Ant Colony Optimization v prostředí Mathematica

Ant Colony Optimization in Mathematica Environment

Bc. Martina Vaculíková

Diplomová práce
2008

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2007/2008

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Martina VACULÍKOVÁ**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Ant Colony Optimization v prostředí Mathematica**

Zásady pro vypracování:

Cílem je vytvořit algoritmus Ant Colony Optimization v prostředí Mathematica a srovnat jeho chování s jinými evolučními algoritmy.

1. **Seznamte se s algoritmem Ant Colony Optimization (ACO).**
2. **Naprogramujte ACO v prostředí Mathematica.**
3. **Vyberte vhodné testovací funkce, na kterých budete demonstrovat chování ACO.**
4. **Srovnejte chování ACO s jinými evolučními algoritmy (Samoorganizující se migrační algoritmus – SOMA, Diferenciální evoluce – DE).**

Rozsah práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **Dorigo M., Stuzle T., Ant Colony Optimization (Bradford Books), The MIT Press, 2004, ISBN: 978-0262042192.**
2. **Dorigo M., Ant Colony Optimization and Swarm Intelligence, Springer, 2006, ISBN 3540226729.**
3. **Zelinka I., Umělá inteligence v problémech globální optimalizace, BEN, 2002, 190 p. ISBN 80-7300-069-5.**
4. **Wolfram S., The Mathematica Book 5, Wolfram Media, 2003, ISBN 1-57955-022-3.**

Vedoucí diplomové práce: **Ing. Zuzana Oplatková**
Ústav aplikované informatiky

Datum zadání diplomové práce: **20. února 2008**

Termín odevzdání diplomové práce: **19. května 2008**

Ve Zlíně dne 20. února 2008

L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Ant Colony Optimization (ACO) je moderní algoritmus používaný pro řešení optimalizačních problémů. Algoritmus je inspirován chováním skutečných mravenčích kolonií a je využíván převážně pro řešení diskrétních problémů. Práce se zabývá modifikací ACO pro spojité systémy. Algoritmus je naprogramován v prostředí Mathematica. Cílem práce je srovnání algoritmu ACO_R a dalších dvou algoritmů – SOMA (SamoOrganizující se Migrační Algoritmus) a DE (Diferenciální Evoluce). Testování algoritmů je provedeno na zadaných testovacích funkcích.

Klíčová slova: Ant Colony Optimization, optimalizace, evoluční algoritmy, mravenčí kolonie, SOMA, SamoOrganizující se Migrační Algoritmus, DE, Diferenciální Evoluce, Mathematica

ABSTRACT

Ant Colony Optimisation is a recent algorithm used for solving optimisation problems. The algorithm is modelled on the behaviour of real ant colonies, and has traditionally been used for solving problems in the discrete domain. This thesis describes a modification of ACO for continuous spaces. The algorithm is programmed in Mahematica environment. The aim of thesis is comparing ACO_R algorithm and another algorithms – SOMA (Self-Organizing Migrating Algorithm) and DE (Diferencial Evolution). Testing of algorithms is performed on given test functions.

Keywords: Ant Colony Optimization, optimization, Evolutionary Algorithms, Ant colonies, SOMA, Self-Organizing Migrating Algorithm, Differential Evolution, DE, Mathematica

Ráda bych poděkovala vedoucí mé diplomové práce Ing. Zuzaně Oplatkové, Ph.D. za odborné vedení, za cenné rady a připomínky k mé práci. Dále bych chtěla poděkovat za provedení testování.

Prohlašuji, že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků, je-li to uvolněno na základě licenční smlouvy, budu uveden jako spoluautor.

Ve Zlíně …….

Podpis diplomanta

OBSAH

ÚVOD

Optimalizační problémy se vyskytují v mnoha odvětvích lidské činnosti, jakými jsou ekonomie, obchod, strojírenství, průmysl nebo medicína. Pro jejich vysokou složitost a komplexnost nejsou dostupné žádné exaktní metody umožňující jejich řešení.

Vedle tradičních metod optimalizace, mezi které patří například numerické metody, lineární a nelineární programování, variační metody nebo dynamické programování existují optimalizační algoritmy, které se dají nazývat inteligentní. Mezi tyto algoritmy patří genetické algoritmy, simulované žíhání, metody Monte-Carlo nebo různé heuristické algoritmy.

Tyto algoritmy nacházejí uplatnění především při řešení problémů, na které neznáme exaktní metodu řešení nebo které jsou příliš složité a rozsáhlé. Tyto metody jsou známy jako heuristické algoritmy. Heuristické metody sice nezaručují nalezení globálně optimálního řešení, ale jsou schopny nalézt kvalitní řešení, které se optimu velmi blíží. Některé z algoritmů využívají znalosti o chování skutečných biologických systémů, například mravenců, včel nebo bakterií.

Do této skupiny heuristických metod můžeme zařadit i optimalizační metodu Ant Colony Optimization (ACO). Je to poměrně nová metoda optimalizace, která je inspirována chováním skutečných mravenčích kolonií. Podle jednoduchých pravidel samoorganizace, jimiž se řídí jednotlivý jedinci, vzniká komplexní chování celku schopné řešit složité optimalizační úlohy. Algoritmus tohoto typu nelze přesně zařadit mezi stochastické nebo deterministické metody, ale jde o směs obou.

Cílem práce je srovnání algoritmu ACO a dalších dvou algoritmů – SOMA (SamoOrganizující se Migrační Algoritmus) a DE (Diferenciální Evoluce). Oba tyto algoritmy jsou velmi robustní a dokážou řešit i velmi náročné optimalizační úlohy.

I. TEORETICKÁ ČÁST

1 OPTIMALIZACE

Teorie optimalizace je matematickou disciplínou, která se zabývá určováním minimálních a maximálních hodnot funkcí při určitých omezujících podmínkách, tj. řešením optimalizačních úloh.

S optimalizačními úlohami nejrůznějšího druhu se v praxi setkáváme velmi často. Většinou jsou formulovány slovně a řeší se na základě zkušeností a intuice. Takový přístup při současné úrovni rozvoje vědy a techniky již zcela nestačí, protože neposkytuje objektivní a vědecké podklady pro řízení a rozhodování.

Analýza problému globální optimalizace ukazuje, že neexistuje deterministický algoritmus řešící obecnou úlohu globální optimalizace v polynomiálním čase, tzn. problém globální optimalizace je NP-obtížný. Přitom globální optimalizace je úloha, kterou je nutno řešit v mnoha praktických problémech, mnohdy s velmi významným ekonomickým efektem, takže je nutné hledat algoritmy, které jsou pro řešení konkrétních problému použitelné.

Nemožnost nalézt deterministický algoritmus obecně řešící úlohu globální optimalizace vedla k využití algoritmů stochastických, které sice nemohou garantovat nalezení řešení v konečném počtu kroků, ale často pomohou nalézt v přijatelném čase řešení prakticky použitelné.

Stochastické algoritmy pro globální optimalizaci heuristicky prohledávají prostor. Heuristikou rozumíme postup, ve kterém se využívá náhoda, intuice, analogie a zkušenost.

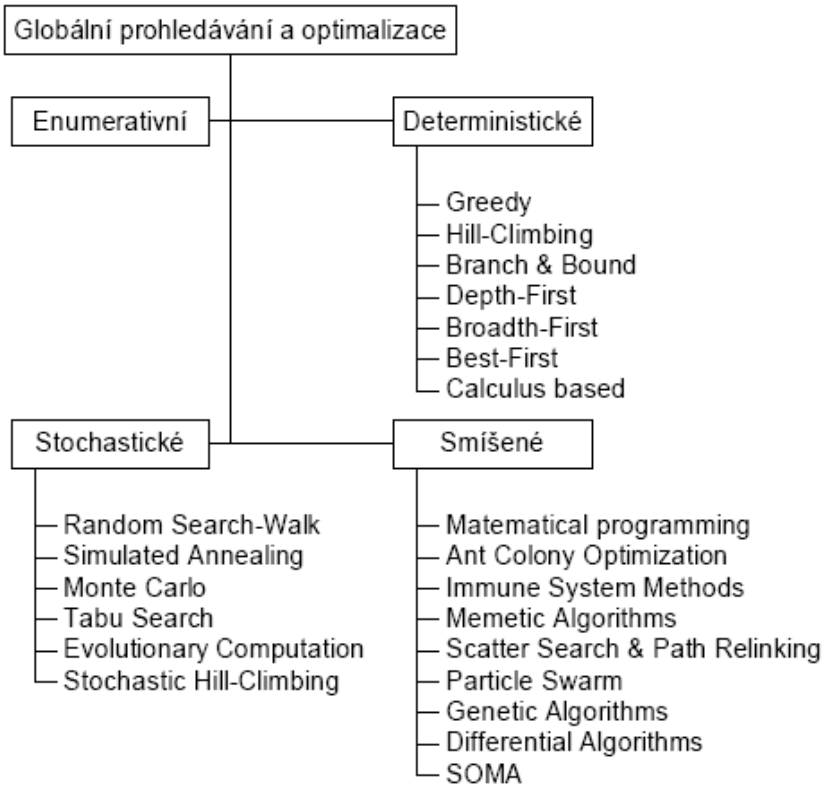
Heuristické metody sice nezaručují nalezení globálně optimálního řešení, ale jsou schopny v přiměřené době poskytnout řešení, jež je uspokojivé. Heuristické optimalizační metody se používají pro optimalizaci mnohaparametrových funkcí s divokým průběhem, tj. s mnoha extrémy nebo neznámým gradientem. Takovými funkcemi je i většina účelových funkcí problémů rozvrhování výroby.

Standardní gradientové metody nebo negradientové metody nejsou vhodnými přístupy, protože požadujeme nalezení globálního extrému funkce s mnoha lokálními extrémy. Tyto metody obvykle konvergují jen k extrému blízko startovního bodu a tento extrém už nejsou schopné opustit.

Stochastické optimalizační metody nutně fungují pomaleji než jakékoli heuristické přístupy. Pokud nemáme dopředu zadané podmínky pro globální extrém, nikdy nevíme, jestli jsme ho dosáhli a máme-li optimalizaci zastavit. Stochastické optimalizační metody

však mají i zásadní výhody: jsou velmi obecně formulované a tedy aplikovatelné téměř na jakýkoli problém a dokážou se dostat z lokálního extrému.

Většina stochastických algoritmů pro hledání globálního minima v sobě obsahuje proces učení. Inspirace k užití heuristik jsou často odvozeny ze znalostí přírodních nebo sociálních procesů.



Obr. 1: Rozdělení optimalizačních algoritmů [3]

Smíšené optimalizační algoritmy představují směs metod deterministických a stochastických, které ve vzájemné spolupráci dosahují překvapivě dobrých výsledků. Poměrně silnou podmnožinou těchto algoritmů jsou evoluční algoritmy. Algoritmy smíšeného charakteru jsou robustní, což znamená, že nezávisle na počátečních podmínkách velmi často naleznou kvalitní řešení, jež je reprezentováno obvykle jedním či více globálními extrémy. Jsou také velmi efektivní a výkonné, to znamená, že jsou schopny nalézat kvalitní řešení během relativně malého počtu ohodnocení účelové funkce. Dále jsou schopné pracovat s problémy typu „černá skříňka“, tzn., že nepotřebují ke své činnosti analytický popis problému.

1.1 Evoluční algoritmy

V posledních desetiletích se s poměrným úspěchem pro hledání globálního minima funkcí užívají stochastické algoritmy zejména evolučního typu. 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.

Evoluční algoritmy jsou zastřešujícím termínem pro řadu přístupů využívajících modelů evolučních procesů v přírodě. Snaží se využít představ o hnacích silách evoluce živé hmoty pro účely optimalizace. Všechny tyto modely pracují s náhodnými změnami navrhovaných řešení. Pokud jsou tato nová řešení výhodnější, nahrazují předcházející řešení.

Evoluční algoritmy byly a jsou předmětem intenzivního výzkumu. Jedním z hlavních motivů jsou především aplikace v praktických problémech, které jinými metodami nejsou řešitelné. Dalšími motivy jsou výzkum umělé inteligence a teorie učení. Hledají se nové inspirace např. v chovatelství a šlechtitelství, modeluje se sexuální reprodukce v populacích nebo chování mravenců.

2 ANT COLONY OPTIMIZATION

Ant Colony Optimization (ACO) je poměrně nová metoda optimalizace, která je inspirována chováním skutečných mravenčích kolonií. Tato metoda byla původně navržena pro řešení kombinatorických optimalizačních problémů. Mravenčí algoritmy patří do kategorie rojové inteligence, která vychází ze studia a pozorování sociálního chování hmyzu jako jsou mravenci, včely, vosy nebo termiti.

Hlavní myšlenkou mravenčích algoritmů je, že principy samoorganizace, které umožňují vysoce koordinované chování reálných mravenců, mohou být použity pro koordinaci populací umělých jedinců, kteří budou vzájemně spolupracovat na řešení výpočetních problémů. Různé aspekty mravenčího chování jsou přitom inspirací pro různé druhy mravenčích algoritmů. Mezi tyto aspekty můžeme zařadit například hledání potravy, stavba mraveniště, rozdělení prací nebo spolupráce při přepravě věcí (potravy apod.).

2.1 Mravenčí kolonie

Mravenci patří mezi sociální hmyz, který žije v koloniích. Hlavní inspirací pro optimalizační metody je chování některých mravenčích druhů při hledání potravy. Mravenci mají schopnost nalézt nejkratší cestu mezi mraveništěm a zdrojem potravy a to i přesto, že jsou téměř slepí.

Mravenci používají zvláštní druh nepřímé komunikace, nazývané stigmergy, kterou řídí své aktivity. Biologové dokázali, že většina sociálního hmyzu, který žije v koloniích, se chová podle určitého modelu a využívá přitom pouze nepřímé (stigmergy) komunikace. Pomocí stigmergy komunikace dosahuje hmyz samoorganizace.

Hlavními charakteristikami stigmergy, které je odlišují od jiných druhů komunikace, jsou následující:

- Stigmergy je nepřímá, nesymbolická forma komunikace, zprostředkovaná pomocí okolního prostředí - hmyz si vyměňuje informace úpravou nebo změnou v jejich okolí.
- Stigmergy informace je lokálního rázu, je tedy přístupná pouze hmyzu, který navštíví dané místo nebo jeho blízké okolí, kde byla tato informace vypuštěna.

Tento druh nepřímé komunikace mezi jedinci je využíván při hledání zdroje potravy a je založen na použití chemikálie produkované mravenci. Když mravenci chodí za potravou

tam a zpět, vypouštějí na zem chemikálii nazývanou feromon. Tím si označují cesty k potravě a zpět. Ostatní mravenci vnímají přítomnost feromonů a mají určitou tendenci následovat cesty, kde je koncentrace feromonu nejvyšší. Díky tomuto mechanismu dokážou mravenci transportovat potravu do mraveniště pozoruhodně efektivním způsobem.

2.2 Chování reálných mravenců

Toto mravenčí chování při hledání potravy bylo prokázáno na důmyslném pokusu s argentinským mravenčím druhem *Linepithema humile*, který vysvětluje chování, jak mravenci naleznou nejkratší cestu mezi zdrojem potravy a mraveništěm. Pokus je následující.

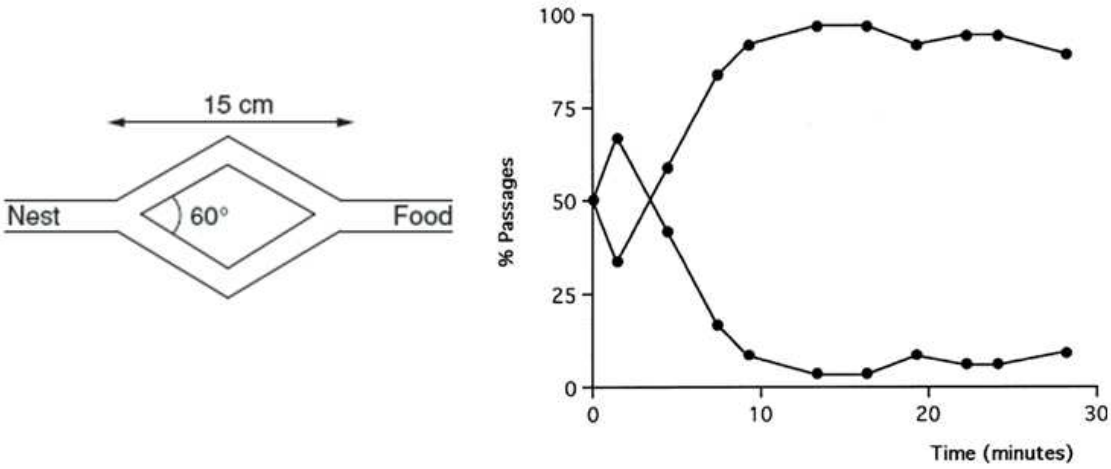
2.2.1 Experiment s dvojitým mostem (Double Bridge Experiment)

Zdroj potravy je spojen s mravenčím hnízdem pomocí mostu se dvěma stejně dlouhými větvemi (*Obr. 2*). Když pokus začal, na mostu nebyla žádná feromonová stopa. Mravenci si vybrali náhodně, se stejnou pravděpodobností, jednu větev a začali prohledávat okolí hnízda, dokud nenarazili na potravu.

Při chození ke zdroji potravy a zpět do hnízda a naopak, označili mravenci cestu feromonem a tím vytvořili feromonovou stopu. Kvůli náhodným fluktuacím je jedna z větví zvolena více mravenci než ta druhá a tak je označkována o trochu větším množstvím feromonu. Vyšší množství feromonu na této větvi mostu podněcuje více mravenců, aby si ji vybrali. Tento proces vede velmi brzy celou kolonii k používání pouze této jedné větve.

Kolektivní chování, které z pokusu vyplynulo, je formou autokatalyckého chování. To znamená, že čím více mravenců následuje stopu, tím více se feromonová stopa stává atraktivní pro další následování. Proces je tedy charakteristický pozitivní zpětnou vazbou, kdy pravděpodobnost, že si mravenec vybere cestu, vzrůstá s počtem mravenců, kteří si předtím vybrali stejnou cestu.

Na *Obr. 2* je znázorněn most a v grafu potom procento mravenců, kteří použili spodní nebo horní větev.

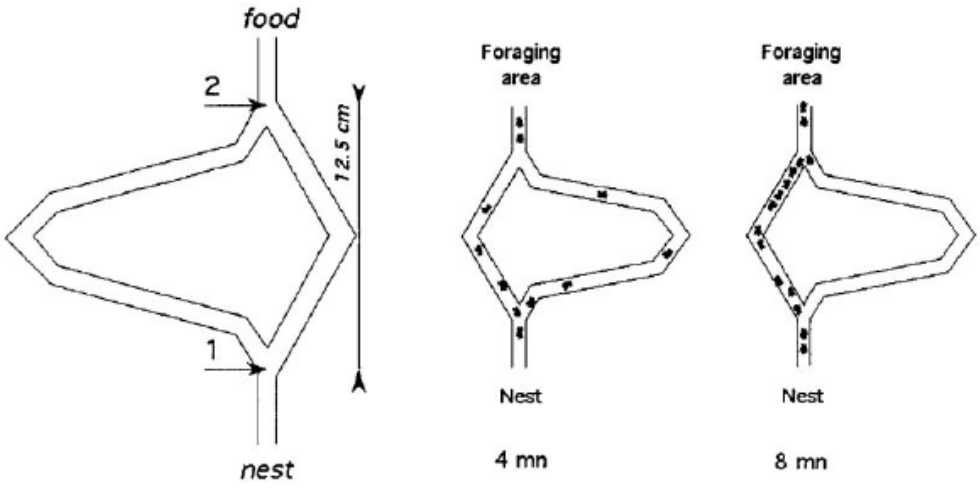


Obr. 2: Dvojitý most se stejnou délkou větví

Experiment byl také proveden pro most s různě dlouhými délkami větví. Poměr délky větví byl nastaven na 2, tedy delší větev byla dvakrát delší než kratší větev (**Chyba! Nenalezen zdroj odkazů.**). Pro většinu pokusů všichni mravenci zvolili pouze kratší větev po nějaké době.

Jako v předchozím případě, mravenci opustili hnízdo, aby prozkoumali okolí a přišli až k místu větvení, kde se museli rozhodnout, kterou cestu zvolit. Zpočátku se obě větve mravencům jeví jako identické, vyberou si tedy náhodně. V průměru se očekává, že si polovina mravenců zvolí kratší cestu a druhá polovina delší cestu. Mravenci, kteří zvolí kratší cestu, dosáhnou zdroje potravy jako první a vracejí se dříve do hnízda. Ale znovu musí zvolit mezi kratší a delší větví.

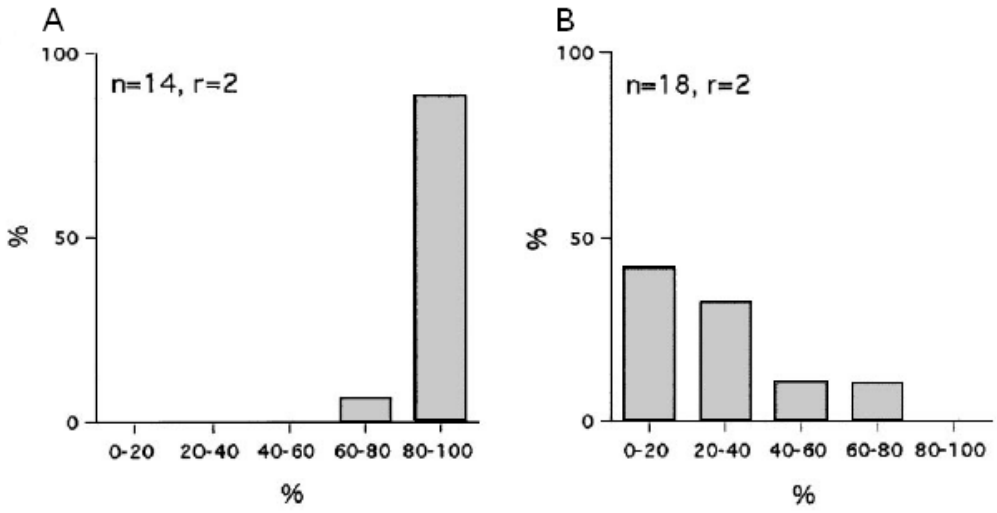
Vyšší koncentrace feromonu na kratší cestě ovlivní jejich rozhodnutí pro výběr této cesty. Feromon se tedy začne rychleji hromadit na kratší větvi, což postupně ovlivňuje výběr ostatních mravenců a nakonec všichni volí kratší cestu.



Obr. 3: Double Bridge experiment s různou délkou větví

Na **Chyba! Nenalezen zdroj odkazů.** je zobrazen výběr kratší větve mravenčí kolonií *Linepithema humile* po 4 a po 8 minutách od umístění mostu a potravy.

Nutno podotknout, že i když je druhá větev dvakrát delší než tak kratší, ne všichni mravenci použijí kratší, ale malé procento zvolí tu delší. To se vysvětluje jako druh určitého průzkumnictví mravenců. Na *Obr. 4* A. je zobrazen procentuální výsledek počtu mravenců, kteří si vybrali kratší cestu. Experiment byl proveden pro *n*=14 pokusů, delší větev je *r* krát delší než kratší větev.



Obr. 4:Procentuální výsledky pro most s různě dlouhými větvemi

Druhý graf (*Obr. 4* B) zobrazuje výsledek experimentu, kdy je do kolonie přidána kratší cesta mezi hnízdo a zdroj potravy 30 minut po delší větvi. Mravenci si tuto kratší cestu

nevyberou a stále se drží delší cesty, protože ta je již silně označkována feromonovou stopou.

Feromonová stopa není trvalá, ale feromon se po nějakém čase vypaří. Díky tomuto procesu méně nadějné cesty postupně ztrácejí feromon a také je používá méně a méně mravenců. Biologické studie dokazují, že feromonové stopy jsou velmi trvalé. Feromon zde zůstane několik hodin až několik měsíců, záleží také na druhu mravenců a typu půdy.

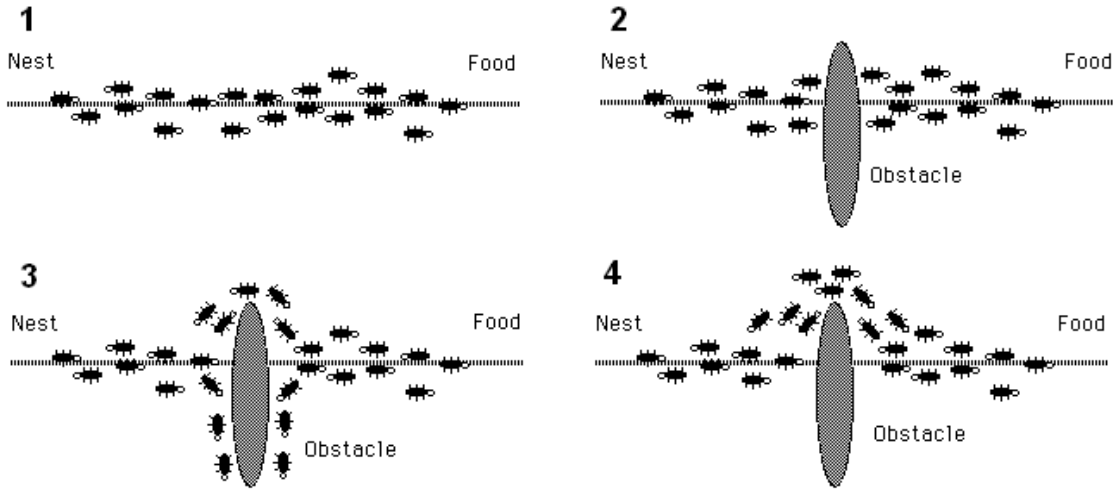
Při srovnání obou experimentů je ve druhém experimentu vliv náhodných fluktuací malý a hlavními mechanismy, které ovlivňují výsledek, jsou stigmergy, autokatalýza založená na feromonu a rozdílná délka větví.

Je také zajímavé poznamenat, že některé mravenčí druhy zanechávají určité množství feromonu podle toho, jak kvalitní je zdroj potravy, který nalezli. Cesty, které vedou k nejlepším zdrojům, označkují mravenci nejvyšším množstvím feromonů.

2.2.2 Experiment s překážkou

Mravenci nejen že dokážou nalézt nejkratší cestu od zdroje potravy do mraveniště bez použití vizuálních podnětů, ale také jsou schopni se lehce přizpůsobit změnám v prostředí. Například dokážou nalézt nejkratší cestu i přesto, že původní cesta není již více dostupná kvůli nové překážce.

Když se na cestě objeví překážka, ti mravenci, kteří jsou těsně u ní, nemohou pokračovat dál po feromonové stopě, ale musí se rozhodnout, kterým směrem se dát. Dá se předpokládat, že polovina mravenců zvolí jeden směr a druhá polovina druhý směr (viz kap. **Chyba! Nenalezen zdroj odkazů.**). Stejná situace nastane na druhé straně překážky. Mravenci, kteří zvolí kratší cestu, rychleji vytvoří novou feromonovou stopu než mravenci na delší cestě. Díky tomu opět mravenci začnou používat kratší cestu (*Obr. 5*).



Obr. 5: Experiment s překážkou v cestě [18]

2.3 Umělí mravenci

Experiment s dvojitým mostem jasně dokázal, že mravenčí kolonie mají schopnost optimalizace. Použitím pravidel pravděpodobnosti založených na lokálních informacích dokážou mravenci nalézt nejkratší cestu mezi dvěma body ve svém prostředí.

ACO algoritmy jsou založeny na kolonii (populaci) umělých mravenců, kteří jsou jednoduchými výpočetními agenty, kteří vzájemně spolupracují a komunikují skrze umělou feromonovou stopu.

Hlavní myšlenky ACO se velmi přesně drží biologické předlohy. Proto je mezi skutečnými a umělými mravenci spousta podobností. Skutečné i umělé mravenčí kolonie jsou založeny na populaci jedinců, kteří vzájemně spolupracují, aby dosáhli jistého cíle. Kolonie je populace jednoduchých, nezávislých, asynchronních agentů, kteří spolupracují na nalezení dobrého řešení daného problému. V případě reálných mravenců je problémem nalezení potravy, zatímco u umělých mravenců je cílem nalézt co nejlepší řešení pro optimalizační problém. Jediný mravenec je schopen řešení nalézt, ale pouze spoluprácí mezi jedinci skrze stigmergy komunikaci lze dosáhnout opravdu nejlepšího možného řešení.

Umělí mravenci mají dvojí charakter. Na jedné straně jsou abstrakcí vzorů chování skutečných mravenců a jejich schopnosti nalezení nejkratší cesty, na straně druhé jsou obohaceni o některé další schopnosti, které jejich přírodní doplňky nemají. Umělí

mravenci tedy mají některé schopnosti navíc, které je dělají více efektivními a výkonnými výpočetními agenty.

Reální mravenci vypouštějí feromony na cestu, po které jdou, a dokážou na ně reagovat. Umělí mravenci žijí ve virtuálním světě, a tak mohou pouze měnit numerické hodnoty umělých feromonů. Sekvence hodnot feromonů je nazývána umělou feromonovou stopou. V ACO jsou feromonové stopy jedinou možnou komunikací mezi jedinci. Mechanismus, analogický k vypařování skutečného feromonu v reálných mravenčích koloniích, umožňuje umělým mravencům zapomínat historii a zaměřit se na nové nadějné směry prohledávání.

Stejně jako skuteční mravenci, umělí mravenci vytváří řešení postupně pohybem z jednoho místa na jiné. Reální mravenci jednoduše chodí, vybírají směr podle koncentrace feromonu a náhodného výběru. Umělí mravenci také vytváří řešení krok za krokem, pohybem po možných místech v řešeném problému a náhodným rozhodovacím procesem v každém kroku.

Umělí mravenci mají některé vlastnosti, které nemají reální mravenci a které umožňují zlepšit algoritmy pro optimalizační úlohy.

- Umělí mravenci žijí v diskrétním světě – pohybují se přes konečnou množinu bodů
- Umělí mravenci mají vnitřní paměť, která zaznamenává doposud vykonané akce
- Umělí mravenci vypouštějí množství feromonu, které je funkcí kvality nalezeného řešení
- Vypařování feromonů zajišťuje algoritmu, aby neuvízl v lokálním optimu. Vypařování také umožňuje pomalu zapomenout horší řešení a směřovat prohledávání i do jiných míst. Tím se lze vyhnout předčasné konvergenci algoritmu do lokálního optima.
- Některé implementace používají další mechanismy, které neexistují u reálných mravenců, například lokální prohledávání, předvídání budoucnosti, backtracking

2.4 ACO metaheuristic

ACO může být použito pro řešení jakéhokoliv diskrétního kombinačního optimalizačního problému, pro který lze použít nějakou konstruktivní heuristickou proceduru.

Metaheuristika je množina algoritmických konceptů, které se jsou použitelné na široké spektrum různých problémů. Je to všestranně použitelná algoritmická konstrukce, která se

dá aplikovat na různé optimalizační problémy s relativně málo modifikacemi. Mezi metaheuristiky patří algoritmy jako simulované žíhání nebo tabu search.

Mravenci použití v ACO fungují jako stochastické konstruktivní procedury, které vytvářejí řešení iterativním přidáváním komponent do částečného řešení, přičemž při výběru každé další komponenty uvažují

- Heuristickou informaci o řešeném problému, která směřuje výpočet ke slibným řešením
- Zkušenosti získané všemi mravenci od začátku výpočtu, reprezentované feromonovými stopami, které se během výpočtu neustále adaptují

Stochastická složka umožňuje vygenerování velkého počtu různých řešení.

2.4.1 Kombinatorický optimalizační problém

Uvažujme minimalizační problém (S, Ω, f), kde S je množina kandidátů řešení, f je objektivní funkce, která má být minimalizována, a Ω je množina omezení. Je dána množina diskrétních proměnných X_i s hodnotami v_i^j. Řešení $s^* \in S$ je globální optimum, pokud $f(s^*) \leq f(s)$ pro všechna $s \in S$. Řešením kombinatorické úlohy nalezneme alespoň jedno optimum s^*.

Model kombinatorického optimalizačního problému je použit pro popis feromonového modelu ACO. Hodnota feromonu τ_{ij} odpovídá každé hodnotě komponenty řešení c_{ij}. Množina všech možných komponent řešení je C.

Umělí mravenci fungují jako konstruktivní procedura, která vytváří řešení procházením konstrukčního grafu G_c (V,E), kde V je množina vrcholů a E je množina hran (pro úplně propojený graf). Mravenci se pohybují mezi vrcholy grafu po hranách a tím postupně vytvářejí částečné řešení. Navíc mravenci vypouštějí určité množství feromonu na hrany, po kterých přešli. Množství feromonu je závislé na kvalitě nalezeného řešení.

2.4.2 The Ant Colony Optimization Metaheuristic algoritmus

ACO metaheuristic je popsán v následujícím algoritmu:

Algorithm 1 Ant Colony Optimization metaheuristic

```
while termination conditions not met do
  ScheduleActivities
    AntBasedSolutionConstruction()
    PheromoneUpdate()
    DaemonActions()        {optional}
  end ScheduleActivities
end while
```

Po počáteční inicializaci následují tři fáze: v každé iteraci mravenci zkonstruují počet řešení, tato řešení mohou být dále vylepšena pomocí metody lokálního prohledávání a nakonec se aktualizuje feromon.

AntBasedSolutionConstruction: množina *m* umělých mravenců zkonstruuje řešení z hodnot komponentů $C=\{c_{ij}\}$, $i=1,\ldots,n$, $j=1, \ldots|D_i|$. Konstrukce řešení započne z prázdného částečného řešení a je postupně rozšířeno o další komponenty, které splňují podmínky omezení. Proces konstrukce řešení znamená chození po grafu G_c. Výběr komponenty z množiny komponent je stochastická metoda, která je závislá na feromonu.

PheromoneUpdate: cílem aktualizace feromonu je zvýšit jeho hodnotu pro dobrá a nadějná řešení a naopak snížit hodnotu pro špatná řešení. Toho dosáhneme vypařováním feromonů.

DaemonActions: zde mohou být použity různé jiné aktivity, například vylepšení nového řešení pomocí lokálních prohledávacích technik.

Vypařování feromonu je důležité pro zabránění příliš rychlé konvergenci algoritmu. Je implementováno jako forma zapomínání a tím se upřednostní prohledávání nových míst v prohledávaném prostoru.

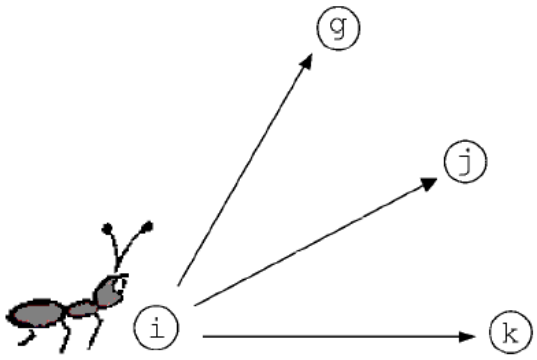
2.4.3 Problém obchodního cestujícího

ACO byl původně vymyšlen pro kombinatorické problémy. Prvním mravenčím algoritmem byl Ant System, který vznikl na počátku devadesátých let. Další algoritmy rychle následovaly. Nejpopulárnější ukázka fungování ACO metaheuristiky je na problému obchodního cestujícího (TSP, Traveling Salesman Problem).

Na mapě je zvoleno *N* měst a jejich vzájemná vzdálenost pro každou z dvojic. Problémem obchodního cestujícího je určit takové pořadí návštěv jednotlivých měst, aby každé město bylo navštíveno právě jednou, výsledná délka cesty byla nejkratší a cestující se vrátil zpět do města, kde cestu začal.

Cílem je tedy najít uzavřenou cestu minimální délky spojující *n* měst. Každé město musí být navštíveno právě jednou. Pohyb z města i do města j je dán komponentou $c_{ij} \equiv c_{ji}$ Graf $G_c = (V,E)$ je definován jako množina bodů, kde města jsou vrcholy *V* a spojení mezi městy jsou hrany grafu *E*. Každé hraně náleží určitá hodnota feromonu. Mravenci začnou budovat řešení z náhodně zvoleného vrcholu grafu, poté se postupně pohybují po grafu z jednoho města do druhého, dokud nedokončí kompletní cestu.

Mravenec ve městě *i* vybírá další město, do kterého půjde. Výběr je ovlivněn pravděpodobností, která je úměrná množství feromonu na dané hraně.



Obr. 6: Mravenec vybírá další město podle množství feromonu na hraně

Každý mravenec udržuje v paměti svou cestu grafem, aby nešel do jednoho místa vícekrát. V každém kroku si mravenec vybere další vrchol podle toho, jaké je na hraně množství feromonu. Na konci iterace je potom feromon aktualizován podle toho, jak dobré řešení bylo nalezeno.

2.5 Základní druhy mravenčích algoritmů

V literatuře lze nalézt množství variant ACO algoritmů. Zde budou popsány tři nejúspěšnější, Ant System (AS) – první realizace ACO algoritmu, dále $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) a Ant Colony System (ACS).

V následující tabulce je přehled některých úspěšných mravenčích algoritmů:

Tabulka 1: Vybrané mravenčí algoritmy

Algoritmus	**Autor**	**Rok**
Ant System (AS)	Dorigo a spol.	1991
Elitist AS	Dorigo a spol.	1992
Ant-Q	Gambardella &Dorigo	1995
Ant Colony System (ACS)	Dorigo & Gambardella	1996
$\mathcal{MAX}$ -$\mathcal{MIN}$ AS	Stüzle & Hoos	1996
Rank-based AS	Bullnheimer a spol.	1997
ANTS	Maniezzo	1999
BWAS	Cordón a spol.	2000
Hyper-cube AS	Blum a spol.	2001

2.5.1 Ant System

Ant System byl prvním algoritmem, který publikován. Jeho hlavní charakteristikou je, že v každé iteraci jsou hodnoty feromonu aktualizovány všemi mravenci. Ukládání feromonu τ_{ij} pro hranu spojující města *i* a *j* je následující:

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \sum_{k=1}^{m}\Delta\tau_{ij}^{k} \qquad (1)$$

kde ρ je faktor vypařování, *m* je počet mravenců a $\Delta\tau_{ij}^{k}$ je množství feromonu na hraně *(i,j).* ρ musí být menší než 1, jinak by se feromon neomezeně kumuloval. Pro mravence *k* platí

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & jesliže\ k-tý\ mravec\ použil\ hranu\ (i,j) \\ 0 & když\ ji\ nepoužil \end{cases} \qquad (2)$$

kde *Q* je konstanta, L_k je délka cesty kterou urazil *k-tý* mravenec.

Při konstrukci řešení mravenci prochází graf a na každém vrcholu musí udělat rozhodnutí, kterou hranou budou pokračovat. Pravděpodobnost přechodu *k*-tým mravencem při pohybu z města *i* do města *j* je dána

$$p_{ij}^{k} = \begin{cases} \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{c_{il} \in N} \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}} & pokud\ c_{il} \in N \\ 0 & jinak \end{cases} \quad (3)$$

kde *N* je množina komponentů, což je množina všech hran *(i,l),* kde *l* je město dosud nenavštívené mravencem *k*. η_{ij} je viditelnost vyjádřená hodnotou $1/d_{ij}$, kdy d_{ij} je euklidovská vzdálenost z města *i* do města *j*. α a β jsou parametry algoritmu, které určují relativní důležitost feromonu a viditelnosti. Pravděpodobnost je počítána jako kompromis mezi viditelností (preferují se bližší města) a intenzitou feromonu (často používaná hrana je žádoucí).

2.5.2 $\mathcal{MAX}$ -$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS)

Tento algoritmus je vylepšením původního algoritmu Ant System. Hlavní změnou je aktualizace feromonu, kdy pouze nejlepší mravenci mohou přidávat feromon a minimální a maximální hodnota feromonu je omezena. Ukládání je následující:

$$\tau_{ij} \leftarrow \left[(1-\rho) \cdot \tau_{ij} + \Delta\tau_{ij}^{best}\right]_{\tau_{min}}^{\tau_{max}} \quad (4)$$

kde τ_{max} a τ_{min} are horní a dolní mez feromonu. $\Delta\tau_{ij}^{best}$ je aktualizace feromonu definována jako

$$\Delta\tau_{ij}^{best} = \begin{cases} \frac{1}{L_{best}} & pokud\ nejlepší\ mravenec\ použil\ hranu\ (i,j) \\ 0 & jinak \end{cases} \quad (5)$$

L_{best} je délka cesty nejlepšího mravence. Hodnoty omezení minimální a maximální hodnoty feromonu se určí empiricky a jejich vylepšení potom závisí na druhu řešeného problému.

Proces aktualizace feromonu v $\mathcal{MM}$AS je ukončen ověřením hodnot feromonu, jestli se nachází všechny ve vymezených mezích:

$$\tau_{ij} = \begin{cases} \tau_{min} & pokud\ \tau_{ij} < \tau_{min} \\ \tau_{ij} & pokud\ \tau_{min} \leq \tau_{ij} \leq \tau_{min} \\ \tau_{max} & pokud\ \tau_{ij} > \tau_{min} \end{cases} \quad (6)$$

2.5.3 Ant Colony System (ACS)

Nejdůležitější změnou oproti původnímu AS je přidání lokální aktualizace feromonu do konečné aktualizace feromonu na konci konstrukce řešení (offline aktualizace). Lokální

aktualizaci feromonu provádí všichni mravenci po každém kroku a aplikují ji na poslední hranu, po které prošli:

$$\tau_{ij} = (1 - \varphi) \cdot \tau_{ij} + \varphi \cdot \tau_0 \tag{7}$$

kde $\varphi \in (0,1]$ je koeficient vypaření feromonu a τ_0 počáteční hodnota feromonu.

Hlavním cílem lokální aktualizace je rozšířit prohledávání dalšími mravenci během iterace. Snížení koncentrace feromonu na hranách, po kterých mravenec už přešel, podpoří další mravence, aby si vybrali jinou hranu a tím vytvořili jiné řešení. Tím je zabráněno případu, kdy několik mravenců vytvoří stejné řešení během jedné iterace.

Konečná aktualizace feromonu je provedena na konci každé iterace pouze jedním mravencem, který našel nejlepší řešení. Aktualizace je následující:

$$\tau_{ij} \leftarrow \begin{cases} (1 - \rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij} & pokud\ hrana\ (i,j)\ patří\ nejl.mravenci \\ \tau_{ij} & pro\ ostatní \end{cases} \tag{8}$$

Další rozdíl mezi AS a ACS lze najít v rozhodovacím pravidle během konstrukce řešení. V ACS se nazývá pseudorandom proporcial pravidlo. Pravděpodobnost mravence při pohybu z místa *i* do místa *j* závisí na náhodné proměnné *q*, která má rovnoměrné rozdělení [0,1].

3 ACO VE SPOJITÉM PROSTORU

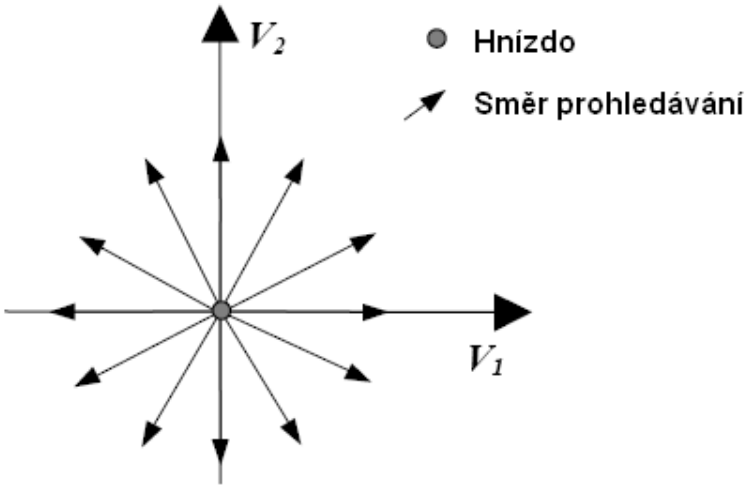
V minulé kapitole bylo popsáno, že ACO algoritmus je výkonným a všestranným nástrojem pro řešení různých kombinatorických problémů. V minulých letech začali vznikat první algoritmy pro spojité optimalizační problémy, i přesto že přímá aplikace ACO metaheuristic na spojité systémy není možná. První návrhy byly sice inspirovány z ACO, ale nenaplňovaly přesně stejnou metodologii. V roce 2004 K. Socha publikoval první skutečný ACO algoritmus pro spojitou optimalizaci $ACO_{\mathbb{R}}$.

3.1 ACO algoritmy pro optimalizaci ve spojitém prostoru

3.1.1 Continuous ACO (CACO)

Prvním pokusem o aplikaci mravenčího algoritmu na spojité optimalizační systémy byl algoritmus Continuous ACO (CACO) [20, 21]

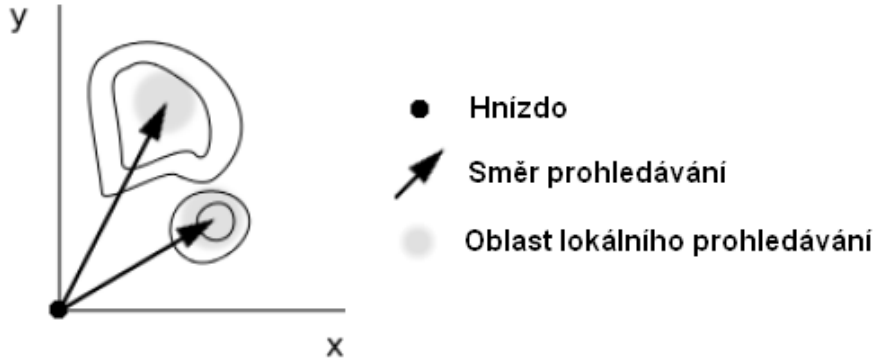
Mravenci začínají lokálně prohledávat své okolí z hnízda umístěného náhodně na ploše. Okolí je prohledáno pomocí *v* vektorů vedoucích z hnízda. Zpočátku jsou vektory rovnoměrně rozmístěny uvnitř poloměru prohledávání a obsahují pouze malé množství feromonu.



Obr. 7: Počáteční stav v CACO

Mravenci si vyberou směr pomocí náhodného výběru - rulety. Na počátku mají všechny vektory stejnou pravděpodobnost výběru. Podle kvality nalezeného řešení je vektorům přidána hodnota feromonu, tudíž nejlepší řešení mají potom nejvyšší pravděpodobnost

výběru mravencem. Po výběru vektoru mravenci přeskočí na nové místo, kam vektor ukazuje a lokálně prohledá okolí.



Obr. 8: Příklad prohledávání v CACO

Z nové pozice se vypočítá hodnota účelové funkce a přiřadí se množství feromonu. Pokud je nalezeno lepší řešení, vektor se změní podle aktuální pozice mravence.

Oproti původní myšlence algoritmus CACO obsahuje navíc položku *hnízdo*, která se v původním ACO vůbec nevyskytuje.

3.1.2 API

API je pojmenován podle mravenčího druhu *Pachycondyla apicalis* a je založen na hledání potravy, ale jejich strategie je odlišná od jiných mravenčích druhů. Tito mravenci mají jednoduchou strategii pro hledání kořisti, kdy každý jedinec loví sám a snaží se pokrýt celou danou oblast kolem hnízda. [22]

Mravenci tedy prohledávají prostor nezávisle na sobě, ale začínají ze stejného místa (hnízda). Hnízdo je potom přemístěno každou periodu *T* na souřadnice nejlepšího nalezeného řešení. Mravenci náhodně prohledávají blízké okolí kolem své oblasti. Pokud jsou pokusy neúspěšné a není nalezeno lepší řešení, je tato oblast zapomenuta a náhodně je zvolena jiná k prohledání. Po přemístění hnízda do nové lokace je vymazána paměť všech bývalých oblastí. Mravenci používají mechanismus, který se nazývá *tandem running*.

Tento algoritmus je možné použít jak pro diskrétní, tak pro spojité problémy.

3.1.3 Continuous Interacting Ant Colony (CIAC)

Třetím algoritmem pro spojitou optimalizaci je Continuous Interacting Ant Colony (CIAC). Oproti ostatním algoritmům obsahuje CIAC nový druh komunikace mezi mravenci. V ostatních algoritmech mravenci používají feromon jako jediný druh komunikace. V CIAC se objevuje kromě komunikace přes feromon přímá komunikace mezi mravenci. Ta probíhá následujícím způsobem:

Jeden mravenec pošle zprávu jinému náhodně vybranému jedinci. Pokud tato zpráva obsahuje lepší hodnotu účelové funkce, příjemce se začne pohybovat k odesílateli. Pokud neobsahuje lepší hodnotu, mravenec pošle novou zprávu.

3.2 ACO for Continuous Domain - $ACO_{\mathbb{R}}$

Všechny popsané předchozí algoritmy pro spojitou optimalizaci nedodržují přesnou původní myšlenku ACO. V roce 2004 vznikl algoritmus $ACO_{\mathbb{R}}$, který se nejvíce podobá původnímu záměru ACO.[5]

Z výše popsaných algoritmů pro spojité systémy byl vybrán právě tento algoritmus pro realizaci v programu Mathematica a následné testování. Hlavním důvodem pro výběr $ACO_{\mathbb{R}}$ je jeho největší podobnost s původním ACO algoritmem.

Algoritmy pro kombinatorické problémy používají feromonový model vhodný pro konstrukci pravděpodobnostního řešení. Hodnoty feromonů vyjadřují schopnost prohledávání. Feromony ovlivňují konstrukci řešení na prohledávaném prostoru nejvíce v místech, kde jsou nejlepší řešení.

$ACO_{\mathbb{R}}$ pracuje na principu přírůstkové konstrukce řešení založené na pravděpodobnostním výběru řešení, které je navíc ovlivněno hodnotou feromonu. Feromon je uložen do tabulky, která se nazývá *solution archive*. Ten obsahuje vždy všechna finální řešení problému. Solution archive je zde to samé jako velikost populace u evolučních algoritmů.

Algoritmus je následující:

Jako první proběhne inicializace solution archivu. Potom v každé iteraci mravenci pravděpodobnostně zkonstruují určitý počet řešení. Tato řešení mohou být dále vylepšena například pomocí lokálního prohledávání nebo gradientních technik. Nakonec se provede aktualizace solution archivu.

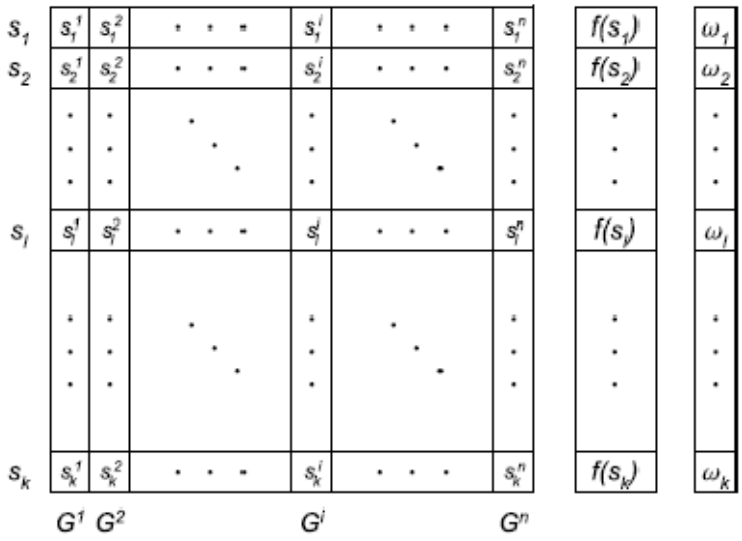
3.2.1 Struktura archivu, inicializace a aktualizace

$ACO_{\mathbb{R}}$ uchovává historii prohledávání a ukládá řešení do *solution archivu T*, který má rozměr $|T|=k$. Za předpokladu, že máme dán *n*-rozměrný spojitý optimalizační problém a *k* řešení, $ACO_{\mathbb{R}}$ ukládá v *T* hodnoty *n* proměnných a hodnotu jejich objektivní funkce *f(s).*

Hodnota *i*-té proměnné a *j*-tého řešení je vyjádřena jako s_j^i. Na *Obr. 9* je zobrazena struktura solution archivu. Řešení jsou seřazena v archivu podle jejich kvality, například pro minimalizační problém platí $f(s_1) \leq f(s_2) \leq \cdots \leq f(s_l) \leq \cdots \leq f(s_{2k})$.

Každé řešení má přiřazenou váhu ϖ, která závisí na kvalitě řešení. Váha je analogií k feromonu.

Proto platí $\omega_1 \geq \omega_2 \geq \cdots \geq \omega_l \geq \cdots \geq \omega_k$.



Obr. 9: Struktura solution archivu

Před startem algoritmu je archiv inicializován *k* náhodnými řešeními, to znamená, že je vygenerována počáteční populace. V každé iteraci *m* mravenců vytvoří novou množinu *m* řešení a tato řešení jsou přidána do archivu. Protože je nyní v archivu *k+m* řešení, tak *m* nejhorších odstraníme, aby archiv nenarůstal.

Zbývajících *k* řešení seřadíme podle jejich kvality (podle hodnoty účelové funkce) a uložíme jako nový archiv *T*. Tímto způsobem prohledávací proces upřednostňuje pouze nejlepší řešení nalezená během prohledávání.

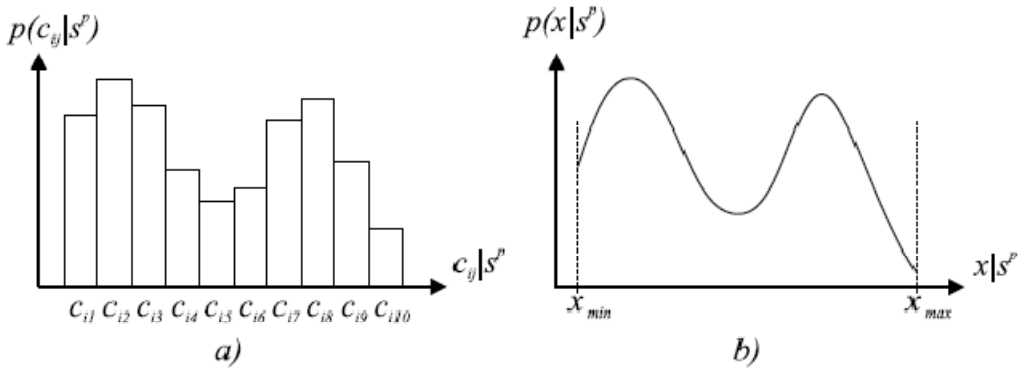
Řešení v archivu jsou vždy seřazena podle jejich kvality, tudíž nejlepší řešení je vždy nahoře (na prvním místě tabulky).

3.2.2 Konstrukce řešení

U ACO je množina komponent řešení definována při formulaci problému. V každém kroku konstrukce řešení si mravenci pravděpodobnostně vyberou jednu komponentu řešení c_i z množiny všech komponent N. Pravděpodobnost každého řešení je dána diskrétním rozdělením pravděpodobnosti.(*Obr. 1Obr. 10*a).

Hlavním znakem u $ACO_{\mathbb{R}}$ je použití spojitého rozdělení pravděpodobnosti místo diskrétního. Tím dostaneme pravděpodobnostní funkci (probability density function – PDF). Viz *Obr. 10*b.

Nejvhodnějším rozdělením pro použití se zdá Gaussovo (normální) rozdělení pravděpodobnosti. Hlavními výhodami je, že je flexibilní a nemá lineární průběh. Díky nelineárnímu průběhu lze nastavit vyšší pravděpodobnost několika nejlepším řešením, zatímco významně zredukujeme pravděpodobnost ostatních.



Obr. 10: Diskrétní (a) a spojité (b) rozdělení pravděpodobnosti

Mravenci provádí konstrukci nového řešení postupně pro každou proměnnou. Mravenec si vybere pravděpodobnostně jedno řešení z archivu. Pravděpodobnost výběru řešení j je dána rovnicí

$$p_j = \frac{\omega_j}{\sum_{r=1}^{k} \omega_r} \tag{9}$$

kde ϖ_j je váha (množství feromonu) přiřazena k řešení j.

Váha může být určena různými způsoby, přičemž záleží na řešeném problému. Nejvhodnější se zdá být opět Gaussovo rozdělení *g(μ,σ)=g(1,qk).*

Váhu potom vypočítáme podle vzorce

$$\omega_j = \frac{1}{qk\sqrt{2\pi}} e^{\frac{-(j-1)^2}{2q^2k^2}} \tag{10}$$

Střední hodnota pro Gaussovo rozdělení je nastavena na 1, takže nejlepší řešení mají nejvyšší váhu. Směrodatná odchylka *qk* je určena parametry *q* a *k*.

k je velikost archivu a *q* je parametr konvergence algoritmu, určuje rychlost konvergence k nalezení optima. Pokud je *q* malé, preferují se nejlepší řešení, pokud je *q* velké, pravděpodobnost výběru řešení se stává více rovnoměrnou.

Jakmile má mravenec vybráno řešení, může začít konstruovat nové řešení. Mravenec hledá řešení pro každou proměnnou *i=1, …, n* samostatně. Vezme se hodnota s_j^i, což znamená, že vezmeme proměnnou *i* vybraného řešení *j* a prohledáme její okolí. To je dáno pravděpodobnostní funkcí. Ta musí splňovat podmínku:

$$\int_{-\infty}^{\infty} P(x)dx = 1 \tag{11}$$

Stejně jako v případě výpočtu váhy, je možné použít více funkcí. Opět ale použijeme Gaussovo rozdělení pravděpodobnosti:

$$P(x) = g(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{12}$$

Funkce má dva parametry, které musíme definovat: μ a σ. Když zvážíme hodnotu s_j^i, tak lze přiřadit $\mu \leftarrow s_j^i$. σ se potom vypočítá podle vzorce

$$\sigma \leftarrow \xi \sum_{r=1}^{k} \frac{\left|s_r^i - s_j^i\right|}{k-1} \tag{13}$$

což představuje průměrnou vzdálenost mezi vybranou *i*-tou proměnnou pro s_j a všemi ostatními *i*-tými proměnnými v archivu.

Celý výsledek potom vynásobíme parametrem ξ, který má efekt podobný k vypařování feromonu v ACO. Čím vyšší je hodnota ξ, tím nižší je konvergence algoritmu.

Celý proces je opakován pro každý rozměr *i=1,…,n* každým z *m* mravenců.

3.2.3 Parametry ACO_R

Rozsah problému je dán definicí účelové funkce a je definován v tzv. *specimenu*.

Dimenze je dána počtem argumentů účelové funkce

Velikost archivu k je dána uživatelem. Tento parametr určuje, kolik jedinců bude tvořit populaci. Velikost populace by neměla být menší než je počet mravenců.

Počet mravenců určuje uživatel. Počet by neměl být menší než dva a zároveň by neměl přesáhnout velikost populace.

Iterace jsou nastaveny uživatelem. Tento parametr je ekvivalentem k parametru migrace nebo generace z jiných evolučních algoritmů. Udává počet kroků, po nichž se výpočet zastaví. V návrhu algoritmu a v samotném programu je použit název migrace.

ksí ξ je dáno uživatelem. $\xi>0$ a má stejný efekt jako vypařování feromonu v ACO. Čím vyšší je hodnota ξ, tím pomaleji algoritmus konverguje k optimu.

q je dáno uživatelem. Představuje rychlost konvergence k nalezení optima. Pokud použijeme větší q, algoritmus prohledává okolí pro více řešení spíše než pro málo nejlepších. Prohledávání je více rozšířené a algoritmus je robustnější na úkor pomalejší konvergenci. Pro *q=0.1* a *velikost archivu=50* dostaneme následující hodnoty: jedno z 5 nejlepších řešení je vybráno s pravděpodobností 68% a jedno z deseti nejlepších řešení s pravděpodobností 95%.

4 ALGORITMY SOMA A DE

Cílem práce je srovnání algoritmu ACO s dalšími dvěma evolučními algoritmy – algoritmem SOMA a DE. V následujícím textu budou stručně popsány oba algoritmy. Více informací lze najít v [3].

4.1 SOMA

Algoritmus SOMA (Samo-Organizující se Migrační Algoritmus) vznikl v roce 1999, je tedy o pár let starší než $ACO_{\mathbb{R}}$ (2004). Jeho činnost je založena na geometrických principech.

Původní myšlenka algoritmu spočívá v napodobení chování skupiny inteligentních jedinců, kteří kooperují při řešení společného problému, jako je např. hledání zdroje potravy. Tímto návrhem se SOMA v mnohém podobá mravenčím koloniím.

Důležitou vlastností u SOMA algoritmu je samoorganizace. Jedinci se ovlivňují navzájem během hledání lepšího řešení, což může vést k tomu, že v prostoru možných řešení vznikají skupiny jedinců, které putují přes prohledávaný prostor, nebo se naopak rozpadají či spojují. Skupina jedinců si sama organizuje vzájemný pohyb – odtud samoorganizace.

Princip algoritmu je následující. Na počátku je vygenerována nová populace jedinců. Každý z nich je ohodnocen účelovou funkcí a je zvolen Leader pro následující migrační kolo. Leader je jedinec, který má nejlepší hodnotu účelové funkce. V dalším kole se začnou ostatní jedinci pohybovat směrem k Leaderovi pomocí skoků. Po skoku je přepočítána hodnota účelové funkce každého jedince, a pokud je lepší, tak je zapamatována. Po ukončení běhu se jedinec vrací na pozici, kde byla nalezena nejlepší hodnota účelové funkce během jeho cesty.

Strategie SOMA algoritmu

AllToOne (všichni k jednomu) – všichni jedinci z populace migrují k Leaderovi

AllToAll (všichni ke všem) – zde neexistuje Leader. Všichni jedinci migrují ke všem ostatním jako v předchozí verzi s tím, že po dokončení migrací jedinec vrací na pozici, kde byla nalezena nejlepší hodnota účelové funkce.

AllToAllAdaptive (adaptivně všichni ke všem) – strategie stejná jako AllToAll, ale jedinec se přesunuje ihned na lepší pozici nalezenou v aktuální migraci. Z této pozice potom provádí migraci k dalším zbývajícím jedincům.

AllToOneRand (všichni k jednomu náhodně) – všichni se jedinci se pohybují k jednomu Leaderovi, který je po každé migraci vybrán náhodně z populace

4.2 DE

Diferenciální evoluce je také poměrně nový typ evolučního algoritmu (1995). Jeho princip je velmi podobný genetickým algoritmům, s nimiž má společné rysy jako například tvorbu potomků nebo používání generací.

Cílem diferenciální evoluce je v cyklech zvaných generace vyšlechtit co nejlepší populaci jedinců. Tvorba populace začíná vygenerováním množiny jedinců a u každého se určí hodnota účelové funkce. Během každé generace se provádí navíc cyklus, který zabezpečuje postupné šlechtění každého jedince. Pro každého jedince je proveden evoluční cyklus. Ten zahrnuje mutaci a křížení. Náhodně se zvolí tři jiní jedinci z populace. První dva se od sebe odečtou a tím se získá *diferenční vektor*. Ten se vynásobí *mutační konstantou*, tím jej zmutuje a dostaneme *váhový diferenční vektor*. Tento vektor přičteme ke třetímu vybranému jedinci a získáme *šumový vektor*. Poté se připraví zkušební vektor a z cílového a šumového vektoru se vyberou postupně prvky a pro tyto dvojice se generuje náhodné číslo z rozsahu 0 až 1 a porovnává se s konstantou CR. Pokud je toto číslo menší než CR, pak se do zkušebního vektoru umístí prvek z vektoru šumového, jinak z vektoru cílového.

Tím dostaneme zkušební vektor a cílový vektor, jejichž hodnoty účelové funkce se porovnají a do nové populace vstupuje ten, který má hodnotu lepší. Tím je zajištěno, že se do nové generace dostanou pouze jedinci s lepšími vlastnostmi. Evoluční cyklus se opakuje až do vyčerpání populace. Tak vznikne nová generace potomků (jedinců).

Diferenciální evoluce má tu zvláštnost, že k tvorbě nového jedince je potřeba ne dvou, ale čtyř rodičů. Ti vytvoří potomka, který nakonec soupeří o místo v nové populaci. V procesu tvorby nového potomka je uplatněna náhoda dvojím způsobem – pomocí křížící konstanty CR a pomocí šumového vektoru. Díky tomu je diferenciální evoluce poměrně robustní algoritmus s poměrně vysokou diverzibilitou.

II. PRAKTICKÁ ČÁST

5 POPIS PROGRAMU

Náplní praktické části práce je porovnání algoritmu $ACO_{\mathbb{R}}$ s dalšími dvěma algoritmy – algoritmem SOMA a algoritmem DE.

Testování bude probíhat na typických testovacích funkcích používaných pro testování optimalizačních algoritmů. Oba výše zmiňované algoritmy jsou naprogramovány v prostředí Mathematica. Toto prostředí bylo zvoleno i pro $ACO_{\mathbb{R}}$, aby bylo možné co nejlépe vyhodnotit a porovnat výsledky.

5.1 Mathematica

Mathematica představuje integrovaný systém umožňující numerické i symbolické výpočty, práci s daty, grafické zpracování výsledků.

Silnou stránkou tohoto systému je vlastní programovací jazyk na bázi jazyků umělé inteligence. Symbolický jazyk, který program využívá, umožňuje representovat data, funkce, grafy, programy a dokonce i celý dokument jednotným způsobem a to jako symbolický výraz. Tento výraz může být matematickou funkcí, grafikou, zvukem, programem i celým Mathematica dokumentem. Může fungovat jako vstup i výstup jiné funkce a umožňovat tak stručné a jednoduché kódování. Toto sjednocení má mnoho praktických výhod od jednoduchosti při učení po rozšiřování oblasti použitelnosti každé funkce. [25]

Další velkou výhodou je využití nezávislého interakčního dokumentu notebook. Notebookové rozhraní kombinuje textový procesor se zřetelně definovanou představou buněk, které jsou svisle seřazeny v rozbalovacím okně jako odstavce textu. Buňky jsou velmi důležité, protože vizuálně a funkčně oddělují text na vstupy, výstupy, grafiku, nadpisy atd. buňky jsou přizpůsobitelné, aby podporovaly jakýkoliv typ a velikost výrazu, poskytly jednoduchou úpravu a vložení a byly snadno rozšiřitelné.

Díky tomu *Mathematica* nachází široké uplatnění v praxi zejména v oblastech vědecko-technických výpočtů, statistickém zpracování dat, finančním managementu atd. Jednotná koncepce systému umožňuje studovat závislost matematických modelů reálných systémů na parametrech jak symbolicky tak numericky Tím se *Mathematica* stává mocným nástrojem pro výzkum a vývoj, ale též názornou pomůckou pro výuku.

Využití softwaru Mathematica:

- Zpracování komplexních symbolických výpočtů, které obsahují velké množství členů
- Zavádění, analýza a vizualizace dat
- Řešení rovnic, diferenciálních rovnic a minimalizace problémů numericky nebo symbolicky
- Provádění numerických modelů a simulací, například modelování regulačních systémů, finančních, složitých biologických systémů, chemických reakcí, vlivů na prostředí
- Výroba profesionálně kvalitních, interaktivních technických dokumentů
- Ilustrace matematických nebo vědeckých konceptů

5.2 Popis programu

Pro naprogramování algoritmu $ACO_{\mathbb{R}}$ v prostředí Mathematica bylo použito několik zdrojových materiálů s popisem tohoto algoritmu. Autoři zde uvádí některé termíny, které ale byly změněny podle algoritmů SOMA nebo DE pro lepší orientaci a pochopitelnost algoritmu.

Například místo termínu iterace je používán termín migrace. Pro termín populace je v ACO zaveden termín *solution archiv* (příp. jen archiv), což může být někdy zavádějící, proto jsou v programu používány obě označení. ACO nezná ani termín jedinec, ale místo toho je zde termín komponent daného řešení. V programu je přesto používán termín jedinec, protože je výstižnější a lépe srozumitelný v řeči evolučních algoritmů

Program se skládá z několika částí.

V první části jsou definovány používané knihovny:

```
<<Statistics`ContinuousDistributions`
<<Graphics`MultipleListPlot`
<<Graphics`Graphics`
<<AGO`Functions`
```

Pro program byly potřeba čtyři knihovny: jedna pro statistiku a dvě pro grafické zobrazení výsledků.

Poslední používanou knihovnou je seznam všech testovacích funkcí a jejich definic pro jakékoliv množství dimenzí. Autorem knihovny je doc. Ing. Ivan Zelinka Ph.D.

V následující tabulce (*Tabulka 2*) jsou popsány jednotlivé kroky algoritmu. Je to jakási sekvence kroků, které algoritmus postupně vykonává. Dále v textu budou všechny kroky podrobně popsány.

Tabulka 2: Sekvence jednotlivých kroků v programu

1. Nastavení parametrů

Specimen, dimenze, počet mravenců, velikost archivu (populace), q, ksi

2. Inicializace solution archivu

Vygenerování nové populace a výpočet hodnot účelové funkce

Seřazení solution archivu podle kvality hodnoty účelové funkce

Vygenerování a přiřazení feromonu jednotlivým řešením

3 Konstrukce řešení

Pro všechny migrace

Pro všechny mravence

Mravenec si vybere jedno řešení z archivu

Pro všechny dimenze

Výpočet pravděpodobnostní funkce (PDF)

Vzorkování okolí a hledání nejlepšího jedince

Přidání řešení do archivu

Aktualizace archivu, seřazení archivu a vymazaní nejhorších řešení

Konec algoritmu

4. **Výsledek = nejlepší nalezené řešení**

5.2.1 Nastavení parametrů

Nastavení parametrů je potřeba věnovat řádnou pozornost, protože jsou pro algoritmus klíčové. Při špatně nastavených parametrech nebude algoritmus správně fungovat. Pro všechny testovací funkce byly nastaveny stejné hodnoty parametrů (kromě specimenu). Jako vzorové hodnoty použijeme nastavení pro funkci De Jong 1st.

```
1 Specimen=Table[{{Real,{-5.12,5.11}}},{i,1,100}];
2 CostFunction=DeJong1st;
3 NumberOfAnts=10;
4 Dim=Dimensions[Specimen][[1]];
5 archiv=50;
6 q=0.1;
7 ksi=0.85;
```

Na prvním místě musíme definovat *specimen*. Specimen je vzorový jedinec, podle kterého se vygeneruje celá počáteční populace. Jeho struktura je následující:

$$Specimen = \{\{typ, \{Lo, Hi\}\}, \{typ, \{Lo, Hi\}\}, \dots, \{typ, \{Lo, Hi\}\}\} \qquad (14)$$

Ve vzorovém jedinci jsou pro každý parametr konkrétního jedince z populace definovány tři parametry. Prvním je typ proměnné (může být například celočíselný, reálný, diskrétní atd.). Druhým parametrem je hranice intervalu, v němž se hodnota každého parametru jedince (argumentu funkce) může pohybovat, a to od spodní hranice *Lo* po horní hranici *Hi*.

Specimen určuje i dimenzi problému (řádek 4) a rozsah testované funkce. Pro všechny testy byla použita dimenze 100. Rozsah pro testování je nastavena reálné hodnoty (parametr `Real`) a rozsah `{-5.12,5.11}`.

Na řádku 2 je definována testovací funkce. Díky knihovně `<<AGO`Functions`` nemusíme zadávat funkci rovnicí, ale pouze pomocí jejího názvu. Například `DeJong1st`, `Rastrigin`, `StretchedSine`.

Další parametr `NumberOfAnts` určuje, kolik mravenců bude použito v každé migraci pro prohledávání prostoru. Mravenec si vybere jedno řešení z archivu, pro které hledá nové lepší řešení prohledáváním okolí. Počet mravenců je nastaven na hodnotu 10, která se zdá být přijatelná pro účely testování. V literatuře [5,6] bylo použito nastavení s dvěma mravenci, nicméně vzhledem k velikosti archivu se mi zdá toto číslo nedostatečné.

Na řádku 5 je parametr `archiv` nastaven na 50. Archiv určuje velikost populace.

Další dva parametry `q` a `ksi` byly popsány v kapitole **Chyba! Nenalezen zdroj odkazů.**, jejich nastavení je opět podle dostupné literatury. Pro všechny funkce je hodnota `q` nastavena 0.1, což zaručuje rychlejší konvergenci k optimu. U většiny multimodálních funkcí by měla být hodnota `q` vyšší, aby populace příliš rychle nesklouzla k lokálnímu minimu.

5.2.2 Popis funkcí v programu

Program je rozdělen do několika funkcí, a to hlavně z důvodů, aby byl kód přehledný a snadno upravitelný.

Vygenerování nové populace

Na začátku algoritmu je funcí `DoPopulation` vygenerována nová populace jedinců pomocí náhodného generátoru čísel s rovnoměrným rozdělením. Pro všechny jedince je vypočítána hodnota jejich účelové funkce (`costvalue`).

```
8 DoPopulation[archiv_, Specimen]
```

`archiv` určuje velikost populace a `Specimen` definuje typ proměnné a rozsah intervalu.

Seřazení populace

Funkce `Serazeni` seřadí počáteční populaci podle kvality řešení – tzn. podle jejich hodnoty účelové funkce (fitness) a vrátí seřazenou populaci.

Při běhu programu seřazení populace probíhá po dokončení každého migračního kola.

Generování feromonu

Po vytvoření počáteční populace je pro tuto populaci vygenerován feromon ve funkci `GenerujFeromon` podle rovnice $\omega_j = \frac{1}{qk\sqrt{2\pi}} e^{\frac{-(j-1)^2}{2q^2k^2}}$

(10). Feromon je generován podle normálního (Gaussova) rozdělení. Parametr `q` určuje, jak moc bude rozdělení stejnoměrné. Pro velké `q` dostaneme téměř rovnoměrné hodnoty, pro malé `q` je v rozdělení strmější.

Výběr řešení z archivu

Výběr řešení z archivu závisí na tom, jak velký

Funkce `VyberReseni` vypočítá z hodnoty feromonu pravděpodobnost výběru řešení. Ta je dána rovnicí $pj = \frac{\omega_j}{\sum_{r=1}^{k} \omega_r}$ (9), kdy pravděpodobnost určíme podílem feromonu přiřazeného k danému řešení a součtu všech hodnot feromonů. Tím dostaneme vektor `vyberzarchivu` o délce populace s hodnotami pravděpodobnosti pro každé řešení. To znamená, že na prvním místě je řešení, které má největší pravděpodobnost, na posledním místě je nejhorší řešení a má přiřazenu nejmenší pravděpodobnost. Ve výsledku jsou nejvíce preferována řešení s nejvyšším feromonem, a tedy s nejvyšší pravděpodobností.

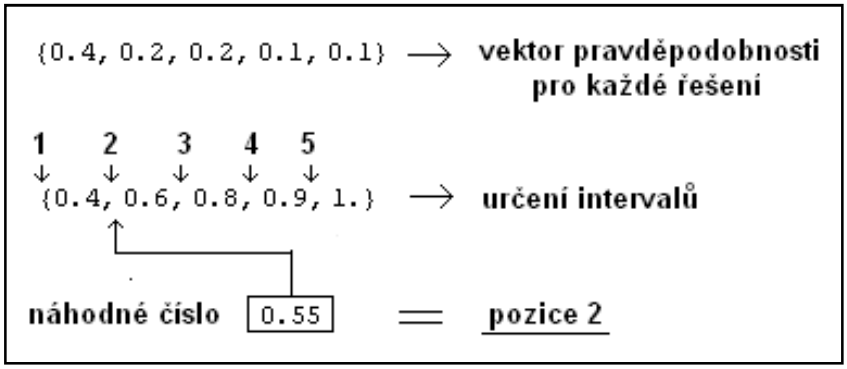
```
 9 soucet=0;
10 pozice=1;
11 intervaly=Table[soucet+=vyberzarchivu[[i]],{i,1,archiv}];
12 nahcislo=Random[];
13 While[nahcislo>intervaly[[pozice]],pozice++];
```

Proměnné `soucet` a `pozice` jsou nastaveny na počáteční hodnoty (řádek 9,10).

V proměnné `intervaly` (řádek 11) je vytvořeno několik intervalů, kde je každá pravděpodobnost přičtena k té předchozí a tím dostaneme součet intervalů 1.

Poté je vygenerováno náhodné číslo `nahcislo` a určíme pozici, do kterého intervalu padlo. Funkce `VyberReseni` vrací právě tuhle pozici – číslo pozice.

Zde je uveden příklad výběru pro velikost populace 5:



Obr. 11: Výběr řešení z archivu

5.2.3 Spuštění běhu programu a simulace

Funkce `StartACO` a `ACO` tvoří hlavní tělo algoritmu. Po spuštění algoritmu je vygenerována nová populace příkazem

```
14 pop=DoPopulation[archiv,Specimen ];
15 StartACO[1000]
```

Potom je spuštěn příkaz `StartACO`, jehož parametrem je počet migrací.

Funkce `StartACO` je následující.

Jejím parametrem je migrace, která udává počet migračních kol neboli iterací algoritmu. Poté je vygenerován `feromon` a do proměnné `populace` je uložena seřazená populace podle kvality řešení, tj. podle hodnoty účelové funkce (řádky 16,17,18)

```
16 StartACO[migrace_]
17 feromon=GenerujFeromon;
18 populace=Serazeni;
19 bestfit=First[populace][[1]];
20 bestfitness={{bestfit}};
```

V dalším kroku je do proměnné `bestfit` uložena nejlepší hodnota fitness za celou migraci. Hodnotu fitness z každé migrace ukládáme pro následné vyhodnocení průběhu nalezení optima. Kvůli následnému zápisu do souboru a snadnému čtení dat ze souboru vložíme `bestfit` do dvojitých závorek a uložíme do nové proměnné `bestfitness`.

Cyklus tvorby nových populací zajistí funkce NestList.

```
21 NestList[ACO,populace,migrace];
```

Jejími parametry jsou `ACO`, což je funkce, ve které probíhají veškeré kroky výpočtu nových řešení, `populace` je vstupní proměnnou do funkce `ACO` a nakonec `migrace` udává počet iterací algoritmu.

Funkce ACO

V solution archivu (populaci) je uložena pro každé řešení hodnota účelové funkce a její argumenty. V dalším běhu programu je zapotřebí pracovat pouze s těmito argumenty, tedy z populace vybereme pouze je a uložíme do proměnné `columns`.

```
22 columns=Table[Part[populace[[i]],2],{i,1,archiv}];
```

Následuje For cyklus, ve kterém jsou všechny operace prováděny postupně pro každého mravence. Postup je následující:

Mravenec si vybere jedno řešení z archivu, to je uloženo do proměnné `vyber`. Do další proměnné `ant` se uloží ten řádek (fitness a její argumenty) z archivu, který si mravenec vybral. Jelikož není potřeba pracovat s fitness, tak do proměnné `individuals` uložíme pouze argumenty.

```
23 vyber=VyberReseni;
24 ant=populace[[vyber]];
25 individuals=ant[[2]];
```

Pro všechny jedince - argumenty (ty jsou dány dimenzí – kolik je dimenze, tolik je argumentů) je prováděno vzorkování okolí a hledání lepší hodnoty. To probíhá v několika krocích.

První musíme určit velikost prohledávaného okolí kolem jedince. To vypočítáme podle vzorce $\sigma \leftarrow \xi \sum_{r=1}^{k} \frac{\left|s_r^i - s_j^i\right|}{k-1}$ (13).

```
26 sigma = (1/(archiv - 1)) * ksi * Sum_{k=1}^{archiv} (Abs[sloupec[[k]] - sloupec[[vyber]]]);
```

Do dalších proměnných uložíme typ proměnné (Real, Integer, atd.) a počáteční a konečný interval. Tyto proměnné se využijí při prohledávání okolí. Prohledávání okolí je uloženo v samostatné funkci `SamplingOkoli`.

```
27 Typ=Specimen[[n]][[1]][[1]];
28 SpecIntFrom=Specimen[[n]][[1]][[2]][[1]];
29 SpecIntTo=Specimen[[n]][[1]][[2]][[2]];
30 SamplingOkoli[jedinec,sigma];
```

Vzorkování okolí

Vzorkování okolí probíhá ve chvíli, kdy má mravenec vybráno jedno řešení z archivu a pro každého jedince se hledají nové lepší hodnoty v jeho okolí. Vstupními parametry do funkce `SamplingOkoli` jsou `ind` a `smodch`. V proměnné `ind` je uložena hodnota jedince, v smodch je hodnota sigma – velikost prohledávaného okolí.

Nejdříve se vygeneruje náhodné číslo x z intervalu okolí jedince a typu definovaného ve *Specimenu*. Pro toto náhodné číslo x se spočítá fitness a porovná se s fitness jedince. Ten, který má lepší hodnotu, se uloží do proměnné bestInd. Aby náhodně vygenerované číslo

nepřekročilo povolený interval funkce, tak je na řádku 35 ověřeno, jestli nový jedinec splňuje podmínku omezení. Pokud ji nesplňuje, jako nového jedince bereme původního jedince.

```
31 SamplingOkoli[ind_,smodch_]
32 x=Random[Typ,{ind-smodch,ind+smodch}];
33  cv=CostFunction[{x}];
34  If[CostFunction[{ind}]<CostFunction[{x}],bestInd=ind ,bestInd=x];
35  If[bestInd>SpecIntTo || bestInd<SpecIntFrom, bestInd=ind];
36  individuals=ReplacePart[individuals,bestInd,n];
37  newParameters=individuals;
```

V proměnné `bestInd` je nyní hodnota nejlepšího nalezeného jedince. Toho uložíme do proměnné `individuals`, kde jím nahradíme předchozího jedince. V této proměnné se nahromadí postupně všichni noví jedinci pro všechny dimenze. Až je prohledávání okolí hotovo, v další proměnné `newParameters` jsou uloženy všechny nové nalezené nejlepší parametry, tedy celé nové řešení.

Pro toto nové řešení je vypočítána hodnota fitness, nové řešení `newSolution` je přidáno do populace `pop` a ta je opět seřazena podle kvality řešení a uložena do proměnné `newPop`.

```
38 newSolution={CostFunction[newParameters ],newParameters};
39 newPop=Sort[AppendTo[pop,newSolution]] ;
```

Ve chvíli, kdy je výpočet hotov pro všechny mravence, je v nové populaci `newPop` o tolik více nových řešení, kolik je mravenců. Aby po každém migračním kole nenarůstala populace o tuto hodnotu, je populace opět uspořádána a je smazáno tolik nejhorších řešení, kolik je počet mravenců. Tím je zaručeno, že po každém migračním kole jsou nejhorší řešení vymazána a nahrazena novými a lepšími (řádek 42).

```
40 indivs=Table[Part[newPop[[i]],2],{i,1,archiv+NumberOfAnts}];
41 cost=Table[First[newPop[[i]]],{i,1,archiv+NumberOfAnts}];
42 populace=Delete[MapThread[{#1,#2}&,{cost,indivs}],
   Partition[Table[-i,{i,1,NumberOfAnts}],1]];
```

V proměnné populace je vždy uložena aktuální populace v každé migraci.

Ukládání hodnot do souboru

Kvůli více simulacím je v programu implementováno ukládání hodnot do souborů. Aby mohly být všechny simulace vyhodnoceny a znázorněny do grafů, je potřeba uložit vždy nejlepší hodnotu fitness z každé populace pro všechny migrace. Zde je kód pro uložení těchto nejlepších hodnot do proměnné `nejlepsifitness`, která se během simulací zapíše do souboru.

```
43 bestfit=First[populace][[1]];
44 nejlepsifitness=AppendTo[bestfitness,{bestfit}];
```

Simulace

Kód pro simulaci vypadá následovně:

```
45 simulations=100;
46 StartACO[1000];
47 Print["simulace  ", sim];
48 vysledky=populace[[1]] ;
49 PutAppend[nejlepsifitness,"DeJong1stFit.txt"];
50 PutAppend[vysledky,"DeJong1st.txt"],
```

Na řádku 45 je nastavení počtu simulací. Dále se spustí funkce `StartACO` s nastavením počtu migrací. Po každé vykonané simulaci je zobrazena informace o tom, kolik simulací už proběhlo. Do proměnné `vysledky` jsou uloženy všechny nejlepší populace pro každou simulaci a ty jsou následně zapsány do souboru, zde například do souboru *DeJong1st.txt*. Každá nejlepší fitness za každou migraci a pro všechny simulace je uložena do dalšího souboru, *DeJong1stFit.txt*.

Pro každou testovací funkci jsou samozřejmě soubory pojmenovány podle názvu této funkce.

Hodnoty do souboru jsou zapisovány průběžně po každé simulaci.

5.2.4 Zobrazení grafů

Uložené výsledky že všech simulací jsou použity pro vykreslení grafů. Výsledné grafy slouží pro porovnání výsledků s jinými algoritmy, zde s algoritmem SOMA a DE.

Program zobrazuje dva grafy.

V prvním je zobrazen průběh hodnot účelové funkce pro všechny migrace.

Druhým grafem je histogram zobrazující v procentech vzdálenost nejlepšího řešení od globálního optima.

6 TESTOVACÍ FUNKCE

Pro testování algoritmu ACO byly vybrány stejné funkce, které již byly použity pro testování algoritmů SOMA a DE.

Funkce byly testovány pro 100D prostor, u každé testovací funkce je uvedeno minimum a maximum ve 100D.

Pro každou funkci je zobrazeno několik grafů. První je graf funkce v jednodimenzionálním prostoru. Pro dvě dimenze je zobrazen 3D graf a density plot. Na tomto grafu je minimum znázorněno černou barvou.

U některých funkcí je více grafů pro 2D. První sada grafů zobrazuje funkci pro zadané intervaly, druhá sada potom zobrazuje funkci více detailně.

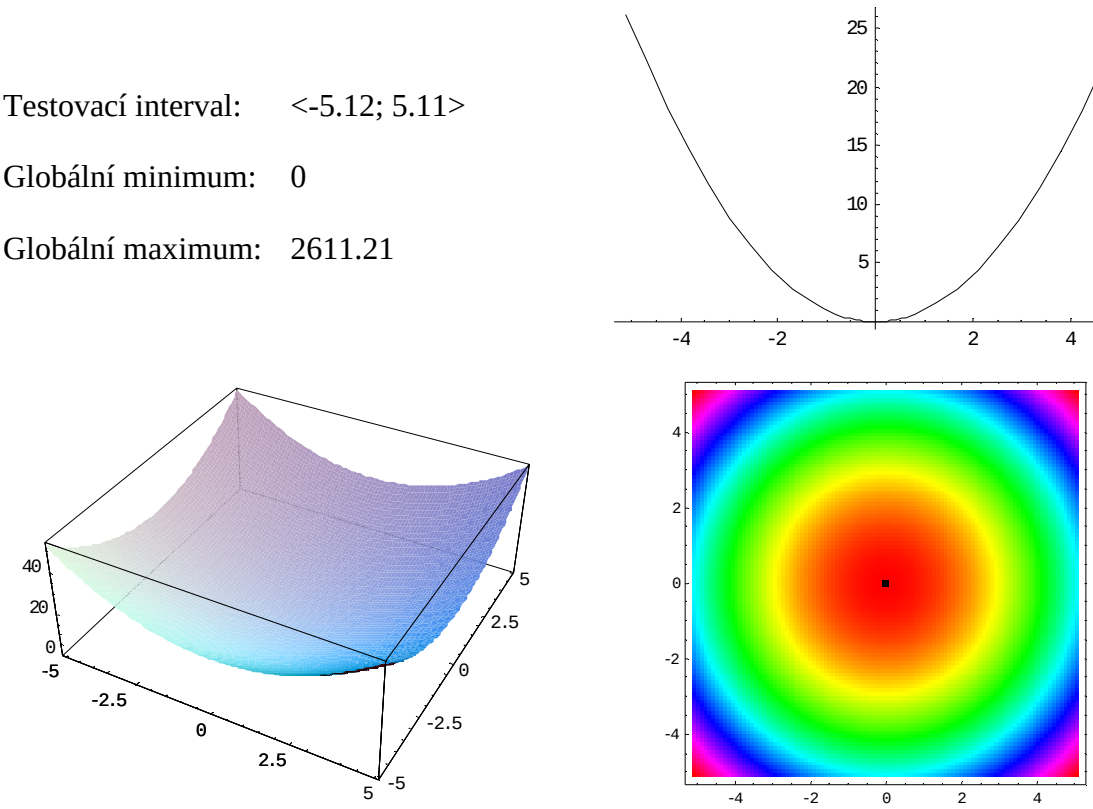
Rovnice testovacích funkcí nebudu uvádět pro velkou složitost u některých. Jejich předpisy lze nalézt v [23]

6.1 1st De Jong's function

Testovací interval: <-5.12; 5.11>

Globální minimum: 0

Globální maximum: 2611.21



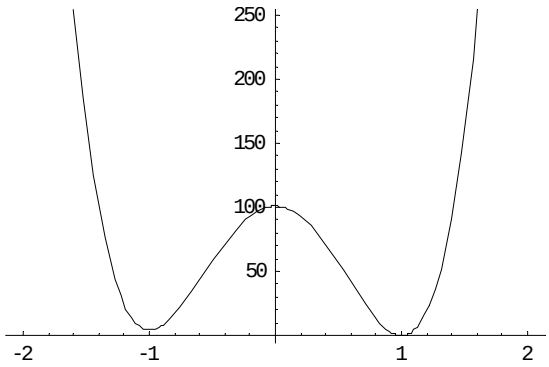
Obr. 12: Grafy 1st De Jong's function

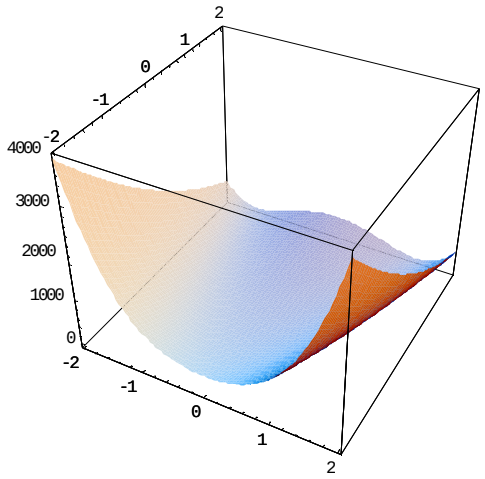
6.2 Rosenbrock's saddle

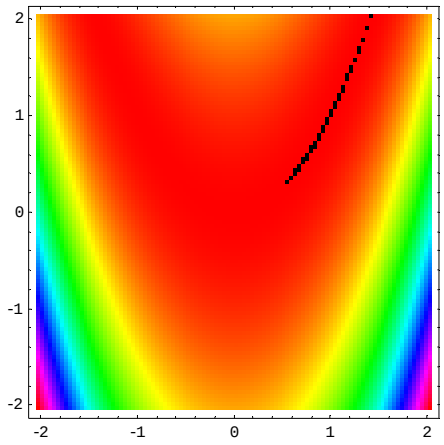
Testovací interval: <-2.048; 2.047>

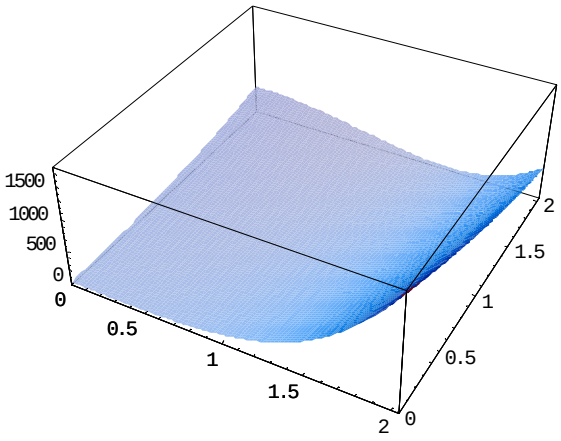
Globální minimum: 0

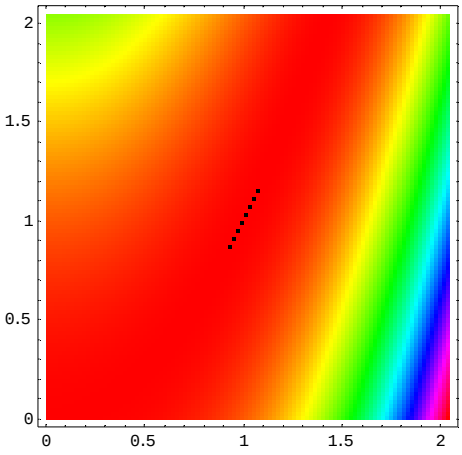
Globální maximum: 357291











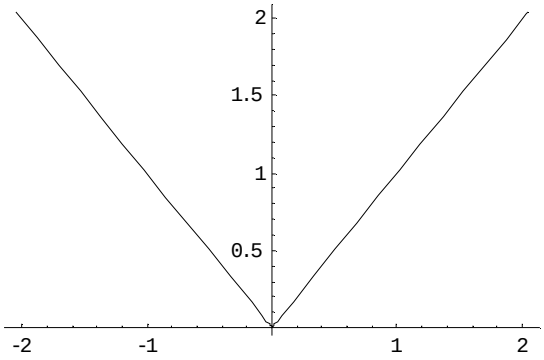
Obr. 13: Grafy Rosenbrock's saddle

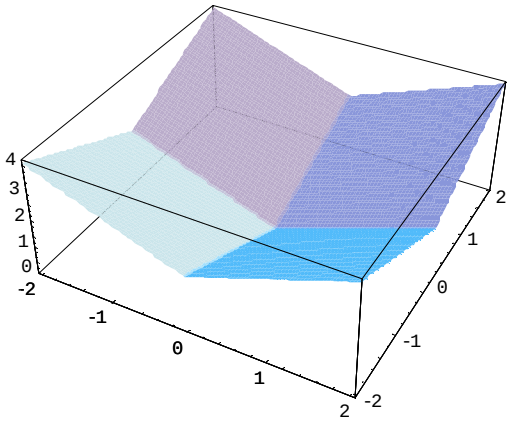
6.3 3rd De Jong's function

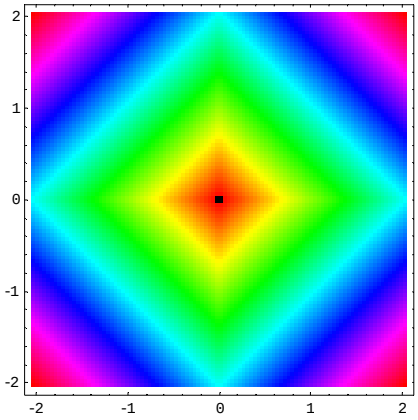
Testovací interval: <-2.048; 2.047>

Globální minimum: 0

Globální maximum: 204.8







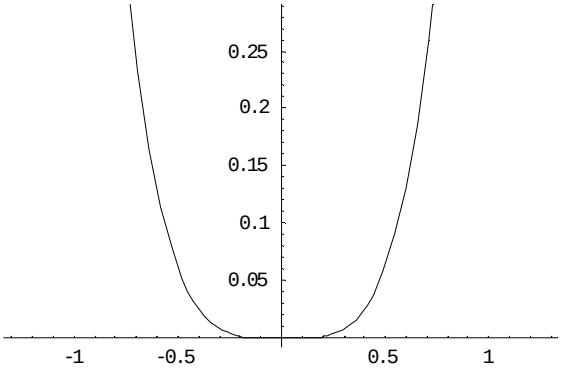
Obr. 14: Grafy 3rd De Jong's function

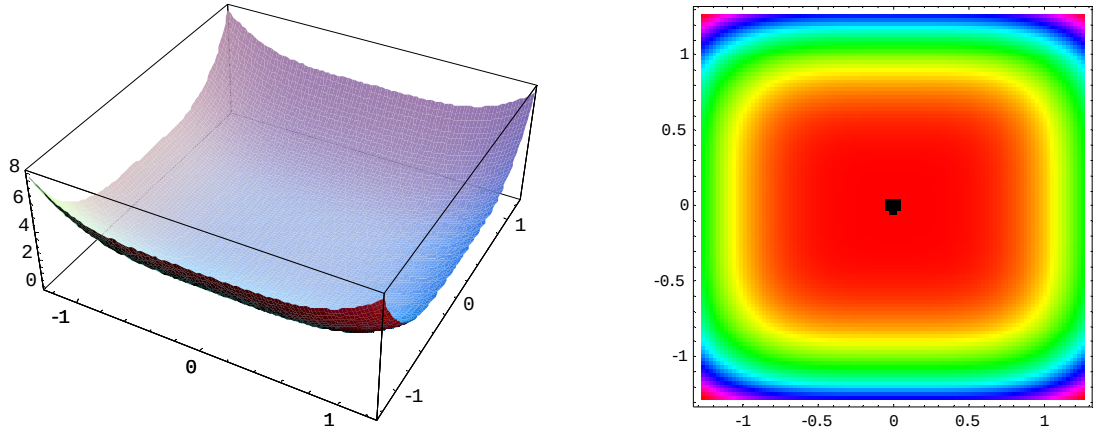
6.4 4th De Jong's function

Testovací interval: <-1.28; 1.27>

Globální minimum: 0

Globální maximum: 13556

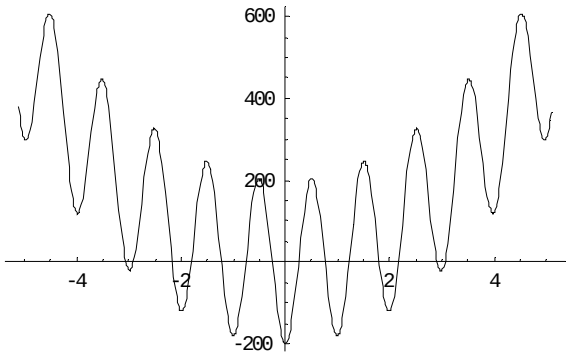


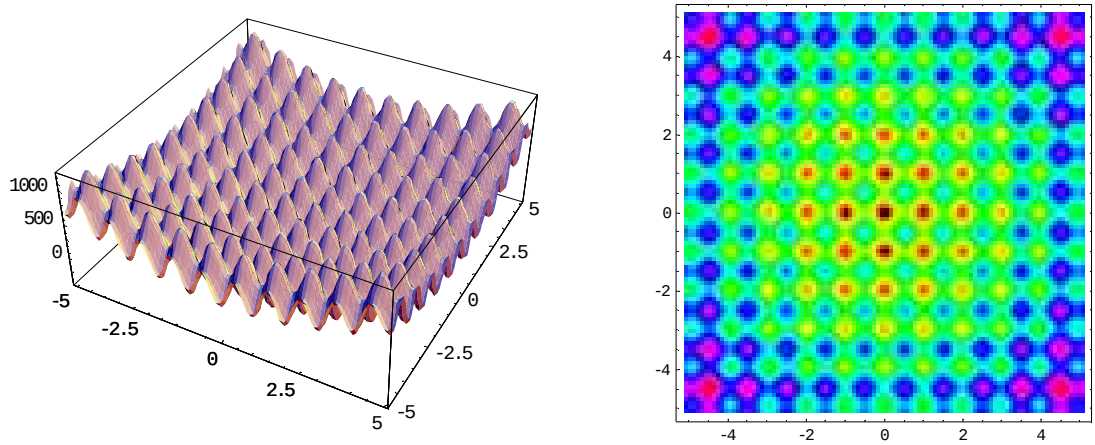


Obr. 15: Grafy 4th De Jong's function

6.5 Rastrigin's function

Testovací interval:	<-5.12; 5.11>
Globální minimum:	-20000
Globální maximum:	60706.6





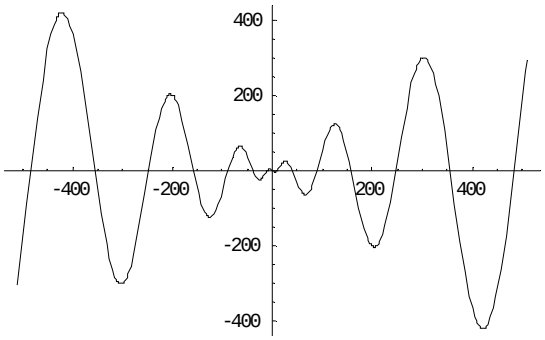
Obr. 16: Grafy Rastrigin's function

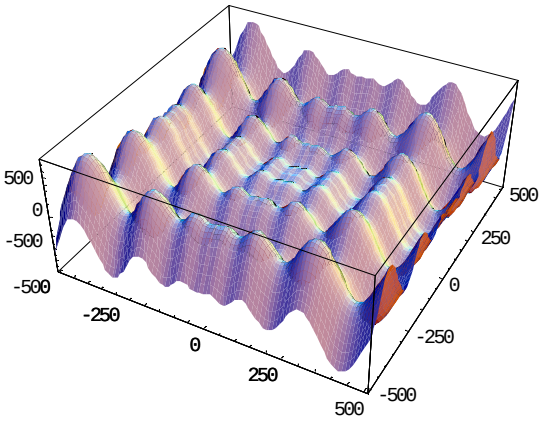
6.6 Schwefel's function

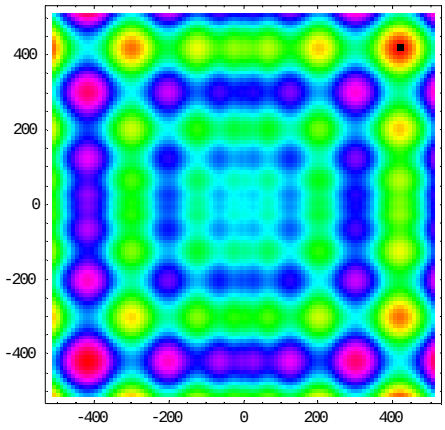
Testovací interval: <-512; 511>

Globální minimum: -41899

Globální maximum: 41898







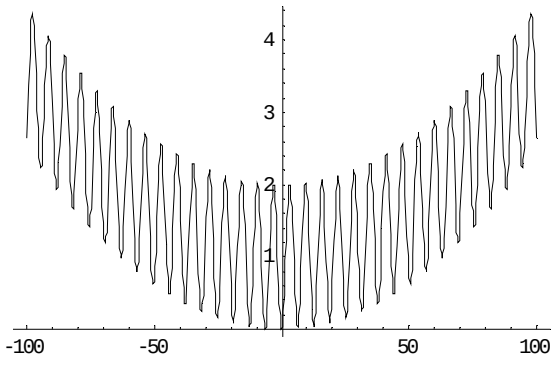
Obr. 17: Grafy Schwefel's function

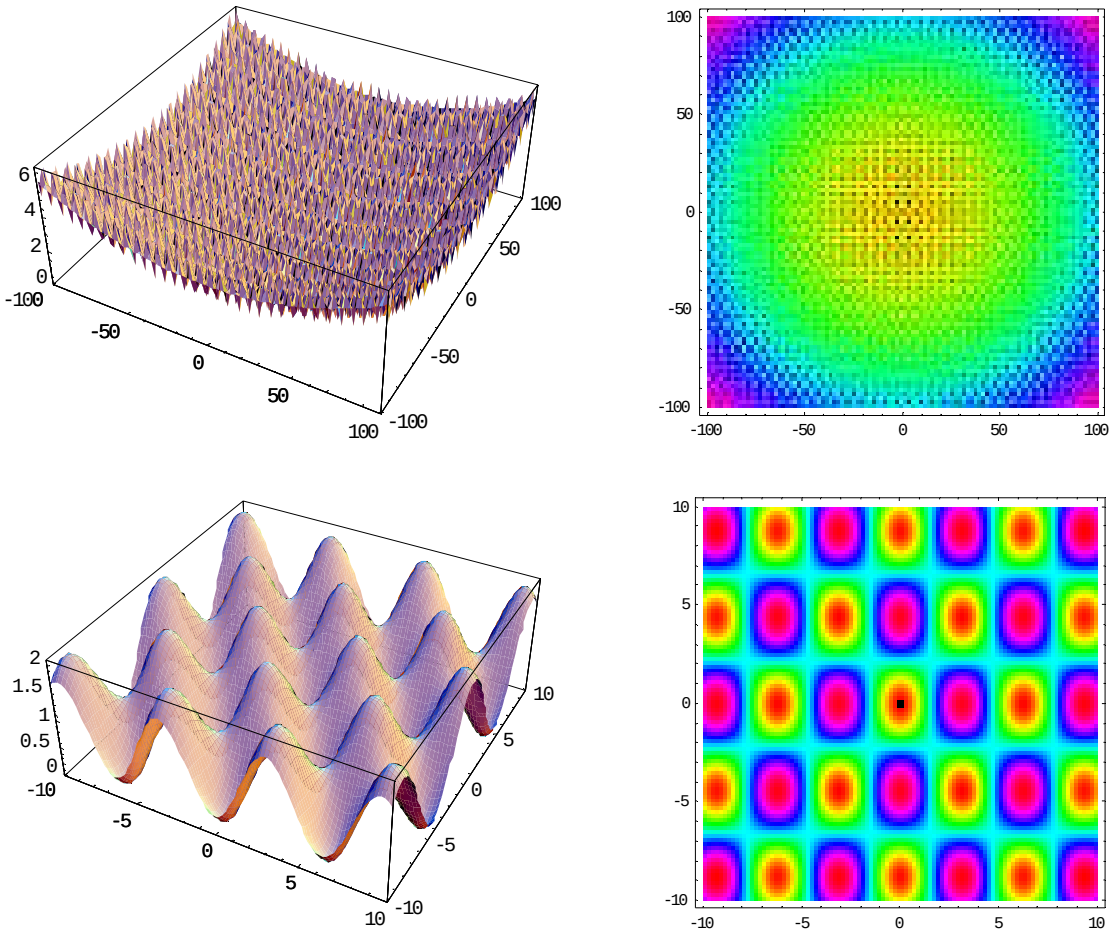
6.7 Griewangk's function

Testovací interval: <-100; 100>

Globální minimum: 0

Globální maximum: 251

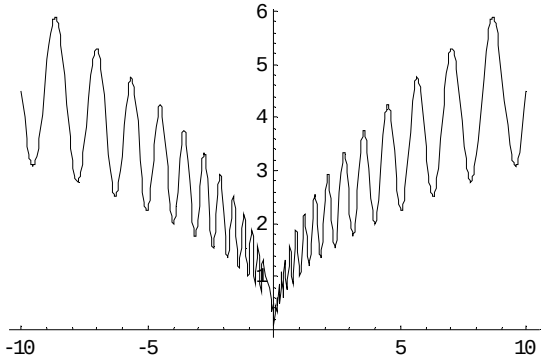


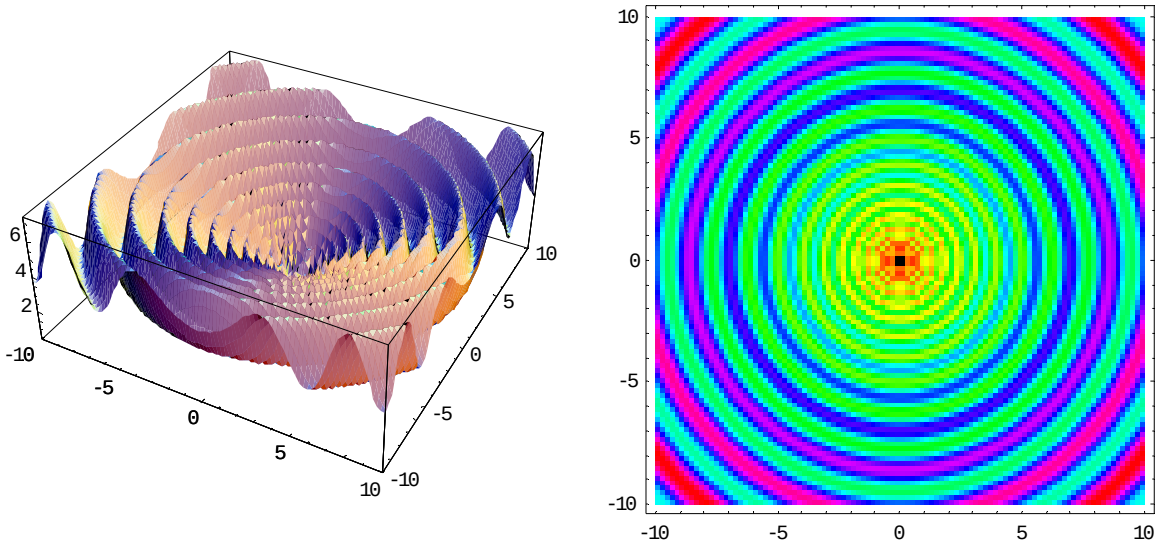


Obr. 18: Grafy Griewangk's function

6.8 Stretched V sine wave function (Ackley)

Testovací interval:	<-10; 10>
Globální minimum:	0
Globální maximum:	708.348

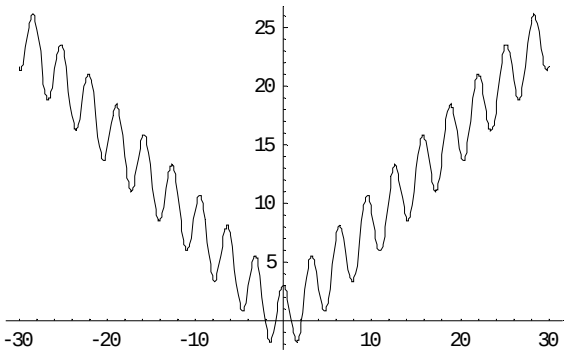


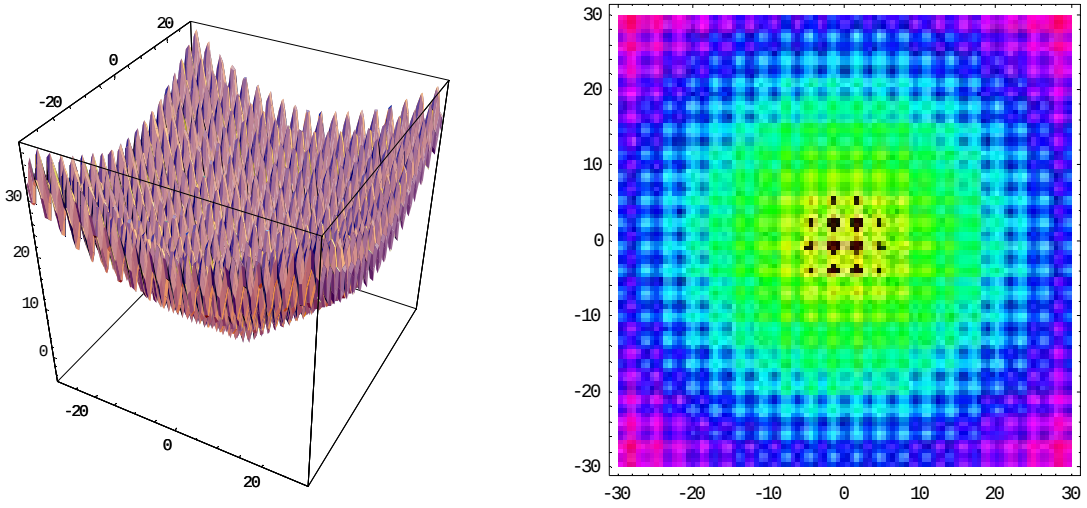


Obr. 19: Grafy Stretched V sine wave function (Ackley)

6.9 Test function (Ackley)

Testovací interval:	<-30; 30>
Globální minimum:	-246.457
Globální maximum:	2315.78

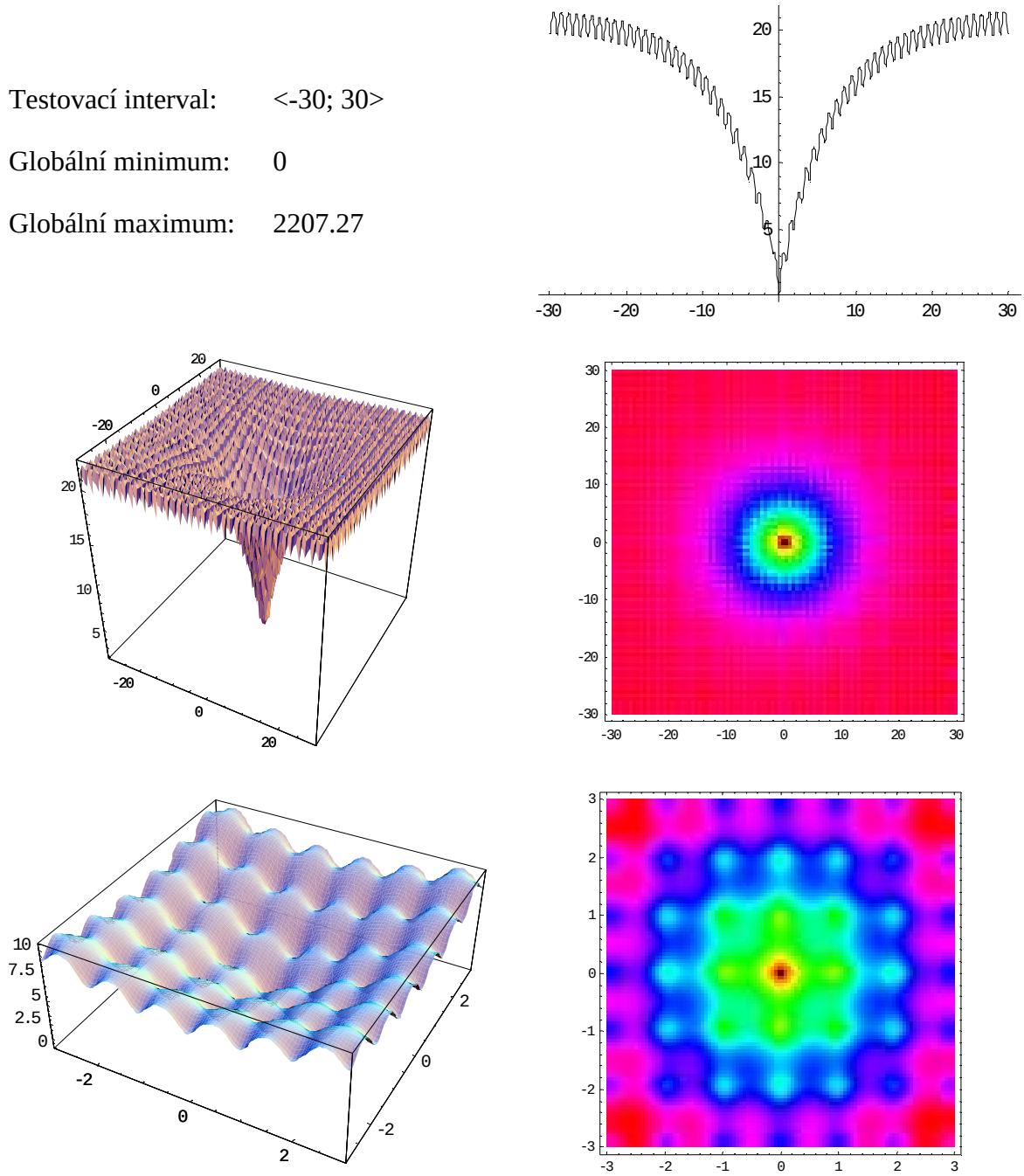




Obr. 20: Grafy Test function (Ackley)

6.10 Ackley's function

Testovací interval:	<-30; 30>
Globální minimum:	0
Globální maximum:	2207.27



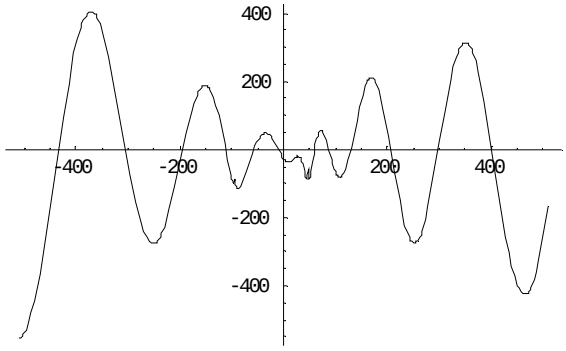
Obr. 21: Grafy Ackley's function

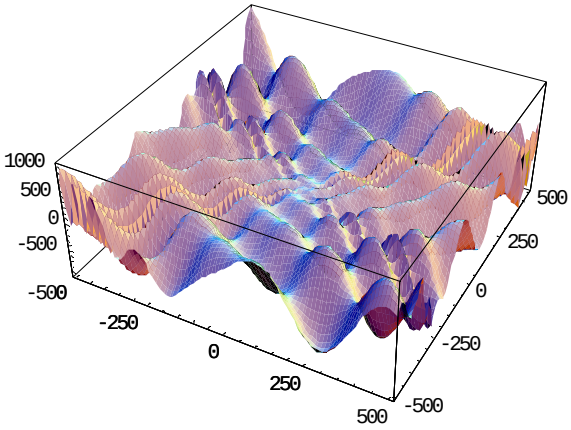
6.11 Egg Holder

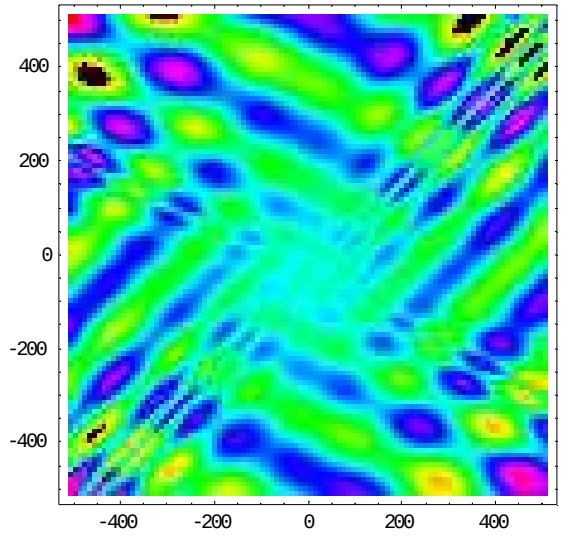
Testovací interval: <-512;512 >

Globální minimum: neznámé

Globální maximum: neznémé







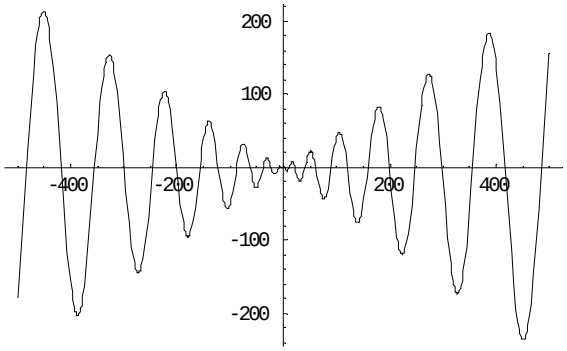
Obr. 22: Grafy Egg Holder function

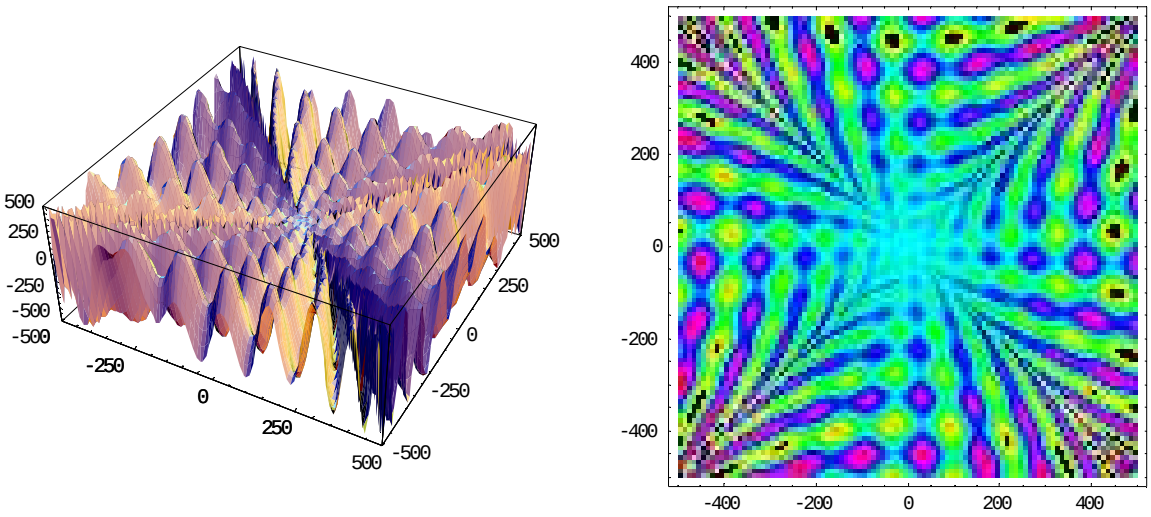
6.12 Rana's function

Testovací interval: <-500;500 >

Globální minimum: neznámé

Globální maximum: neznámé

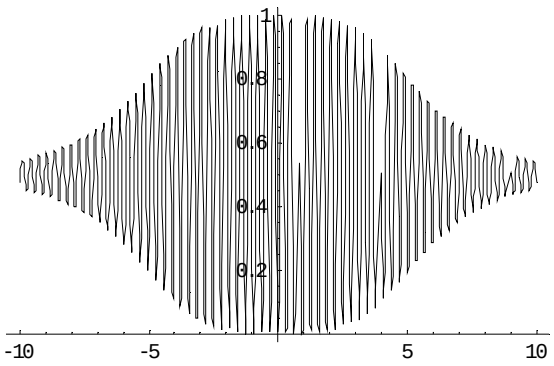


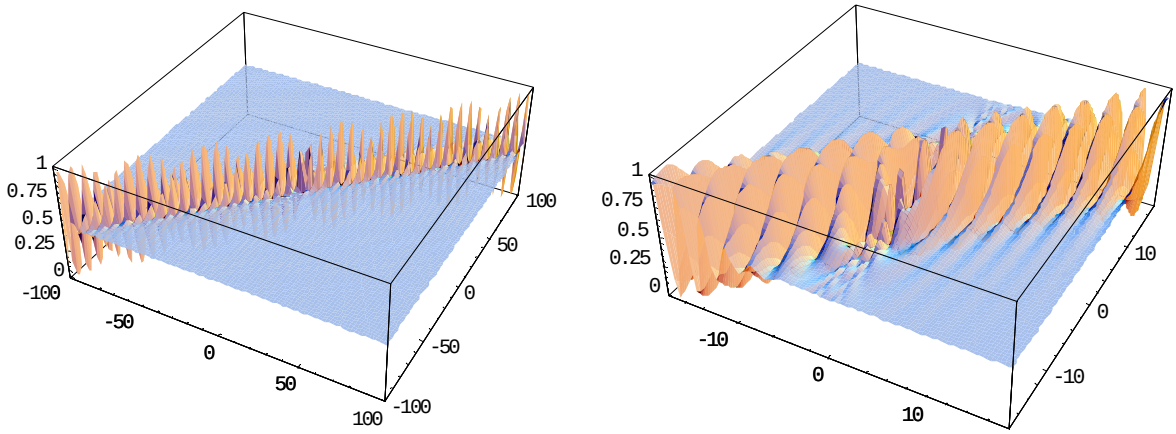


Obr. 23: Grafy Rana's function

6.13 Pathological function

Testovací interval:	<-100; 100>
Globální minimum:	neznámé
Globální maximum:	neznámé





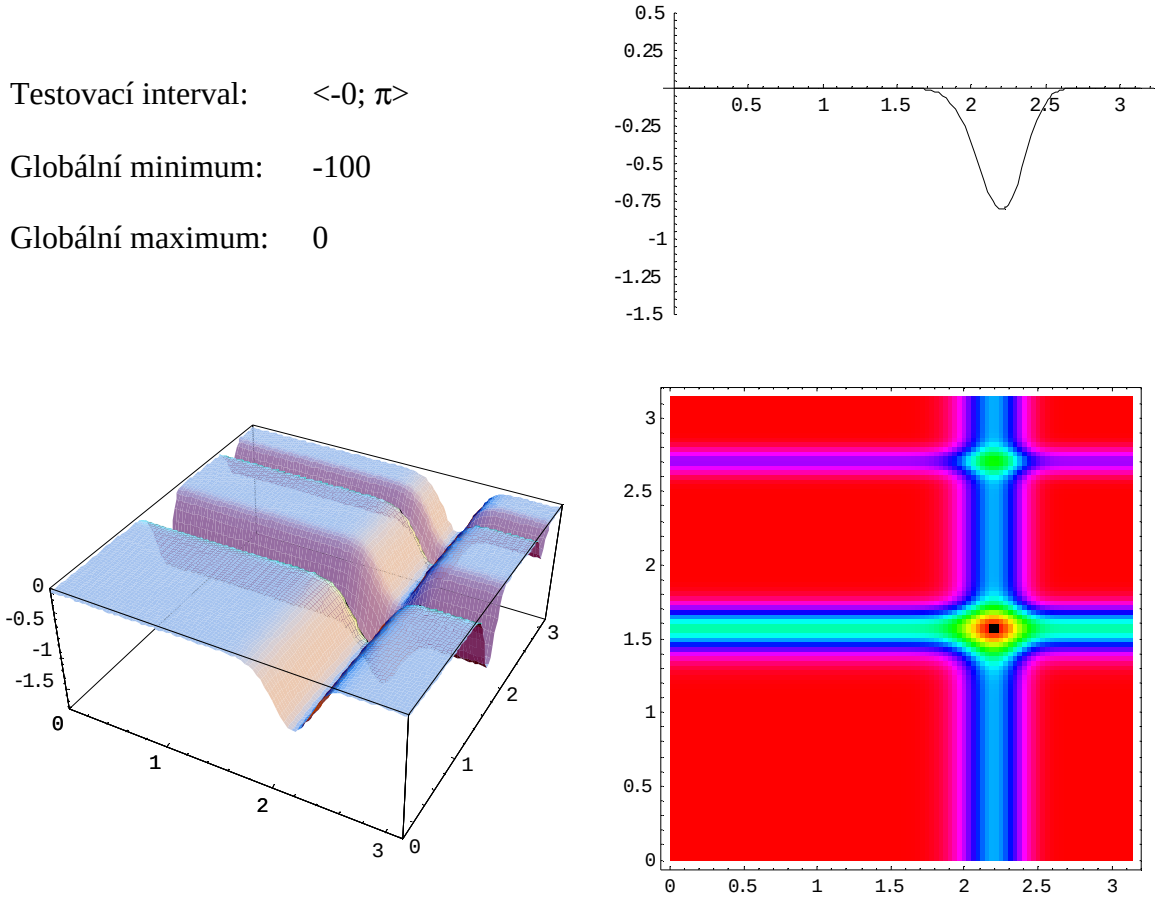
Obr. 24: Grafy Pathological function

6.14 Michalewicz's function

Testovací interval: <-0; π>

Globální minimum: -100

Globální maximum: 0



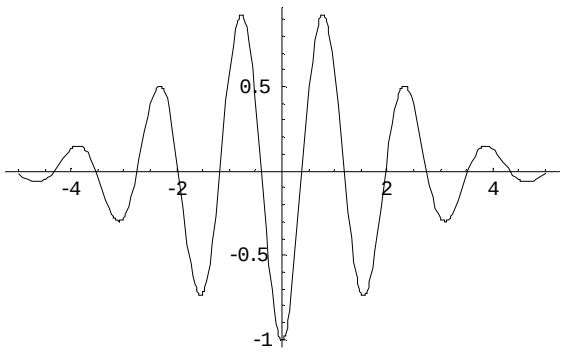
Obr. 25: Grafy Michalewicz's function

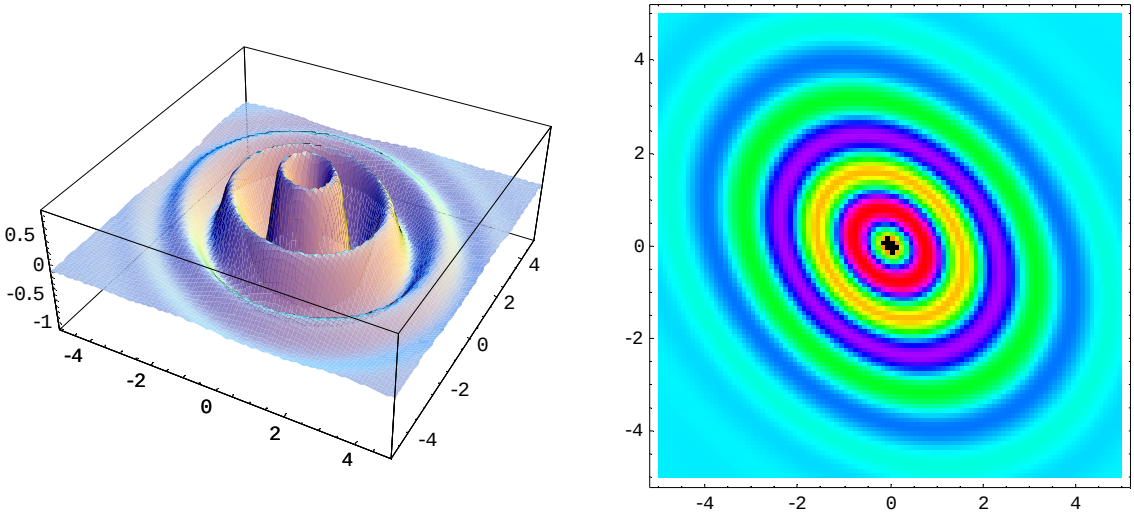
6.15 Cosine wave function (Masters)

Testovací interval: <-5; 5 >

Globální minimum: -99

Globální maximum: 91





Obr. 26: Grafy Cosine wave function (Masters)

7 VÝSLEDKY TESTŮ

Všechny funkce byly testovány ve 100 dimenzionálním prostoru. Každá funkce měla tedy 100 argumentů. Pro každou funkci je vytvořen soubor s nastavením parametrů a zobrazením grafů.

Počet simulací byl 100 pro všechny tři algoritmy (ACO, SOMA, DE).

V tabulce (*Tabulka 3*) uvedeny nejlepší a nejhorší hodnoty účelové funkce pro jednotlivé algoritmy a jsou uvedeny jako procentuální vzdálenosti jedinců od globálního minima. U některých funkcí jsou uvedeny absolutní hodnoty, protože pro tyto funkce není známa hodnota globálního extrému pro 100D. Výsledky zobrazují hodnoty nejlepších a nejhorších jedinců vždy z poslední migrace.

První dva sloupce obsahují hodnoty získané algoritmem ACO, druhé dva sloupce jsou výsledky algoritmu SOMA a poslední dva sloupce zobrazují výsledky DE.

V tabulkách s grafy jsou v prvním řádku grafy ACO, ve druhém SOMA a ve třetím DE. Nastavení parametrů je pro každou funkci zobrazeno u grafu.

V prvním sloupci jsou průběhy algoritmů, kde je zachycen vývoj nejlepších účelových hodnot během simulace. Všechny simulace jedné funkce jsou zobrazeny v jednom grafu.

Druhý sloupec grafů zobrazuje procentuální vzdálenost nejlepšího výsledku od skutečného optima.

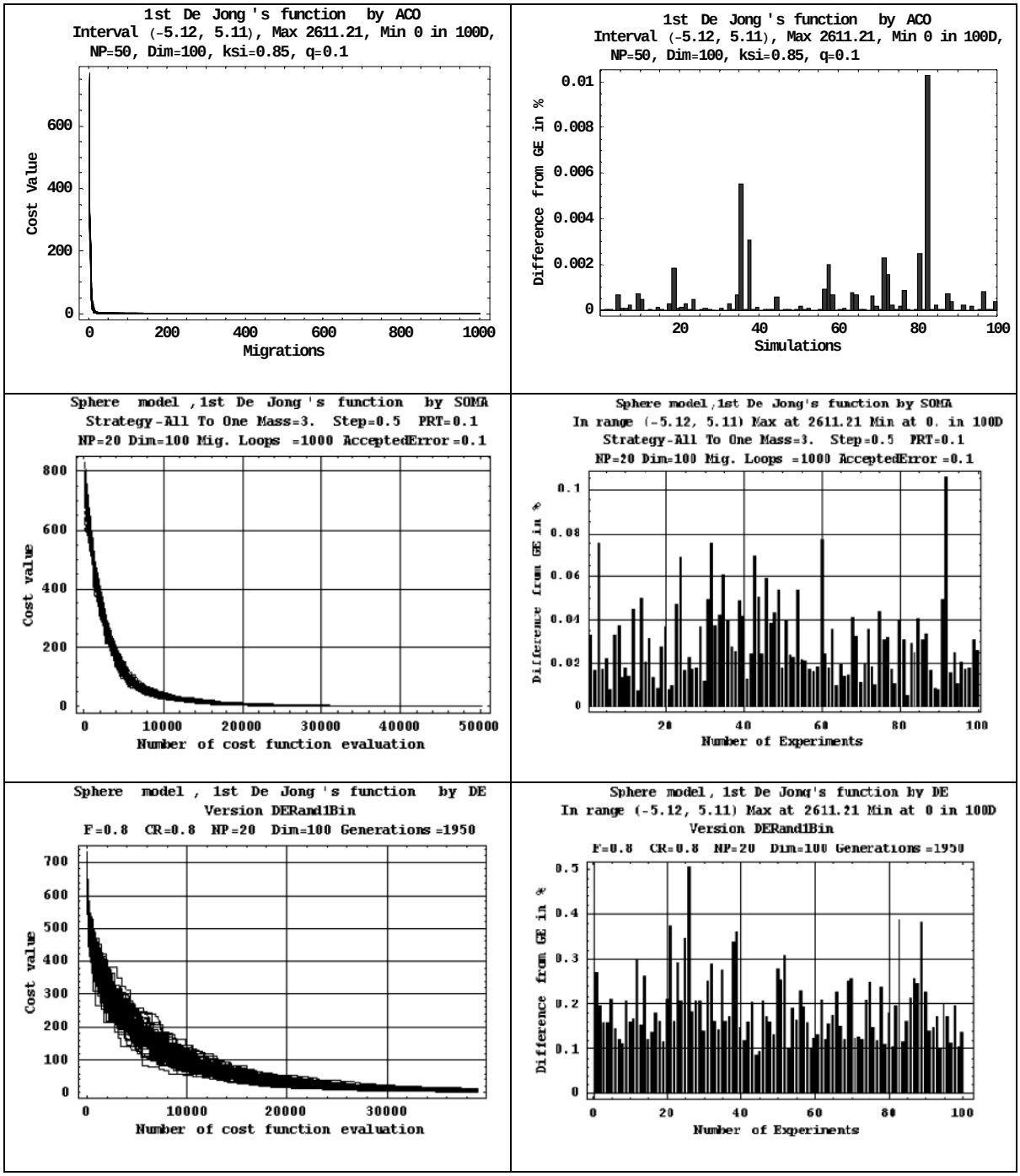
7.1 Zobrazení výsledků v tabulce

Tabulka 3: Nejlepší a nejhorší výsledky jednotlivých algoritmů ve funkcích

	ACO		**SOMA**		**DE**	
	Nejlepší výsledky %	Nejhorší výsledky %	Nejlepší výsledky %	Nejhorší výsledky %	Nejlepší výsledky %	Nejhorší výsledky %
Sphere model, 1st De Jong's function	$4.9x10^{-9}$	0.01	0.01	0.11	0.09	0.5
Rosenbrock's saddle	0.57	1.86	0.027	0.055	0.027	0.04
3rd De Jong's function	$3.6x10^{-7}$	0.01	0.27	1.46	0.3	2.4
4th De Jong's function	$6.5x10^{-13}$	0.00003	0.0007	0.0072	0.005	0.09
Rastrigin's function	2.26	6.86	0.71	1.6	0.8	5.4
Schwefel's function	10.4	17.4	0.02	1.27	0.025	0.12
Griewangk's function	3.9	8.5	0.42	0.525	0.00001	0.04
Stretched V sine wave function (Ackley)	24.9	33.4	0.55	2.47	$4.2x10^{-16}$	$8x10^{-16}$
Test function (Ackley)	26	39.8	0.0	1.0	1.5	1.75
Ackley's function	47.1	62.8	0.25	3.7	$6x10^{-6}$	$8x10^{-6}$
Egg Holder	*-40759*	*-11374*	*-52000*	*-37000*	*-29000*	*-25000*
Rana's function	*-24342*	*-6548*	*-25000*	*-22000*	*-18500*	*-16000*
Pathological function	*45*	*47*	*25*	*32*	*35*	*37.5*
Michalewicz's function	30.2	41.8	1.8	5.2	0.12	1.6
Cosine wave function (Masters)	30	45	6.4	14	34	33

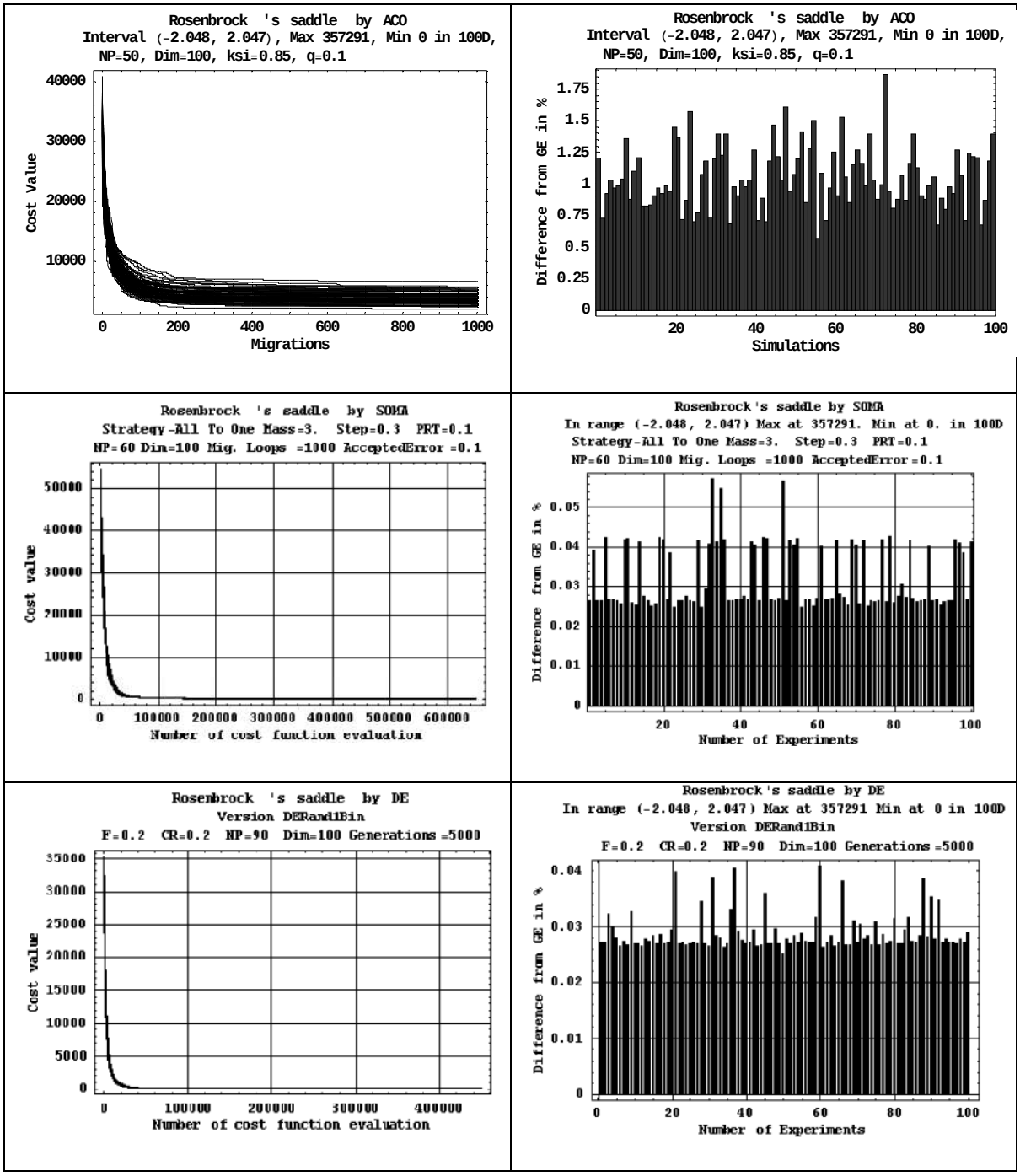
7.2 Grafické zobrazení výsledků

7.2.1 1st De Jong's function



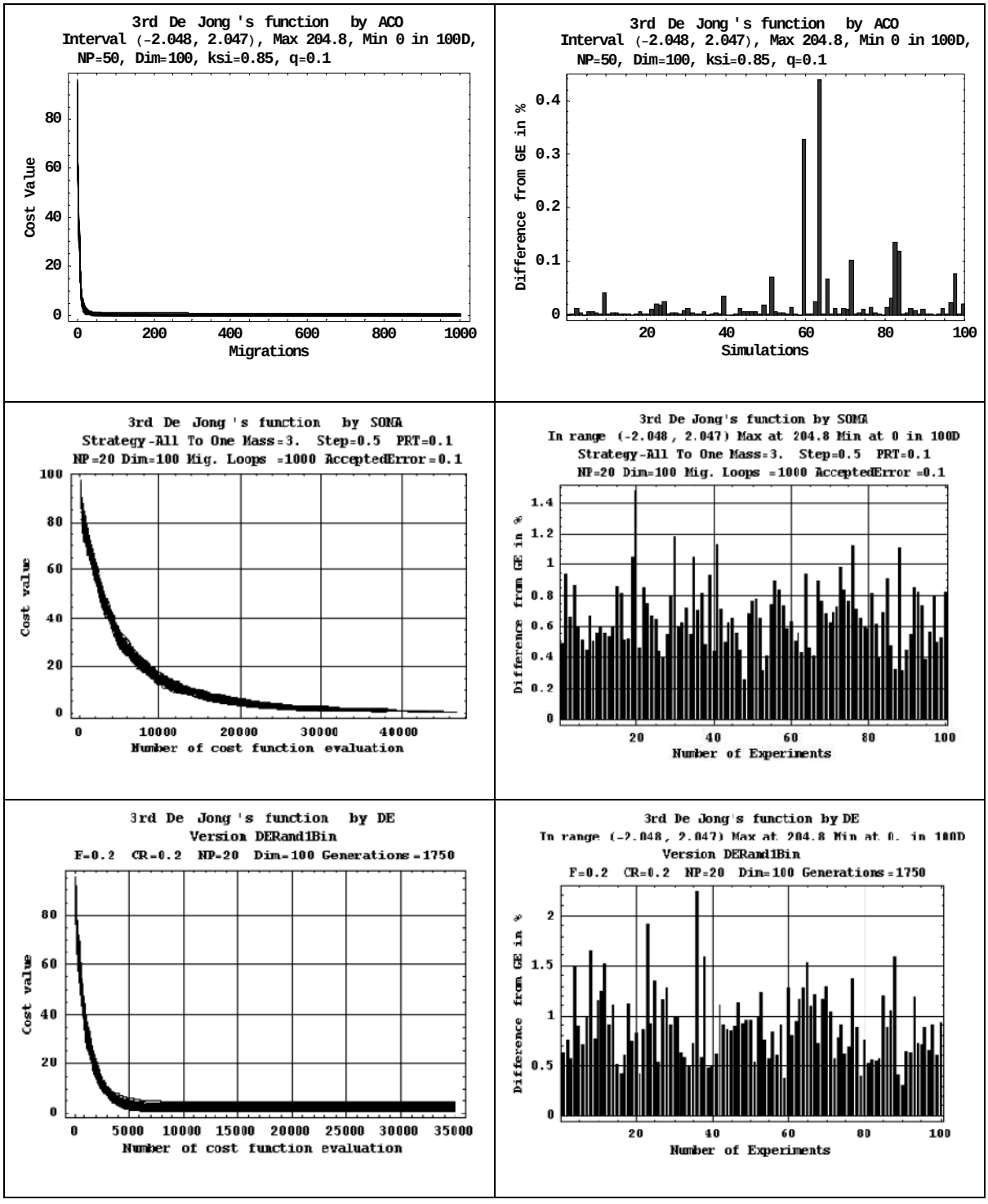
Obr. 27: Výsledky -1st De Jong's function

7.2.2 Rosenbrock's saddle



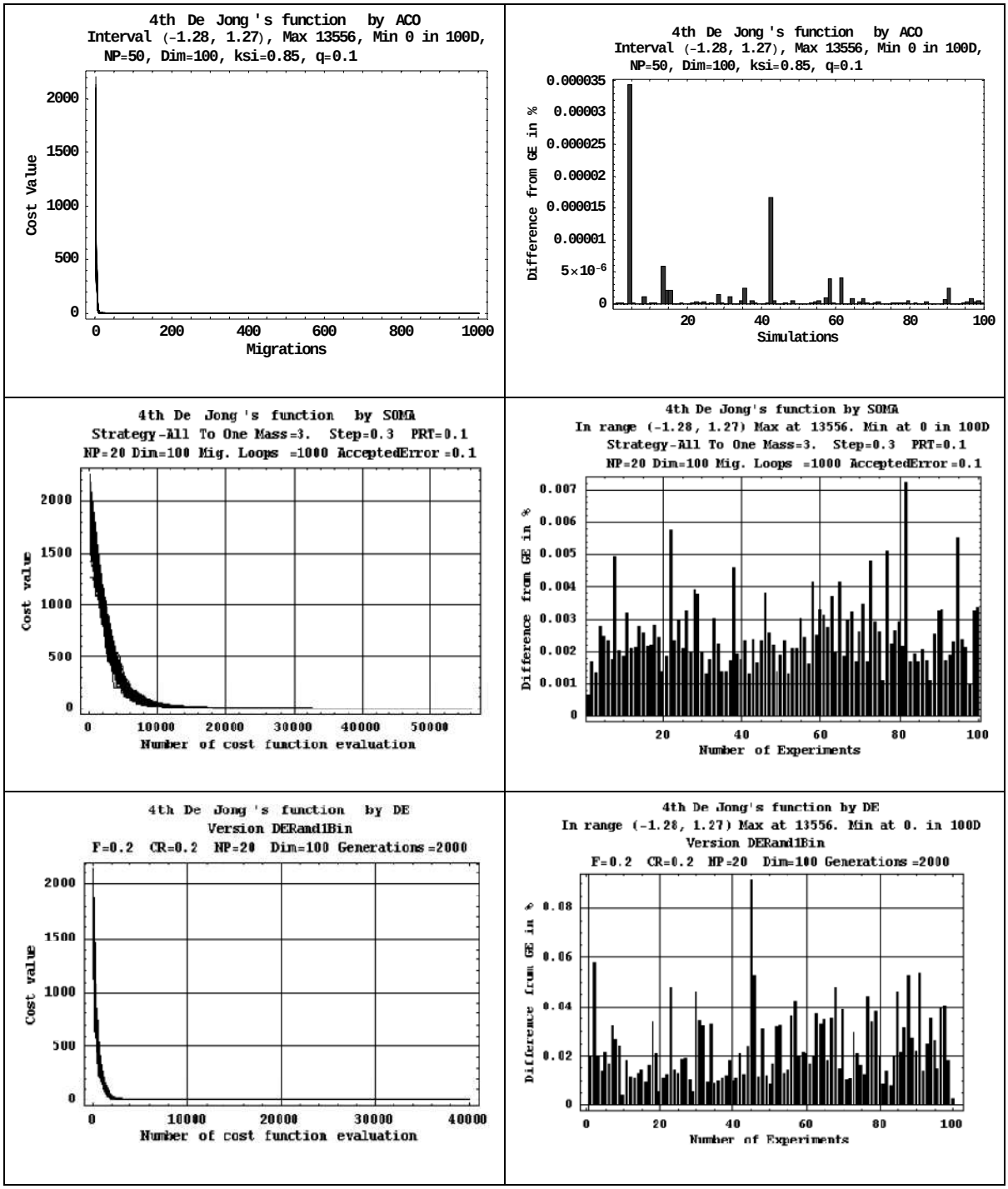
Obr. 28: Výsledky - Rosenbrock's saddle

7.2.3 3rd De Jong's function



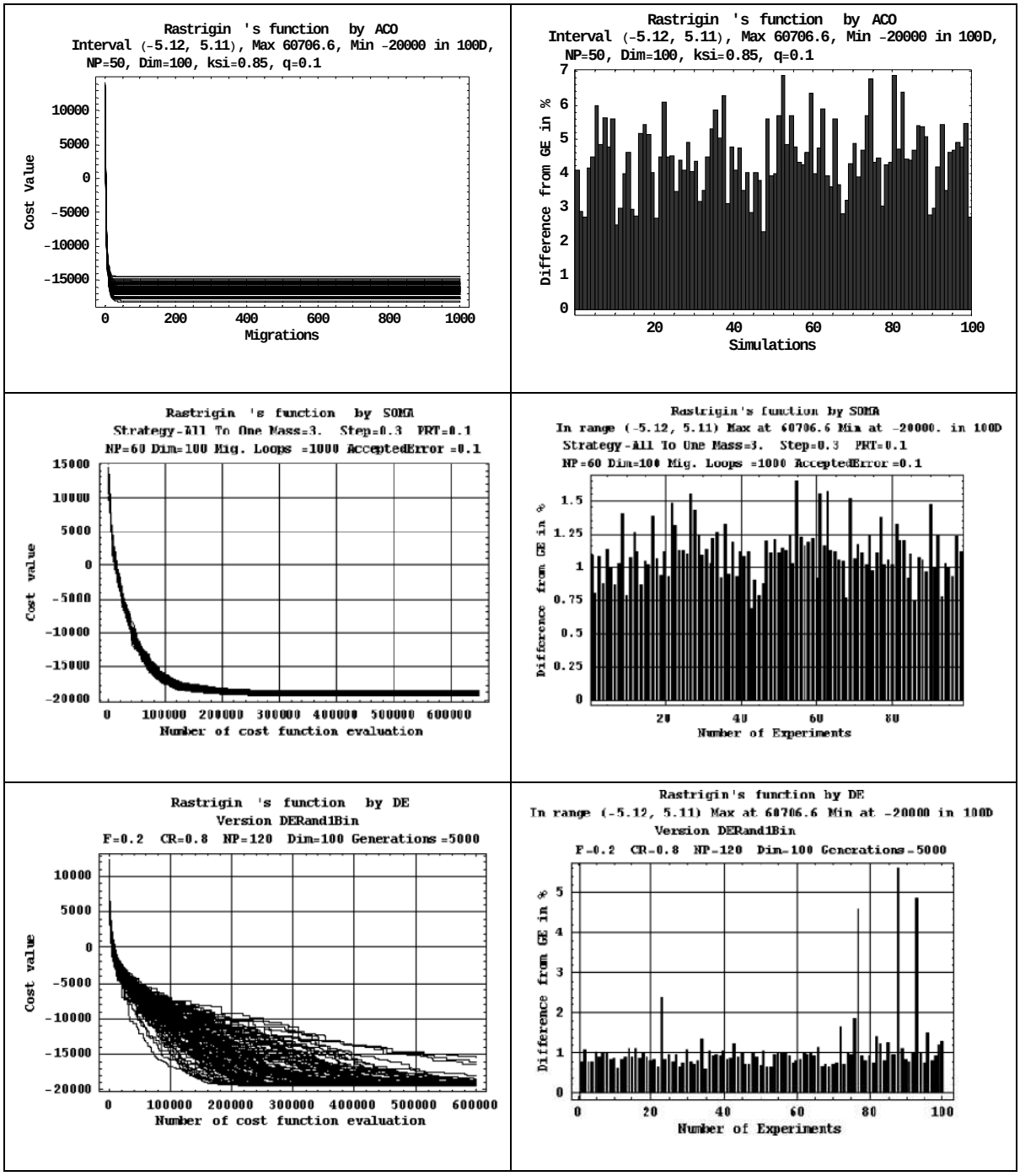
Obr. 29: Výsledky -3rd De Jong's function

7.2.4 4th De Jong's function



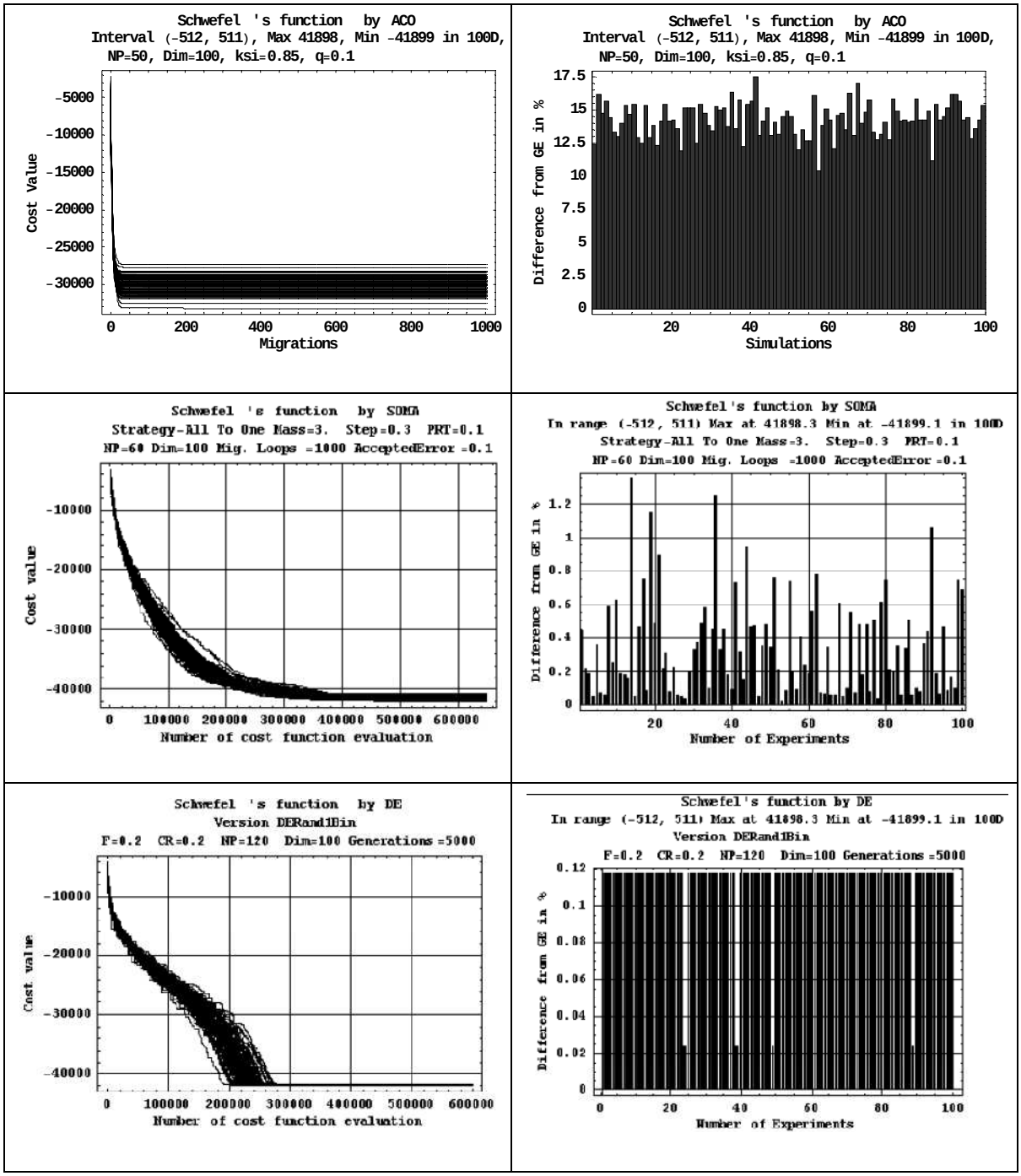
Obr. 30: Výsledky - 4th De Jong's function

7.2.5 Rastrigin's function



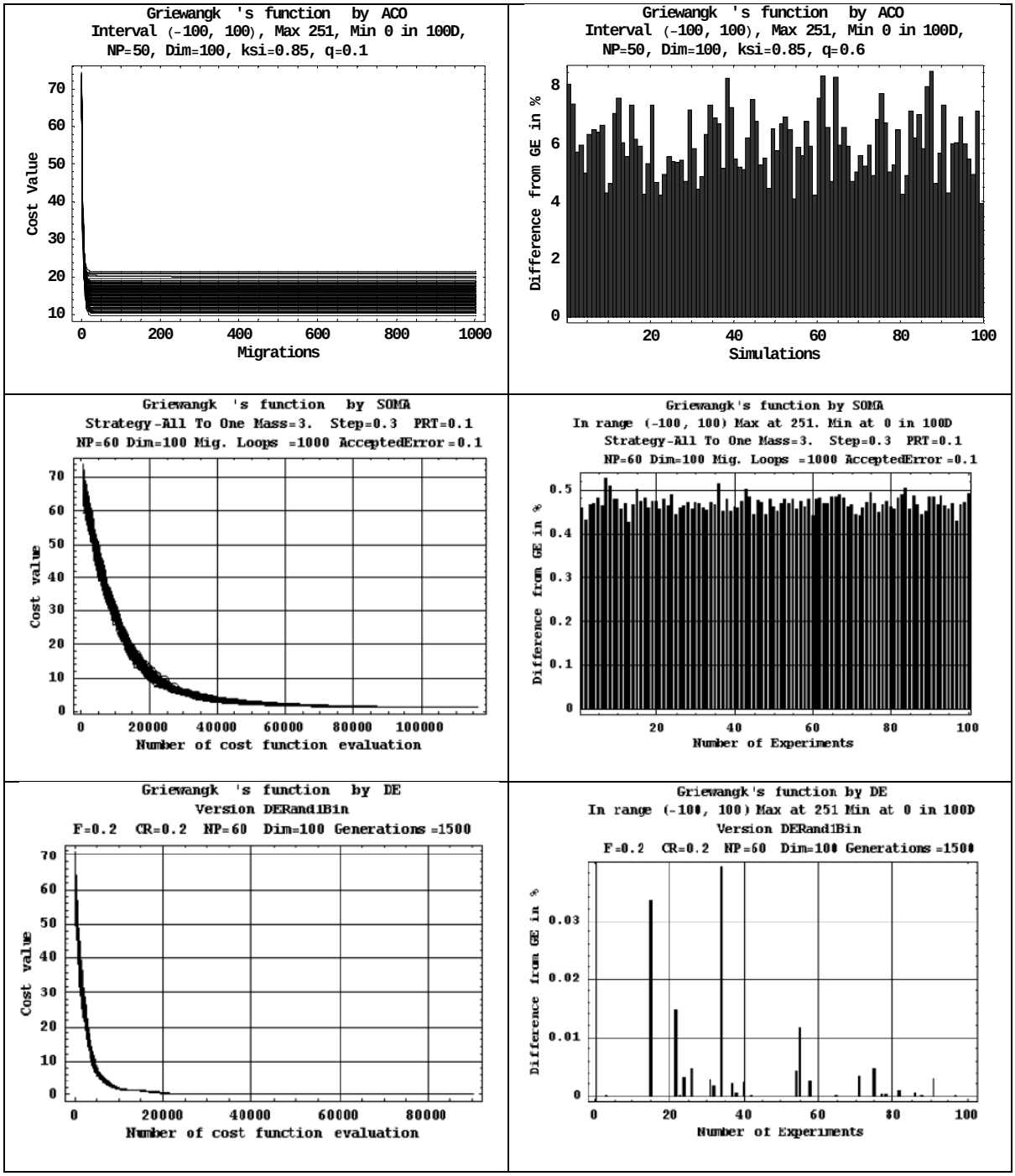
Obr. 31: Výsledky - Rastrigin's function

7.2.6 Schwefel's function



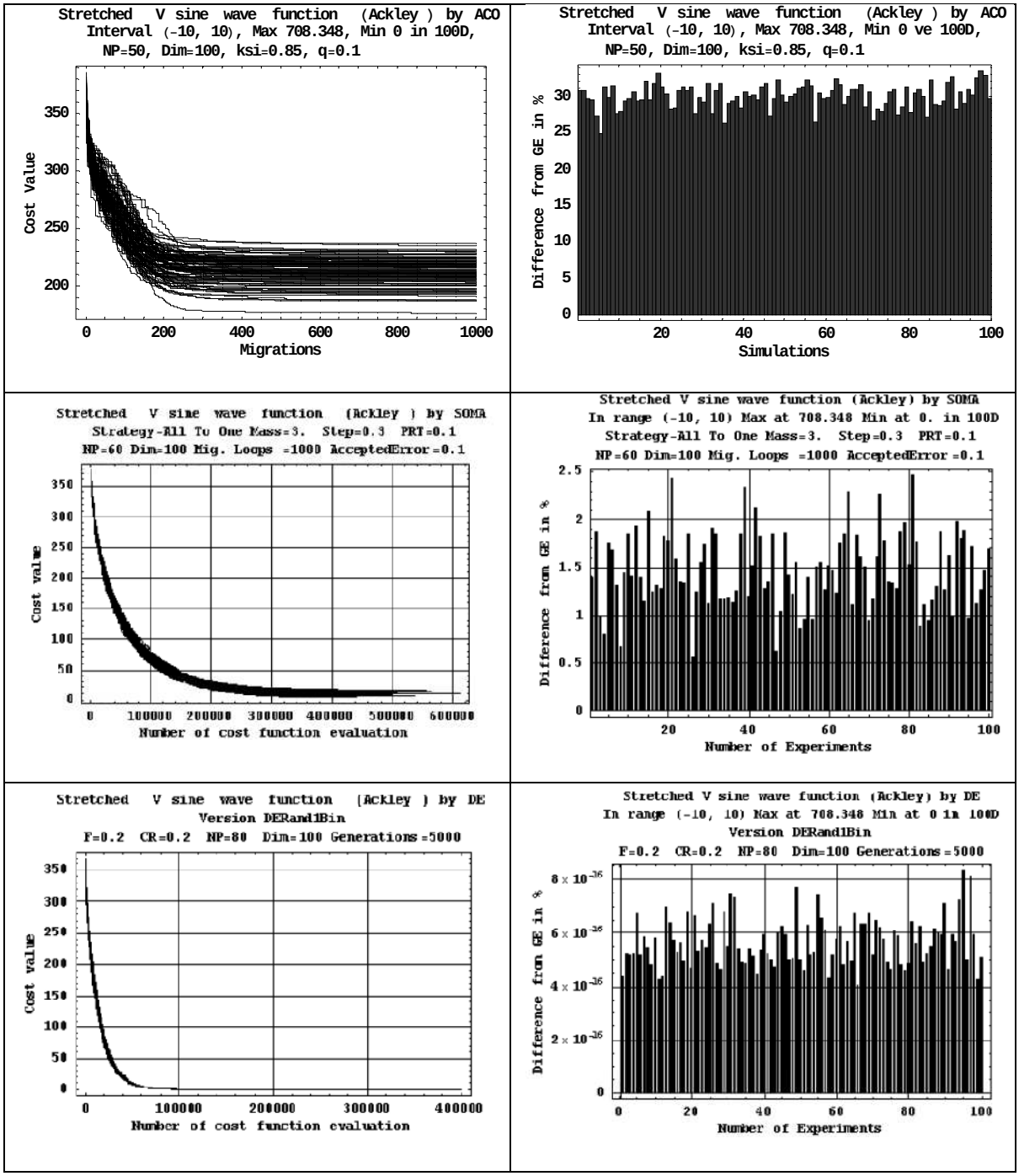
Obr. 32: Výsledky - Schwefel's function

7.2.7 Griewangk's function



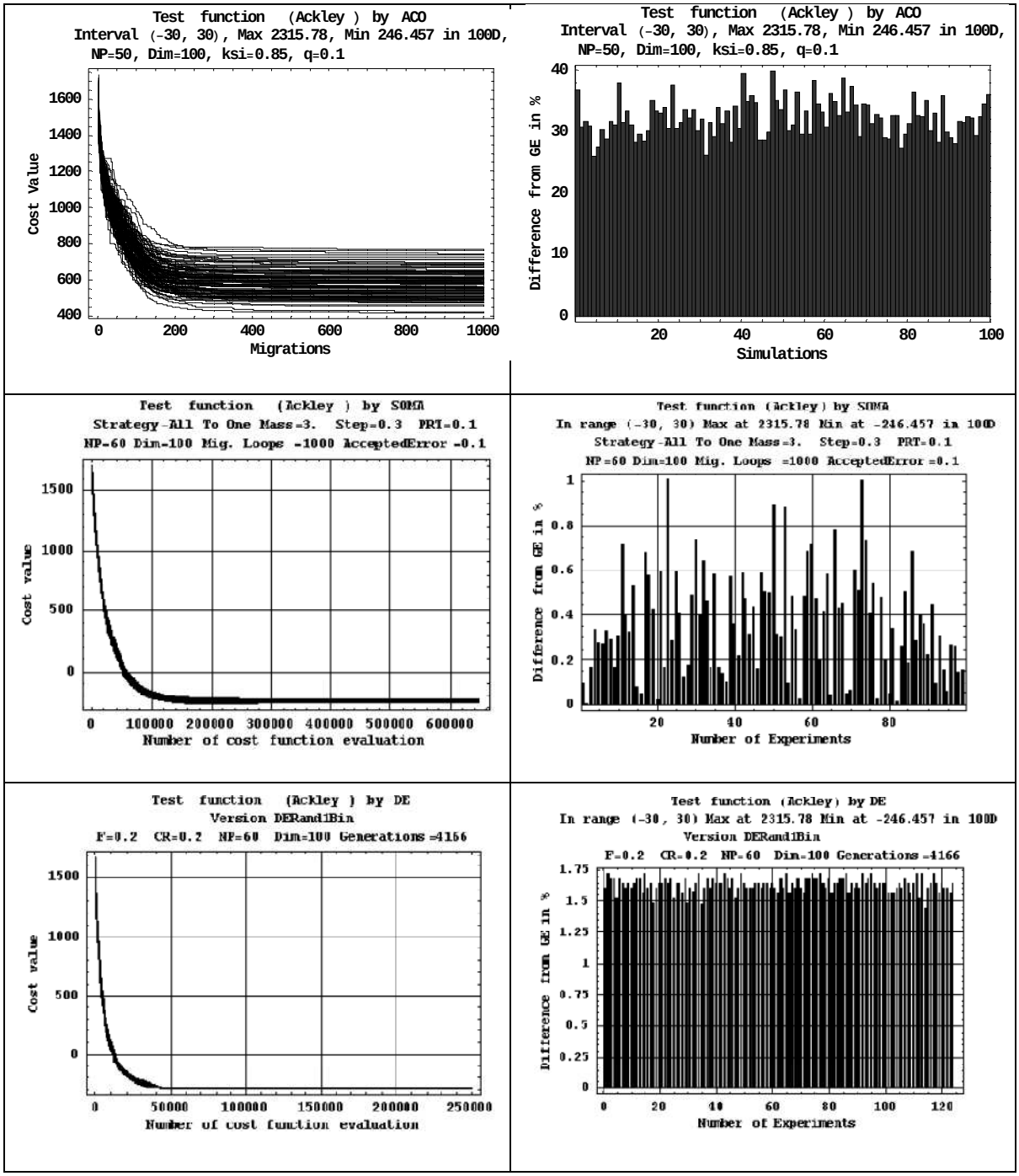
Obr. 33: Výsledky - Griewangk's function

7.2.8 Stretched V sine wave function (Ackley)



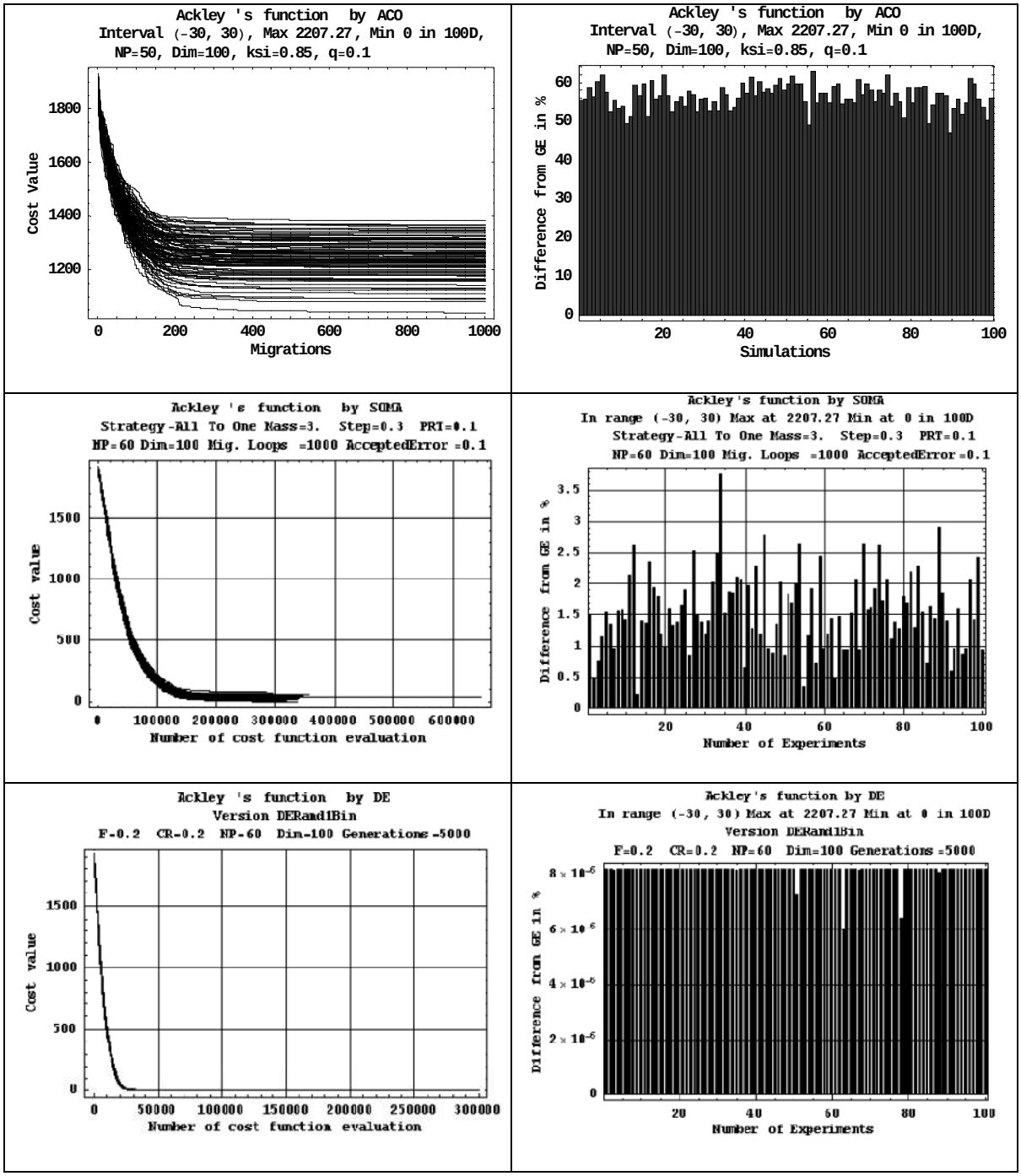
Obr. 34: Výsledky - Stretched V sine wave function (Ackley)

7.2.9 Test function (Ackley)



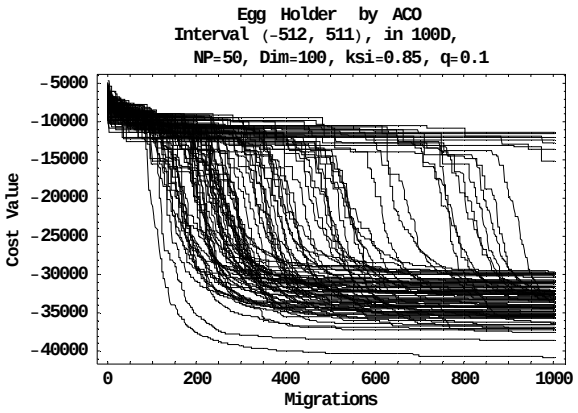
Obr. 35: Výsledky - Test function (Ackley)

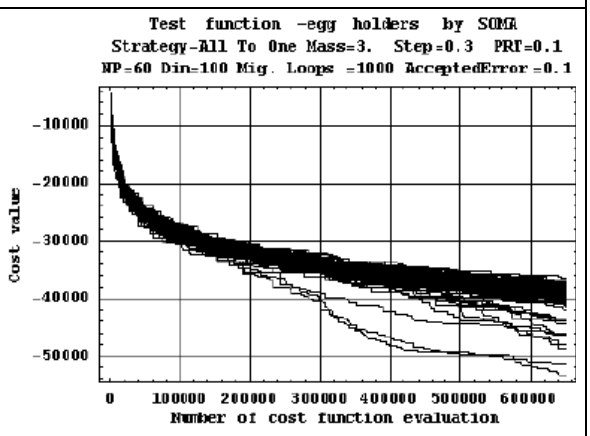
7.2.10 Ackley's function

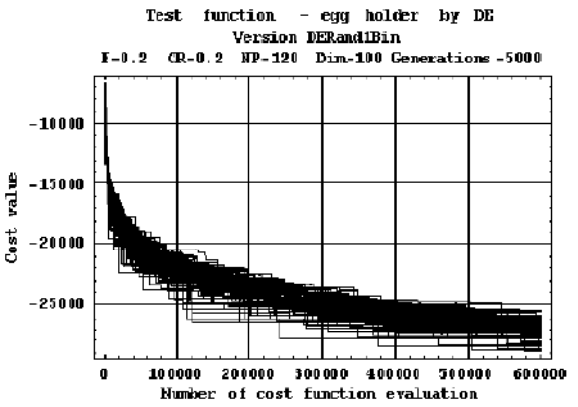


Obr. 36: Výsledky - Ackley's function

7.2.11 Egg Holder



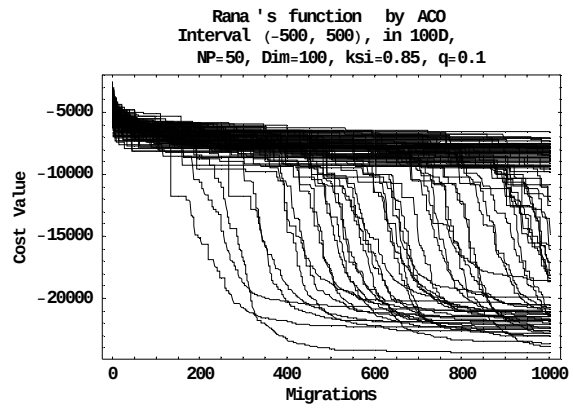


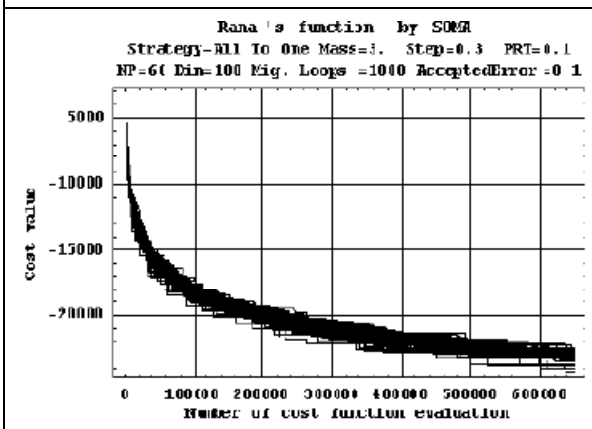


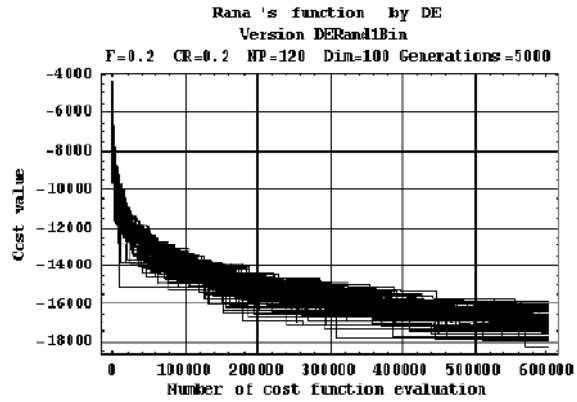
Procentuální vzdálenost od globálního extrému nešlo určit z důvodů vysoké a nepravidelné multimodality funkce

Obr. 37: Výsledky - Egg Holder function

7.2.12 Rana's function



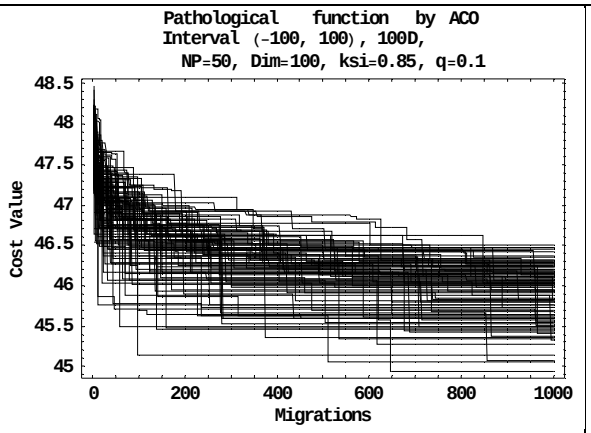


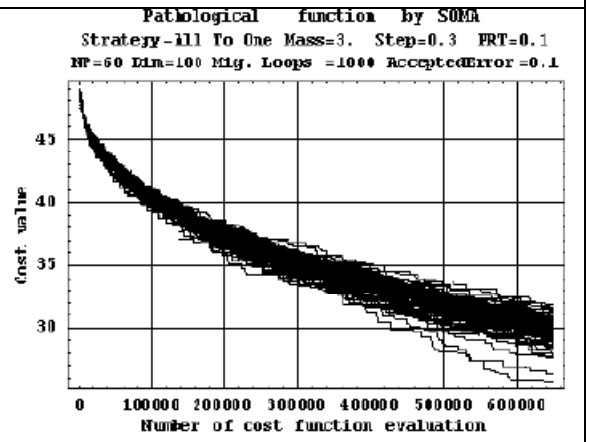


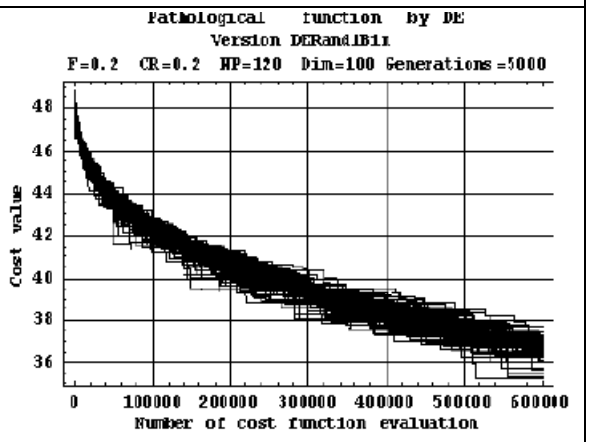
Procentuální vzdálenost od globálního extrému nešlo určit z důvodů vysoké a nepravidelné multimodality funkce

Obr. 38: Výsledky - Rana's function

7.2.13 Pathological function



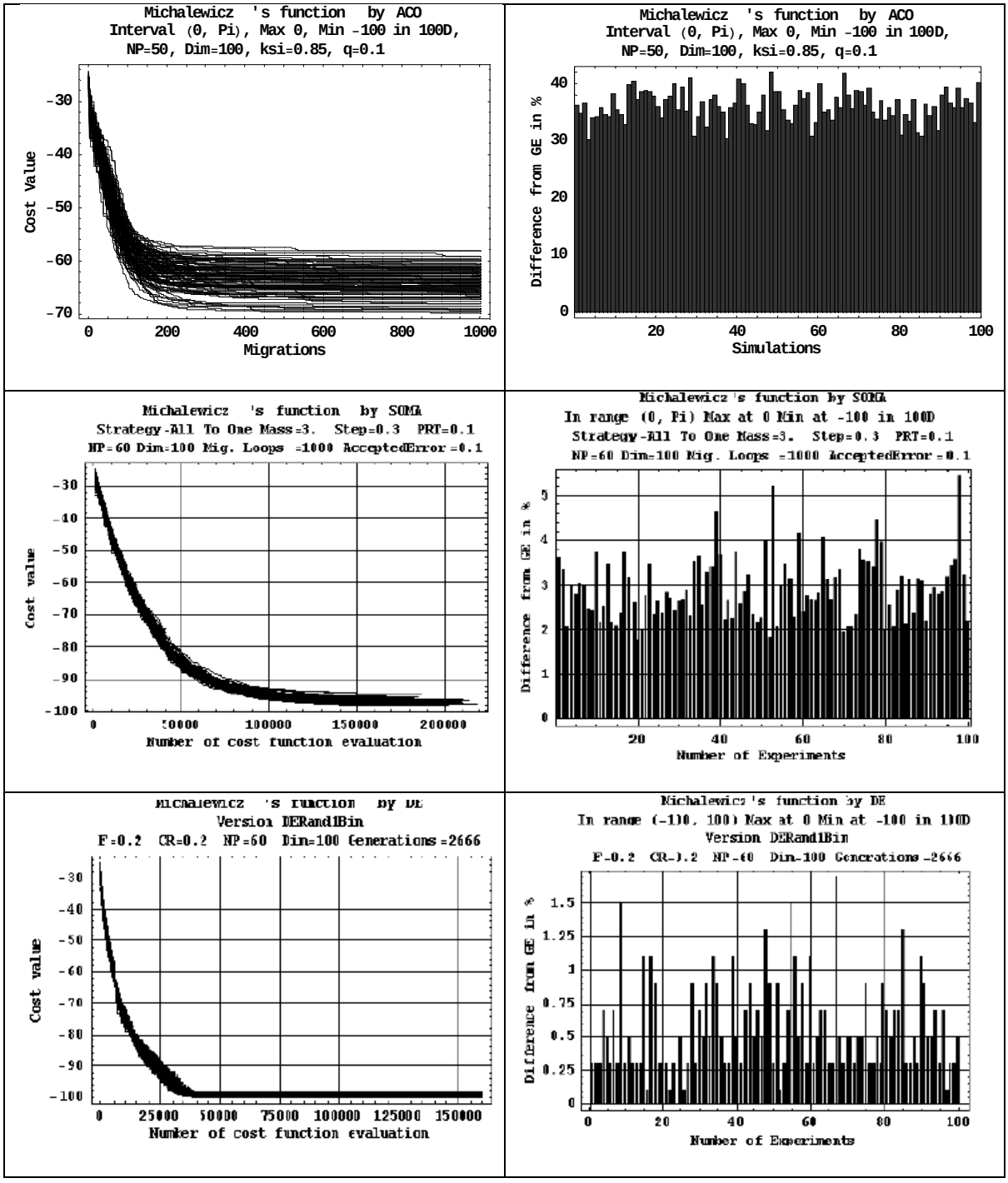




Procentuální vzdálenost od globálního extrému nešlo určit z důvodů vysoké a nepravidelné multimodality funkce

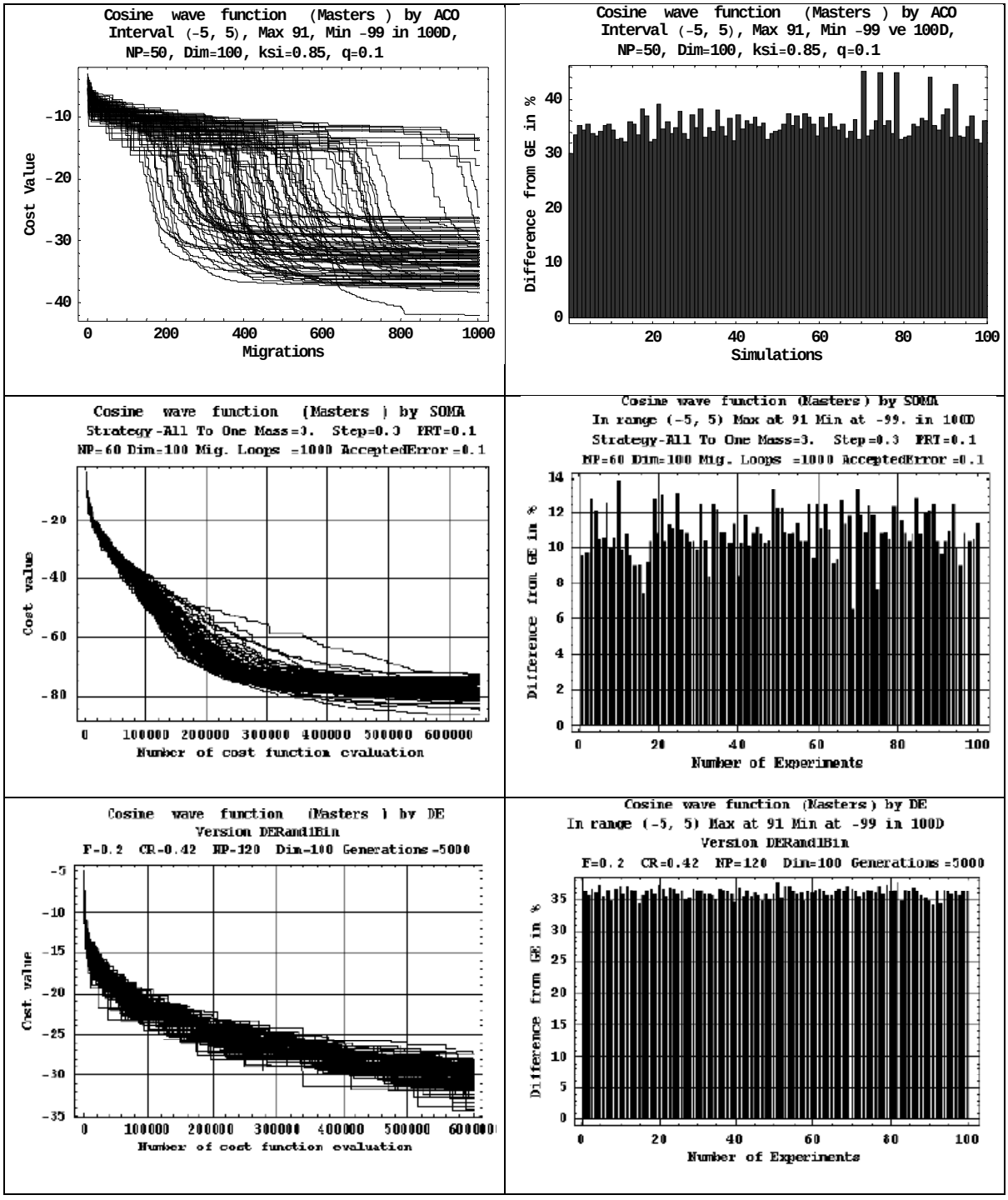
Obr. 39: Výsledky - Pathological function

7.2.14 Michalewicz's function



Obr. 40: Výsledky -Michalewicz's function

7.2.15 Cosine wave function (Masters)



Obr. 41: Výsledky - Cosine wave function (Masters)

7.3 Zhodnocení výsledků

Algoritmus $ACO_{\mathbb{R}}$ prokázal velmi dobré výsledky pro jednoduché unimodální funkce (De Jong 1st, De Jong 3rd, De Jong 4th), kde dokázal najít minimum již po několika málo migracích.

U složitějších funkcí již byly výsledky horší. Nejhůře dopadly všechny Ackleyho funkce, kde se $ACO_{\mathbb{R}}$ vůbec nedokázal ani přiblížit k minimu, ale velmi brzy zkonvergoval do některého lokálního minima.

Pro funkce Rastrigin, Schwefel a Griewangk algoritmus příliš rychle uvízl v některém lokálním minimu. Jelikož byl pro všechny funkce parametr *q* nastaven na hodnotu *0.1*, kdy jsou preferováni pouze nejlepší řešení, algoritmus ztratil svou robustnost při prohledávání prostoru. Zvýšením tohoto parametru alespoň na hodnotu *0.6* nebo více by algoritmus prohledával i méně nadějná řešení a tím by tak rychle neuvízl v lokálním minimu. Z hlediska časové a hlavně výpočetní náročnosti již nebyly tyto testy provedeny.

Překvapivé byly výsledky pro dvě velmi složité funkce – Egg Holder a Rana. V obou případech $ACO_{\mathbb{R}}$ nalezl lepší řešení než DE, a srovnatelné řešení jako SOMA. Jak bylo řečeno výše, nastavením vyšší hodnoty parametru q by mohly být výsledky ještě lepší.

U některých funkcí je z grafů patrné, že počet migračních kol byl naprosto nedostačující a algoritmus nestihl nalézt minimum (například u funkce Pathological).

ZÁVĚR

Cílem práce bylo naprogramovat algoritmus Ant Colony Optimization (ACO) a provést testování a srovnání s dalšími dvěma algoritmy - algoritmy SOMA (SamoOrganizující se Migrační Algoritmus) a DE (Diferenciální Evoluce). Oba tyto algoritmy byly naprogramovány již dříve v prostředí Mathematica, proto byl zvolen tento program i pro mravenčí algoritmus.

Algoritmus ACO byl původně vymyšlen pro diskrétní optimalizační problémy, ale s postupem času se vyvinulo mnoho různých variant i pro jiné úlohy. V této práci je podrobně popsáno několik druhů Ant Colony algoritmů pro kombinatorické problémy i pro spojité systémy. Hlavím úkolem práce bylo provést testování algoritmu na spojitých testovacích funkcích, proto byla zvolena varianta ACO algoritmu pro spojité systémy.

Těchto algoritmů již existuje celá řada, ale pro účely práce byla vybrána jedna z nejnovějších variant – algoritmus $ACO_{\mathbb{R}}$. Byl vytvořen v r. 2004 K. Sochou a oproti jiným variantám pro spojité problémy se velmi přesně drží původní myšlenky mravenčího chování. Popis algoritmu zahrnuje princip, na jakém ACO pracuje a jaké postupy využívá a s jakými parametry pracuje.

Mravenčí algoritmy jsou založeny na chování kolonie mravenců a jejich schopnosti nalézt nejkratší cestu mezi zdrojem potravy a mraveništěm. Mravenci využívají principy samoorganizace. Tato problematika je velmi podrobně popsána v teoretické části práce.

Jedna kapitola je věnována algoritmům SOMA a DE, kde je krátce vysvětlen jejich princip fungování.

V praktické části je stručný popis matematického prostředí Mathematica, který byl použit pro naprogramování algoritmu. Algoritmus je popsán velmi podrobně i s ukázkami zdrojového kódu.

V poslední částí práce je představeno 15 testovacích funkcí. $ACO_{\mathbb{R}}$ byl testován na všech funkcích a u některých prokázal velmi dobré výsledky. U složitějších multimodálních funkcí již byly výsledky horší, kdy algoritmus velmi rychle zkonvergoval do některého lokálního minima. Rychlost konvergence a robustnost algoritmu při prohledávání se dá ovlivnit nastavením jeho parametrů, přičemž při testování bylo použito nastavení pro velmi rychlou konvergenci, což se ukázalo jako špatná volba.

CONCLUSION

The aim of this work is programming Ant Colony Optimization (ACO) algorithm, realize testing and compare it with another two algorithms – SOMA (Self-Organizing Migrating Algorithm) and DE (Differential Evolution). Both algorithms were creating previously in Mathematica environment, so this environment was chosen for ant algorithm.

Algorithm ACO was originated for solving discrete optimization problems, but afterwards was coming up with many different variant of ACO for solving another tasks. Several types of ACO for combinatorial and continuous problems are closely described in this work. Main assignment of work was realized a testing on continuous test functions, thus ACO for continuous domain was chosen.

There are many different types of ant algorithms, but for purposes of this work was selected one recent variant - algorithm ACOR. It was built in 2004 by K. Socha and over against other variants for continuous domain is very strongly keep original intention of ant's behavior. Description of algorithm contains principles, which ACO work with and which techniques it uses.

Ant algorithms are based on behaviors of ant colony and their ability to find shortest path from nest to food source. Ants use rules of self-organization. This field is very closely described in theoretical part.

One chapter is appropriated for SOMA and DE algorithms, where is their principles explained.

In practical part is a brief characterization of mathematical environment Mathematica, which was used for programming algorithm. Description of algorithm contains examples of source code.

In the last part are introduced 15 test functions. ACO was tested on all functions and proved very good results for some of them. For complex multimode functions were worse results. Algorithm converged very fast to some local minima. Speed of convergence and robustness of algorithm can be affected with better settings of parameters. For testing parameters was set for fast speed of convergence, which was bad idea.

SEZNAM POUŽITÉ LITERATURY

[1] Dorigo M., Stuzle T., Ant Colony Optimization (Bradford Books), The MIT Press, 2004, ISBN: 978-0262042192

[2 Dorigo M., Ant Colony Optimization and Swarm Intelligence, Springer, 2006, ISBN 3540226729

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

[4] Wolfram S., The Mathematica Book 5, Wolfram Media, 2003, ISBN 1-57955-022-3

[5] Socha K, Dorigo M, Ant Colony Optimization for Continuous Domains, IRIDIA - Technical Report Series, Technical Report No. TR/IRIDIA/2005-037, 2005

[6] Socha, K, ACO for Continuous and Mixed-Variable Optimization, Proceedings of the Fourth International workshop on Ant Colony Optimization and Swarm Intelligence, Brussels, Belgium, 2004.

[7] Socha K, Blum Ch, Ant Colony Optimization, Technical Report Series, Technical Report No. TR/IRIDIA/2006-009, 2006

[7] Dorigo M, Socha K., An Introduction to Ant Colony Optimization: To appear in T. F. Gonzalez, Approximation Algorithms and Metaheuristics, CRC Press, 2007.

[8] Socha K, Dorigo M.,Ant Colony Optimization for Mixed-Variable Optimization Problems, Technical Report Series, Technical Report No. TR/IRIDIA/2007-019, 2007

[9] Blum C, Dorigo M., Ant colony optimization: Introduction and recent trends. Physics of Life Reviews, 2(4):353-373, 2005

[10] Cordon O., Herrera F., Stutzle TA Review on the Ant Colony Optimization Metaheuristic: Basis, Models and New Trends. Mathware and Soft Computing, 9(2-3), pp. 141--175, 2002.

[11] Dorigo M, Birattari M, Stuzle T, Ant Colony Optimization - Artificial Ants as a Computational Intelligence Technique, IEEE Computational Inteligence Magazine,2006

[12] Bonabeau E, Dorigo M, Theraulez G, Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, NY, 1999.

[13] Dorigo M, Stuzle T, The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances, Handbook of Metaheuristics, 2002

[14] Dorigo M, Di Carlo G., Gambardella L. M., Ant Algorithms for Discrete Optimization. Artificial Life, 5(2):137-172, 1999

[15] Stutzle T, Hoos H. H., MAX-MIN Ant System. Future Generation Computer Systems. 16(8):889--914,2000

[16] Dorigo M., Maniezzo V, Colorni A, Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26(1):29-41,1996

[17] Molga M. Test functions for optimization needs [online] Dostupný z www: < www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf >

[18] Dorigo M, Ant Colony Optimization Metaheuristic , Dostupný z www: < http://www.aco-metaheuristic.org>

[19] Dorigo M, Ant Colony Optimization ,Bibliography, Dostupný z www: < http:// http://www.aco-metaheuristic.org/publications.html>

[20] G. Bilchev and I. C. Parmee. The Ant Colony Metaphor for Searching Continuous Design Spaces., Proceedings of the AISB Workshop on Evolutionary Computation, volume 993 of LNCS, Springer-Verlag, Berlin, Germany, 1995.

[21] Lachlan Kuhn, "Ant Colony Optimization for Continuous Spaces" Bachelor's Thesis, The University of Queensland, 2002.

[22] Deneuboug, J.-L., S. Aron, S. Goss and J.-M. Pasteels. "The Self-Organizing Exploratory Pattern of the Argentine Ant. Insect Behavior 3, 1990.

[23] Zelinka I, SOMA homepage, Dostupný z www: < http://www.ft.utb.cz/people/zelinka/soma/>

[24] Wikipedia, Dostupný z www: < http://en.wikipedia.org/wiki/Mathematica>

[25] Chramcov B, Základy práce v prostředí Mathematica, Skriptum 2005, Univerzita Tomáše Bati ve Zlíně, Fakulta technologická

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ACO	Ant Colony Optimization.
SOMA	SamoOrganizující se Migrační Algoritmus
DE	Diferenciální Evoluce
TSP	Travelling Salesman Problem
AS	Ant System
MMAS	Max-Min Ant System
ACS	Ant Colony System
CACO	Continuous ACO
CIAC	Continuous Interacting Ant Colony
ksi	Vypařování feromonu
q	Rychlost konvergence algoritmu
$ACO_{\mathbf{R}}$	ACO for continuous domain

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD ROM

Na CD je uložena diplomová práce ve formátu .doc a .pdf.

Dále je zde program ve formátu .nb. Pro každou testovanou funkci byl vytvořen samostatný soubor.