

Algoritmus Scatter Search v programovém prostředí Mathematica

Algorithm Scatter Search in
program Mathematica

Jindřich Zdráhal

Bakalářská práce
2005

Univerzita Tomáše Bati ve Zlíně
Fakulta technologická

Univerzita Tomáše Bati ve Zlíně
Fakulta technologická
Institut řízení procesů a aplikované informatiky
akademický rok: 2004/2005

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: Jindřich ZDRÁHAL
Studijní program: B 3902 Inženýrská informatika
Studijní obor: Informační technologie

Téma práce: Algoritmus Scatter Search v programovém prostředí Mathematica

Zásady pro vypracování:

Evoluční algoritmy se používají na řešení poměrně obtížných problémů. Typickým představitelem moderních algoritmů jsou algoritmy jako je SOMA, DE, atp. Cílem DP bude sestavení programu algoritmu Scatter Search a provést ověření jeho činnosti na vybraných problémech. Postup DP je následující:

1. vypracovat přehled problematiky moderních evolučních metod
2. naprogramovat algoritmus Scatter Search
3. vybrat vhodné již řešené příklady na ověření jeho funkčnosti
4. Provést závěr

Rozsah práce:

Rozsah příloh:

Forma zpracování bakalářské práce: tištěná/elektronická

Seznam odborné literatury:

[1] ZELINKA, Ivan. Umělá inteligence I. Volume 1. Zlin: Vutium, Brno, 1998. 126 p. ISBN 80-214-1163-5.

[2] ZELINKA, Ivan. Umělá inteligence /kap.6 "Diferenciální evoluce", Academia, 33 p.

[3] Kvasnička V., Pospíchal J., Tiňo P. 2000, Evoluční algoritmy, STU Bratislava, ISBN 85-246-2000, 2000

[4] ZELINKA, Ivan, New Optimization Techniques in Engineering / kap. 7 "SOMA - Self Organizing Migrating Algorithm", Springer-Verlag

[5] David W. Corne, Morco Dorigo, Fred Glover, editors., New Ideas in Optimization - Mechanical Engineering Design Optimization by Differential Evolution. Volume 1. London: McGraw-Hill, 1999. 20p. ISBN 007-709506-5.

[6] Zelinka Ivan Umělá inteligence v problémech globální optimalizace, BEN, 2002, 190 p. ISBN 80-7300-069-5

[7] Ken Price, Rainer Storn, Jouni Lampinen, "Differential Evolution - A Practical Approach to Global Optimization", Springer, ISBN 3-540-20950-6

Vedoucí bakalářské práce:

Ing. Ivan Zelinka, Ph.D.

Institut řízení procesů a aplikované informatiky

Datum zadání bakalářské práce:

21. února 2005

Termín odevzdání bakalářské práce:

10. června 2005

Ve Zlině dne 18. února 2005

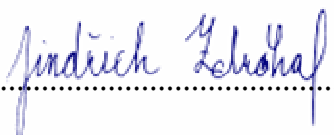

prof. Ing. Josef Šimoník, CSc.
děkan




prof. Ing. Vladimír Vašek, CSc.
ředitel institutu

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

Ve Zlíně, 01. 06. 2005


.....
jméno diplomanta

Abstrakt

Tato práce se snaží lehce nastínit dělení optimalizačních algoritmů, u některých významnějších je objasněn princip jakým pracují. Pak je pozornost soustředěna na algoritmus Scatter Search (rozptylové prohledávání). Jsou zmíněny jak myšlenky, které vedly k tvorbě tohoto algoritmu, tak principy na jejichž základě tento algoritmus pracuje. Ke konci práce je také popsána činnost algoritmu, jak jsem jej naprogramoval.

This work tries to sketch division of optimisation algorithms. There is explanation of fundaments of the most important algorithms. Main attention is directed to the Scatter Search algorithm. There are fundaments of this algorithm and ideas which were necessary for development of this algorithm. At the end of my work I try to explain how SS work.

OBSAH

ÚVOD	7
1 TEORETICKÁ ČÁST	8
1.1 PŘEHLED OPTIMALIZAČNÍCH ALGORITMŮ.....	8
1.1.1 Enumerativní algoritmy	8
1.1.2 Deterministické algoritmy.....	9
1.1.3 Stochastické algoritmy	9
1.1.4 Smíšené algoritmy	9
1.2 POPIS ČINNOSTI VYBRANÝCH ALGORITMŮ.....	10
1.2.1 Hill-Climbing Algorithm (Horolezecký algoritmus).....	10
1.2.2 Tabu Search (Zakázané prohledávání).....	10
1.2.3 Diferenciální evoluce.....	11
1.2.4 SOMA (SamoOrganizující se Migrující Algoritmus).....	12
1.3 ALGORITMUS SCATTER SEARCH (ROZPTYLOVÉ PROHLEDÁVÁNÍ)	13
1.3.1 Úvod k Scatter Search algoritmu	13
1.3.2 Princip fungování SS.....	15
1.3.3 Popis programu SS	18
2 EXPERIMENTÁLNĚ-SIMULAČNÍ ČÁST	25
2.1 SIMULACE NASTAVENÍ FUNKCE FINDMINIMUM	25
2.2 SIMULACE FUNKCE SS ALGORITMU PRO NĚKOLIK VYBRANÝCH FUNKCÍ..	27
2.2.1 Funkce použitá v příkladu naprogramovaném v C++	27
2.2.2 Rastriginova funkce	30
2.2.3 Griewangkova funkce.....	31
2.2.4 Druhá De Jongova funkce (Rosenbrockovo sedlo).....	32
2.2.5 Shrnutí simulací pro různé funkce.....	33
ZÁVĚR	34
SEZNAM POUŽITÝCH ZDROJŮ	36
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	37
SEZNAM OBRÁZKŮ	38
SEZNAM TABULEK	39

ÚVOD

Většina problémů se kterými se setkáváme v praxi buď je definována jako optimalizační problémy nebo se tak dají definovat. Se vzrůstající náročností a komplexností těchto problémů potřebujeme sofistikovanější metody k jejich řešení. Naštěstí tyto metody existují a jsou reprezentovány právě optimalizačními algoritmy.

Myslím že je to zajímavé vědní odvětví, které má široké uplatnění, navíc v dnešní době, která je podle mě posedlá snižováním nákladů a zvyšováním efektivity (což mimo jiné spadá do optimalizační problematiky) je toto odvětví perspektivní.

Rozhodl jsem se svou práci rozdělit na několik částí. V první části zmíním přehled optimalizačních algoritmů a jejich dělení. V další části bych chtěl popsat principy fungování vybraných algoritmů. Tyto algoritmy byly vybrány tak, že některé mají vazbu na algoritmus Scatter search, jiné ne, ale zdály se mi dostatečně důležité na to abych je uvedl.

V třetí části jsem se zaměřil na algoritmus Scatter Search. Nejprve na to z jakých myšlenek vznikl, pak jak byly tyto myšlenky převedeny na principy a pak na to jak byly z těchto principů vytvořeny metody. V této části by také mělo být vysvětleno vlastní fungování SS algoritmu. Dále bych si dovilil popsat činnost SS algoritmu, jak jsem jej naprogramoval.

Toto vše bylo v teoretické části. Na ni navazuje část experimentálně-simulační v níž se snažím ukázat jak úspěšně funguje SS algoritmus.

Nenapadlo mě lepší místo, kam bych měl uvést své poděkování, tak jej napíšu zde.

Chtěl bych poděkovat všem lidem, kteří mi pomohli při tvorbě této práce. Největší dík patří těm lidem s jejichž literaturou jsem pracoval a jejichž myšlenky jsem použil. Jmenovitě to jsou: Ivan Zelinka, Fred Glover, Manuel Laguna a Rafael Martí. Pak to jsou všichni ostatní uvedeni v bibliografii.

Pokud naleznete v této práci nějakou skvělou myšlenku tak je jisté, že pochází od někoho z nich. Pokud bych použil nejen myšlenku ale i slova těchto lidí a nedostatečně na to upozornil, pak se omlouvám a prohlašuji, že to není záměrně.

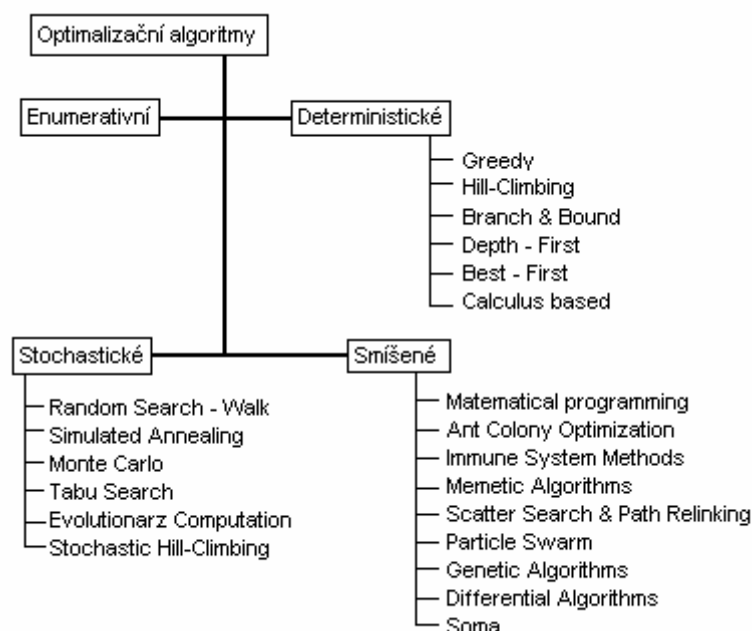
Doufám, že tato práce obohatí ty kteří ji čtou alespoň tak, jak obohatila mě když jsem ji psal.

1 TEORETICKÁ ČÁST

1.1 Přehled optimalizačních algoritmů

Optimalizační algoritmy slouží k nalezení minima dané účelové funkce tak, že hledají optimální numerickou kombinaci jejich argumentů. Tuto definici používá Zelinka (1, s. 20). Je možno dělit tyto algoritmy podle různých hledisek, ale já se budu ve své práci držet dělení podle Zelinky (1, s. 20), protože mi přijde logické a přehledné. Stejně tak si dovolím použít i charakteristiku jednotlivých algoritmů (1, s.21-23) a jejich funkci.

Algoritmy se dělí na čtyři třídy podle způsobu řešení daného problému. Viz obr. 1.



Obr. 1 Dělení optimalizačních algoritmů

1.1.1 Enumerativní algoritmy

Jedná se o matematický výpočet všech možných kombinací daného problému. Tento postup je vhodný pouze u funkcí, jejichž argumenty jsou diskrétní a nabývají jen malého množství hodnot. Na jiné funkce se nehodí z důvodu velké časové náročnosti.

1.1.2 Deterministické algoritmy

Jsou takové algoritmy, které používají metody klasické matematiky. Požadují ke své efektivní činnosti určité předpoklady:

- Funkce je lineární
- Funkce je konvexní
- Prohledávaný prostor možných řešení je malý
- Prohledávaný prostor možných řešení je spojitý
- Účelová funkce má pokud možno jen jeden extrém (je unimodální)
- Mezi parametry na účelové funkci nejsou nelineární interakce
- Jsou dostupné doplňující informace jako např.: gradient
- Problém je definován v analytickém tvaru
- Tato metoda dává jako výsledek jediné řešení.

1.1.3 Stochastické algoritmy

Algoritmy této skupiny jsou založeny na využívání náhody. V podstatě jde o náhodné hledání argumentů funkce. Algoritmus si zapamatuje to nejlepší řešení na které během své činnosti přišel a to je použito jako výsledek.

Tyto algoritmy bývají většinou pomalé. Vhodné jen pro funkce s malým rozsahem argumentů. Jsou vhodné pro hrubý odhad.

1.1.4 Smíšené algoritmy

Jedná se o kombinaci metod předchozích dvou skupin algoritmů, které se vzájemně doplňují. Tyto algoritmy jsou:

- Robustní, nezávisle na počátečních podmínkách často naleznou kvalitní řešení
- Naleznou řešení během relativně krátké doby
- Nemají takové požadavky na předběžné informace

- Pracují i s problémy typu černá skříňka – tzn.: není potřeba analytického popisu problému
- Jsou schopny nalézt více řešení během jednoho spuštění

1.2 Popis činnosti vybraných Algoritmů

Myslím, že by bylo vhodné stručně popsat činnost některých algoritmů. Jedná se o algoritmy, které souvisí nebo předcházejí algoritmu Scatter Search. Také jsem se rozhodl uvést dva nejvýznamnější evoluční algoritmy. Jmenovitě to jsou diferenciální evoluce a SOMA. Diferenciální evoluci jsem si vybral z důvodu její rozšířenosti a taky toho, že ostatní algoritmy jsou s ní často porovnávány. Somu jsem si vybral proto, že mi dovolila pochopit podstatu optimalizačních algoritmů.

1.2.1 Hill-Climbing Algorithm (Horolezecký algoritmus)

Patří mezi gradientní algoritmy. Prohledává prostor možných řešení ve směru největšího spádu. To je také jeho nevýhodou, protože velmi často uvázne v lokálním extrému. Základní verze vychází z náhodného bodu v prostoru možných řešení. Pro toto řešení se navrhne okolí a daná funkce se zkoumá jen v tomto okolí. Získané lokální řešení se pak použije jako střed pro výpočet nového okolí. Celý proces se pak opakuje. Algoritmus si zaznamená nejlepší nalezené řešení a to pak použije jako konečný výsledek.

Stochastická verze tohoto algoritmu pracuje stejně jako normální verze. Jen postupně opakuje prohledávání z více náhodných míst.

Nevýhodou tohoto algoritmu je možnost zacyklení v místech lokálních extrémů.

1.2.2 Tabu Search (Zakázané prohledávání)

Jedná se o vylepšenou verzi Horolezeckého algoritmu. Vylepšení spočívá v tom, že zde byla zavedena krátkodobá a dlouhodobá paměť.

Krátkodobá paměť slouží k uchovávání informací o použitých transformacích, které byly použity pro výpočet aktuálního středu. Tyto transformace jsou pak zakázány. Proto se algoritmus jmenuje Zakázané prohledávání.

Dlouhodobá paměť si zase pamatuje nejčastěji používané transformace. Tyto transformace jsou pak penalizovány. To má za následek, to, že se pak používají méně.

Toto vylepšení se projevuje tím, že algoritmus na rozdíl od Horolezeckého algoritmu tak často neuvízne v lokálních extrémech.

1.2.3 Diferenciální evoluce

Tento algoritmus patří k novějším algoritmům. Roku 1995 jej vyvinuli a použili Ken Price a Rainer Storm. Svou činností je velmi podobný genetickým algoritmům. Má s nimi několik společných rysů:

- o Jednoduchost.
- o Používání jedinců.
- o Používání populací.
- o Používání mutací a křížení.

Má však také několik odlišností.

- o Nepracuje s binárními čísly, ale s dekadickými.
- o Z toho vyplývá, že nepoužívá logické operace na binárních číslech, ale operace s vektory.
- o Místo 2 rodičů se používají 4.
- o Pořadí akcí je mutace a pak křížení (u genetických algoritmů je tomu naopak).
- o Nezávislost křížení na kvalitě rodičů.

Nový jedinec se u tohoto algoritmu tvoří tak, že se vezme jeden jedinec a k němu se náhodně vyberou 3 jiní (různí) jedinci. Mutací těchto tří jedinců se vytvoří tzv. šumový vektor. Tento vektor se zkříží se 4. doposud nepoužitým jedincem. Takto vytvořený jedinec nahradí v populaci tohoto 4. jedince.

Standardně se ukončí tento algoritmus tehdy, když se vytvoří uživatelem definovaný počet populací. Nicméně je možno vymyslet a použít spoustu jiných podmínek.

1.2.4 SOMA (SamoOrganizující se Migrující Algoritmus)

Tento algoritmus se dá zařadit do dvou skupin, záleží jen na úhlu pohledu, který použijeme. Dá se zařadit mezi evoluční algoritmy, protože na konci migračního kola jsou jedinci umístěni na nové pozice (což odpovídá fungování evolučních algoritmů). Druhý pohled je podle způsobu, jakým se jedinci pohybují, tedy na princip fungování algoritmu. Podle tohoto pohledu se algoritmus řadí mezi memetické algoritmy. Memetické algoritmy mohou být definovány jako algoritmy strategie soutěže a spolupráce, což popisuje jejich princip.

Algoritmus SOMA je podobný s jinými algoritmy tím, že na začátku náhodně generuje populaci. Nicméně na této populaci neprovádí mutace ani křížení. Místo toho používá „pohyb“ jedinců po ploše řešení. Tento pohyb záleží na zvolené strategii.

- All to one – zjistí se jedinec s nejmenší hodnotou účelové funkce a ten je označen jako „leader“ ostatní jedinci se k němu snaží přiblížit.
- All to rand – pro každého jedince se vybere jeden „leader“ (náhodně bez ohledu na vhodnost) a k němu tento jedinec zamíří.
- All to all – každý zkusí cestovat ke všem a pak se přesune na nejlepší místo ze všech cest.
- All to all adaptivně – jedinec jde jednou cestou k náhodně zvolenému jedinci. Na této cestě najde nejvhodnější místo a z něj vyrazí na cestu k dalšímu jedinci a to pak opakuje dokud jako cíl nepoužije všechny jedince. Toto provedou všichni jedinci.

U všech strategií mohou jedinci urazit jen předem stanovenou dráhu a na ní se pohybují v předem definovaných skocích. Na své cestě si zapamatuje nejvýhodnější místo a tam se přesune. Jakmile toto provedou všichni jedinci, tak je konec migračního kola a celý postup se opakuje, dokud populace nesplní jednu z podmínek. Ty jsou většinou tyto: je dosažen definovaný počet kol migrace nebo dosáhnou předem definovaného maximálního rozdílu mezi jedinci.

Všem zájemcům o tento algoritmus bych doporučil ke shlédnutí stránky jeho autora I. Zelinky (2), ze kterých jsem čerpal, stejně tak jako z jeho přednášek.

1.3 Algoritmus Scatter Search (rozptylové prohledávání)

Předně bych chtěl poděkovat, že M. Laguna na svých internetových stránkách (3) popsal jednoduchou a srozumitelnou formou problematiku SS algoritmu. Tyto stránky také slouží jako oficiální stránky SS algoritmu a také jako rozcestník pro ostatní stránky s touto problematikou. Musím zdůraznit, že jsem používal tyto stránky hojně. Myslím si totiž, že nejsem tak dobrý, abych dokázal vysvětlit fungování SS lépe než lidé, kteří této problematice obětovali celý život. Zkusím jejich myšlenky obléct do svých slov a nevím jestli se mi to povede, takže pokud se tady vyskytnou nějaké chyby tak bych poprosil o shovívavost.

1.3.1 Úvod k Scatter Search algoritmu

Jeden z důležitých principů, na kterých SS staví, je způsob kombinace různých metod řešení. Takže bychom se mohli podívat na metody, které jsou založené na zlepšování řešení za pomoci kombinace rozhodovacích pravidel a které byly podkladem pro vznik SS

Rozvoj těchto metod je spojen s plánovacími metodami použitými pro získávání lepších pravidel rozhodování při řešení „Job shop scheduling problems“. Myšlenka byla jednoduchá, stvořit systém číselných vah pro kombinování existujících pravidel.

Tato koncepce byla motivována myšlenkou, že informace o relativní vhodnosti různých možností řešení je uložena pro různá pravidla v různé formě. Pokud by se vytvořil ucelený systém, který by byl seříděn podle možností různých kombinací mechanismů, tak by se tyto informace mohly využívat efektivněji. A bylo by možno na ně aplikovat běžné strategie výběru použitých vhodných pravidel.

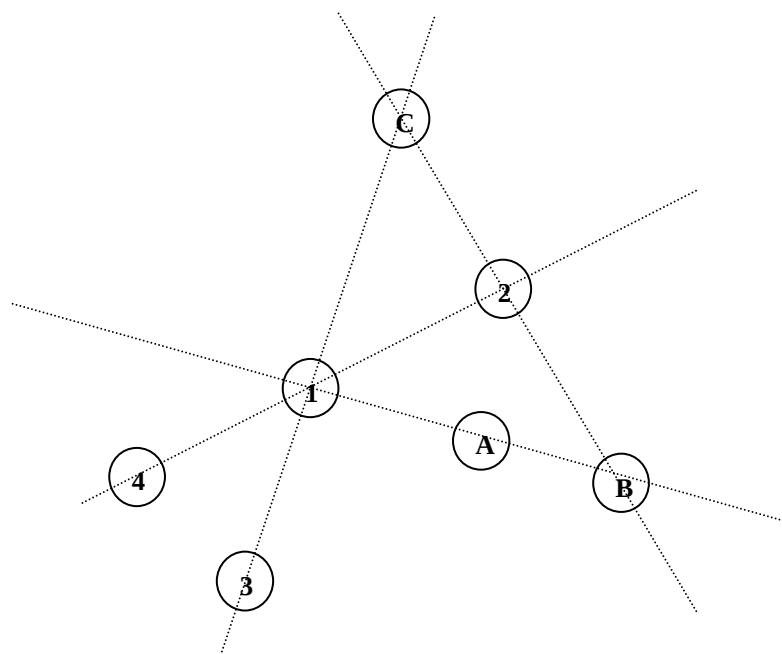
Tato metoda se navíc liší od ostatních metod v tom, že se po dosažení lokálního optima nezastaví, ale místo toho pokračuje s obměněnými parametry, které ovlivňují kombinaci pravidel a používá tato pravidla pro konstrukci dalších zkušebních řešení. Tato strategie se stala jednou ze základních myšlenek při tvorbě Tabu Search algoritmu.

Rozhodovací pravidla tvořená kombinací strategií produkují lepší výstupy než standardní aplikace lokálních rozhodovacích pravidel.

Nedlouho po úspěšném uvedení strategie řešení, založené na kombinaci pravidel rozhodování, byly podobné procedury použity na kombinaci omezení. Stejný mechanismus byl použit k tvorbě číselných kombinačních vah (použitím nezáporných čísel) na tvorbu nových omezení. Tato omezení byla nazvána „náhradní omezení“. Byly použity izolované podmnožiny omezení. Tyto podmnožiny byly kritičtěji prověřeny. Toto prověřování bylo uskutečněno podle vypočítaných zkušebních řešení, založených na náhradních omezeních a produkci nových vah, které odrážely stupeň toho, jak uspokojily nebo neuspokojily jednotlivá omezení. Kombinace omezení nachází částečná uplatnění v oboru celočíselného a nelineárního programování. Princip fungování náhradních omezení společně s metodami pro kombinaci rozhodujících pravidel poskytuje způsob, jak získávat možnosti, které mohou být použity k tvorbě a modifikaci zkušebních řešení. Z této úvahy vychází různé heuristické postupy tvořené používáním náhradních omezení a výpočtem zkušebních řešení na jejich základě.

V SS algoritmu se vyskytují dva doplňující se postupy, první z nich je kombinace pravidel rozhodování a druhým postupem je kombinace omezení.

SS pracuje s množinou řešení, tzv. „referenční množina“. Tvorba nových řešení se děje kombinací řešení z této množiny. Pokud se jako hlavní mechanismus pro kombinaci řešení vezme takový, že nové řešení je tvořeno lineární kombinací dvou jiných řešení, tak se referenční množina může vyvíjet jako na obrázku 2. Tento obraz zahrnuje originální referenční množinu, kolečka označená A, B a C. Po konkávní kombinaci řešení označených A a B je stvořeno řešení 1. Ačkoliv bylo stvořeno více zkušebních řešení tak pouze toto řešení bylo přidáno do referenční množiny. Podmínky pro zařazení do této množiny budou probrány později. Stejným způsobem konvexní a konkávní kombinací originálních a nových řešení stvoříme body 2, 3 a 4. Celá referenční množina se nyní skládá ze 7 řešení (nebo prvků).



Obr. 2 Ukázková množina řešení SS

SS nenechává řešení v surovém stavu, jak je produkuje kombinační mechanismus, ale kandidáty pro vložení do referenční množiny nechává projít heuristickým vylepšením. Tento postup bude důkladněji probrán dále.

Na rozdíl od populace v genetických algoritmech je referenční množina v SS relativně malá. GA vybere dvě řešení z populace a zkombinuje je, v tomto procesu hraje velkou roli náhoda. Běžná populace v GA se může skládat ze 100 prvků. Na rozdíl od toho SS vybere 2 řešení a zkombinuje je, ale náhoda zde hraje až druhořadou úlohu a podřizuje se zvolené kombinační strategii. Typická referenční množina v SS se skládá z 20 a méně řešení.

1.3.2 Princip fungování SS

SS procesy jsou založeny na myšlence náhradních omezení, které nejsou uchovávány odděleně v jednotlivých vektorech. Další důležitá myšlenka je ta, že se používá výhod heuristické analýzy k získávání výhodných kombinací a k tvorbě nových vektorů.

Jednotlivé kroky použité při práci SS by se daly zapsat takto:

- 1) Tvorba startovací množiny řešení. Je snaha o to, aby řešení v této množině byla co nejrozdílnější. Tzn. pokud vezmeme každé řešení jako samostatný vektor, tak je snaha

o to aby byla vzdálenost těchto vektorů co největší. Pokud máme zaručenu různost, tak ještě zbývá použít „vylepšení“ těchto řešení za pomoci heuristické metody, která je vhodná pro náš problém. Do referenční množiny se vezmou nejvhodnější prvky. Tady není jediným kritériem hodnota účelové funkce. Prvek může být zařazen do referenční množiny pokud je hodnota účelové funkce menší než hodnoty v referenční množině, ale také tehdy pokud se tím zvýší různorodost prvků v množině.

- 2) Tvorba nových řešení. Tato tvorba se provádí tak, že se použije strukturovaná kombinace podmnožin z nynější referenční množiny. Strukturální kombinace jsou:
 - a) Výběr bodů k produkci uvnitř i venku konvexní oblasti zastoupené referenčními řešeními.
 - b) Změna k získání vhodných řešení.
- 3) Aplikace heuristických postupů použitých v kroku 1 k získání lepších řešení vytvořených v kroku 2. Tyto heuristické postupy musí být schopny pracovat s takovými řešeními, které nejsou vhodné (např.: funkce není pro dané řešení definována) a může, ale nemusí získat vhodné řešení.
- 4) Vytáhnout nejlepší řešení z kroku 3 a přidat je k referenční množině. Pojem nejlepší se opět musí chápat v trochu širších souvislostech. Opakovat kroky 2, 3 a 4 dokud se referenční množina mění. Obměňovat referenční množinu opětovným stratem od kroku 1. Zastavit se po dosažení předem definovaného počtu iterací.

První pozoruhodný rys v SS je, že strukturální kombinace je navržena tak, aby vznikly vážená centra vybraných podmnožin. Toto přidává konkávní kombinace tak, že navrhuje nová centra v oblastech, které jsou mimo původní referenční řešení (např.: řešení 3 v Obr. 2). Tento vzor, tvořený centry a jejich projekcí do vnějších oblastí je užitečný v několika aplikačních oblastech.

Jiný důležitý rys se týká strategie pro výběr jednotlivých podmnožin řešení pro kombinaci podle kroku 2. Tyto strategie jsou navrženy tak, aby využili shlukování k poskytnutí nového řešení „uvnitř shluku“ a „za shlukem“. Metoda je organizována k užívání dodatečných metod vylepšení, které jsou schopny pracovat s nevhodnými řešeními, odstraňují omezení,

že řešení musí být vhodné (musí existovat) v souladu s tím, že je zahrnuto referenční množině.

Tyto principy shrnují základní metody SS:

- Užitečné informace o optimálním řešení jsou většinou obsaženy v odpovídající různorodé sbírce vybraných řešení.
- Když jsou řešení kombinována strategií pro získávání těchto informací, tak je důležité, aby byl mechanismus schopný získat takové kombinace, které jsou i mimo oblast již posouzených řešení. Stejně tak je také důležité, aby jednotlivé heuristické procesy použily tato řešení ke kombinaci na další řešení. Důvodem těchto kombinací je jak různorodost, tak i kvalita.
- Získávání několika řešení naráz jako základ pro tvorbu kombinací, zvyšování možnosti získat informace obsažené v sjednocení nejlepších řešení

Fakt, že mechanismus SS není omezen pouze na jednotný návrh umožňuje získat strategické možnosti, které mohou dokázat efektivnost v jednotlivých použitích SS. Tato pozorování a principy vedly k následujícímu modelu pro implementaci SS.

1. Metoda různorodého generování slouží k tvorbě souboru různorodých zkušebních řešení. Používá dohodnuté zkušební řešení (nebo počáteční řešení) jako vstup.
2. Zlepšující metoda slouží k transformaci zkušebního řešení na jedno nebo více vylepšených řešení. Ani vstup ani výstup nemusí být řešitelný, nicméně u výstupu se očekává, že bude. Pokud se po provedení této metody nenalezne vylepšené řešení, pak se použije stejné řešení jako bylo na vstupu.
3. Metoda aktualizace referenční množiny je potřeba k vytvoření a udržování referenční množiny, které se skládá z b nejlepších řešení, která byla nalezena (kde b je většinou malé číslo, málokdy větší než 20). Tato množina je organizována tak, aby poskytla ostatním metodám co nejlepší přístup k datům v ní obsaženým. Řešení se mohou dostat do této množiny pro jejich kvalitu a různorodost.
4. Metoda generování podmnožin pracuje na referenční množině a produkuje podmnožiny řešení z této množiny jako podklad pro kombinování řešení.

5. Metoda kombinace řešení je určena k tomu aby z podmnožiny řešení dodávané metodou generování podmnožin kombinovala jedno nebo více vektorů řešení.

1.3.3 Popis programu SS

Na tomto místě bych chtěl opět poděkovat za to že pánové F. Glover, M. Laguna a R. Martí popsali nejen teoreticky jak SS pracuje, ale dokonce popsali toto fungování na příkladu naprogramovaném v C++ (4). Vzhledem k tomu, že prostředí Mathematica, v němž jsem měl tento algoritmus realizovat se dost liší od programovacího jazyka C++, tak jsem mohl využívat tento program jen jako velmi volné vodítko.

Jak jsem již psal v předchozí kapitole, tak se SS algoritmus skládá z 5 metod. Nyní bych chtěl postupně probrat všechny tyto metody a nakonec by bylo možno shrnout jak se tyto metody „skládají dohromady“ aby z nich byl funkční program.

Ještě bych rád uvedl, že funkce, pro kterou se hledá minimum je tato funkce:

$$f(x)=100(x_2-x_1^2)^2+(1-x_1)^2+90(x_4-x_3^2)^2+(1-x_3)^2+10,1((x_2-1)^2+(x_4-1)^2)+19,8(x_2-1)(x_4-1) \quad (1)$$

Tato funkce bylo použita i pro demonstraci programu v C++ a tak jsem ji použil také, mimo jiné i pro kontrolu funkčnosti mého programu. Avšak jak vyplývá z grafů (ty jsou uvedeny při simulacích v kapitole 2.2.1), tak se mi zdá tato funkce poněkud málo „základná“. Kromě toho, že se u ní nachází nepatrné naznačení sedla, tak se mi zdá poměrně jednoduchá. Může však skýtat jiná úskalí, nicméně já jsem je neodhalil.

Omezení pro tuto funkci bylo, že proměnné budou nabývat pouze hodnot -10 až 10.

1.3.3.1 Metoda různorodého generování (*diversifikační metoda*)

Cílem této metody je vygenerovat různorodou množinu řešení. Tato množina se používá jako startovací množina. Aby bylo dosaženo potřebné různorodosti, které bychom nedosáhli použitím generátoru pseudonáhodných čísel, tak je použit tento postup.

Rozsah v němž se nalézají možná řešení se rozdělí na čtyři intervaly. Nejprve se náhodně vybere první interval. Z něj se teprve náhodně generuje řešení. Tento interval je potom

penalizován při dalším generování a tak má menší šanci být vylosován znovu, to se opakuje do té doby dokud nenaplníme požadavky pro startovní množinu.

Tabulka I. Rozložení pravděpodobností do jednotlivých intervalů

Interval	X_1	X_2	X_3	X_4
-10 až -5	26	20	26	26
-5 až 0	23	23	17	32
0 až 5	25	24	24	21
5 až 10	26	33	33	21

Tabulka 1 ukazuje kolikrát byl který interval generován pro startovní množinu. Vzhledem k tomu, že množina měla 100 prvků a z tabulky je jasné patrné, že žádný interval se neopakoval více než 33-krát a méně než 17-krát a žádaný interval 25 se objevil jednou tak můžeme říct, že tato metoda pracuje uspokojivě.

Pro každé řešení byla ihned spočítána i hodnota hledané funkce (1). A tato hodnota nabývá hodnot od 55,8701 do 1768772,4 při průměrné hodnotě 444100.

Tabulka II. Prvních deset řešení vygenerovaných diversifikační metodou

Řešení	X_1	X_2	X_3	X_4	$f(x)$
1	9.28026	-8.74071	-2.328	6.41449	725273.
2	0.608701	1.78185	1.83785	-0.143	1312.19
3	-9.83663	6.77817	1.67906	4.84196	154805.
4	-3.44711	8.00747	-4.71976	-7.65202	395574.
5	1.58812	6.91459	-1.71699	5.26963	168786.
6	-2.30584	1.30511	9.32222	3.15067	631469.
7	-9.95792	-3.95913	5.16503	1.59624	95495.9
8	4.96957	-8.57204	1.06433	-8.0036	684232.
9	-3.71107	-6.61028	-1.36739	-9.49624	268004.
10	4.76259	7.76769	5.54996	-2.40754	75771.

1.3.3.2 Zlepšující metoda

V mém vzorovém programu (4) byla použita Nelder and Mead's zjednodušující metoda (metoda pro vyhledávání lokálních extrémů).

Nicméně já jsem měl tuto metodu zjednodušenou tím, že program Mathematica má vlastní implementovanou funkci pro vyhledávání lokálních minim „FindMinimum“. Nechal jsem původní nastavení, kde si tato funkce vybírá automaticky použitou metodu. Jen jsem změnil počet iterací pro tuto metodu z původních 100 na 10. Důvodem pro to byla skutečnost, že

tato metoda byla tak účinná, že i přesto, že měla vyhledávat lokální minima, tak se jí velmi často podařilo najít globální minimum a tak se nedal program řádně otestovat.

Z důvodu fungování této funkce jsem stanovil podmínku, že všechna řešení, která se budou optimalizovat musí být řešitelná.

Tabulka III. ukazuje jak se zkušební řešení změnila po provedení této metody.

Tabulka III. Řešení upravená zlepšující metodou

Řešení	X_1	X_2	X_3	X_4	$f(x)$
1	9.2637	-0.17042	-2.51387	6.30396	259.69
2	0.664049	0.917286	0.909235	0.841628	1.2986
3	-9.69708	-0.204318	2.14312	4.31406	175.341
4	-3.41106	0.619488	1.42794	-1.57374	1287.64
5	1.58394	-0.127637	-2.1215	4.77688	91.6083
6	-2.30124	1.19242	1.88242	3.71719	104.951
7	-7.8294	-0.0746856	1.65819	2.49924	87.1419
8	4.95443	2.12864	-0.12401	-4.98142	3081.76
9	-3.65382	1.78823	1.69292	1.62238	379.909
10	4.75463	1.20146	-0.343446	-1.30912	250.102

Minimální hodnota je 1,2986 maximální hodnota je 3081,76 a průměr je 571,944. Při porovnání s tabulkou II. je patrné zlepšení. Pokud by byla metoda použita s původními 100 iteracemi, tak bychom již v tuto chvíli měli globální minimum.

Tato při svém hledání opustí prostor povolených řešení. Program je nastaven tak, že hlídá a pokud se tak stane, tak toto řešení vrátí zpět do prostoru povolených řešení.

Algoritmus používá tuto metodu pokaždé, když vygeneruje nová zkušební řešení a to ať už metodou diversifikační nebo metodou kombinační.

1.3.3.3 Aktualizace referenční množiny

Algoritmus SS má množinu, kde se uchovávají nejvhodnější řešení. Pro rozhodování o vhodnosti řešení jsou dvě kritéria.

- První kritérium se vztahuje k hodnotě optimalizované funkce.
- Druhé kritérium sleduje „různorodost“ prvků.

Celá tahle množina se nazývá referenční množinou a obsahuje b prvků. Skládá se ze dvou podmnožin. První podmnožina uchovává prvky s nejlepší hodnotou a má b_1 prvků, druhá

podmnožina uchovává naopak prvky nejvzdálenější od ostatních a má b_2 prvků. Platí tedy, že:

$$b = b_1 + b_2 \quad (2)$$

V našem příkladu je nastaveno $b_1=3$ a $b_2=2$. Zaplnit první podmnožinu z tabulky III. nebude žádným problémem. Je hned patrné, že se do této množiny dostanou tato řešení:

Tabulka IV. Řešení uložená do referenční množiny podle hodnot $f(x)$

Řešení	X_1	X_2	X_3	X_4	$f(x)$
2	0.664049	0.917286	0.909235	0.841628	1.2986
7	-7.8294	-0.0746856	1.65819	2.49924	87.1419
5	1.58394	-0.127637	-2.1215	4.77688	91.6083

Tato řešení se také vymažou z množiny možných řešení a pro zbylá řešení je nutno spočítat klasickou euklidovskou vzdálenost od všech řešení, která jsou uložena v referenční množině. Podle tohoto požadavku se do druhé podmnožiny referenční množiny dostala tato řešení:

Tabulka V. Řešení uložená do referenční množiny podle vzdáleností

Řešení	X_1	X_2	X_3	X_4	$f(x)$
1	9.2637	-0.17042	-2.51387	6.30396	259.69
8	4.95443	2.12864	-0.12401	-4.98142	3081.76

Je samozřejmé, že se referenční množina musí doplňovat. Pokaždé když se vygenerují nová řešení díky zlepšující metodě, tak jsou porovnána s referenční množinou. Pokud je řešení lepší než nejhorší řešení z podmnožiny nejlepších řešení, tak jej nahradí. Pokud je řešení vzdálenější než nejméně vzdálené řešení z množiny různorodých řešení, tak jej nahradí. Tím se udržuje referenční množina neustále aktualizovaná.

1.3.3.4 Generování podmnožin

Generování podmnožin nám slouží k výběru prvků, které budeme kombinovat aby nám vznikla nová pokusná řešení. Tyto prvky vybíráme z referenční množiny. Samozřejmě, že je několik postupů, jak tvořit podmnožiny. V našem případě však bude postačovat nejjednodušší postup a to ten, že vezmeme všechny možné dvojice. Čímž z naší referenční množiny o 5 prvcích získáme 10 dvojic.

1.3.3.5 Kombinování prvků

Stejně jako pro generování podmnožin, tak i pro kombinování prvků může být spousta různých postupů. Tyto postupy většinou vycházejí ze znalosti řešené problematiky. Protože jsem se rozhodl algoritmus zkoušet na stejné funkci jako pánové kteří jej naprogramovali v C++, tak jsem se rozhodl držet se jejich příkladu a použít stejný postup jako oni.

V této fázi programu se berou dvojce, které jsme si v předcházejícím kroku vygenerovali a za pomoci lineární kombinace se hledá nové zkušební řešení. V mnou použité metodě se nejprve zjistí do kterých podmnožin referenční množiny kombinované prvky patří a podle toho se s nimi provedou určité operace. Tyto operace jsou tři a jsou to tyto:

$$C1: \quad x = x' - d$$

$$C2: \quad x = x' + d$$

$$C3: \quad x = x'' + d$$

Kde $d = r(x'' - x')/2$ a r je náhodné číslo v rozsahu 0 až 1.

Pro výběr které lineární kombinace se použijí jsou tato pravidla:

- Pokud x' a x'' jsou prvky podmnožiny nejlepších řešení (první podmnožina), tak se provedou C1 a C3 jednou a C2 dvakrát. Tzn. získáme 4 zkušební řešení.
- Pokud je x' nebo x'' prvkem podmnožiny nejlepších řešení, pak se provedou tyto kombinace: C1, C2 a C3 každé jednou. Tím získáme 3 zkušební řešení.
- Pokud ale x' ani x'' nejsou prvky podmnožiny nejlepších řešení, pak se provede kombinace: C2 jednou a náhodně se vybere jestli se použije C1 nebo C3. Tím získáme 2 řešení.

Díky tomuto postupu získáme v našem vzorovém příkladu celkem 32 nových zkušebních řešení.

1.3.3.6 Návaznost kroků v SS

Nyní máme probrány jednotlivé kroky použité v algoritmu SS a teď bychom se mohli podívat jak jsou řazeny za sebou a jak jsou použity cykly.

Nejprve bych ale rád objasnil zkratky, které budou použity.

Psize	startovní množiny, kterou nám vygeneruje diversifikační metoda
P	množina zkušebních řešení
b	velikost referenční množiny
b1	velikost podmnožiny referenční množiny ve které jsou uložena nejlepší řešení
b2	velikost podmnožiny referenční množiny ve které jsou uložena různorodá řešení
MaxIter	maximální počet iterací
RefSet	referenční množina
RefSet1	podmnožina referenční množiny s nejlepšími řešeními
Refset2	podmnožina referenční množiny s nejrozumnějšími řešeními

Tady je nástin funkce mnou naprogramovaného SS algoritmu

- 1) Začni s $P = \emptyset$, použijeme diversifikační metodu k tvorbě zkušebních řešení x . Toto se opakuje tak aby $|P| = Psize$
- 2) Použijeme na všechna x zlepšující metodu a dostaneme $x \rightarrow x^*$, tímto prvkem nahradíme původní prvek.
- 3) Setřídíme řešení v P podle hodnot účelové funkce
For ($Iter = 1$ **to** $MaxIter$)
 - 4) Stvoř $Refset = RefSet1 \cup RefSet2$ z P kde $|RefSet| = b$, $|RefSet1| = b1$ a $|RefSet2| = b2$. Vezmi $b1$ nejlepších řešení z P a vlož je do $RefSet1$, pak pro všechna $x^* \in P$ spočítej jejich vzdálenost od všech prvků $RefSet$. Seřaď tyto prvky podle vzdálenosti a vyber $b2$ nejlepších a vlož je do $Refset2$.
 $NovePrvky = TRUE$**While** ($NovePrvky = TRUE$) **do**
 - 5) Vygeneruj podmnožiny
 - 6) Zkombinuj prvky a tyto prvky vlož do P
 - 7) Použij zlepšující metodu $x \rightarrow x^*$
 - 8) $StaryRefSet = Refset$
 - 9) $NovePrvky = FALSE$**For** ($i = 1$ **to** $|P|$)
 - If** ($x_i^* \notin RefSet1$ a hodnota účelové funkce x_i je lepší než hodnota účelové funkce nejhoršího prvku $RefSet1$) **then**
 - 10) Přidej x_i^* do $RefSet1$ a smaž nejhorší prvek této množiny
 - 11) $NovePrvky = TRUE$
 - Else**

```

If ( $x_i^* \notin RefSet2$  a vzdálenost  $x_i^*$  od prvků je lepší než nejhorší
prvek v  $RefSet2$ ) then
12) Přidej  $x_i^*$  do  $RefSet1$  a smaž nejhorší prvek této
množiny
13)  $NovePrvky = TRUE$ 
End If
End If
End For
End While
If ( $Iter < MaxIter$ ) then
14) Stvoř novou množinu  $P$  za pomoci diversifikační metody.
15)  $P = P \cup RefSet$ 
End If
End For

```

2 EXPERIMENTÁLNĚ-SIMULAČNÍ ČÁST

Nyní se pokusím na simulacích předvést a dokázat funkčnost mnou naprogramovaného SS algoritmu. Tady bych chtěl upozornit na jednu skutečnost, zatímco v jiných algoritmech se za iteraci bere určitý cyklus algoritmu, tak v mých simulacích se za jednu iteraci bere použití metody pro aktualizaci referenční množiny. Tato metoda je za jeden cyklus, který se bere standardně jako iterace pro SS, volána i několikrát a tak se mi zdá směřodatnější za iteraci brát použití této metody.

V tabulkách uvedených u jednotlivých funkcí se počítá vždy s konečnou hodnotou SS (tzn. s tím, co nám algoritmus vypíše jako výsledek po 100 iteracích). Jako procentuelní úspěšnost se bere kolikrát se podařilo algoritmu najít globální minimum (procentuelně).

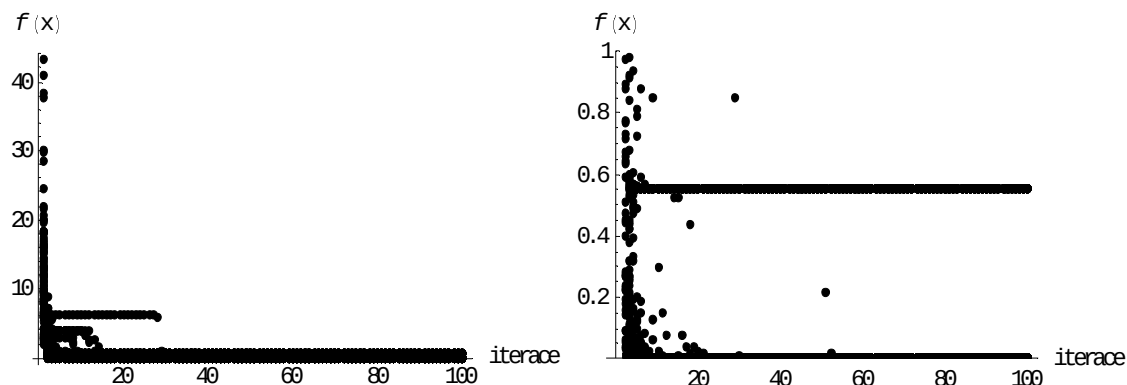
2.1 Simulace nastavení funkce FindMinimum

Nejprve jsem se rozhodl znázornit závislost výsledků na počtu opakování funkce „FindMinimum“. Kterou používám pro vyhledání lokálního minima. Tato funkce je přímo naprogramována v *Mathematice*. Funkce ke své práci používá 5 metod: "ConjugateGradient", "Gradient", "LevenbergMarquardt", "Newton" a "QuasiNewton" a má standardně nastaven automatický výběr toho, kterou funkci použije. Toto nastavení jsem neměnil. Další důležité nastavení je počet iterací pro tuto funkci. Standardní nastavení je na 100 iterací, což mi přišlo pro potřeby mých simulací velmi mnoho, jelikož algoritmus našel optimální řešení velmi brzo a tak by tyto výsledky nebyly vhodné pro účely mé simulace.

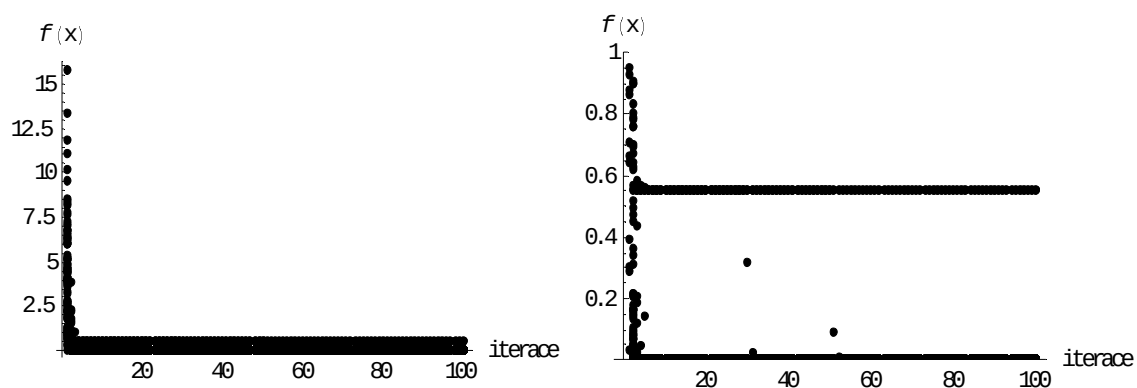
Toto jsem se rozhodl předvést na příkladu první funkce.

Jedná se o funkci (1) použitou již v příkladu v kapitole 1.3.3 a která je znovu připomenuta v kapitole 2.2.1, kde se nachází i její grafy. V této kapitole (1.3.3) jsem psal o tom, že se mi zdá tato funkce málo záludná, že kromě náznaku sedla, je poměrně jednoduchá. Ze simulací však vyplývá, že i toto nepatrné sedlo je záludné.

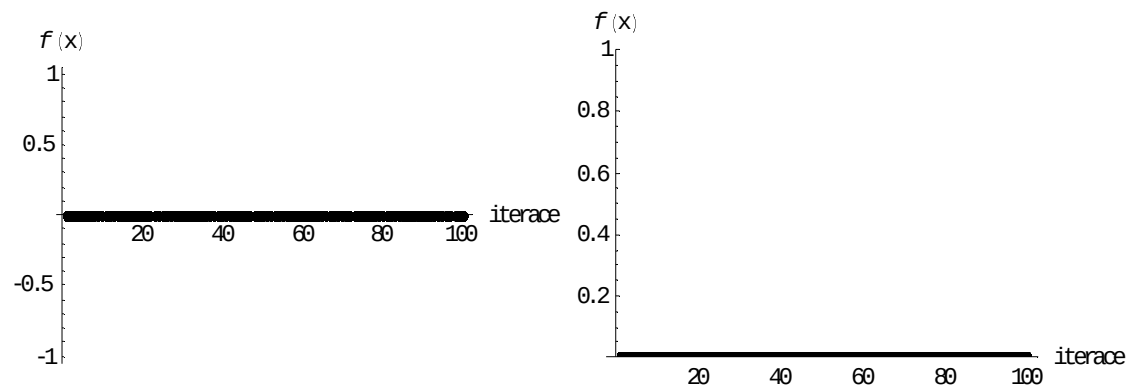
Ve všech mnou prováděných simulacích jsou shodné tyto parametry: počet iterací algoritmu je 100 a počet spuštění algoritmu je také 100. Osa x v grafu udává počet iterací algoritmu a osa y udává velikost účelové funkce.



Obr. 3 Simulace pro funkci (1), počet iterací pro „FindMinimum“ je 5



Obr. 4 Simulace pro funkci (1), počet iterací pro „FindMinimum“ je 10



Obr. 5 Simulace pro funkci (1), počet iterací pro „FindMinimum“ je 100

Tabulka VI. Úspěšnost SS pro funkci (1) při různém nastavení FindMinimum

Počet iterací	5	10	100
Maximální hodnota	0.5566697007	0.5566697003	0.
Minimální hodnota	0.	0.	0.
Průměrná hodnota	0.0612336670	0.0111333940	0.
Procentuelní úspěšnost	89%	98%	100%

Z grafů a této tabulky je zřejmé, že pokud se nastaví počet iterací pro funkci FindMinimum na malou hodnotu, tak se snižuje úspěšnost algoritmu, naopak to platí samozřejmě také. Čím větší číslo, tím větší úspěšnost. Nicméně si myslím, že pokud se nastaví příliš velká hodnota, tak se výpočty nejen neúměrně prodlouží, ale také se nějak ztrácí hodnota algoritmu, protože jak je zřejmé z poslední simulace, tak funkce FindMinimum nastavená na 100 iterací si nejen poradí s řešenou funkcí, ale udělá tak hned v prvním kroku. Tím pádem nedá šanci aby se algoritmus předvedl.

Na základě těchto simulací jsem se rozhodl, že pro další simulace využiji nastavení FindMinimum na 10. Jsem si tím samozřejmě vědom, že tím utrpí úspěšnost algoritmu, nicméně simulace budou průkaznější.

2.2 Simulace funkce SS algoritmu pro několik vybraných funkcí

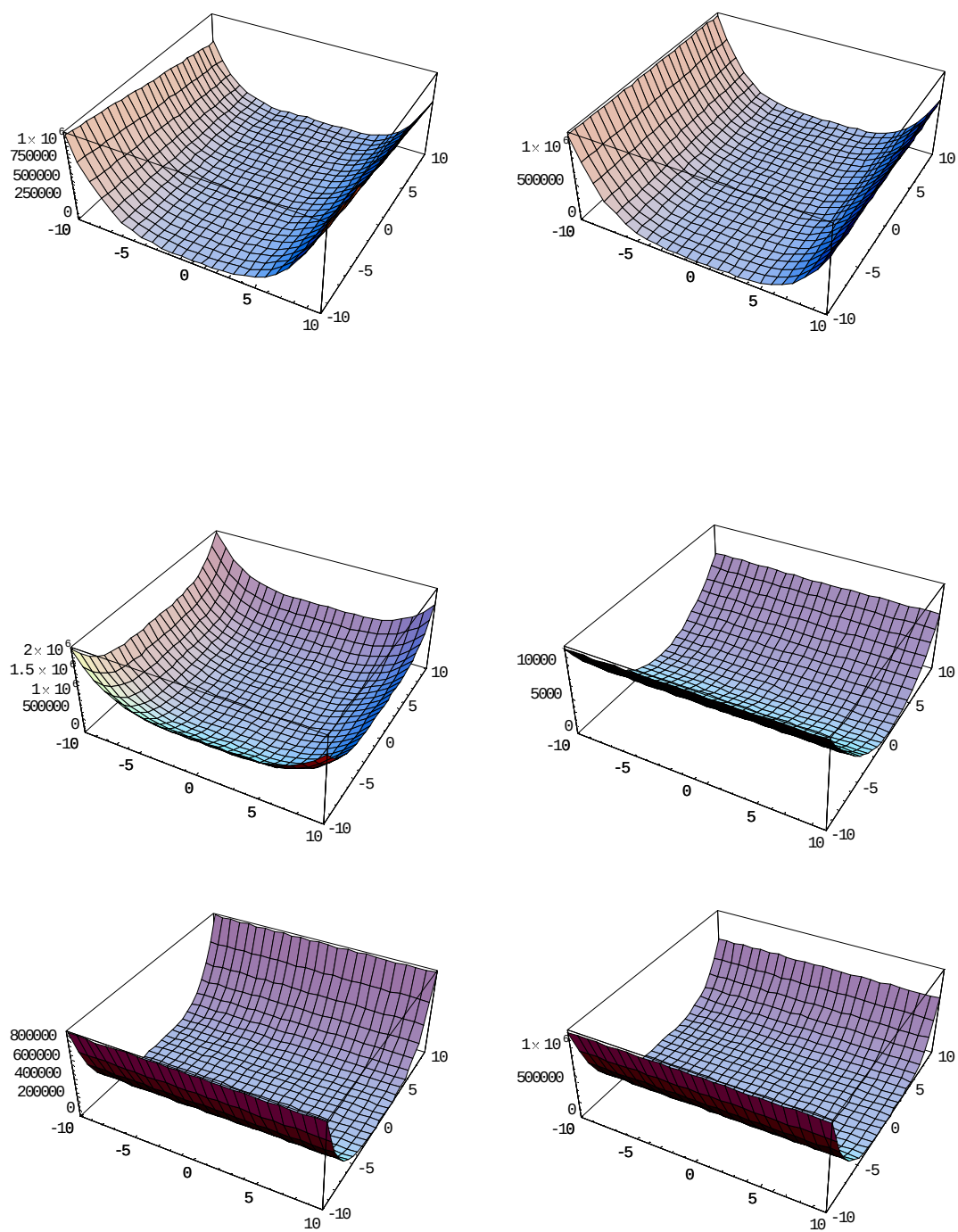
V této kapitole bych chtěl provést postupně simulace pro několik různých funkcí. I když jsou simulace pro různé funkce, tak několik věcí mají společných. V první řadě to jsou následující skutečnosti. Počet iterací pro funkci FindMinimum je 10. Počet iterací pro SS algoritmus je 100. Počet spuštění SS algoritmu je také 100.

2.2.1 Funkce použita v příkladu naprogramovaném v C++

Tuto funkci jsem použil už ve dvou kapitolách (1.3.3 a 2.1) a tak si myslím, že bych ji měl trochu přiblížit. Jelikož její zápis:

$$f(x)=100(x_2-x_1)^2+(1-x_1)^2+90(x_4-x_3)^2+(1-x_3)^2+10,1((x_2-1)^2+(x_4-1)^2)+19,8(x_2-1)(x_4-1) \quad (3)$$

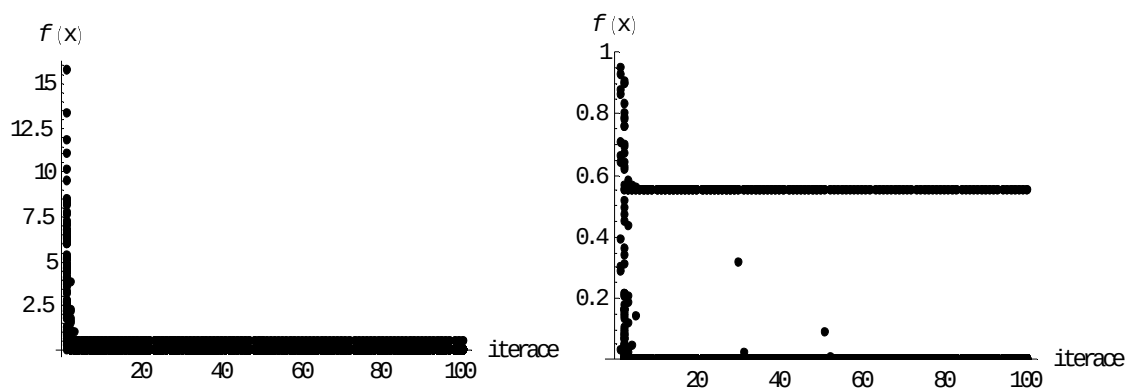
nevypovídá nic o jejím průběhu, tak jsem si připravil několik grafů, které snad nastíní průběh této funkce. Funkce je kromě malého sedla poměrně jednoduchá.



Obr. 6 Grafy účelové funkce (1)

Nevím jak bych měl lépe tuto funkci přiblížit.

Tento graf ukazuje jak se dařilo SS algoritmu na této funkci:



Obr. 7 Simulace pro funkci (1)

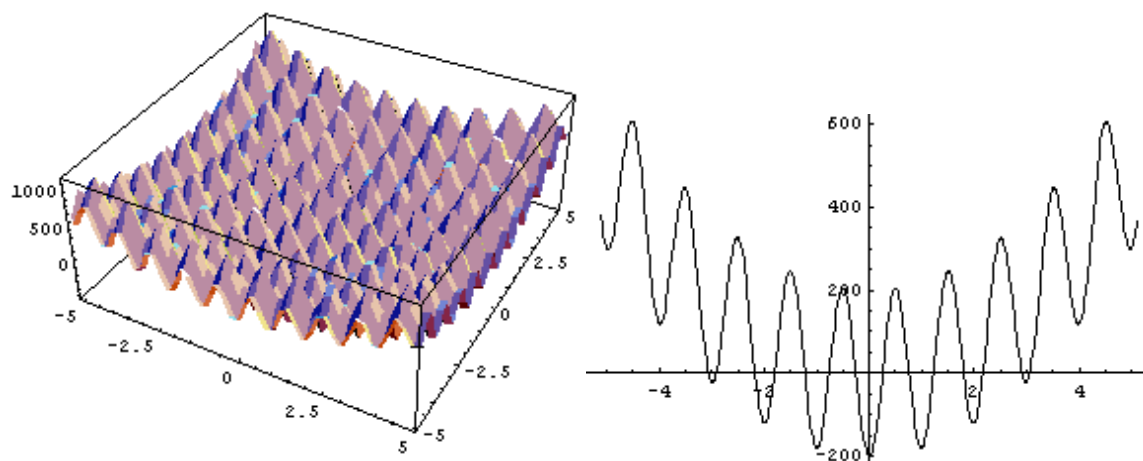
Tabulka VII. Úspěšnost SS pro funkci (1)

Maximální hodnota	0.5566697003
Minimální hodnota	0.
Průměrná hodnota	0.0111333940
Procentuelní úspěšnost	98%

2.2.2 Rastriginova funkce

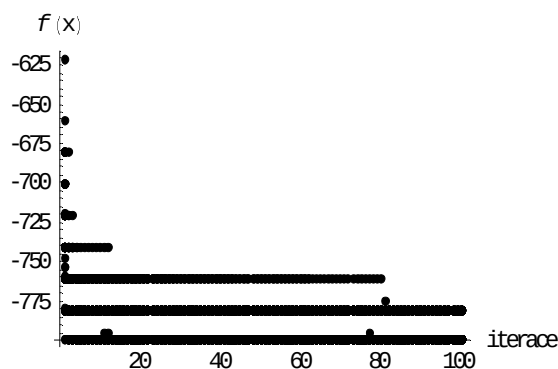
$$20 \left(\sum_{i=1}^{\text{Dim}} (x_i^2 - 10 \cos(2\pi x_i)) \right) \quad (4)$$

Jedná se o funkci kterou jsem (stejně jako všechny následující funkce) přebíral z literatury (6). Následující grafy ji snad trochu přiblíží.



Obr. 8 Graf Rastriginovy funkce

Následující graf ukazuje jak se dařilo algoritmu na této funkci, už z grafu je patrné, že kromě globálního extrému ($f(x) = -800$) se algoritmus také zasekával v lokálním extrému ($f(x) = -780,1$).



Obr. 9 Simulace pro Rastriginovu fci

Tabulka VIII. Úspěšnost SS pro Rastriginovu funkci

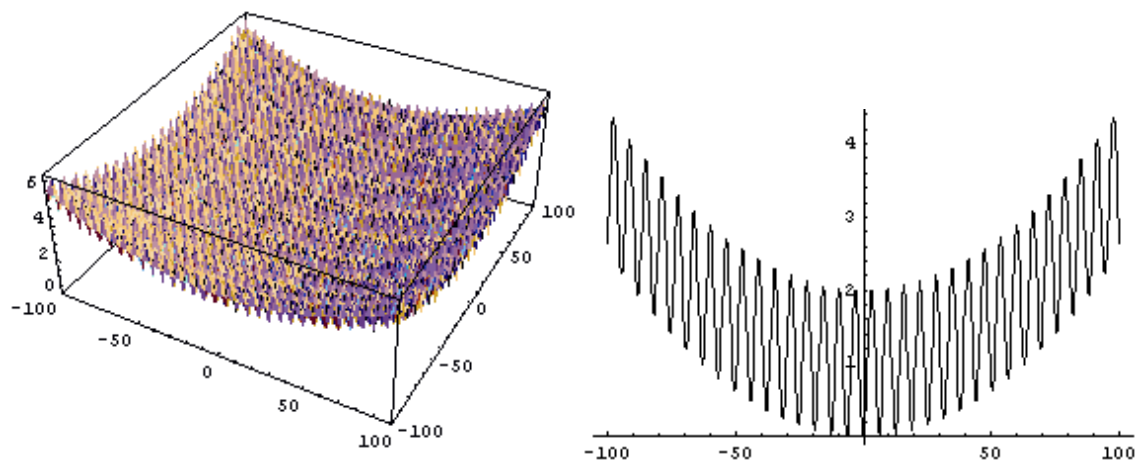
Maximální hodnota	-780.1008188581
Minimální hodnota	-800.

Průměrná hodnota	-795.2241965259
Procentuelní úspěšnost	76%

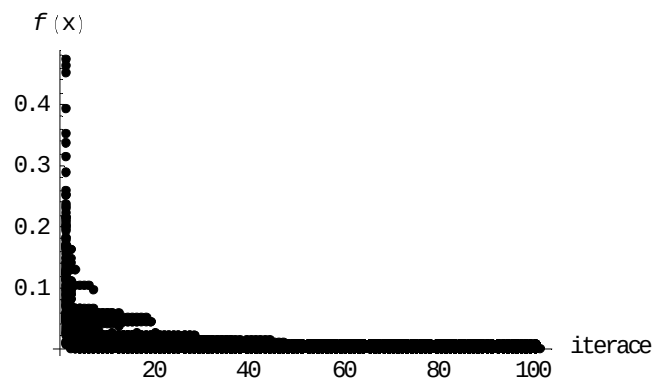
2.2.3 Griewangkova funkce

$$-\prod_{i=1}^{\text{Dim}} \cos\left(\frac{x_i}{\sqrt{i}}\right) + \sum_{i=1}^{\text{Dim}} \frac{x_i^2}{4000} + 1 \quad (5)$$

Jedná se o další testovací funkci převzatou z odborné literatury (6).



Obr. 10 Graf Griewangkovy funkce



Obr. 11 Simulace pro Griewangkovu fci

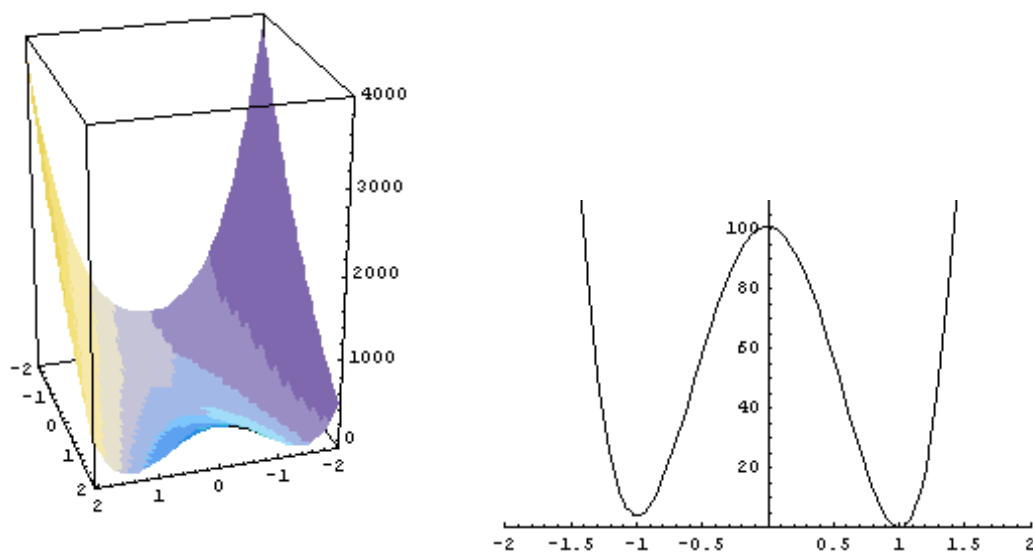
Tabulka IX. Úspěšnost SS pro Griewangkovu funkci

Maximální hodnota	0.0098646721
Minimální hodnota	0.
Průměrná hodnota	0.0012820131
Procentuelní úspěšnost	83%

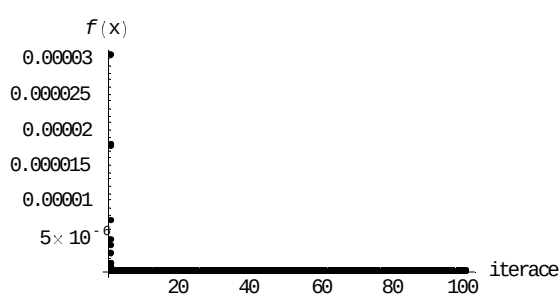
2.2.4 Druhá De Jongova funkce (Rosenbrockovo sedlo)

$$\sum_{i=1}^{\text{Dim}} (100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2) \quad (6)$$

Tato funkce v sobě skrývá jistou záludnost, touto záludností je právě ono zmiňované sedlo. Ze simulací vyplývá, že pro SS to není žádnou překážkou, neboť většinou již při druhé iteraci objeví globální minimum. Funkce převzata ze stejného zdroje jako předchozí (6).



Obr. 12 Graf druhé De Jongova funkce



Obr. 13 Simulace pro druhou De Jongovu funkci

Tabulka X. Úspěšnost SS pro druhou De Jongovu funkci

Maximální hodnota	0.
Minimální hodnota	0.
Průměrná hodnota	0.

Procentuelní úspěšnost	100%
------------------------	------

2.2.5 Shrnutí simulací pro různé funkce

Tento algoritmus má své slabiny, stejně jako přednosti. Netestoval jsem dostatečné množství funkcí, abych mohl prohlásit v čem jsou tyto slabiny a přednosti, což ani nebylo zadáním této práce. Nicméně na základě provedených simulací mohu prohlásit, že je algoritmus funkční. Pro přehlednost ještě shrnu jeho úspěchy na jednotlivých funkcích v přehledné tabulce.

Tabulka XI. Úspěšnost SS pro jednotlivé funkce

Jméno funkce	Funkce (1)	Rastriginova f.	Griewangkova f.	2. De Jongova f.
Rozsah proměnných	$<-10;10>$	$<-5;5>$	$<-100;100>$	$<-100;100>$
Maximální hodnota	0.5566697003	-780.100818	0.0098646721	0.
Minimální hodnota	0.	-800.	0.	0.
Průměrná hodnota	0.0111333940	-795.22419	0.0012820131	0.
Procentuelní úspěšnost	98%	76%	83%	100%

ZÁVĚR

Ve své práci jsem chtěl nastínit pohled na problematiku optimalizačních algoritmů. Myslím, že se mi to více či méně podařilo. Sám uznávám, že v této části jsem mohl být sdílnější a mohl jsem popsat funkci více druhů optimalizačních algoritmů. Nicméně jsem si řekl, že vše v této práci zkusím vztáhnout k SS algoritmu a tak jsem nastínil jen ty, které byly důležité pro vznik SS, neodpustil jsem si však nezmínit se o Diferenciální evoluci z toho důvodu, že je zmiňována ve spoustě odborné literatury. Získal jsem pocit, že tento algoritmus je používán k porovnávání s ostatními algoritmy. Rozhodl jsem se tedy s ním alespoň povrchně seznámit. Další algoritmus který jsem uvedl, přestože nemá k SS žádný vztah je Soma. Tu jsem uvedl z toho důvodu, že jsem se s ní seznámil jako s prvním optimalizačním algoritmem a na ní jsem pochopil, co to vlastně optimalizační algoritmy jsou a k čemu jsou dobré.

Pak jsem se ve své práci zaměřil již na samotný SS algoritmus. Nejprve jsem váhal nad tím, jak tento algoritmus popsat. Pak jsem se inspiroval od povolanějších (4) a rozhodl jsem se použít jejich osnovu. Takže jsem nejprve popsal nástin historie. Tady jsem mohl asi také být upovídanější, ale snažil jsem se vystihnout jen to nejdůležitější pro vznik SS algoritmu. Toto vychází z mého osobního názoru na historii. Myslím si totiž že historie je důležitá pouze ve vztahu k přítomnosti a tak si myslím že je zbytečné vláčet historický balast. Pak jsem popsal teoreticky fungování SS. Tady jsem si zamiloval SS pro jeho „modulovou“ architekturu. Líbí se mi, jak se skládá z 5 modulů, které se opakují a mají na sebe návaznost. Bylo to také výhodné při programování, protože se tím program rozdělil na několik dílčích podprogramů a také je to výhodné pokud by se někdo pokusil tento program optimalizovat. V další části jsem začal popisovat na příkladu, jak funguje mnou naprogramovaný SS. Při programování jsem zjistil dva poznatky o programovém prostředí *Mathematica*, které považuji za důležité. Prvním z nich je fakt, že *Mathematica* je skvělý matematický nástroj. Druhý fakt je, že zde chybí vhodný nástroj pro vyhledávání a odstraňování chyb, na jaký je člověk zvyklý z jiných programových prostředí.

V experimentálně-simulační části jsem dokázal, že algoritmus je funkční nebo alespoň to, že pro uvedené testovací funkce byl schopen najít globální minima.

Mám pocit, že jsem splnil vytýčené cíle, nicméně nepřísluší mi abych to hodnotil. Toto hodnocení je v kompetenci jiných lidí a já jen doufám, že budou alespoň trochu sdílet můj názor.

Na úplný závěr bych ještě jednou chtěl poděkovat všem, kteří mi ať už vědomě nebo nevědomě pomohli. Nechci se opakovat a tak jen všichni se můžou najít v seznamu použité literatury. Pokud se tam však nenajdou, tak se jim omlouvám. Nebylo to mým úmyslem, ale jen nepříjemným opomenutím.

Taky bych chtěl poděkovat všem, kteří dočetli mou práci až sem a nezhodili ji po přečtení pár řádků. Mohu jen doufat, že Vás čtení této práce obohatilo alespoň tak jako mě její psaní.

SEZNAM POUŽITÝCH ZDROJŮ

- 1) Zelinka Ivan, Umělá inteligence v problémech globální optimalizace, BEN, 2002, ISBN 80-7300-069-5
- 2) SOMA [online]. [cit. 31.10.2003] Dostupný z URL: <<http://www.ft.utb.cz/people/zelinka/soma>>
- 3) Scatter Search [online]. [cit. 15.1.2005] Dostupný z URL: <<http://leeds-faculty.colorado.edu/laguna/index.htm>>
- 4) Scatter Search [online] [cit.] Dostupný z URL: <<http://leeds-faculty.colorado.edu/laguna/articles/ss4.html>>
- 5) Scatter Search [online] [cit.] Dostupný z URL: <<http://leeds-faculty.colorado.edu/laguna/articles/ss3.html>>
- 6) David W. Corne, Morco Doroto, Fred Glover, editors., New Ideas in Optimization – Mechanical Engineering Design Optimization by Difererntial Evolution, London: McGraw-Hill, 1999, ISBN 007-709506-5
- 7) <http://www.sciencedirect.com>:
- 8) Zelinka Ivan, Umělá inteligence I. Volume 1. Zlín:Vutium, Brno, 1998, ISBN 80-214-1163-5
- 9) Kvasnička V., Pospíchal J., Tiňo P.2000, Evolučné algoritmy, STU Bratislava, ISBN 85-246-2000
- 10) Ken Price, Rainer Storn, Jouni Papinem, Differential Evolution – A practical Approach to Global Optimization“, Springer, ISBN 3-540-20950-6

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SS	Scatter Search (někde též Scatter Search algoritmus)
GA	Genetický algoritmus
fce	funkce

SEZNAM OBRÁZKŮ

Obr. 1 Dělení optimalizačních algoritmů.....	8
Obr. 2 Ukázková množina řešení SS	15
Obr. 3 Simulace pro funkci (1), počet iterací pro „FindMinimum“ je 5.....	26
Obr. 4 Simulace pro funkci (1), počet iterací pro „FindMinimum“ je 10.....	26
Obr. 5 Simulace pro funkci (1), počet iterací pro „FindMinimum“ je 100.....	26
Obr. 6 Grafy účelové funkce (1).....	28
Obr. 7 Simulace pro funkci (1).....	29
Obr. 8 Graf Rastriginovy funkce.....	30
Obr. 9 Simulace pro Rastriginovu fci.....	30
Obr. 10 Graf Griewangkovy funkce.....	31
Obr. 11 Simulace pro Griewangkovu fci.....	31
Obr. 12 Graf druhé De Jongovy funkce	32
Obr. 13 Simulace pro druhou De Jongovu funkci	32

SEZNAM TABULEK

Tabulka I. Rozložení pravděpodobností do jednotlivých intervalů.....	19
Tabulka II. Prvních deset řešení vygenerovaných diversifikační metodou.....	19
Tabulka III. Řešení upravená zlepšující metodou.....	20
Tabulka IV. Řešení uložená do referenční množiny podle hodnot $f(x)$	21
Tabulka V. Řešení uložená do referenční množiny podle vzdáleností.....	21
Tabulka VI. Úspěšnost SS pro funkci (1) při různém nastavení FindMinimum.....	26
Tabulka VII. Úspěšnost SS pro funkci (1).....	29
Tabulka VIII. Úspěšnost SS pro Rastriginovu funkci.....	30
Tabulka IX. Úspěšnost SS pro Griewangkovu funkci.....	31
Tabulka X. Úspěšnost SS pro druhou De Jongovu funkci.....	32
Tabulka XI. Úspěšnost SS pro jednotlivé funkce.....	33