - práh křížení (0-1), **NP** - počet jedinců v populaci (prozatím "nejrozšířenější" metoda je $NP=D*10$), **D** - rozměr jedince - jsou to v podstatě argumenty účelové funkce). Dále je nutné nadefinovat prototyp jedince (Specimen) - z jakých typů čísel se budou skládat jedinci např. {(Real,-10,15),(Integer,-1,1)} atd.

2. **Tvorba populace** - populace se tvoří vygenerováním množiny jedinců (matice) podle prototypového vektoru. U každého jedince se musí počítat s jedním prvkem navíc - hodnota funkcionálu (dále už jen HF)
3. **Započetí cyklu generace** - během každé generace se provádí ještě cyklus, který zabezpečuje postupné evoluční šlechtění každého jedince z populace. V tomto cyklu se postupně vybírá jeden jedinec (aktivní jedinec, cílový vektor) za druhým až do konce populace (tím je ukončena jedna generace) a pro každého z nich je proveden evoluční cyklus.
4. **Evoluční cyklus** - v tomto cyklu se náhodně zvolí tři další **různé** vektory (jedinci) z populace. První dva se od sebe odečtou a získá se tak tzv. "**diferenční vektor**". Ten se vynásobí mutační konstantou, která jej tím pádem změní (zmutuje), a získá se "**váhovaný diferenční vektor**". Ten se přičte k třetímu náhodně vybranému vektoru a získá se tzv. "**šumový vektor**". Poté se připraví tzv. "zkušební vektor" a z cílového a šumového vektoru se bere postupně jeden prvek za druhým (první z obou, druhý z obou...) a pro takto vybranou každou dvojici se generuje náhodné číslo v rozsahu 0-1 a porovnává se s konstantou CR. Pokud je toto číslo menší než CR, pak se do příslušné pozice v tzv. "**zkušebním vektoru**" umístí prvek z vektoru šumového a v opačném případě z vektoru cílového. Tak získáme zkušební vektor, jehož HF porovnáme s HF cílového vektoru. Na pozici cílového vektoru v nové populaci je vybrán ten vektor - jedinec, který má HF lepší. V případě Tab. 1 je hledáno minimum, a proto je na pozici cílového vektoru v nové populaci umístěn vektor zkušební. Tím je zajištěno, že se do nové generace dostanou jedinci s lepšími vlastnostmi. Vybere se druhý jedinec – cílový vektor a celý evoluční cyklus se opakuje až do vyčerpání populace. Tak vznikne nová generace potomků (jedinců).
5. **Vyhodnocení** – Celý proces generací se opakuje, dokud není vyčerpán zadaný počet generací nebo dokud HF nedosáhne zadané hodnoty. Během každé generace se uschová HF nejlepšího jedince do vektoru historie, který po ukončení znázorňuje průběh evolučního procesu.

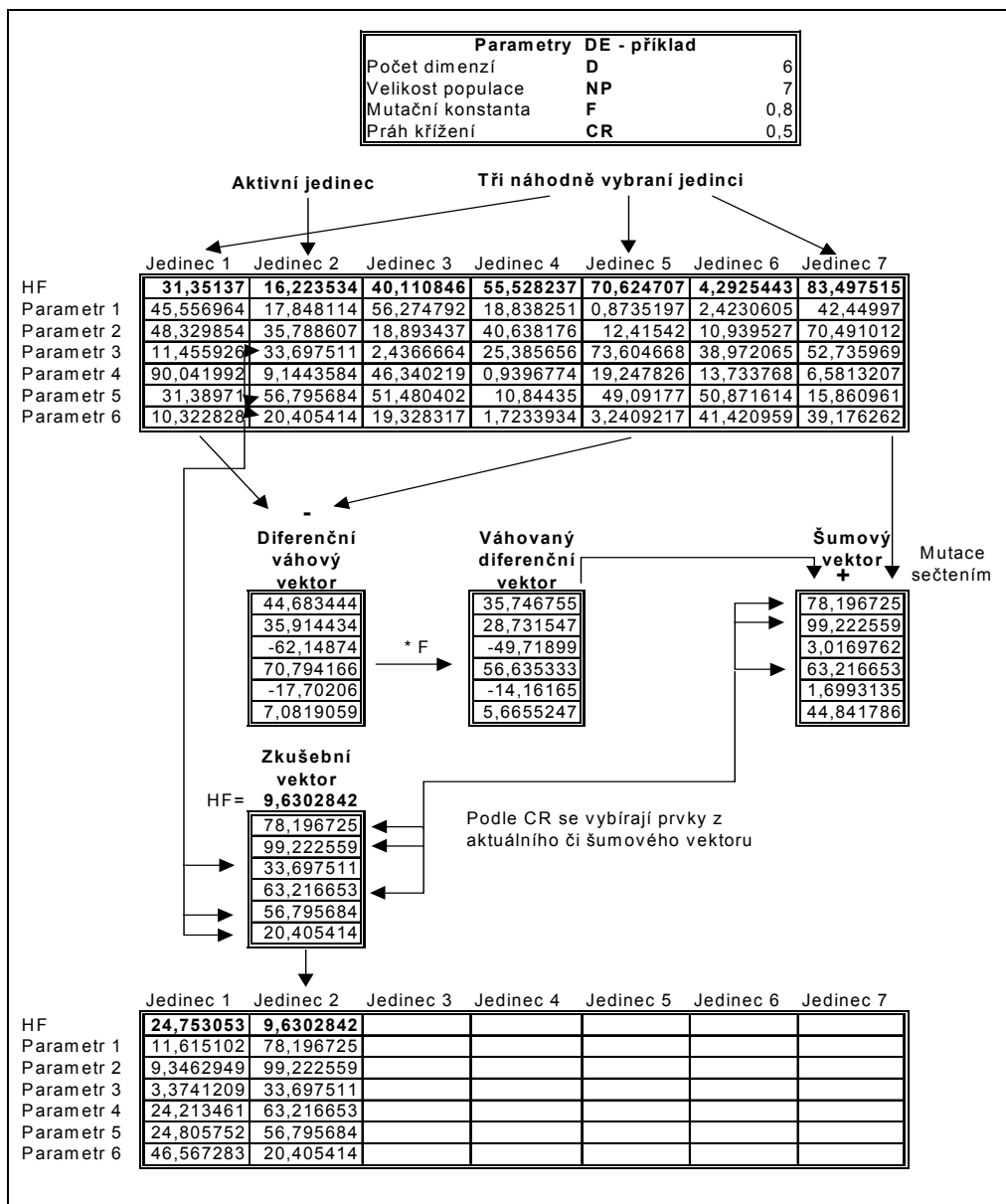
Celý evoluční proces je schématicky znázorněn v Tab. 1. Parametry jsou voleny jako pevné, nicméně lze je během evoluce měnit podle toho, jak kvalitně probíhá. Kvalitu a průběh šlechtění lze ovlivnit mnoha faktory, a to:

1. **Nastavením konstant** – kombinace konstant může mít vliv na průběh evoluce. Např. nízká hodnota CR může evoluci zbrzdit (bude vznikat méně nových jedinců).

2. **Velikostí populace** – při malé populaci bude horší výběr, při velké bude potřeba více času na její postupný průchod.
3. **Počtem generací** – při zadaném malém počtu generací může evoluce skončit dříve než nalezne extrém. Obecně platí, že čím více, tím lépe.
4. **Definicí HF** – pokud je nevhodně či špatně nadefinována HF, může evoluce probíhat pomalu nebo vůbec ne.
5. **Definicí intervalu** – Při evoluci je vždy definován interval, ve kterém se má evoluce držet. To ze zcela jednoduchého důvodu. Pokud poběží evoluce na funkci, jaká je na Obr. 1a) a nebude omezená intervalem, pak bez problémů nalezne extrém – extrémy se „koncentrují“ okolo hodnoty 0,0 a tudíž evoluce bude konvergovat k těmto hodnotám. Pokud se však použije např. funkce z Obr. 1b), pak se evoluce zastaví až v nekonečnu.



Obr. 7 Ukázkové funkce a, b.



Tab. 1 Princip diferenciální evoluce

Diferenciální evoluce má několik zvláštností. Ke tvorbě nového jedince je potřeba ne dvou, ale celkem tří rodičů, kteří vytvoří potomka, jenž nakonec soupeří o místo v nové populaci ne s některým ze tří rodičů, ale s aktuálně zvoleným jedincem, pro nějž byl potomek vytvořen. V procesu tvorby nového potomka je uplatněna náhoda dvojím způsobem. Je to pomocí křížící konstanty CR (tak jako v jiných evolučních algoritmech) a pomocí tzv. šumového vektoru. Díky tomu je diferenciální evoluce poměrně robustní algoritmus s poměrně vysokou diverzibilitou.

1.2.3.2 Stagnace

Diferenciální evoluce nemá jen samé výhody. Jednou z „nevýhod“ je tzv. stagnace, která je vlastní tomuto algoritmu. Stagnace je jev, při kterém dochází k zastavení vývoje hodnoty funkcionálu směrem k nižším hodnotám ještě před dosažením globálního extrému. K tomuto jevu dochází bez zřejmých důvodů a liší se od předčasné konvergence k suboptimálnímu řešení tak, že populace stále zůstává diverzibilní zatímco optimalizační proces dále nepokračuje. Obecně je známo, že v případě použití evolučních algoritmů či algoritmů jim příbuzných dochází k předčasné konvergenci k suboptimálnímu řešení, a to tehdy, jestliže

- Optimalizační proces zavedl populaci do lokálního extrému daného funkcionálu
- Populace ztratila diverzibilitu
- Optimalizační proces probíhá pomalu nebo vůbec neprobíhá

Na rozdíl od těchto obecných poznatků optimalizační proces běžící pod diferenciální evolucí vykazuje za určitých podmínek stagnaci navzdory faktu, že

- Populace nezkonvergovala do lokálního extrému
- Populace neztratila diverzibilitu
- Optimalizační proces dále neprobíhá navzdory faktu, že vznikají nové jedinci

Tato situace je nazývána stagnací. Stagnace je jev, který může nastat při nevhodné velikosti populace a řídicích parametřích v diferenciální evoluci. Tyto poznatky je tudíž dobré využít při návrhu populace a řídicích parametřích u diferenciální evoluce.

Křížící konstanta CR

Faktorem, který se podílí na tvorbě nových řešení, je konstanta křížení. V případě jejího nastavení na $CR=1.0$ opět nenarůstá počet nových řešení, protože se stává novým kandidátem přímo šumový vektor. Nicméně nepatrná změna např. na $CR=0.99$ již rapidně zvyšuje možnost tvorby nových kandidátů – vektorů. Z tohoto důvodu je doporučeno nenastavovat CR na hodnotu 1, ale těsně pod ni v případě, kdy je potřeba silné mutace.

Velikost populace NP

Větší populace silně redukuje riziko stagnace a předčasné konvergence. Počet možných nových řešení rychle vzrůstá s velikostí populace NP. Na druhou stranu se tím ale zpomaluje rychlost nalezení globálního extrému.

Stagnace tedy závisí na počtu nových zkušebních řešení, ale také na jejich schopnosti získat místo v nové populaci. Stručně řečeno tedy závisí na

- Velikosti populace NP
- Mutaci F
- Křížicí konstantě CR
- Aktuální populaci
- Funkcionálu

Větší populace snižuje riziko. Obvykle stačí $NP > 20$, $F = 1.0$ stejně jako u $CR = 1.0$ se nedoporučuje z důvodů vysokého rizika vzniku stagnace. V případě potřeby hodnot blízkých jedné je doporučeno používat 0.99 apod. Zanedbá-li se problematika stagnace, pak na základě mnoha simulací lze tvrdit, že DE je velmi robustní algoritmus jak z hlediska nalezení globálního extrému. Rovněž nastavení jeho parametrů citelně nesnižuje schopnost nalézt globální extrém. Co citelně ovlivňuje je pouze doba potřebná k nalezení globálního extrému.

1.2.3.3 Diferenciální evoluce – závěr

Tento algoritmus i přes své mládí byl použit s velkým úspěchem na poměrně bohatou množinu problémů a byly na něm provedeny komparativní studie s již existujícími algoritmy jako je například simulované žihání, genetické algoritmy, dvouúrovňové paralelní evoluční strategie, evoluční programování apod. Tento algoritmus se ve všech testech osvědčil výborně a lze ho tedy považovat za vhodný k řešení mnoha komplikovaných problémů.

V dalším byl použit na komparativní studii s algoritmem SOMA při řešení úlohy obchodního cestujícího.[1]

1.2.4 SOMA– SamoOrganizující se Migrační Algoritmus

SOMA je algoritmus, jehož činnost je založena na geometrických principech. Vzhledem k tomu, že pracuje s populacemi podobně jako např. genetické algoritmy a výsledek po jednom migračním kole je totožný s diferenciální evolucí, lze jej řadit např. mezi evoluční algoritmy navzdory faktu, že během jeho běhu nejsou vytvářeni noví potomci, jak je tomu u jiných evolučních algoritmů. Pokud by se hledala biologická analogie, pak by se dal tento algoritmus spíš přirovnat k tvorbě potomků ve stádu (jelen a laně...) než ke klasickému výběru rodičů z populace.

1.2.4.1 Princip činnosti

SOMA je založen na napodobení chování skupiny inteligentních jedinců, kteří kooperují na společném úkolu, což se v konečném důsledku projevuje synergetickými efekty. Pro názorné ukázky takové činnosti z reálného světa se nemusí chodit daleko. Je to např. spolupráce skupiny predátorů při lovu, nevědomá, nicméně synergetické efekty přinášející spolupráce mnoha příslušníků některých druhů jako jsou mravenci či včely apod. Synergetickým efektem, který je atributem tohoto algoritmu, je samoorganizace jedinců v populaci, která v konečném důsledku vede v drtivé většině případů k nalezení globálního extrému daného funkcionálu. Samoorganizace jedinců vzniká jejich vzájemným ovlivňováním během tzv. migračních kol, která jsou jakýmsi ekvivalentem generací známých z klasických evolučních algoritmů.

Konkrétní činnost SOMA lze popsat následujícími kroky:

1. Nadefinování parametrů a jejich význam - SOMA je řízen a ukončován množinou parametrů, které musí být nadefinovány ještě před vlastním během algoritmu. Jsou to Migrations, AcceptedError, NP, Mass, Step, PRT a Specimen. První dva parametry slouží k ukončení běhu algoritmu. Migrations, je parametr, který udává počet tzv. migračních kol, kde migrační kolo znamená cyklus, během kterého se jedinci přemísťují po „N+1“ rozměrné hyperploše. AcceptedError udává, jaký rozdíl mezi nejlepším a nejhorším jedincem v populaci je akceptovatelný. Pokud je počet migračních kol větší než Migrations nebo AcceptedError je větší než rozdíl mezi nejlepším a nejhorším jedincem z aktuální populace, pak algoritmus ukončí svou činnost. Další parametry se podílí na kvalitě běhu algoritmu. NP udává počet jedinců v populaci – aktuální velikost tohoto parametru závisí na uživateli a možnostech jeho hardware. Mass udává, jak daleko od místa svého startu se aktivní

jedinec zastaví (jestliže se $Mass = 1$, jedinec se zastaví na pozici jedince, ke kterému cestuje; jestliže je $Mass = 2$, pak se migrující jedinec zastaví za jedincem ke kterému cestuje ve vzdálenosti, jenž je stejná jako vzdálenost startu od jedince ke kterému cestuje atd.). Step udává velikost kroku, jakým migrující jedinec vykonává prohledávání dané hyperplochy. PRT je tzv. perturbace (rušení), která je jakýmsi ekvivalentem křížící konstanty CR známé z genetických algoritmů či diferenciální evoluce. Je to vektor o stejné dimenzi jako jedinci z populace. Skládá se z 0 a 1 a má významný vliv na samoorganizační proces a robustnost daného algoritmu (viz krok 3).

2. Specimen je definice vzorového jedince (jakéhosi Adama), podle nějž je pak vygenerována celá populace.

$$\text{Specimen} = \{ \{ \text{Real}, \{ \text{Lo}, \text{Hi} \} \}, \{ \text{Integer}, \{ \text{Lo}, \text{Hi} \} \}, \dots, \{ \text{Real}, \{ \text{Lo}, \text{Hi} \} \} \} \quad (1.4)$$

Tento jedinec se skládá z trojic údajů, které definují typ, horní a dolní hranici každého parametru jedince. Každý jedinec se tedy skládá z parametrů (argumentů funkcionálu) a populace z jedinců (množiny různých aktuálních řešení daného funkcionálu)

3. Tvorba populace. Populace je vygenerována podle Specimenu, který udává dimenzi řešeného problému. Počet jedinců je dán parametrem NP. Při generování jedinců tvořících populaci se vychází z definice parametrů jedince. Každý parametr je generován pomocí generátoru náhodných čísel v povoleném intervalu, jenž je definován pomocí horní (Hi) a dolní (Lo) hranice ve Specimenu. Generování probíhá pomocí vztahu

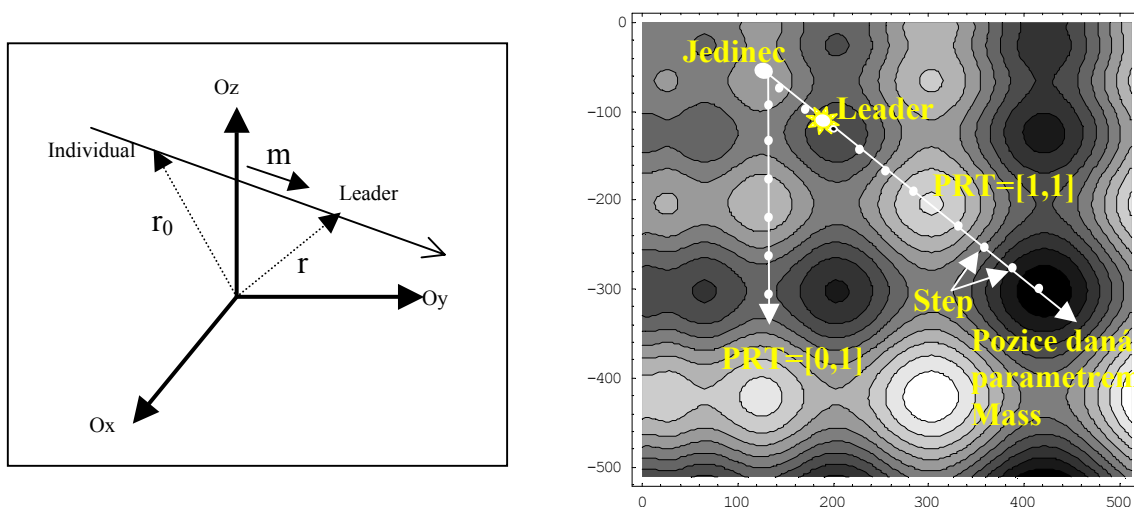
$$P^{(0)} = x_{i,j}^{(0)} = \text{rnd}_{i,j} (x_j^{(Hi)} - x_j^{(Lo)}) + x_j^{(Lo)} \quad i = 1, \dots, n_{pop} \quad , \quad j = 1, \dots, n_{param} \quad (1.5)$$

Součástí jedince je i návratová hodnota funkcionálu (CostValue), která se na výsledném evolučním procesu nepodílí, pouze nese informaci o úspěšnosti daného jedince.

4. Start SOMA algoritmu. Jakmile je vygenerována prvotní populace a jsou nadefinovány potřebné parametry, začnou probíhat tzv. migrační kola, ve kterých probíhá samoorganizace jedinců. Tato samoorganizace probíhá několika způsoby, které mají obecné označení „Strategie“. Základní a také původní strategií je strategie „AllToOne“ (všichni k jednomu), na jejímž základě fungují i strategie ostatní. „AllToOne“ pracuje tak, že po vytvoření první populace je vlastně

vytvořena množina jedinců – bodů na hyperploše. Každý jedinec ohodnotí svou CostValue (geometricky hloubku na hyperploše) a ten jedinec, jehož CostValue je nejmenší se stane tzv. Leaderem (vůdcem). Od tohoto okamžiku se všichni jedinci začnou pohybovat směrem k Leaderovi po skocích danými parametrem Step až do pozice dané parametrem Mass podle rovnice

$$\vec{r} = \vec{r}_0 + m t PRT \quad \text{kde } t \in \langle 0, \text{by Step to, Mass} \rangle \quad (1.6)$$



Obr. 8 Princip algoritmu SOMA

Při přesnějším rozepsání lze rovnici 1.6 rozepsat jako

$$x_{i,j}^{MK+1} = x_{i,j,start}^{MK} + (x_{L,j}^{MK} - x_{i,j,start}^{MK}) t PRT \quad \text{kde } t \in \langle 0, \text{by Step to, Mass} \rangle \quad (1.7)$$

kde „x“ znamená parametr „j“ jedince „i“ (indexy) v daném migračním kole (horní index MK). Dolní index „L“ znamená, že jde o leadera a „start“ značí startovní pozici aktuálního jedince. Po každém skoku ohodnotí jedinec svou CostValue s tím, že si pamatuje pozici, na níž byla nalezena nejnižší CostValue během jeho cesty. Po ukončení cesty na pozici danou parametrem Mass se jedinec vrací na pozici nejnižší CostValue z jeho cesty..

$$x_{i,j}^{MK+1} = \begin{cases} \text{Min}(f_{\text{cost}}(x_{i,j}^{MK})) < x_{i,j,start}^{MK} \\ x_{i,j,start}^{MK} \end{cases} \quad (1.8)$$

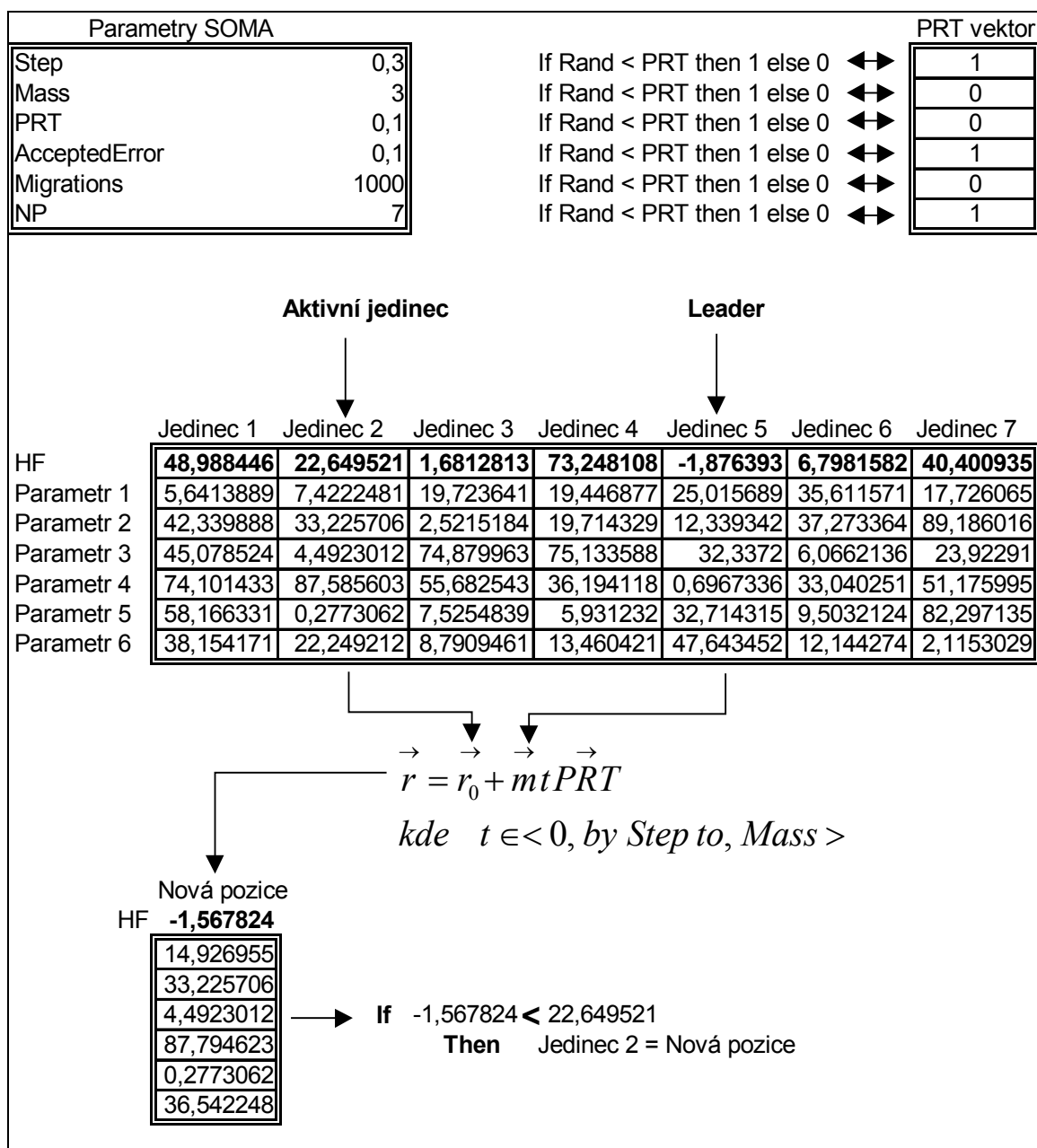
Pohyb jedince je ovlivňován tzv. PRT parametrem, který představuje perturbaci – narušení trajektorie, po které se jedinec pohybuje. Podle PRT parametru se generuje tzv. PRT vektor tak, že se pro každý parametr jedince generuje náhodné číslo mezi 0 a 1 a porovná se s PRT parametrem. Pokud je vygenerované číslo menší než PRT, pak je daný prvek PRT vektoru nastaven na 0, jinak na 1 (viz. 1.9).

$$PRT_j = \begin{cases} 0 & \text{if } Random_j < 0,1 > < PRT \\ 1 & \text{if } Random_j < 0,1 > > PRT \end{cases} \quad (1.9)$$

Je to tedy vektor o stejné dimenzi jako jedinec a skládá se z 0 a 1. Jak vyplývá ze vztahu 1.9 ten parametr, jemuž je z PRT vektoru přiřazena 0, je zafixován a ten, kterému je přiřazena 1, je přepočítáván. Pokud jsou všechny prvky PRT vektoru rovny 1, pak se jedinec pohybuje k Leaderovi přímo podél směrového vektoru. Pokud je „m“ prvků z PRT vektoru rovno nule, pak se jedinec pohybuje v „N-m“ rozměrném prostoru, který je kolmý na původní „N“ rozměrný prostor. Ukončení migrace všech jedinců je označováno jako jedno migrační kolo. Slovo „migrace“ vystihuje podstatu jejich pohybu – jedinci se pohybují - migrují po hyperploše podobně jako jejich biologické protějšky po povrchu krajiny. Princip je znázorněn v Tab. 3, kde je naznačen výpočet nové pozice a „diagnostika“ hlubší.

Návrat na krok 3 dokud není rozdíl mezi nejlepším a nejhorším jedincem v populaci menší než AcceptedError nebo dokud není vyčerpán povolený počet migračních kol. Jinak na krok 5.

5. Konec a návrat nejlepšího jedince (řešení) z populace.



Tab. 2 Jádru SOMA

1.2.4.2 Strategie

V současné době existuje několik variací základního algoritmu SOMA, pro jejichž obecné označení se používá rovněž výraz „strategie“ místo „variace“, aby se podtrhl fakt geometrického přesouvání jedinců populace v „N+1“ dimenzionálním prostoru možných řešení. Všechny verze jsou navzájem prakticky plně porovnatelné ve smyslu nalezení globálního extrému.[1]

1.3 Již řešené příklady úlohy obchodního cestujícího

Na úlohu obchodního cestujícího bylo v minulosti aplikováno mnoho algoritmů. Dá se říci že je to jedna z úloh na které se dají dobře testovat vlastnosti různých výpočetních metod. Jedna z nejrozšířenějších je metoda pomocí genetických algoritmů. Pro tuto metodu již byla vytvořena řada modifikací, právě pro úlohu obchodního cestujícího. V této části budou tyto již vytvořené modifikace popsány a následně předvedeny na zvoleném příkladu.

1.3.1 Vytvořené modifikace genetických algoritmů pro úlohu TSP

Základní přirozená reprezentace úlohy obchodního cestujícího je pomocí řetězců délky M , které jsou tvořeny symboly (číslky nebo názvy M měst). Taková reprezentace má tendenci prohledávat prostor s kardinalitou M^M oproti přípustné $M!$. K takové reprezentaci je tedy nutné najít vhodné rekombinační operátory.

1.3.1.1 Ordinální reprezentace

Ordinální reprezentace pochází z jednoho z historicky nejstarších pokusů o řešení problému obchodního cestujícího metodami GA. Nevychází tak ze zmíněné přirozené reprezentace, nýbrž reprezentuje cestu M městy jako seznam o délce M , přičemž i -ty element seznamu je číslo v rozsahu $(1, 1 - (M - i + 1))$. Interpretace takového seznamu je pak dána tím, že je referenční seznam (např. pro $M = 5$)

$$C = (a \ b \ c \ d \ e).$$

Pak cesta $b-a-e-d-c(-b)$ je reprezentována jako řetězec (seznam) (21321), který je dekódován postupem naznačeným na obr. 3.18.

Základní významnou vlastností této reprezentace je, že umožňuje použití klasického jednobodového křížení, které je analogické křížení používanému SGA. Například rodiče:

$$\begin{aligned} p1 &= (2 \ 1 \ 3 \ 1 \ 21), & c1 &= b-a-e-d-c, \\ p2 &= (3 \ 21 \ 1 \ 11), & c2 &= c-b-a-d-e \end{aligned}$$

vytvoří potomky

$$\begin{aligned} o1 &= (213111), & d1 &= b-a-e-c-d, \\ o2 &= 321!21), & d2 &= c-b-a-e-d. \end{aligned}$$

Genotyp	Cesta	Referenční seznam
2 1 3 2 1 	O	(a, b, c, d, e)
2 1 3 2 1 	(b)	(a, c, d, e)
2 1 3 2 1 	(b,a)	(c, d, e)
2 1 3 2 1 	(b, a, e)	(c,d)
2 1 3 2 1 	(b, a, e, d)	(c)
	(b, a, e, d, c)	O

Tab. 3 Dekódování ordinální reprezentace.

Klasické jednobodové křížení tedy vytvoří ze dvou přípustných rodičů vždy dva přípustné potomky. Jejich vlastnosti však nejsou pro problém obchodního cestujícího příliš vhodné. Část řetězců vlevo od bodu křížení se nemění, avšak pravá část je tvořena zcela náhodně. Experimenty s touto reprezentací ukázaly její neadekvátnost pro řešení problému obchodního cestujícího, a proto byla zavržena. Jiné experimenty pak prokázaly, že může však být výhodně použita pro některé optimalizační úlohy permutačního charakteru z oblasti tvorby rozvrhů.[7]

1.3.1.2 Přirozená reprezentace cesty a specializované rekombinační operátory

Přirozená reprezentace cesty se nakonec ukázala jako nejen nejjednodušší, ale i jako nejúspěšnější, pokud je doplněna vhodně koncipovanými operátory křížení. V tomto zápisu je například cesta a - e - d - c - b (a zpět do a) reprezentována jako chromozómový řetězec (a e d c b). V průběhu vývoje experimentů s řešením problému obchodního cestujícího pomocí GA bylo pro tuto reprezentaci navrženo několik speciálních operátorů křížení.

Operátor křížem se zachováním pořadí - OX: Operátor křížení se zachováním pořadí je označován jako OX (*order crossover*). Je to dvoubodový křížící operátor, který pracuje tak, že náhodně vybere dva křížící body a rozdělí tak oba rodiče na tři (vzhledem k cykličnosti cesty ve skutečnosti na dva) úseky. Operátor vytváří potomky tak, že zachovává relativní pořadí měst z rodičů. Potomci se pak konstruují postupně z počátečního kroku spočívajícího v okopírování prostředního úseku řetězců a zbývajících úseků jsou ponechány jako zatím neznámé (* značí dosud neznámý symbol)

A: a b c | d e f g | h i j
 B: a i b | d f c j | g e h

A': * * * | d e f g | * * *
 B': * * * | d f c j | * * *

Počínaje druhým řezem se nyní kopírují města z druhého z rodičů za současného vynechávání měst již na cestě existujících

A': b c j | d e f g | h a i
 B': b c g | d f c j | h i a

Operátor OX zachovává sice pořadí měst, avšak často rozbíjí slibné úseky cest. Z hlediska účelu, tedy pro úlohu obchodního cestujícího, není pořadí měst příliš zajímavé. Optimální cesta je jistě tvořena spíše zřetěžením kratších úseků pocházejících z kvalitních rodičů. Zachování pořadí však může být velmi výhodné pro některé úlohy tvorby výrobních rozvrhů.

Operátor křížení s částečným zobrazením - PMX: Operátor křížení s částečným zobrazením označovaný jako PMX (*partially mapped crossover*) je opět dvoubodový křížící operátor. Potomci se pak konstruují postupně z počátečního kroku, kdy se okopírují prostřední úseky řetězců. Zbývající pozice se postupně doplňují.

A: a b c | d e f g | h i j
 B: a i b | d f c j | g e h

A': * * * | d e f g | * * *
 B': * * * | d f c j | * * *

Střední částí řetězců implikují částečné zobrazení

$d \leftrightarrow d, e \leftrightarrow f, c \leftrightarrow f, g \leftrightarrow j,$

kterého je využito v dalším postupu. V druhém kroku lze z rodičů doplnit další města, která nejsou konfliktní, tj. nezpůsobí opakování některého města v řetězci. Doplnění se děje vždy z druhého z rodičů, než ze kterého pochází střední část.

A: a b c | d e f g | h i j
 B: a i b | d f c j | g e h

A': a i b | d e f g | * * h
 B': a b * | d f c j | h i *

V posledním, třetím, kroku se využije částečných zobrazení k doplnění chybějících symbolů. Dočasně se zapíše na místa křížků ta zobrazení, v nichž se zobrazuje příslušný symbol z rodiče. Pak zpravidla zbude jedno nebo jen několik málo neznámých symbolů a ty se doplní náhodně tak, aby byl splněn požadavek bezkonfliktnosti.

A: a i b | d e f g | g->j * h
 B: a b * | d f c j | h i g<-j

A': a i b | d e f g | j c h
 B': a b e | d f c j | h i g

Základní vlastností PMX je, že zachovává úseky cest a částečně i pořadí měst, což jsou vlastnosti pro řešení úlohy obchodního cestujícího potřebné.

Operátor křížení s rekombinací hran - ERX: Operátor křížení s rekombinací hran je zpravidla označován jako ERX (*edge recombination crossover*). Je to nejúspěšnější známý operátor křížení pro úlohu obchodního cestujícího. Zvláštností operátoru ERX je, že ze dvou rodičů buduje jen jednoho potomka. Celý GA se pak musí přizpůsobit tomuto faktu, neboť dva rodiče vytvoří jediného potomka a do algoritmu je tedy nutné zahrnout vhodnou náhradovou strategii. Každý úsek cesty ve vytvářeném potomku pochází z některého z rodičů. Operátor ERX k tomu využívá tzv. *hranovou tabulku*, která je použita, jakmile jsou zvoleni rodiče pro křížení. Tato tabulka připisuje ke každému městu seznam jeho sousedů vyskytujících se v obou rodičích.

A:

a	b	c	d	e	f	g	h	i	j
---	---	---	---	---	---	---	---	---	---

 B:

a	i	b	d	f	c	j	g	e	h
---	---	---	---	---	---	---	---	---	---

Hranová tabulka:

Město	Sousedé	Město	Sousedé
a	b, h, i, j	f	c, d, e, g
b	a, c, d, i	g	e, f, h, i
c	b, d, f, j	h	a, e, g, i
d	b, c, e, f	i	a, b, h, j
e	d, f, g, h	j	a, c, g, i

Nyní se postupuje tak, že mezi městy s minimálním počtem sousedů se náhodně zvolí jedno město a to se vyškrtne ze všech seznamů sousedů v hranové tabulce. Volba mezi těmi, která mají minimální počet sousedů představuje prevenci možné izolace města, která by mohla vzniknout v dalším postupu. Stav po náhodné volbě města „a“ je:

O:

a									
---	--	--	--	--	--	--	--	--	--

Hranová tabulka:

Město	Sousedé	Město	Sousedé
a	b, h, i, j	f	c, d, e, g
b	c, d, i	g	e, f, h, j
c	b, d, f, j	h	e, g, i
d	b, c, e, f	i	b, h, j
e	d, f, g, h	j	c, g, i

Nyní mezi sousedy města „a“ náhodně zvolíme další město, přičemž opět upřednostňujeme města s minimálním počtem sousedů. Necht' volba padne na h:

O:

a	h								
---	---	--	--	--	--	--	--	--	--

Opakováním tohoto postupu se dospěje například k potomku ve tvaru

O:

a	h	i	j	g	e	f	c	d	b
---	---	---	---	---	---	---	---	---	---

Experimentálně bylo prokázáno, že tento operátor křížení dává jednoznačně nejlepší výsledky. Je to pochopitelné, neboť potomek přebírá od rodičů potenciálně dobré segmenty celé výsledné cesty.[7]

1.3.1.3 Operátory mutace pro TSP

Mutace v úlohách se složitějšími reprezentacemi obvykle pracuje tak, že s malou pravděpodobností P_m se v populaci vybírají jedinci a ti podstoupí mutaci. Nepracuje se tedy s jednotlivými alelami jako v SGA. Pro úlohu obchodního cestujícího existuje několik příkladů mutačních operátorů. Zde jsou některé z nich:

Zamíchání vygeneruje úplně novou permutaci.

Vložení náhodně vyjme jedno město a vloží je na novou náhodně vybranou pozici.

Inverze vezme úsek cesty a invertuje ho. Jde o poměrně úspěšnou mutaci, zejména v pokročilých fázích vývoje algoritmu, neboť má charakter lokálního vylepšování jedince.

Přesun úseku vyjme náhodně vybraný úsek cesty o náhodné délce a vloží ho na jiné náhodně určené místo (rovněž úspěšná mutace).

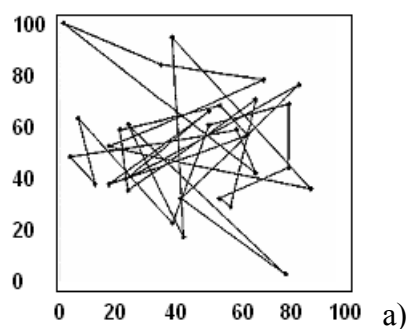
Závěrem je nutno dodat, že mutace se při řešení úlohy obchodního cestujícího používají zřídka.[7]

1.3.1.4 Příklad řešení pomocí operátoru PRX

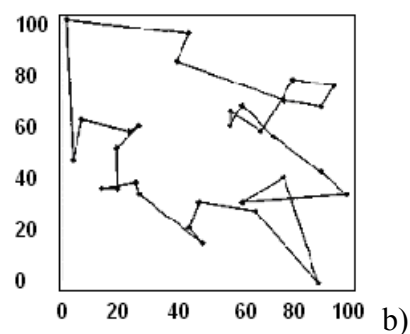
Jde o úlohu s třiceti městy, přičemž pro rozmístění měst byla zvolena testovací úloha používaná jako soutěžní právě pro tuto úlohu. Nejkratší známá cesta pro tuto úlohu má délku $l = 423,74$. Rozmístění měst je zřejmé z obr. 3.19. GA měl tyto parametry: $M = 30$,

velikost populace $N = 500$, pravděpodobnost křížení $P_x = 0,95$, operátor křížení PRX, selekce ruletovým kolem bez úpravy selekčního tlaku.

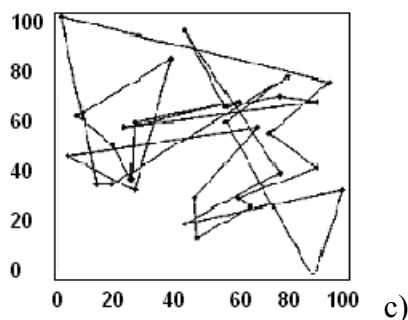
Na obrázku 3.19 je ilustrován jeden náhodně vybraný běh v předchozím výkladu uvedené úlohy obchodního cestujícího. Grafy a) a b) ukazují nejhoršího a nejlepšího jedince z počáteční náhodně vygenerované populace 500 permutací. Nejlepší jedinec měl délku cesty $l=1038$. Graf c) ukazuje nejlepšího dosud nalezeného jedince v 25. generaci. V 94. generaci byl nalezen jedinec s délkou cesty $l = 423,74$. Byla tedy nalezena nejkratší známá cesta pro tuto úlohu.



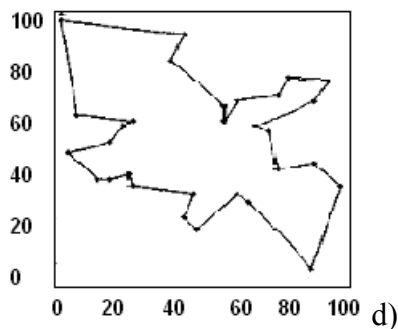
Nejhorší cesta pro $t = 0$; $l = 1349,58$



Nejlepší cesta pro $t=0$, $l=1038,08$

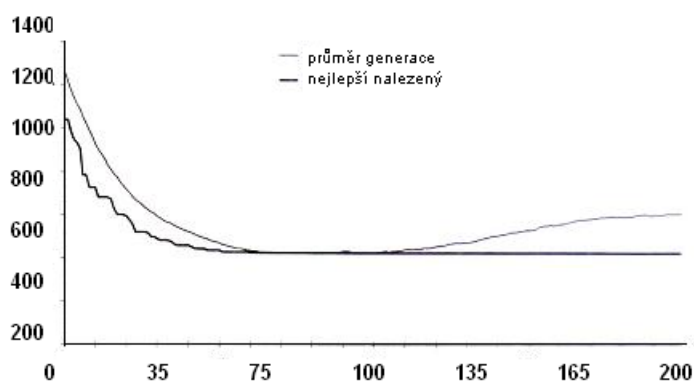


$t=25$, $l=518,49$



$t=100$, $l=423,74$

Obr. 9 Jeden běh řešení úlohy obchodního cestujícího.



Graf. 2 Průběh genetického algoritmu

Běžně dostupné publikace tvrdí, že v úlohách kombinatorických optimalizací je mutace vždy jen druhotnou rekombinační operací. V tomto experimentu však byla mutace typu *přesun úseku* použita, avšak poměrně speciálním způsobem. Pravděpodobnost mutace P_m byla totiž nulová, pokud docházelo k vylepšování řešení. Jakmile během posledních pěti generací nenastalo zlepšení dosud nejlepšího jedince, začala pravděpodobnost mutace růst o 0,05 v každé generaci, a to až do dosažení hodnoty $P_m = 0,5$. Zvětšující se pravděpodobnost mutace vnáší do jinak téměř homogenní populace zcela nové prvky. Jakmile došlo ke zlepšení, P_m opět byla nastavena na nulu. Efekt této modifikace je dobře vidět na grafu 2. Zhruba od 65. generace jsou populace téměř homogenní (jejich průměrná kvalita v podstatě shodná s nejlepším jedincem). Okolo generace č. 90 se začíná projevovat vliv mutace - populace se mírně diverzifikuje. V generaci č. 94 je pak nalezeno nejlepší řešení, mající za následek pokles P_m opět na nulu s následnou homogenizací populace. V dalším průběhu experimentu již pravděpodobnost P_m jen roste a s ní i průměrná délka cesty v populaci.

Takto nakonfigurovaný experiment byl opakován a pouze v jediném z dvaceti experimentálních běhů se stalo, že do sté generace nebylo nalezeno ono řešení s délkou cesty $l = 423,74$. Naproti tomu při zmenšení velikosti populace na $N = 300$ bylo toto řešení nalezeno jen zřídkakdy. Velikost populace je tedy poměrně kritickým parametrem GA.(7)

Tato úloha má prohledávací prostor o kardinalitě $30! = 2 \cdot 10^{32}$. V tomto experimentu však během 100 generací stačilo prohlédnout $100 \cdot 500 = 50000$ cest, aby byl mezi nimi nalezeno takto vysoce kvalitní řešení. To jistě nepřímě vypovídá o síle GA a především o síle tohoto operátoru křížení ERX.

2 EXPERIMENTÁLNÍ ČÁST

V této části byl úkol navrhnout a naprogramovat úpravu vybraných evolučních algoritmů, tak aby tyto algoritmy šly aplikovat na tzv. Úlohu obchodního cestujícího a tím vytvořit alternativní řešení zejména ke Genetickým algoritmům. Dále bylo úkolem provést testování těchto algoritmů na úloze obchodního cestujícího. Po dohodě s vedoucím diplomové práce byly zvoleny z časových důvodů algoritmy Diferenciální Evoluce a SOMA.

2.1 Zvolené zadání úlohy obchodního cestujícího

Po dohodě s vedoucím diplomové práce bylo zvoleno následující zadání úlohy obchodního cestujícího:

Je dáno m měst. Všechna města jsou propojena silnicemi, délka silnic je známa. Úkolem obchodního cestujícího je navštívit všechna města právě jednou. Nalezněte takovou túru (permutaci měst), aby délka uražené cesty byla co nejmenší.

Navrhnete algoritmy Diferenciální Evoluce a SOMA řešící problém obchodního cestujícího. Algoritmy testujte na odpovídajícím vzorku vstupních dat.

Zadání úlohy je na první pohled celkem standardní, ovšem dramatický rozdíl oproti ostatním metodám nastává při samotném přístupu k řešení.

V naprosté většině algoritmů řešící tuto úlohu (tzn. úlohu ve variantě s průchodem každého města právě jednou) je určité „zabezpečení“, které zaručuje, že algoritmus hledá řešení pouze v oblasti přípustných řešení tzn. bez možnosti duplicity měst. Jinými slovy algoritmus je uzpůsoben tak aby vždy generoval pouze nekonfliktní jedince tzn. jedince kde se neopakují města. Prohledávací prostor těchto algoritmů má pak kardinalitu $M!$, kde M je počet měst.

V případě aplikace algoritmů Diferenciální Evoluce resp. SOMA tomu bude jinak. Tyto algoritmy nemají zatím žádné varianty které by zaručovaly tuto bezkonfliktnost. Tyto algoritmy budou mít tedy prohledávací prostor o kardinalitě M^M . Proto při hledání minima byl zvolen systém penalizací kterým je možno docílit přijatelných výsledků. Pokud se v nalezené cestě vyskytne nějaké město více jak jednou, bude k výsledné délce cesty připočtena penalizační hodnota 2000 tolikrát, kolikrát se bude ve zkoumané trase vyskytovat duplicita. Tato hodnota nebyla zvolena náhodně, nýbrž tak, aby byla vyšší než maximální možná délka cesty bez opakování pro počet měst roven třiceti a maximální

vzdálenosti jednotlivých měst mezi sebou rovnu padesáti. Tato maximální délka cesty by pak byla 1500. Takováto penalizace duplicitních tras pak zásadně přispívá ke konvergenci.

2.2 Programové prostředky

Pro samotnou realizaci programátorské části bylo použit vývojový prostředek MATLAB 6.0 a již vytvořený toolbox Evolučních algoritmů. Celé grafické rozhraní tohoto toolboxu je řešeno jako soubor funkcí beze skriptů, z toho vyplívá nutná vlastnost pro toolboxy a to nepřidávání nebo nepřekrývání proměnných ve workspace Matlabu vnitřními proměnnými funkcí v toolboxu. Bylo tedy nutné takto koncipovat i nově vytvořené funkce. Bylo vytvořeno 8 nových funkcí které obsahují cca. 2500 řádků zdrojového textu ve vývojovém prostředí Matlab 6.0.

2.2.1 Vytvořené funkce

Seznam vytvořených funkcí vložených do toolboxu Evolučních algoritmů ve vývojovém prostředí MATLAB 6.0

TSPgenv.m

TSPread.m

TSP.m

AlgdeTSP.m

AlgsomaTSP.m

UiTSP.m

UiTSPd.m

UiTSPs.m

Popis jednotlivých funkcí

Funkce **TSPgenv.m** - Funkce pro náhodnou generaci dat o vzdálenostech měst do souboru typu CSV.

Zadáva se v tomto formátu:

```
function [pocvzd] = tspgenv(poc_mest,limit,filename)
```

Vstupní parametry:

POC_MEST – udává počet měst pro které se mají generovat vzdálenosti

LIMIT – udává maximální možnou vzdálenost mezi městy

FILENAME – jméno souboru do kterého se budou generovat data o vzdálenostech

Výstupní parametr:

POCVZD – vektor s náhodně vygenerovanými vzdálenostmi mezi městy

Funkce **TSPread.m** - Funkce pro čtení dat o vzdálenostech ze souboru CSV.

Zadáva se v tomto formátu:

```
function [vzdalenosti] = tspread(filename)
```

Vstupní parametr:

FILENAME(string) – Jméno souboru

Výstupní parametr:

VZDALENOSTI – vektor obsahující vzdálenosti mezi městy

Funkce **TSP.m** – Jedna z nejdůležitějších funkcí, přizpůsobuje algoritmy Diferenciální Evoluce a SOMA pro použití na úlohu obchodního cestujícího. Dá se říci že je to účelová funkce. Vstupními parametry jsou CESTA což je pořadí měst a VZDALENOSTI mezi jednotlivými městy. Pořadí měst tvoří vektor, který je totožný s jedincem v populaci tzn. tento parametr si bere ze samotného průběhu algoritmu. Parametr VZDALENOSTI si bere

ze souboru CSV který tyto data obsahuje. Výstupem pak parametr DELKA kterému odpovídá přímo délka zkoumané trasy, je to tedy hodnota účelové funkce (CF).

Zadává se v tomto formátu:

```
function [delka] = tsp(cesta, vzdalenosti)
```

Vstupní parametry:

CETSA - vektor, který je totožný s jedincem v populaci tzn. tento parametr si bere ze samotného průběhu algoritmu

VZDÁLENOSTI – vektor vzdáleností který si bere z vygenerovaného souboru CSV

Výstupní parametr:

DELKA – délka zkoumané trasy (CF)

Funkce **AlgdeTSP.m** – Je to funkce která obsahuje samotné jádro algoritmu Diferenciální Evoluce. Je uzpůsobena pro použití s účelovou funkcí TSP.m. Vychází z toolboxové funkce Algde.m.

Zadává se v tomto formátu:

```
function [min,paramd,mat,pmat,actmat] = algdetsp(vzorec,dim,f,cr,np,gen)
```

Vstupní parametry:

VZOREC(string) – vzorec účelové funkce, v tomto případě to bude funkce TSP.m

DIM – značí počet měst (dimenzí)

F(0..1) – mutační konstanta

CR(0..1) – práh křížení

NP – počet jedinců v populaci

GEN – počet generací

Výstupní parametry:

MIN – minimum funkce (nejkratší hodnota nalezené trasy)

PARMD – parametry minima (vektor s pořadím měst s nejlepší CF)

MAT – aktuální populace

PMAT – počáteční populace

ACTMAT – aktuální populace

Funkce **AlgsomaTSP.m** - Je to funkce která obsahuje samotné jádro algoritmu SOMA. Je uzpůsobena pro použití s účelovou funkcí TSP.m. Vychází z toolboxové funkce Algsoma.m.

Zadáva se v tomto formátu:

Function

```
[mmin,parms,kmat,pmat,actmat]=algsomatasp(fname,mig,np,mass,step,PRT,acceptederror, variace)
```

Vstupní parametry:

FNAME – soubor ze kterého se mají číst data o vzdálenostech měst

MIG – počet migrací

NP – počet jedinců v populaci

MASS – pozice jedince

STEP – krok jedince

PRT - perturbace

ACCEPTEDERROR – maximální odchylka jedinců od minima

VARIACE – 1 ALL TO ONE

- 2 ALL TO ALL

Výstupní parametry:

MMIN - minimum funkce (nejkratší hodnota nalezené trasy)

PARMS - parametry minima (vektor s pořadím měst s nejlepší CF)

KMAT - konečná populace

PMAT – počáteční populace

ACTMAT – aktuální populace

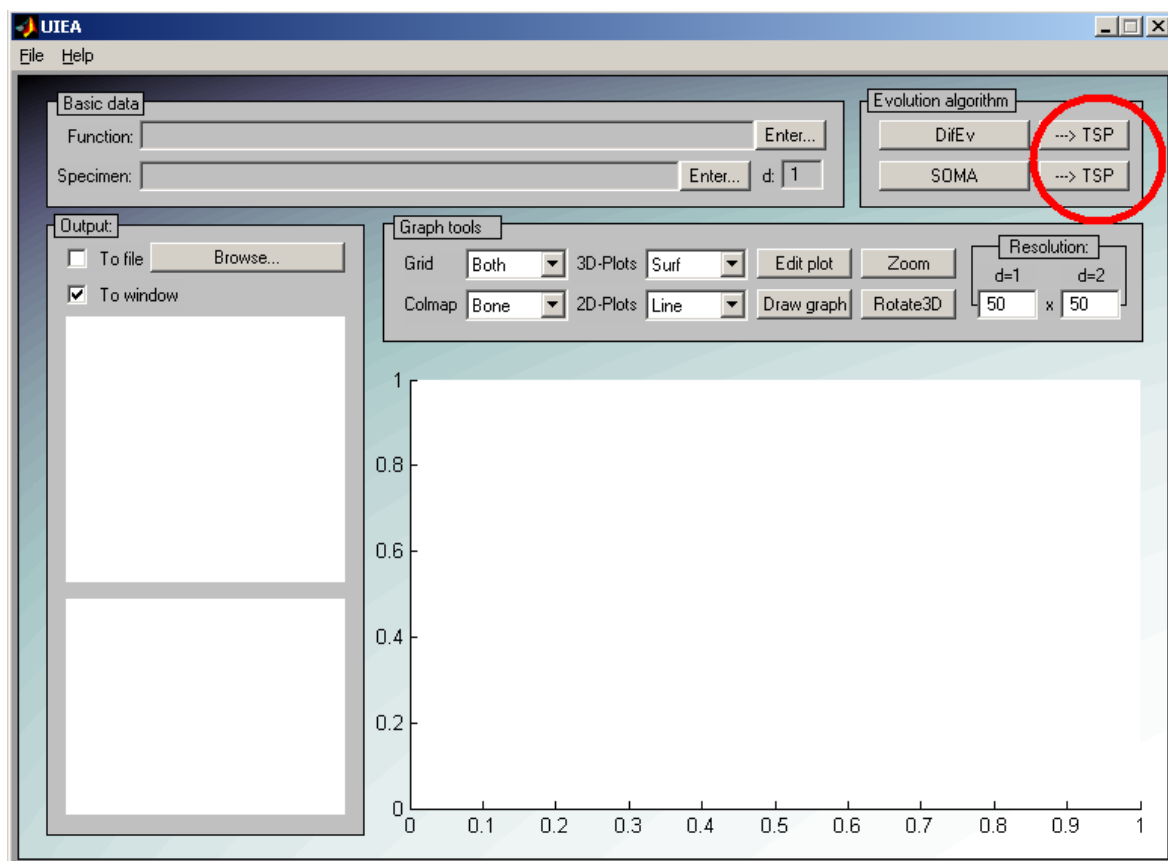
Funkce **UiTSPd.m**, **UiTSPs.m** – tyto funkce tvoří grafické uživatelské rozhraní funkcí algoritmu Diferenciální Evoluce resp. SOMA. Jsou implementovány do grafického rozhraní toolboxu evolučních algoritmů. Jejich ukázka je v kapitole Grafické rozhraní.

2.2.2 Grafické rozhraní

V této části bylo úkolem implementovat veškeré funkce do jednotného uživatelského rozhraní toolboxu evolučních algoritmů. Záměrem je co nejvíce zjednodušit ovládání uživateli který tento toolbox bude používat. Spuštění grafického rozhraní se provádí příkazem UIEA

- Vzhled hlavního okna UIEA a změny v něm vytvořené.

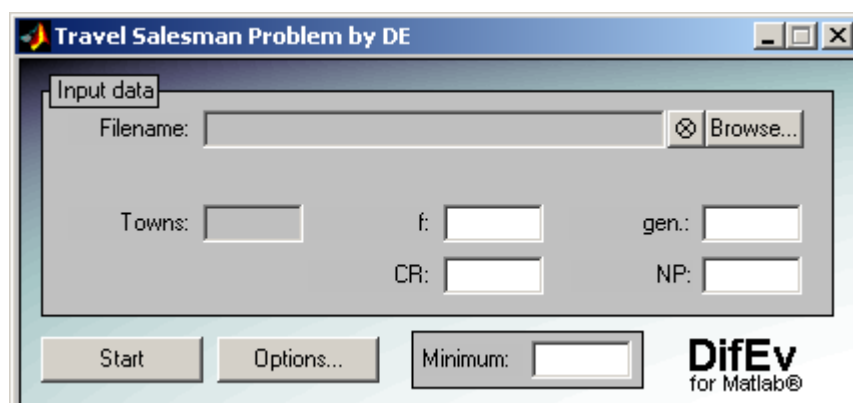
Zde bylo nutnosti vytvoření dalších dvou tlačítek s názvem „TSP“ pro spouštění nově vytvořených funkcí UiTSPd.m resp. UiTSPs.m. Změny jsou ukázány na obr. 10. Po kliknutí na tyto tlačítka se přímo spouští okna pro práci s Úlohou obchodního cestujícího.



Obr. 10 Změny v hlavním okně UIEA

➤ Okno Travel Salesman Problem by DE

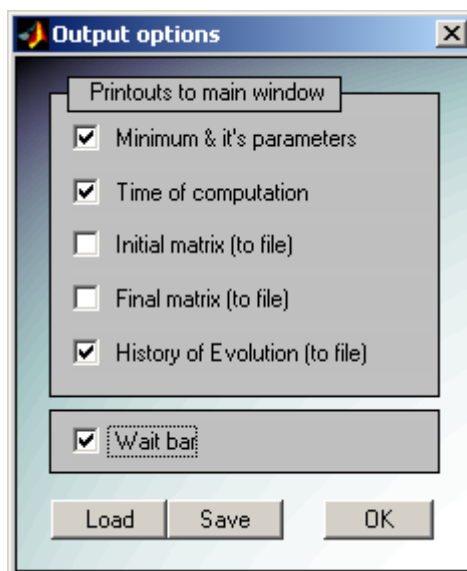
Okno pro zadávání vstupních parametrů algoritmu Diferenciální Evoluce. Popis viz Str. 25. Spouštění algoritmu se provádí tlačítkem „Start“, pro nastavení výstupních hodnot je zde volba „Options“.



Obr. 11 Okno TSP by DE

➤ Okno Output options

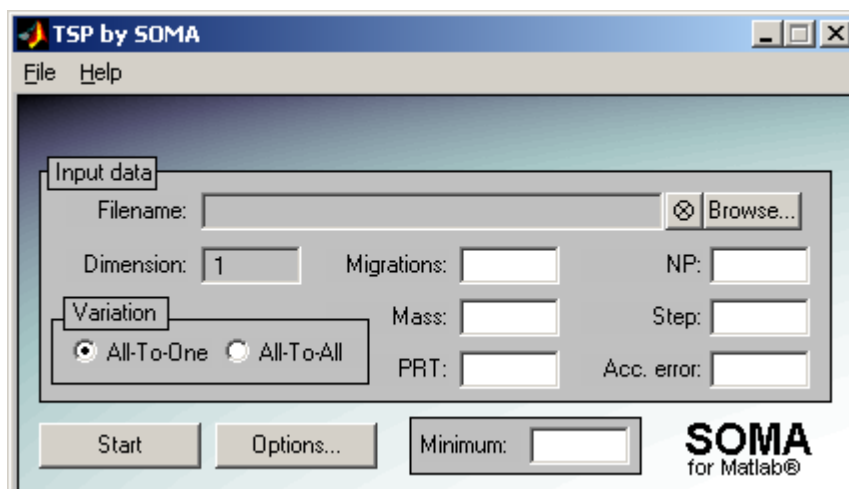
V tomto okně má uživatel možnost volit, jaké informace ho při výpočtu nejkratší trasy dále zajímají. Je zde možnost ukládání do souboru který se nastavuje v hlavním okně UIEA. Potvrzení se provádí tlačítkem „OK“.



Obr. 12 Okno Output options

➤ Okno TSP by SOMA

Okno pro zadávání speciálních parametrů pro algoritmus SOMA. Popis viz Str. 30. Spouštění algoritmu se provádí tlačítkem „Start“, pro nastavení výstupních hodnot je zde volba „Options“ shodná s volbou u algoritmu Diferenciální Evoluce.



Obr. 13 Okno TSP by SOMA

2.3 Testovací část

Zde bylo úkolem testovat nově vytvořené algoritmy na co největším souboru dat s cílem nalézt optimální parametry k oběma algoritmům.

Výpočty zejména příkladů kdy počet měst byl více než 20 byly z časového hlediska velmi náročné. Doba výpočtu příkladu kdy počet měst byl 30, počet jedinců v populaci 20, počet generací resp. migrací 300 dosahovala hodnoty téměř pěti hodin. Z tohoto důvodu bylo hned od počátku zvoleno, že oba algoritmy se budou testovat na příkladech s počtem měst od 3 do 30 a parametr NP (počet jedinců v populaci) bude roven 20. Je nutno dodat že velikost generace samozřejmě ovlivní rychlost konvergence, ovšem zhruba takovým způsobem, že pokud je zvětšena velikost generace na dvojnásobek, konverguje algoritmus (přibližně) dvakrát rychleji - měřeno počtem generací. Vše samozřejmě myšleno v rámci rozumných mezí. Takto byl aplikován na oba algoritmy.

Další parametry shodné pro oba algoritmy byly parametry GEN a MIG, tedy počet generací pro Diferenciální Evoluci, resp. migrací pro SOMU. Tyto parametry byly shodně zvoleny na hodnotě 200 pro soubory s počtem měst od 3-10. Pro soubory s počtem měst 10-30 měly tyto parametry hodnotu 260.

Další parametry již byly pro oba algoritmy rozdílné a jejich nastavení předcházelo zdlouhavé testování těchto algoritmů.

2.3.1 Volba parametrů Diferenciální Evoluce

Zde tedy zůstávaly jako neznámé parametry F a CR.

Parametr **CR** $\in <0, 1>$. Jde o tzv. práh křížení. V případě že se jedná o funkci, která je separabilní, pak je doporučeno nastavit tento parametr na hodnoty blízké 0. V opačném případě jsou výhodné hodnoty, které se blíží 1. V případě, že se CR nastaví na 0 dojde k tomu, že se mutace nedostane do zkušebního jedince, který díky tomu bude čistou kopií aktuálního (čtvrtého rodiče). Vývoj evoluce se pak zastaví. V případě že CR bude nastaven na 1, bude zkušební jedinec tvořen pouze ze tří náhodně vybraných rodičů – jedinců z populace a diferenciální evoluce se bude spíše podobat náhodnému hledání nežli evolučnímu algoritmu (všichni tři jedinci, byť evolučně vytvoření, jsou náhodně vybráni bez ohledu na jejich kvalitu). Je proto vhodné, aby CR nikdy nenabývalo těchto hodnot.

Pro případ obchodního cestujícího se jako optimální jevila hodnota konstanty $CR = 0.5$. Při této hodnotě bylo zajištěno toho, že se jedinci dostatečně křížili, ale ne zase zbytečně hodně, a tím se rozhazovaly slibné části jedinců.

Parametr $F \in \langle 0, 2 \rangle$. Mutační konstanta je poslední řídicí parametr diferenciální evoluce a její hodnota se pohybuje v intervalu $\langle 0, 2 \rangle$. Pro případ obchodního cestujícího byla zvolena hodnota 0.2. Tato poměrně nízká hodnota zaručí nízkou mutaci diferenčního váhového vektoru a tím pádem i menší riziko výstupu jedince mimo povolený interval. Nutno dodat že v případě úlohy obchodního cestujícího mutační konstanta F neměla nijak zásadní vliv na chod evoluce.

Pro další měření byly tedy vybrány tyto hodnoty parametru:

Soubory s počty měst: 3-30

Generace: 200 pro počty měst 3-10, a 260 pro počty měst 11-30

NP: 20

CR: 0.5

F: 0.2

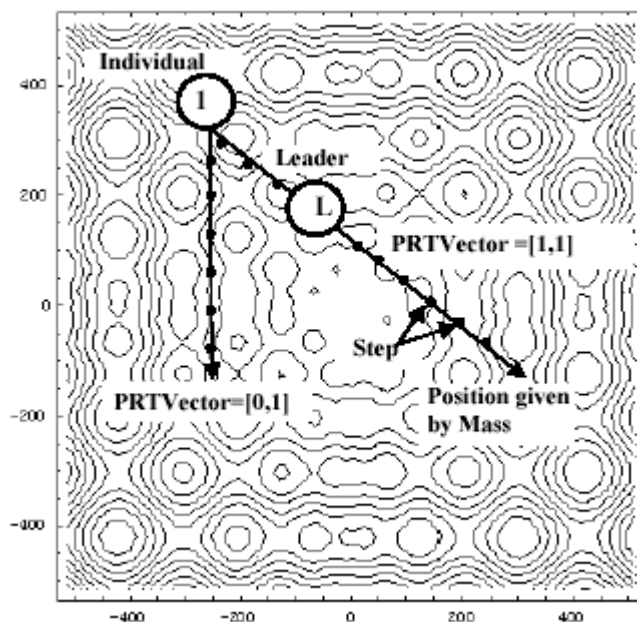
2.3.2 Volba Parametrů SOMA

Zde tedy zůstávaly jako neznámé parametry STEP, MASS, PRT, AcceptedError.

Parametr **AcceptedError** – udává, jaký rozdíl mezi nejlepším a nejhorším jedincem v populaci je akceptovatelný. Pokud je počet migračních kol větší než Migrations nebo AcceptedError je větší než rozdíl mezi nejlepším a nejhorším jedincem z aktuální populace, pak algoritmus ukončí svou činnost. Zvolení tohoto parametru bylo nejsnadnější. Pro naši úlohu je požadavek aby proběhly všechny generace které jsou naplánované. Proto tedy byla hodnota parametru AcceptedError zvolena záporná, konkrétně na hodnotu -1 .

Zvolení ostatních parametrů již bylo daleko komplikovanější.

Parametry **STEP**, **MASS**, **PRT** – tyto parametry mají zásadní vliv na běh celé evoluce. Jejich význam je znázorněn na obrázku 14. Zde je jasně vidět jak tyto parametry ovlivňují chod jedince směrem k leaderovi.



Obr. 14 Pohyb jedince k leaderovi

Je tedy jasné že parametr **MASS** musí být dostatečně velký, rozhodně větší než 1. V opačném případě by jedinec vůbec k leaderovi nedošel a algoritmus by konvergoval do lokálního minima. Pro úlohu obchodního cestujícího se nejlepší volba zdála na hodnotě 3. Tato hodnota ještě nebyla příliš velká s tím že by neúměrně protahovala dobu výpočtu, ale zase dostatečná aby algoritmus prohledával dostatečně velkou oblast.

Volba parametru **STEP** se zdála být jednoznačná. Zde totiž platí že čím bude menší, tím samozřejmě jedinec při postupu směrem k leaderovi bude nabývat více hodnot. Při úloze obchodního cestujícího ovšem příliš drobné postupování nemá smysl. Zde se pracuje na oboru celých čísel a je tedy jasné že příliš malé kroky po zaokrouhlení budou nabývat stejných hodnot a tedy se jen prodlužuje doba výpočtu bez potřebného efektu. Další poznatek byl že je dobré volit tento parametr tak, aby nebyl celočíselný dělitel parametru **MASS**. Při této podmínce totiž jedinec při postupu leadera přeskočí a tedy nebude nabývat jeho hodnot. To také snižuje riziko konvergence. Při testování se jako nejlepší volba jevila hodnota parametru **STEP**=0.65.

Konečně parametr **PRT** byl zvolen na hodnotě 0.5 z obdobného důvodu jako parametr **CR** u Diferenciální Evoluce. Tím bude trasu měnit přiměřeně, je tedy zaručeno že

jedinec bude postupovat k Leaderovi po sice ne ideální trajektorii, ale ne zase po velmi změněné.

Pro další měření byly tedy vybrány tyto hodnoty parametru:

Soubory s počty měst: 3-30

Migrace: 200, 260 pro počty měst 3-10, 11-30

NP: 20

MASS: 3

STEP: 0.65

PRT: 0.5

2.4 Výsledky výpočtů

V první části budou porovnány algoritmy Diferenciální Evoluce a SOMA na základě výsledných hodnot po vykonání určitého počtu generací resp. migrací. Výsledky jsou brány pro příklady s počty měst 3-10, zde bylo provedeno 200 generací resp. migrací a pro počty měst 11-30, zde bylo provedeno 260 generací resp. migrací. Pro každý příklad s určitým počtem měst byly provedeny tři výpočty a následně byl generován průměr těchto měření. Takto se postupovalo u obou algoritmů. Výsledné hodnoty jsou zanesené v tabulkách Tab. 4-6. Pro přehlednost jsou barevně zvýrazněna pole která obsahují minimum z měření obou algoritmů. Oranžovou barvou jsou zvýrazněna pole která obsahují nejlepší hodnoty z výsledků obou algoritmů. Modrou barvou jsou zvýrazněna pole která obsahují nejlepší průměrné hodnoty z výsledků obou algoritmů. Šedá pole znázorňují hodnoty kde oba algoritmy dosahovaly shodných výsledků.

Towns	3	4	5	6	7	8	9	10	11
DE1	44	65	87	41	75	149	121	137	145
DE2	44	65	102	38	82	168	110	102	226
DE3	44	65	104	41	75	159	97	105	168
Prům.	44,00	65,00	97,67	40,00	77,33	158,67	109,33	114,67	179,67
Soma1	44	65	87	36	72	156	112	103	135
Soma2	44	65	87	40	77	144	110	73	142
Soma3	44	65	87	36	75	140	91	121	115
Prům.	44,00	65,00	87,00	37,33	74,67	146,67	104,33	99,00	130,67

Tab. 4 Výsledková tabulka (oblast příkladů s počty měst 3-11)

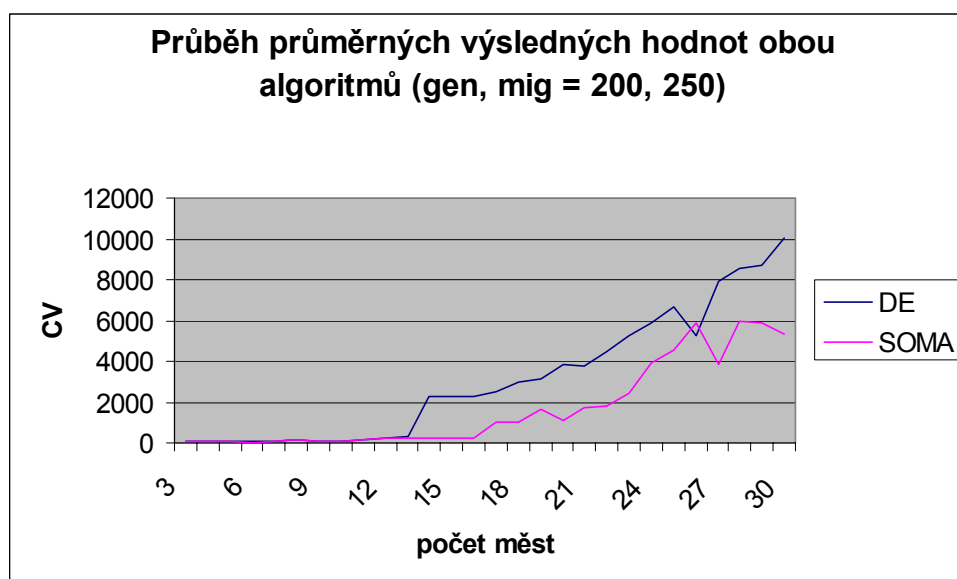
Towns	12	13	14	15	16	17	18	19	20
DE1	194	393	2273	2306	2245	2418	2351	2410	4420
DE2	169	359	2244	2258	2270	2559	4283	2506	4456
DE3	292	244	2243	2298	2306	2494	2405	4443	2544
Prům.	218,33	332,00	2253,33	2287,33	2273,67	2490,33	3013,00	3119,67	3806,67
Soma1	176	248	243	186	220	367	408	2317	389
Soma2	221	235	233	292	282	2231	2305	2350	2495
Soma3	208	192	221	233	227	365	393	297	358
Prům.	201,67	225,00	232,33	237,00	243,00	987,67	1035,33	1654,67	1080,67

Tab. 5 Výsledková tabulka (oblast příkladů s počty měst 12-20)

Towns	21	22	23	24	25	26	27	28	29	30
DE1	2489	4549	4547	4579	6667	4540	8634	8531	8719	8880
DE2	4467	4393	6545	6595	6583	6566	6673	8586	8621	10689
DE3	4398	4491	4566	6574	6742	4665	8532	8516	8689	10563
Prům.	3784,66	4477,66	5219,33	5916	6664	5257	7946,33	8544,33	8676,33	10044
Soma1	2392	2436	4455	4619	4623	4598	4539	4601	6574	6721
Soma2	335	2572	2360	4584	6466	6589	6411	6676	4499	4582
Soma3	2518	336	471	2520	2529	6545	465	6545	6644	4736
Prům.	1748,33	1781,33	2428,67	3907,67	4539,33	5910,67	3805,00	5940,67	5905,67	5346,33

Tab. 6 Výsledková tabulka (oblast příkladů s počty měst 21-30)

Grafická závislost zprůměrovaných výsledných hodnot (CV) snímaných po stejném počtu generací resp. migrací (200 pro počet měst 3-10; 260 pro počet měst 11-30) na počtu měst v příkladu je patrná z grafu Graf 1.



Graf 1

Z tabulek výsledků Tab 1-3 výsledků a z grafu Graf 1, je zcela evidentní že po proběhnutí stejného počtu generací resp. migrací vrací daleko lepší hodnoty algoritmus SOMA. Toto je očekávaný a zcela logický výsledek srovnání těchto algoritmů na tomto principu. Algoritmus Diferenciální Evoluce totiž v průběhu jedné generace mnohem méně často vypočítává hodnotu účelové funkce (v tomto případě délku trasy). Vzorce pro výpočet počtu ohodnocení účelové funkce (CFE) v jedné generaci/migraci je následující:

Algoritmus Diferenciální evoluce: $CFE = NP$

Algoritmus SOMA: $CFE = [(NP-1) * MASS] / STEP$

V dalším je provedeno srovnání obou algoritmů na základě stejného množství ohodnocení účelové funkce (Cost Function Evolutions - CFE). Takto pojaté srovnání je daleko objektivnější a dá se říci že nejčastěji používané pro srovnání algoritmů evolučního typu. Výsledné hodnoty jsou snímány při 4000 ohodnocení účelové funkce pro příklady počtu měst 3-10 a při 5120 pro příklady počtu měst 11-30. Výsledky jsou zanesené v tabulkách Tab. 7-9. Pro přehlednost jsou opět zvýrazněná pole dle stejného principu jako v tabulkách Tab. 4-6.

Towns	3	4	5	6	7	8	9	10	11
DE1	44	65	87	41	75	149	121	137	145
DE2	44	65	102	38	82	168	110	102	226
DE3	44	65	104	41	75	159	97	105	168
Prům.	44,00	65,00	97,67	40,00	77,33	158,67	109,33	114,67	179,67
SOMA1	44	65	102	53	83	172	148	139	192
SOMA2	44	65	87	40	77	150	121	73	146
Soma3	44	65	87	36	87	144	143	138	177
Prům.	44,00	65,00	92,00	43,00	82,33	155,33	137,33	116,67	171,67

Tab. 7 Výsledková tabulka (oblast příkladů s počty měst 3-11)

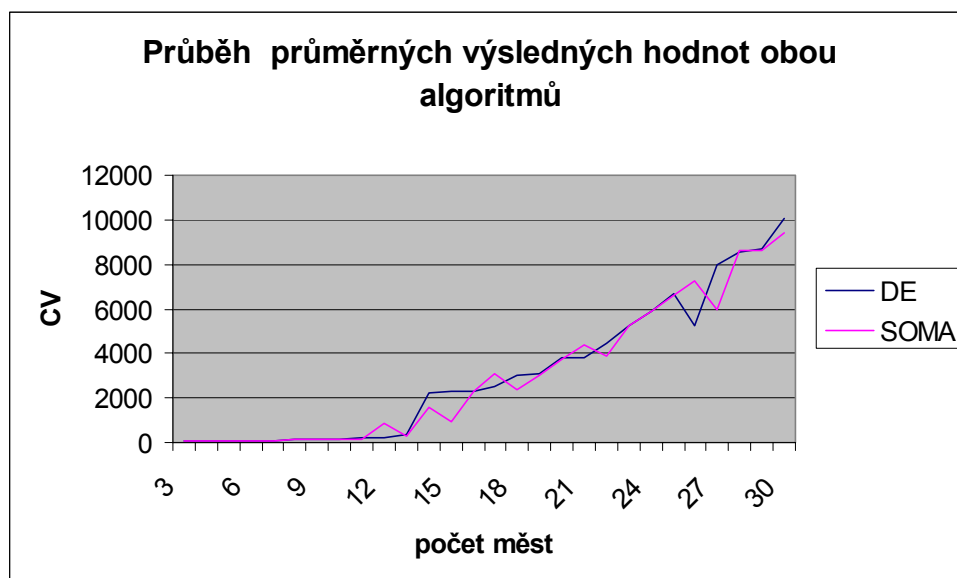
Towns	12	13	14	15	16	17	18	19	20
DE1	194	393	2273	2306	2245	2418	2351	2410	4420
DE2	169	359	2244	2258	2270	2559	4283	2506	4456
DE3	292	244	2243	2298	2306	2494	2405	4443	2544
Prům.	218,33	332,00	2253,33	2287,33	2273,67	2490,33	3013,00	3119,67	3806,67
SOMA1	215	248	291	225	2247	2502	2417	2363	4398
SOMA2	2194	288	2238	301	2261	4306	2357	2387	4373
SOMA3	258	256	2221	2242	2313	2415	2364	4319	2402
Prům.	889,00	264,00	1583,33	922,67	2273,67	3074,33	2379,33	3023,00	3724,33

Tab. 8 Výsledková tabulka (oblast příkladů s počty měst 12-20)

Towns	21	22	23	24	25	26	27	28	29	30
DE1	2489	4549	4547	4579	6667	4540	8634	8531	8719	8880
DE2	4467	4393	6545	6595	6583	6566	6673	8586	8621	10689
DE3	4398	4491	4566	6574	6742	4665	8532	8516	8689	10563
Prům.	3784,67	4477,67	5219,33	5916,00	6664,00	5257,00	7946,33	8544,33	8676,33	10044,00
SOMA1	4343	4558	6500	6523	6583	8557	8567	8647	10566	10642
SOMA2	4385	4585	4606	6546	8478	6617	6674	8667	6588	8716
SOMA3	4447	2472	4567	4572	4739	6643	2553	8603	8703	8832
Prům.	4391,67	3871,67	5224,33	5880,33	6600,00	7272,33	5931,33	8639,00	8619,00	9396,67

Tab. 9 Výsledková tabulka (oblast příkladů s počty měst 21-30)

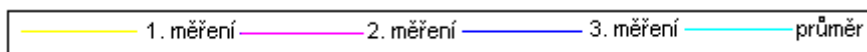
Grafická závislost zprůměrovaných výsledných hodnot (CV) snímaných po stejném počtu ohodnocení účelové funkce (CFE – 4000 pro počet měst 3-10; 5120 pro počet měst 11-30) na počtu měst v příkladu je patrná z grafu Graf 2.

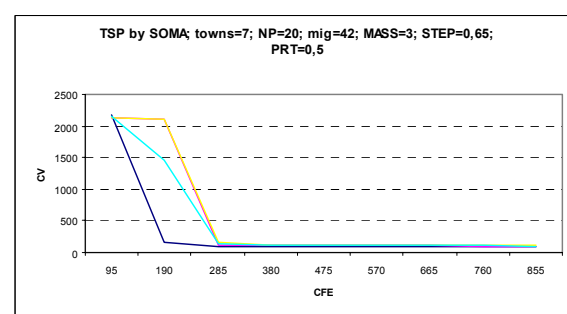
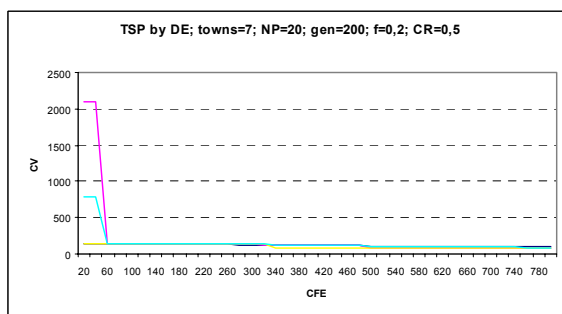
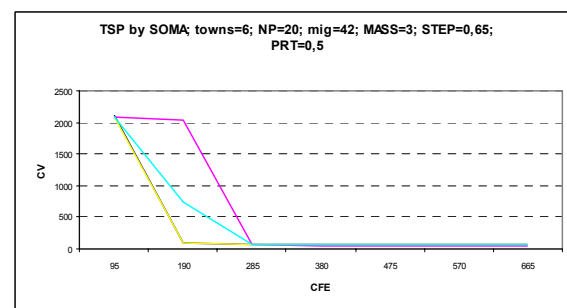
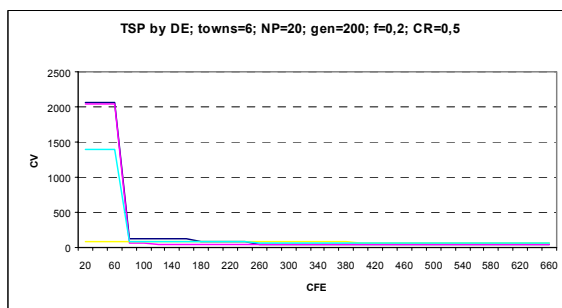
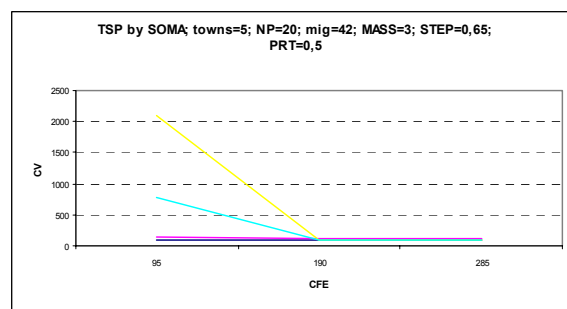
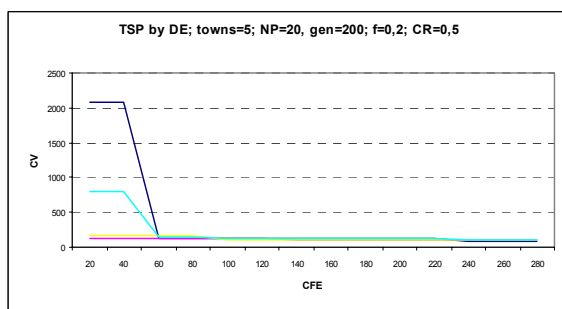
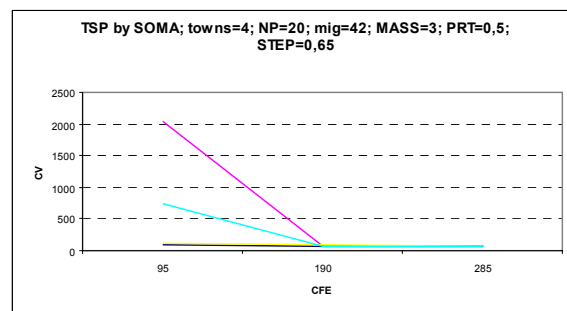
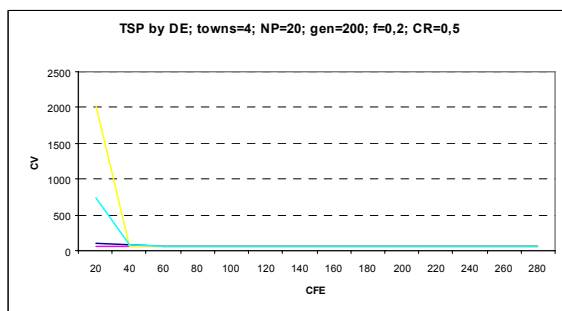
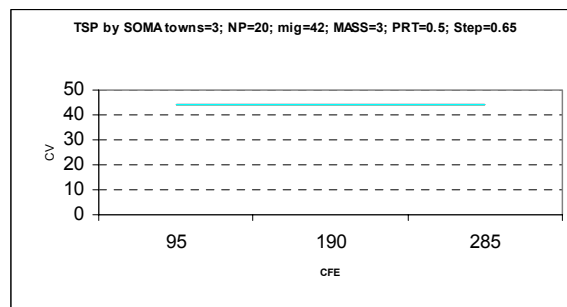
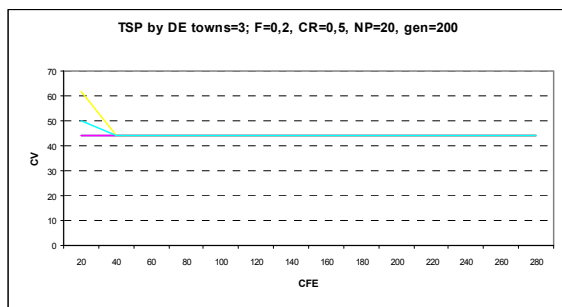


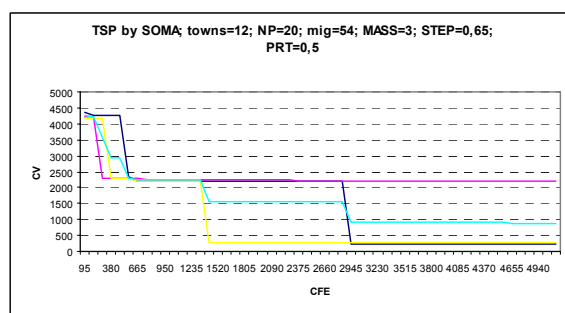
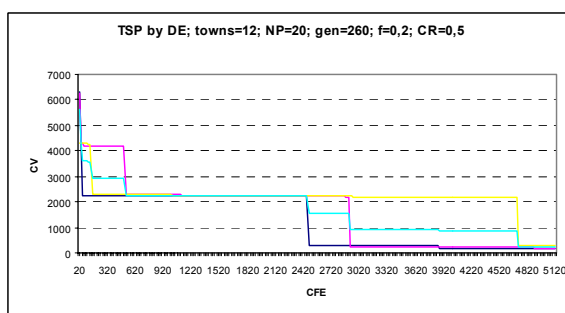
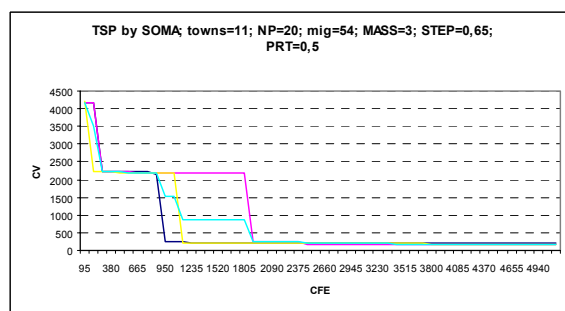
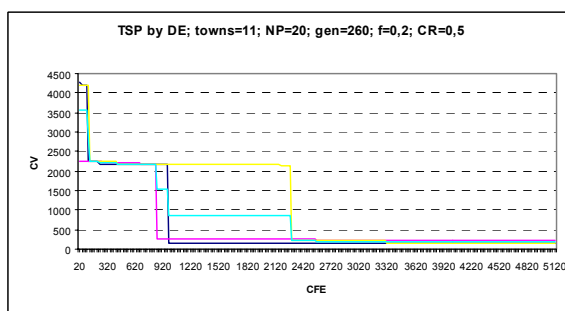
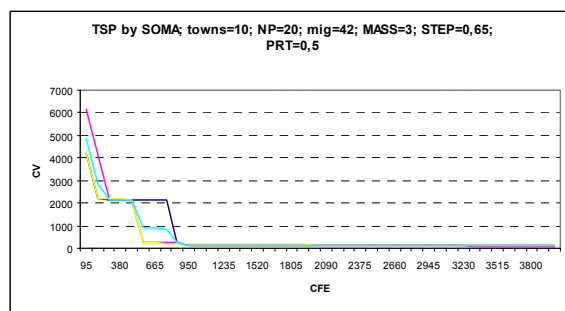
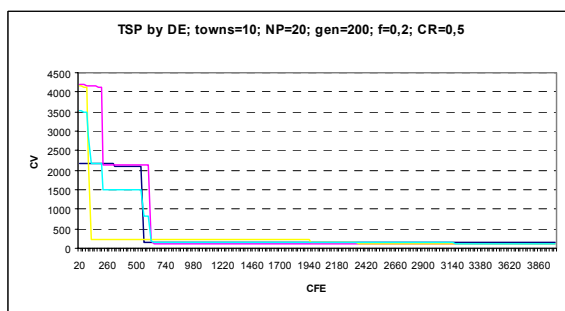
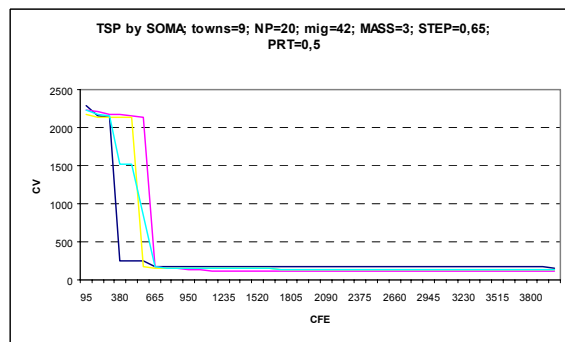
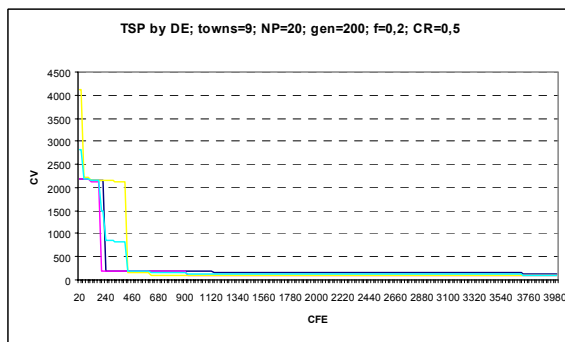
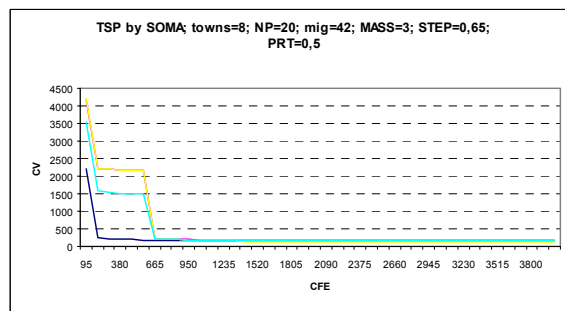
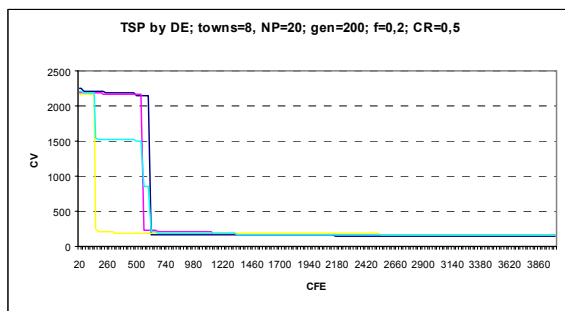
Graf 2

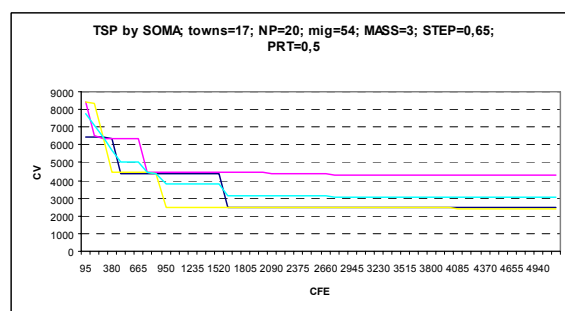
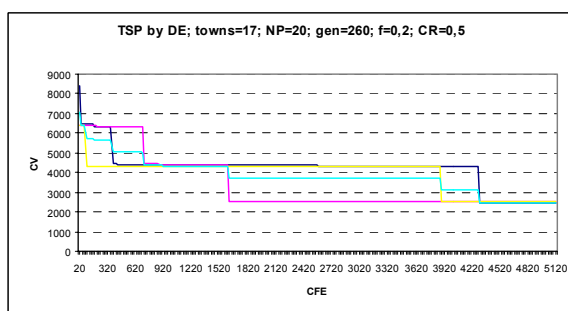
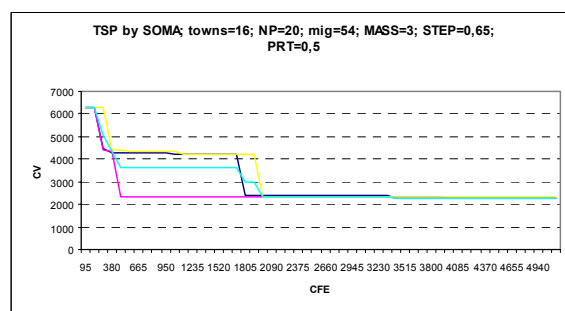
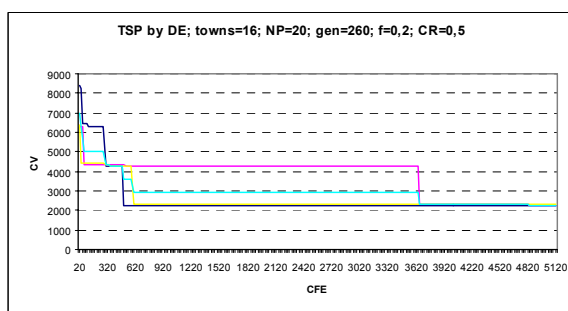
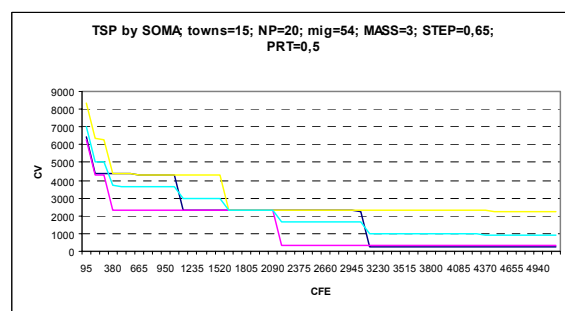
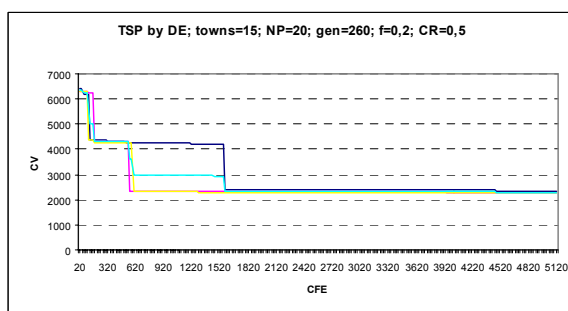
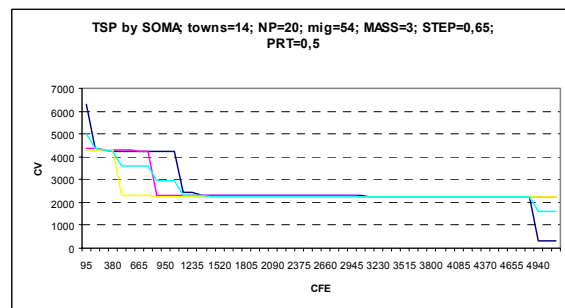
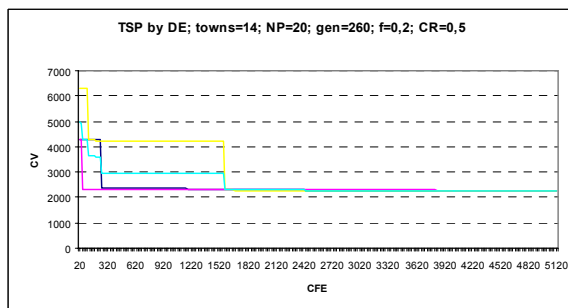
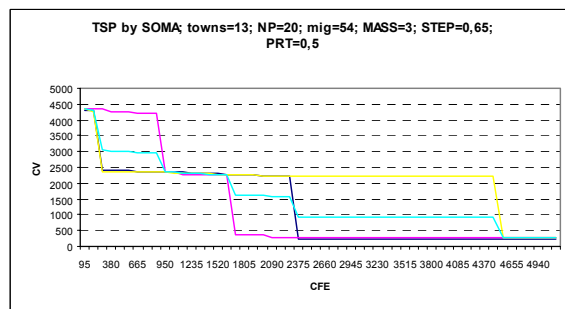
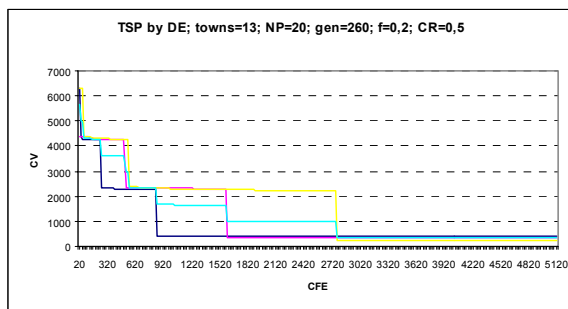
Grafická závislost vypočítané délky trasy, tedy hodnoty účelové funkce (CV) na počtu ohodnocení účelové funkce (CFE) je patrná z grafů Graf. 3-30. V grafech s počty měst 3-7 je z důvodu přehlednosti vyřezaná pouze počáteční dynamická oblast. Odpovídají tomu hodnoty na ose X, tedy ohodnocení účelové funkce (CFE).

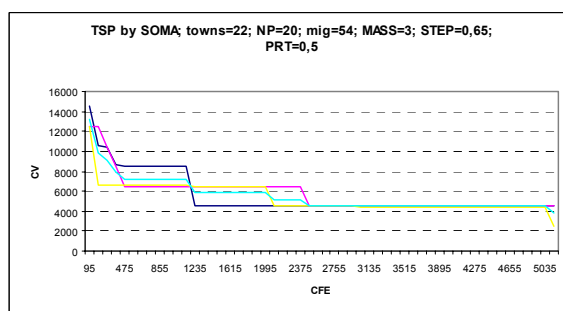
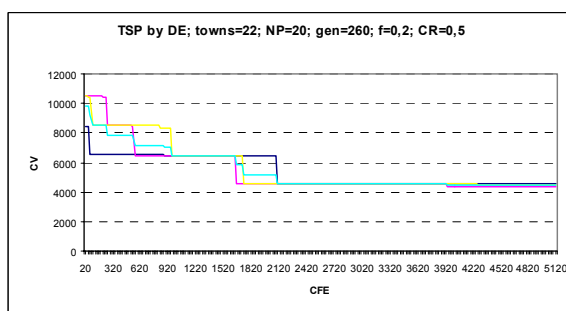
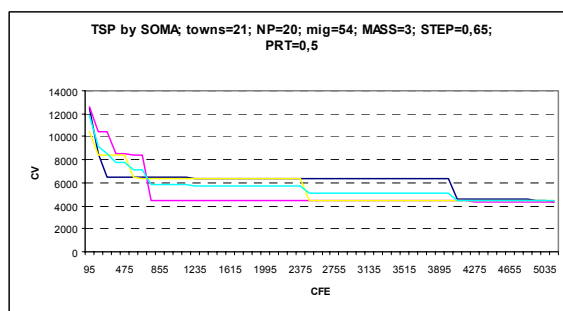
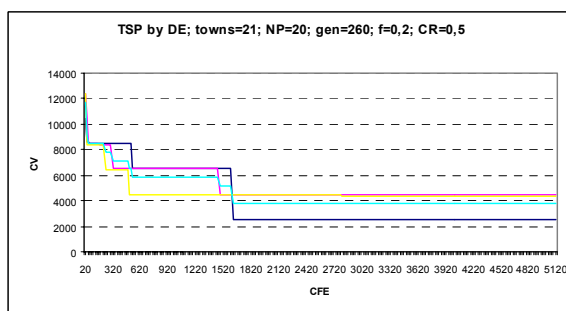
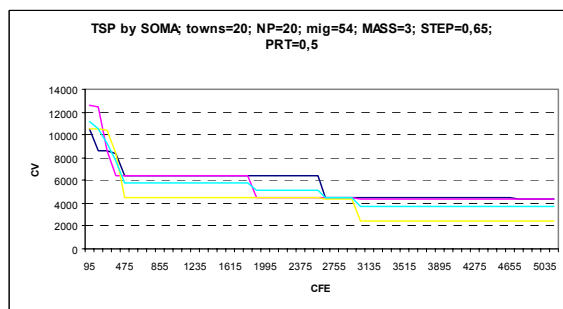
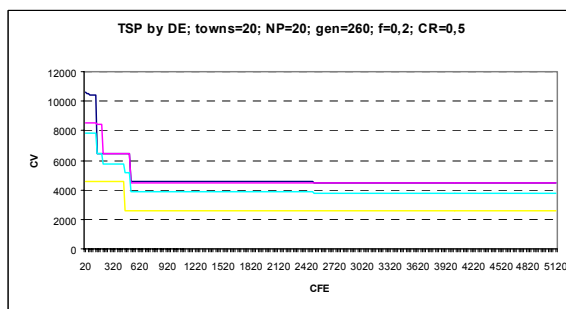
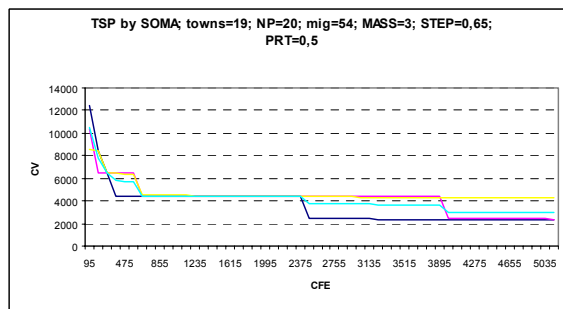
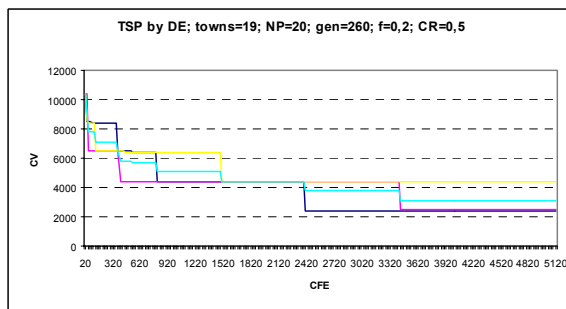
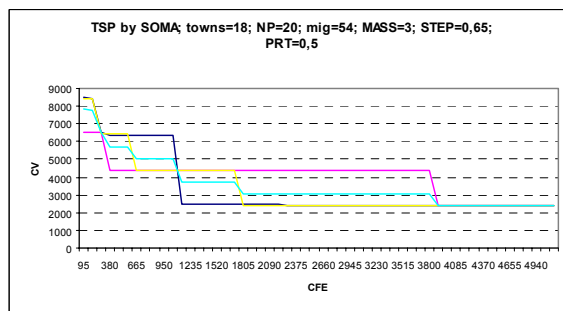
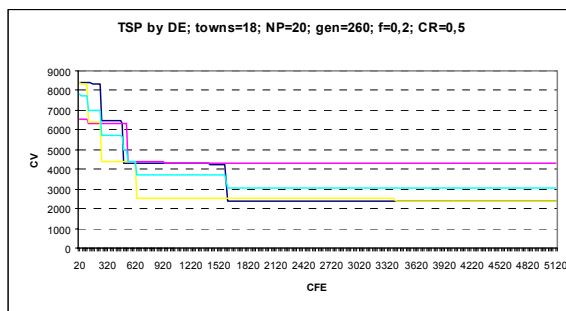
Legenda ke grafům Graf 3-30 :

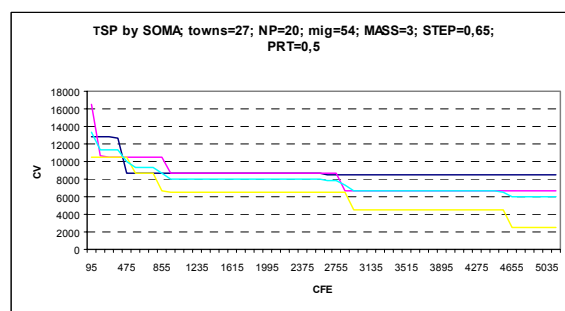
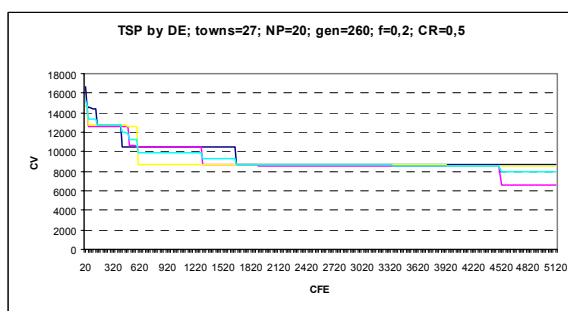
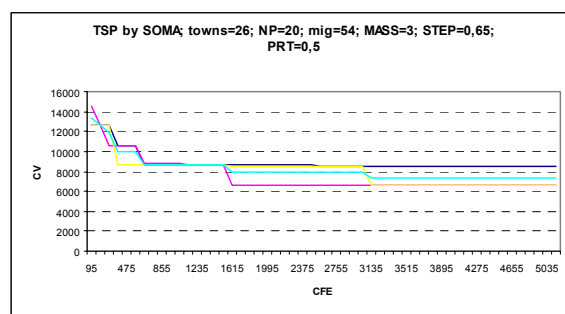
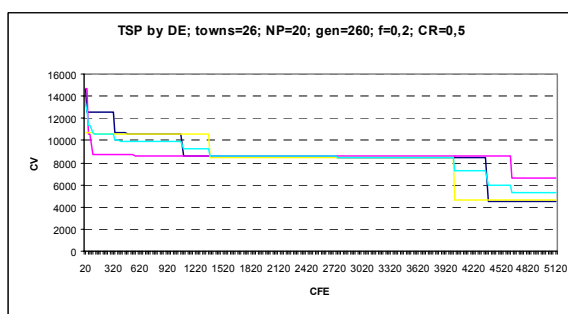
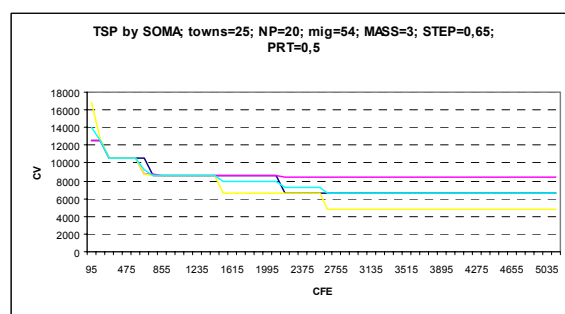
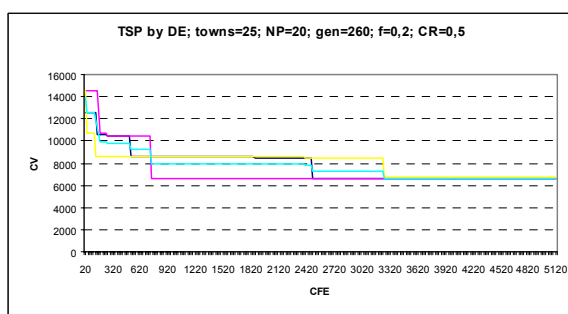
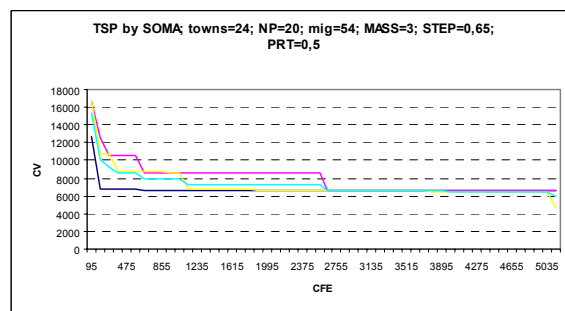
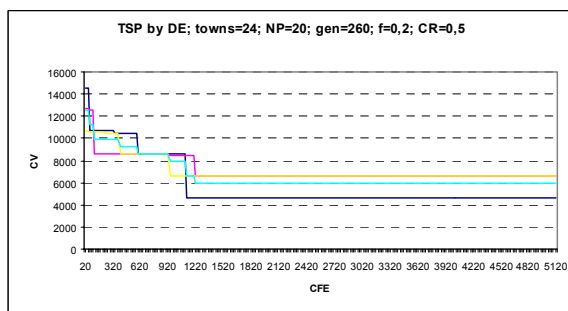
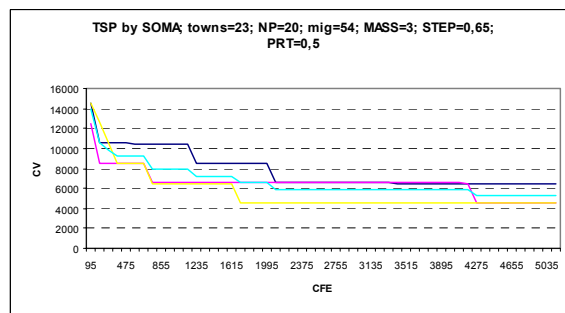
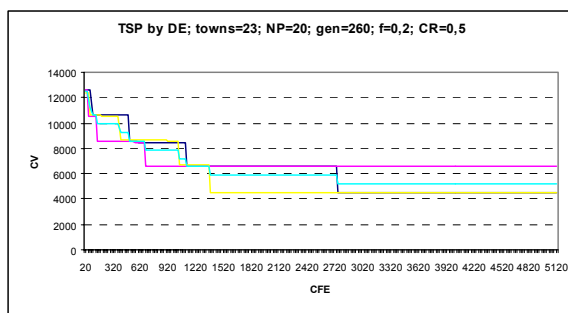


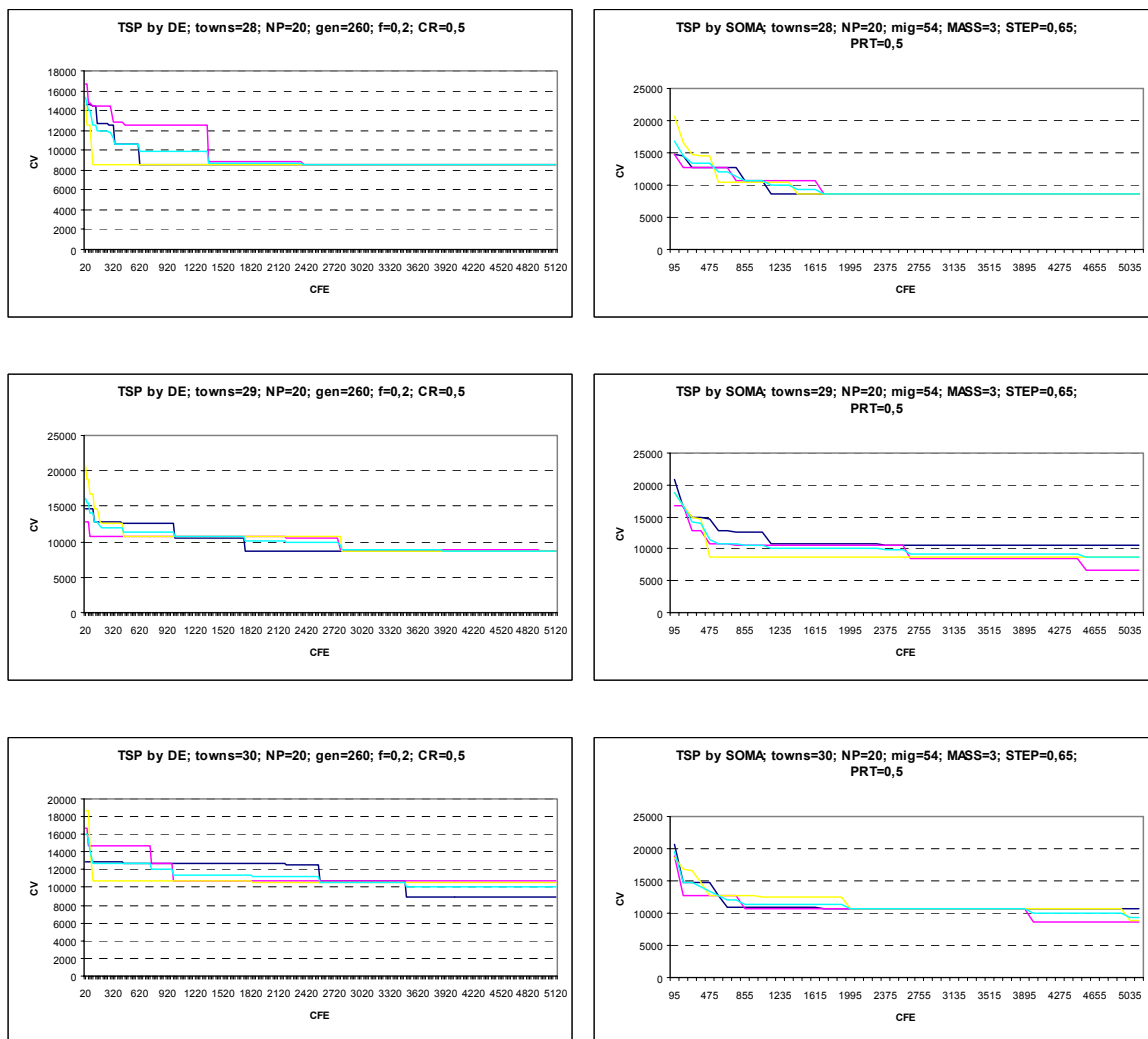












Grafy 3-30, průběh algoritmů DE resp. SOMA

Kompletní výsledky jsou zaznamenány v textových souborech, které jsou na přiloženém CD.

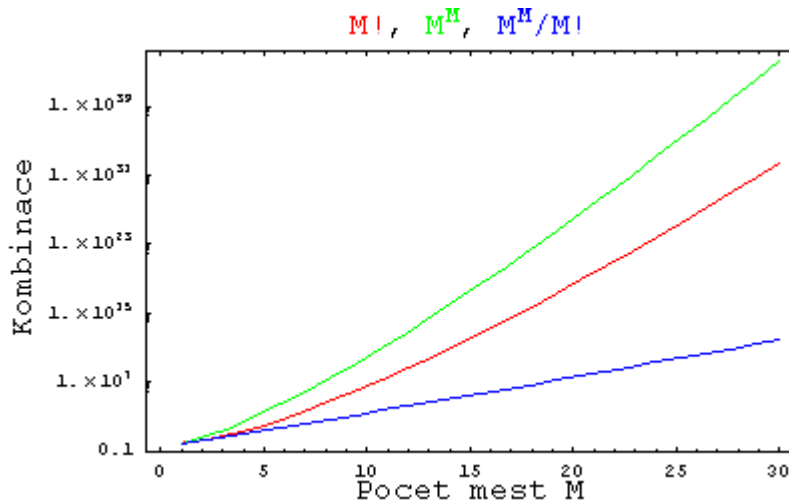
2.4.1 Hodnocení výsledků

Oba algoritmy tzn. Diferenciální Evoluce i SOMA jsou algoritmy, které nejsou nijak uzpůsobeny pro řešení problémů typu obchodního cestujícího. Je tedy jasné že bez nějaké úpravy nemohou konkurovat algoritmům které obsahují variace pro řešení právě takovýchto problémů. Ovšem i bez úpravy prokázaly schopnost konvergence a k vykazování „přijatelných“ výsledků. Systém penalizací se ukázal jako účinný nástroj a jako jedna z možností jak k takovému problému přistupovat.

Průběh obou algoritmů je samozřejmě silně závislý na zvolených parametrech. Pokud jde o samotné srovnání obou algoritmů pak mírně lepší výsledky vykazoval algoritmus SOMA. Po stejném počtu ohodnocení účelové funkce vracel ve 14 případech

z 28 kvalitnější trasu. U diferenciální Evoluce tomu bylo v 11 případech. Ve 3 případech našly oba algoritmy stejné řešení. Stejně řešení nacházely oba algoritmy pouze v případech nízkého počtu měst(3-5). V ostatních případech vrátil každý algoritmus jinou nejlepší hodnotu délky trasy. Dost vypovídající je i porovnání průměrů výsledků obou algoritmů. I zde mírně lepší výsledky vykazoval algoritmus SOMA. Z 28 případů vykázal SOMA v 15 případech lepší průměrnou hodnotu minimálních tras než Diferenciální Evoluce. Ta naopak vykazala lepší hodnotu než SOMA pouze v 10 případech. Ve třech případech vykázaly oba algoritmy stejné průměrné řešení. Opět to bylo zejména v případech s nízkým počtem měst.

Samozřejmě pokud jsou tyto algoritmy srovnány s nejlepšími optimalizačními metodami jakou je například Genetický algoritmus kde je použit operátor křížení s rekombinací hran – ERX, pak dostáváme v obou případech algoritmů mnohem horší řešení než je tomu právě u tohoto genetického algoritmu. Toto zjištění je zcela logické a nepřekvapující už z toho hlediska že prohledávací prostor algoritmů Diferenciální Evoluce resp. SOMA je o kardinalitě M^M kdežto u nejlepších metod přímo určených pro tuto úlohu je tento prostor „pouze“ o kardinalitě $M!$, kde M značí počet měst, tedy počet dimenzí účelové funkce. Tento rozdíl je patrný z logaritmického grafu 31.



Graf 31 Velikost prohledávacích prostorů o kardinalitě $M!$ a M^M

Další důvod je neexistence optimálních variací algoritmů Diferenciální Evoluce resp. SOMA, které by braly v úvahu specifika úlohy obchodního cestujícího, tak jak je tomu například u algoritmů genetických.

ZÁVĚR

Hlavním záměrem této diplomové práce bylo nalézt možná řešení úlohy obchodního cestujícího pomocí Evolučních algoritmů a dále pak navrhnout alternativní řešení v podobě algoritmů Diferenciální Evoluce a SOMA. V teoretické části byl vypracován přehled na poli NP problémů obecně a dále pak přehled optimalizačních algoritmů, které lze pro řešení těchto úloh použít. Jako ideální se jeví použití genetických algoritmů. Existuje totiž již několik variací tohoto algoritmu, které jsou přímo pro úlohu Obchodního cestujícího optimalizované. Jednoznačně nejlepší výsledky v tomto smyslu vykazuje genetický algoritmus, kde je použit operátor křížení s rekombinací hran – ERX. Principu tohoto algoritmu je tedy věnována značná pozornost v teoretické části práce. Stejná pozornost je zde věnována i algoritmům Diferenciální Evoluce a SOMA. Pomocí těchto algoritmů byl navržen alternativní přístup k řešení úlohy obchodního cestujícího.

V praktické části bylo vytvořeno několik funkcí, které algoritmy Diferenciální Evoluce a SOMA uzpůsobují k použití právě na úlohu obchodního cestujícího. Jako vývojové prostředí byl použit Matlab 6.0. Nově vytvořené funkce byly vsazeny do již existujícího toolboxu Evolučních algoritmů. Bylo vytvořeno i pohodlné grafické rozhraní pro snadnou obsluhu těchto algoritmů. Pomocí nově vzniklých funkcí byly algoritmy podrobeny důkladné analýze z důvodu nalezení optimálních parametrů a dále pak byly takto uzpůsobené algoritmy aplikovány na samotnou úlohu obchodního cestujícího. Byl získán poměrně široký soubor dat, který je tvořen jak výstupy tak i vstupy z jednotlivých měření. Na základě výsledků byly oba algoritmy porovnány.

Z výsledků měření jsou jasné i největší slabiny těchto algoritmů a tedy i to v jakém směru lze tuto práci v budoucnu rozvíjet. Pro smysluplné využití těchto algoritmů pro úlohu obchodního cestujícího je nezbytně nutné nalézt optimální variace uzpůsobené právě pro tuto úlohu tak, jak je tomu již nyní u jiných algoritmů zejména pak u algoritmů genetických. V tom případě pak mají algoritmy Diferenciální Evoluce a SOMA předpoklady k tomu, aby mohly konkurovat již ověřeným výpočetním metodám pro úlohu obchodního cestujícího.

SEZNAM POUŽITÝCH ZDROJŮ

- 1) Zelinka, Ivan. Umělá Inteligence I. Volume 1. Zlín: Vutium, Brno, 1958, 126 p. ISBN 80-214-1163-5
- 2) Zelinka, Ivan. Umělá Inteligence /kap.6 “Diferenciální evoluce“, Academia, 33p.
- 3) Kvasnička, V., Pospíchal J., Tiňo P. 2000, Evoluční algoritmy, STU Bratislava, ISBN 85-264-2000
- 4) Zelinka, Ivan. New Optimization Techniques in Engineering /kap.7 “SOMA – Self Organizing Migrating Algorithm, Springer-Verlag”
- 5) Zelinka, Ivan. 1999a. Aplikovaná informatika. Volume 1. Zlín: Ediční středisko, Fakulty technologické, VUT Zlín, 1999.
- 6) Lampinen J., Zelinka, I. 2000d, On Stagnation of Differential Evolution, Mendel 2000, 6th International Conference on Soft Computing, Brno, Czech Republic
- 7) Mařík, V., Štěpánková, O., Lažanský, J. 2001, Umělá Inteligence 3, Academia Praha, ISBN 80-200-0472-6
- 8) Evoluční algoritmy: biologická podstata [online]. [cit. 2004-10-15]. Dostupný z URL: < http://lucifer.fav.zcu.cz/uir/gen_alg2/Bioevol.htm>.
- 9) Úvod do TSP: Užití GA pro TSP [online]. [cit. 2004-10-12]. Dostupný z URL: < http://moon.felk.cvut.cz/~pjv/Jak/_info/i396/uvod.html#TSP>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

M – počet měst

$\mathfrak{O}$ – složitost algoritmu

p() – polynom

$\propto_p$ – polynomiální redukce

GA – Genetický algoritmus

DE – Diferenciální evoluce

ACO - Ant Colony Optimization

TSP – Travel salesman problem

SOMA – Samo organizující se migrační algoritmus

NP – počet jedinců v populaci

MIG – počet migrací

GEN – počet generací

MASS – dráha jedince k leaderovi

STEP – krok jedince k leaderovi

F – mutační konstanta

CR – práh křížení

PRT – vektor perturbace

SEZNAM OBRÁZKŮ

Obrázek 1 Možné uspořádání optimalizačních algoritmů	17
Obrázek 2 Princip horolezeckého algoritmu	18
Obrázek 3 Princip evoluční strategie	19
Obrázek 4 ACO algoritmus, princip	20
Obrázek 5 GA, princip činnosti	20
Obrázek 6 křivka pravděpodobnosti přijmutí nového stavu	21
Obrázek 7 Ukázkové funkce a, b	25
Obrázek 8 Princip algoritmu SOMA	31
Obrázek 9 GA, Jeden běh řešení úlohy obchodního cestujícího	39
Obrázek 10 Změny v hlavním okně UIEA	47
Obrázek 11 Okno TSP by DE	47
Obrázek 12 Okno Output options	48
Obrázek 13 Okno TSP by SOMA	48
Obrázek 14 SOMA, Pohyb jedince k leaderovi	51

SEZNAM TABULEK

Tab. 2 Princip diferenciální evoluce	26
Tab. 2 Jádro SOMA	33
Tab. 3 Dekódování ordinální reprezentace	35
Tab. 4 Výsledková tabulka (oblast příkladů s počty měst 3-11)	53
Tab. 5 Výsledková tabulka (oblast příkladů s počty měst 12-20)	53
Tab. 6 Výsledková tabulka (oblast příkladů s počty měst 21-30)	53
Tab. 7 Výsledková tabulka (oblast příkladů s počty měst 3-11)	55
Tab. 8 Výsledková tabulka (oblast příkladů s počty měst 12-20)	55
Tab. 9 Výsledková tabulka (oblast příkladů s počty měst 21-30)	55

SEZNAM PŘÍLOH

- P 1 : Ukázka kostry zdrojových kódů vybraných nově vytvořených funkcí.
- P 2 : CD-ROM s kompletními zdrojovými kódy vytvořených funkcí, kompletní výsledné data zaznamenané v textových souborech TXT.

PŘÍLOHY

Stěžejní část funkce TSP.m

```
function [delka] = tsp(cesta, vzdalenosti)
```

```
if ischar(vzdalenosti)
```

```
    fid=fopen(vzdalenosti);
```

```
    vzdalenosti=fscanf(fid,'%g ; ');
```

```
    fclose(fid);
```

```
end
```

```
pocet_mest=length(cesta);
```

```
delka=0;
```

```
pk=0;
```

```
%Kontrola opakování měst
```

```
for l=1:length(cesta)-1
```

```
    for o=l+1:length(cesta)
```

```
        if cesta(o)==cesta(l)
```

```
            delka=delka+2000;
```

```
            % return
```

```
        end
```

```
    end
```

```
end
```

```
%Výpočet délky cesty
```

```
for i=1:length(cesta)-1
```

```
    mn=[cesta(i) cesta(i+1)];
```

```
    m1=1; n1=2;
```

```
    j=0;
```

```
    for m1=1:pocet_mest;
```

```
        for n1=m1+1:pocet_mest;
```

```
            j=j+1;
```

```
            if [m1 n1]==mn | [n1 m1]==mn
```

```
                delka1=vzdalenosti(j);
```

```
            end
```

```
        end
```

```
    end
```

```
delka=delka+delka1;
```

```
end
```

Stěžejní část funkce AlgSoma.m

```

for m=1:mig

    if (mmax-mmin)<acceptederror
        m=mig;
        disp('acceptederror');
    end
    %-----HLEDANI LEADERA-----

    lmin=inf;
    for i=1:np
        if mat(1,i)<lmin
            lmin=mat(1,i);
            wh=i;
            lead=mat(2:d+1,wh);
        end
    end

    actmat(m)=mat(1,wh);

    for n=1:np

        x=mat(2:d+1,n);
        if n~=wh
            xm=zeros(d+1,floor(mass/step)+1);
            xm(2:d+1,1)=x;

            t1=2;
            for t=step:step:mass

                for p=1:d
                    r=rand;
                    if PRT<r
                        PRTv(p)=1;
                    end
                    if PRT>=r
                        PRTv(p)=0;
                    end
                end
            end
        end
    end
end

```

end

xm(2:d+1,t1)=x+(lead-x).((t.*PRTv)');*

for w=1:d

*if xm(w+1,t1)<intr(2*w-1)|xm(w+1,t1)>intr(2*w);*

*xm(w+1,t1)=rand*rozmer(w)+intr(2*w-1);*

end

if ri(w)==73; xm(w+1,t1)=round(xm(w+1,t1)); end

end

t1=t1+1;

end

xmin=inf;

for z=1:floor(mass/step)+1

j=0;

x=xm(2:d+1,z);

j=eval([vzorec]);

xm(1,z)=j;

if xm(1,z)<xmin

xmin=xm(1,z);

whxm=z;

end

end

if xmin<mat(1,n)

mat(:,n)=xm(:,whxm);

end

end

end

for npw=1:np

if mat(1,npw)<mmin

mmin=mat(1,npw);

whmi=npw;

end

end

for npw=1:np

if mat(1,npw)>mmax

```

    mmax=mat(1,npw);
    whma=npw;
end
end

```

```

if p_uiea | p_uism;
    cnt=m;
    if mod(cnt,bc)==0;
        dwait(cnt/cntn);
    end
end

```

```

end

```

```

for mi=2:(d+1)
    if ri(mi-1)=='I'
        mat(mi,:)=round(mat(mi,:));
    end
end

```

```

parms=mat(2:d+1,whmi);
kmat=mat

```

Stěžejní část funkce TSPgenv.m

```

function [pocvzd] = tspgenv(poc_mest,limit,filename)

% [filename,pocvzd] = TSPGENV(poc_mest,limit,filename)

%

% Funkce pro náhodnou generaci dat o vzdálenostech měst do souboru CSV

%

% Příklad:

%  tspgenv(4,10,'tspex04.csv')

pocvzd=0;

for i=1:poc_mest

    for j=i+1:poc_mest

        pocvzd=pocvzd+1;

        vzdalenosti(pocvzd)=round(rand*limit+1);

    end
end

```

end

fid=fopen(filename,'w');

vzdalenosti=fprintf(fid,'%g ; ',vzdalenosti);

fclose(fid);