UNIVERZITA PARDUBICE

FAKULTA EKONOMICKO SPRÁVNÍ

ÚSTAV SYSTÉMOVÉHO INŽENÝRSTVÍ A INFORMATIKY

EVOLUČNÍ ALGORITMY V OPTIMALIZAČNÍCH PROBLÉMECH VEŘEJNÉ SPRÁVY

DISERTAČNÍ PRÁCE

AUTOR PRÁCE: Ing. Jan Panuš
ŠKOLITEL: Doc. RNDr. Bohdan Linda, CSc.

2008

UNIVERSITY OF PARDUBICE

FACULTY OF ECONOMICS AND ADMINISTRATION

INSTITUTE OF SYSTEM ENGINEERING AND INFORMATICS

EVOLUTIONARY ALGORITHMS IN OPTIMIZATION'S PROBLEMS OF PUBLIC ADMINISTRATION

DISSERTATION

AUTHOR: Ing. Jan Panuš
SUPERVISOR: Doc. RNDr. Bohdan Linda, CSc.

2008

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně Univerzity Pardubice.

V Pardubicích dne 8. 7. 2008 Ing. Jan Panuš

Poděkování

Na tomto místě bych rád poděkoval především svému školiteli Doc. RNDr. Bohdanu Lindovi, CSc. za jeho ochotu, vstřícný přístup a trpělivost během mého studia a také zejména za jeho odborné vedení a cenné rady, kterých se mi během zpracování této práce dostalo.

Dále bych rád poděkoval prof. Ing. Vladimíru Olejovi, CSc. za včasný a správný impuls pro zpracování této práce a za jeho cenné rady.

Nakonec bych velice rád poděkoval své rodině – rodičům, prarodičům, bratrovi a přítelkyni za jejich podporu a neutuchající zájem o výsledky mého snažení, čímž mne neustále nutili práci dokončit.

Anotace

V této doktorské disertační práci je předveden vytvořený algoritmus, jehož základem jsou některé algoritmy založené na lokálním prohledávání. Do algoritmu lokálního prohledávání byl přidán penalizační faktor spolu s aspiračním kritériem, které by mělo urychlit vyhledávání optimálního (nebo alespoň suboptimálního) řešení ve vybraných problémech. Následně je předvedena funkcionalita algoritmu na vybraných úlohách z oblasti problematiky obchodního cestujícího a také funkcionalita na vybraných testovacích funkcích. Jednotlivé výsledky měření jsou porovnány a na jejich základě je zkoumáno, zda je tento algoritmus úspěšný či nikoliv.

Annotation

In this thesis we show how local search algorithm can be applied to a set of problems, and we show that this algorithm improves its performance. We added an aspiration criterion from Tabu Search algorithm to improve local search algorithm in order to advance the performance of some problem types and parameter settings. We demonstrate then the functionality of the algorithm on some types of problems of Travelling Salesman Problem and on some test's functions; we make some search monitors for this extension to analyse in case this extension fails or succeeds.

Klíčová slova

Optimalizace, stochastické optimalizační algoritmy, penalizační lokální prohledávání, zakázané prohledávání, aspirační kritérium, problém obchodního cestujícího

Key words

Optimization, stochastic optimization algorithms, penalty local search, tabu search, aspiration criterion, travelling salesman problem

OBSAH

Seznam použitých zkratek

ACO	Optimalizace mravenčí kolonie (Ant Colony Optimization)
AK	Aspirační kritérium
B&B	Metoda větví a mezí (Branch and Bound)
CSP	Constrain Satisfaction Problems
DLM	Diskrétní Lagrangeův multiplikátor (Discrete Lagrange Multipliers)
FLS	Rychlé lokální prohledávání (Fast Local Search)
GA	Genetický algoritmus
HCwL	Horolezecký algoritmus s učením
ILS	Iterační lokální prohledávání
IP	Celočíselné programován (Integer Programming)
LP	Lineární programován
MA	Memetický algoritmus
MAX-SAT	Problém maximální uspokojitelnosti (Maximum Satisfiability Problem)
MP	Matematické programování
NP	Nedeterministicky polynomiální problémy
P	Polynomiální problémy
PLP	Algoritmus penalizačního lokálního prohledávání
QAP	Quadratic Assignment Problem
ReTS	Reaktivní tabu prohledávání (Reactive Tabu Search)
RLFAP	Radio Link Frequency Assignment Problem
RTS	Robustní tabu prohledávání (Robust Tabu Search)
SA	Simulované žíhání (Simulated Annealing)
SAT	Problém uspokojitelnosti (Satisfiability Problem)
SJ	Simulované skákání (Simulated Jumping)
TS	Tabu Search
TSP	Problém obchodního cestujícího (Travelling Salesman Problem)
VNS	Nastavitelné prohledávání sousedství (Variable Neighbourhood Search)

Seznam použitých symbolů

α	Koeficient pro snižování teploty v SA
D	Matice vzdáleností v TSP
d_{ij}	Vzdálenosti mezi městy i a j
D'	Upravená matice vzdáleností v TSP o penalizaci
E_n	Euklidovský n-rozměrný prostor
$f(x)$	Účelová funkce
$g(x)$	Zvýšená účelová funkce použitím penalizací
λ	Parametr, který se používá pro úpravu vlivu penalizací na velikost účelové funkce
n	Velikost problému (např. počet měst v TSP)
P	Matice penalizací v TSP
p_{ij}	Penalizace přiřazená trase i a j v TSP
X	Množina všech přípustných řešení
Tep	Koeficient vyjadřující teplotu v SA
$využitelnost(x, e_{ij})$	Využitelnost trasy e mezi městy i a j v řešení x

Úvod

Optimalizační problematika je řešena v matematických či technických oborech již poměrně dlouhou dobu. Dříve (od doby po druhé světové válce) byla tato problematika řešena pomocí dnes již klasických metod. Postupem času se rodina těchto problémů dále rozvíjela, avšak aparát založený na infinitezimálním počtu, variačních metodách aplikovaných ve funkcionálních prostorech či numerických metodách nebo v problémech, které se týkají teorie grafů, umožňuje hledání globálních extrémů pouze v problémech jednoduššího charakteru. Tyto metody již však současným problémům nemohou postačovat a to z několika důvodů. Například již není žádným problémem definovat argumenty účelové funkce v různých oborech (reálný, celočíselný, logický atp.), nebo některý argument účelové funkce se může v určitých subintervalech měnit anebo na něj mohou být aplikována různá omezení, která vyplývají z fyzikální či ekonomické realizovatelnosti. Z výše definovaného pak tedy vyplývá, že je potřeba mnohem výkonnějších metod, které ulehčí řešení složitých optimalizačních úkolů.

A právě pro řešení takových úloh, které víceméně vychází z reálného života, byla vyvinuta množina nového typu algoritmů a to tzv. evolučních algoritmů. Evoluční algoritmy se vyznačují několika zvláštnostmi oproti klasickým metodám, jako je například využití náhody, a tím je činí robustnějšími a široce použitelnými. Je vhodné, aby řešitel dobře znal oblast, kterou se snaží optimalizovat a tedy i správně definoval účelovou funkci, kterou hodlá optimalizovat. Tyto algoritmy jsou určeny i pro hledání globálního extrému, což jim také samozřejmě přidá na výhodě. U klasických numerických metod se také stávalo, že nabídly pouze jediné řešení, kdežto metody z oblasti evoluční strategie mohou nabídnout řešení několik.

Jako vše v životě i tyto techniky mají své kladné nebo záporné stránky. Mezi několik nevýhod, které tyto techniky mají, patří to, že se velice těžko sestavují matematické důkazy o jejich konvergenci. Většinou se při zpracování těchto algoritmů vychází ze zkušeností, jež jednoznačně ukazují na jejich životaschopnost.

Disertační práce se zabývá právě takovými metodami, které pochází z této rodiny a jež za jistou dobu prokázaly svou kvalitu a robustnost. Ke splnění definovaného cíle práce bude využita smíšená heuristická technika vytvořená na základě znalostí a vlastností tří známých metod využívaných v optimalizaci, a to konkrétně metodou lokálního prohledávání,

simulovaného žíhání a zakázaného prohledávání. Vytvořený algoritmus má za úkol efektivně projít prohledávaný prostor vybraného optimalizačního problému a nalézt tak kvalitní řešení. Navržený algoritmus je následně otestován na několika vybraných problémech.

1. Současný stav řešené problematiky

1.1 Formulace úlohy

V praktickém životě se setkáváme s mnoha problémy, kdy z různých variant vybíráme jednu, vhodnou pro naše potřeby. Tyto různé varianty nebývají z hlediska subjektu vždy rovnocenné a pak tedy vzniká problém výběru té nejvhodnější (optimální) varianty. Z tohoto hlediska se pak o takovém výběru hovoří jako o optimálním rozhodování.

Pokud vytvoříme pomocí matematických prostředků model nějakého jevu (jež se skládá z různých znaků a charakteristik), pak tento model nazýváme *matematický model*. Dále matematický model problému optimálního rozhodování budeme nazývat *optimalizačním modelem* [LAŠČ90]. Optimalizační model lze vyjádřit soustavou rovnic a nerovnic a cílem rozhodování je pak nalézt co možná nejvyšší (resp. nejnižší) hodnoty jedné, nebo více funkcí na množině všech řešení dané soustavy. Řešením takových modelů se zabývá disciplína zvaná *matematické programování* a modely uvedeného typu se nazývají *úlohy matematického programování* (dále jen MP) [LAŠČ90].

Z obecného hlediska je problematika matematického programování za omezených podmínek definována následujících modelem [KOOP57], [LAŠČ90]

$$\textit{optimalizuj} \qquad f(\boldsymbol{x}) \tag{1}$$

$$\textit{za podmínek} \qquad h_i(\boldsymbol{x}) \geq 0 \quad i = 1,2,\ldots,m \tag{2}$$

$$\boldsymbol{x} \geq \boldsymbol{0},\ \boldsymbol{x} \in E_n \tag{3}$$

kde $f(\boldsymbol{x})$ je obecná funkce, kterou hodláme optimalizovat a jíž nazýváme *účelovou funkcí*. Funkce $h_i(\boldsymbol{x}),\ i = 1,2,\ldots,m$ spolu s omezením $\boldsymbol{x} \geq \boldsymbol{0}$ se nazývají soustava omezení úlohy MP. Funkce $h_i(\boldsymbol{x})$ se nazývá *strukturní omezení*, $\boldsymbol{x} \geq \boldsymbol{0}$ se pak nazývá *podmínka nezápornosti* řešení. V protikladu k mnoha existujícím teoriím a metodám nelineárního programování nemá tato naše formulace žádné požadavky na konvexnost, diferencovatelnost a spojitost účelové funkce či omezujících podmínek.

1.2 Obecný koncept

Pro charakteristiku hledaných řešení je vhodné představit některé obecné koncepty [SCHW95], [TAYL97] ze kterých teorie optimalizace vychází. Jsou jimi pojmy přípustné řešení a lokální nebo globální minimum.

Každý vektor $\boldsymbol{x}$ z euklidovského n-rozměrného prostoru (E_n) nazýváme *přípustným řešením* úlohy MP v případě, že $\boldsymbol{x}$ splňuje všechny podmínky, tzn. $h_i(\boldsymbol{x}) \geq 0$ a $\boldsymbol{x} \geq \boldsymbol{0}$. Množinu všech těchto vektorů nazýváme *množinou přípustných řešení* úlohy MP a ta se značí X. Vektor $\boldsymbol{x}$, ve kterém účelová funkce $f(\boldsymbol{x})$ nabývá optimum, nazýváme *optimálním řešením*.

Účelová funkce nabývá *lokální optimum* v bodě $\boldsymbol{x}' \in X$, když $\boldsymbol{x}'$ je přípustným řešením a když existuje takové okolí $N(\boldsymbol{x}')$ bodu $\boldsymbol{x}'$, že pro každé přípustné řešení $\boldsymbol{x} \in N(\boldsymbol{x}')$ platí $f(\boldsymbol{x}') \leq f(\boldsymbol{x})$ anebo $f(\boldsymbol{x}') \geq f(\boldsymbol{x})$. Takové řešení někdy nazýváme *suboptimálním* (resp. s*uboptimem)*. Funkce má v bodě $\boldsymbol{x}'$ ostré lokální optimum, jestliže existuje okolí bodu $\boldsymbol{x}'$ takové, že platí $f(\boldsymbol{x}') < f(\boldsymbol{x})$, resp. $f(\boldsymbol{x}') > f(\boldsymbol{x})$ pro všechna $\boldsymbol{x}$ z tohoto okolí, s výjimkou bodu $\boldsymbol{x} = \boldsymbol{x}'$.

Globální optimum nabývá v $\boldsymbol{x}' \in X$ tehdy, když uvedené nerovnosti platí pro všechna $\boldsymbol{x} \in X$. V případě, že X je prázdná množina, úloha nemá přípustná řešení a tudíž ani optimální.

Pokud platí, že pro všechny j = 1, 2, ..., n se každá proměnná může spojitě měnit na celé ose reálných čísel, tj. že obor přípustných hodnot D_j proměnné $\boldsymbol{x_j}$ je:

$$D_j = x_j; -\infty \leq x_j \leq \infty \qquad (4)$$

potom úlohu (1) nazýváme spojitou úlohou MP. Pokud jsou všechny D_j diskrétní množiny, pak hovoříme o *diskrétní úloze* MP. V případě, že jsou všechny koeficienty funkcí $f(\boldsymbol{x})$ a $h_i(\boldsymbol{x})$ konstantní, nazývá se příslušná úloha *deterministickou úlohou MP*. O *stochastických úlohách* MP hovoříme v případě, že se ve funkcích $f(\boldsymbol{x})$ a $h_i(\boldsymbol{x})$ vyskytují náhodné parametry, tj. některé parametry jsou náhodné veličiny.

Pokud proměnné x_j v úloze (1) nejsou časově závislé, pak se příslušná úloha MP nazývá *statickou úlohou*. Pokud je některá z proměnných funkcí času, nebo charakterizuje mnohokrokové procesy, pak tyto úlohy nazýváme *dynamické úlohy* MP (resp. úlohy dynamického programování).

Úlohu (1), ve které požadujeme maximalizaci účelové funkce, nazýváme maximalizační úlohu MP. Pokud požadujeme minimalizaci účelové funkce, hovoříme o minimalizační úloze MP.

Optimálním řešením úlohy MP pak nazýváme takové přípustné řešení $\boldsymbol{x}^* \in X$ úlohy (1), které v případě maximalizační úlohy splňuje, že

$$f(\boldsymbol{x}^*) = \max_{\boldsymbol{x} \in X} f(\boldsymbol{x}) < \infty, \tag{5}$$

resp. v případě minimalizační úlohy splňuje, že

$$f(\boldsymbol{x}^*) = \min_{\boldsymbol{x} \in X} f(\boldsymbol{x}) > -\infty. \tag{6}$$

Veličinu $f(\boldsymbol{x}^*)$ nazýváme optimální hodnotou účelové funkce [LAŠČ90].

1.3 Obtížnost úloh

Existuje mnoho algoritmů, které jsou navrženy pro prohledávání určitého prostoru řešení a jež slouží k nalezení optimálního řešení. V následujících odstavcích se zaměříme na exaktní a heuristické algoritmy spolu s některými pojmy, které s touto oblastí souvisí.

Tradiční exaktní optimalizační metody mohou být velmi efektivní v případě, že jsou vhodným způsobem přizpůsobeny dané úloze. Vyplácí se je aplikovat, pokud víme kdy kterou použít, resp. nepoužít. Obecně se dají rozdělit na dvě třídy [MICH02] a to buď algoritmy, které vyhodnotí kompletní řešení nebo na algoritmy, které k dosažení výsledku potřebují nějaké přibližné řešení nebo částečně vygenerované řešení. Kompletní řešení daného problému znamená, že máme specifikovány všechny možnosti daného problému, např. pro problém obchodního cestujícího (TSP) jsou známy všechny permutace všech vrcholů problému.

Na druhou stranu existuje celá řada algoritmů, jež porovnává několik (v mnoha případech dvě) nalezených řešení. Jakmile je nalezeno lepší řešení než nějaké předcházející, pak je původní řešení nahrazeno novým. Mezi tyto metody lze považovat lokální prohledávání, horolezecký algoritmus a do této třídy patří i např. metody jako simulované žíhání nebo algoritmus zakázaného prohledávání (jež patří mezi heuristické metody).

Pro mnoho typů problémů však nelze použít takové metody, které spočítají všechny možnosti vybraného problému. Důvodem je velký počet proměnných a jejich následná kombinace a tím pádem i velké množství vygenerovaných řešení. Pak je vhodné použít ty, jež počítají s částečným řešením. Částečné řešení problému může mít dvě formy a to buď neúplné řešení daného problému, anebo daný problém je zredukován (zjednodušen). Pro neúplné řešení problému je typické využití nějaké podmnožiny prohledávaného prostoru, např. pro TSP se použije permutace měst, jež obsahuje sekvenci nějakých konkrétních měst (např. 7 – 11 – 5 – 16). Pokud se použije daná možnost, bez zpětného ověření není jisté, zda je obsažena i v konečném (resp. v optimálním) řešení daného problému.

Alternativním východiskem může být použití dekompozice celého problému na množinu jednodušších (resp. menších) podproblémů. Naděje spočívá ve vyřešení každého z těchto jednodušších problémů, které lze eventuálně zkombinovat a dostat tak řešení původní úlohy. Příkladem pro TSP je nalezení nejkratší cesty z města *i* do města *j*, jež prochází přes vybraných (i náhodně) *k* měst z celkového počtu *n* měst.

Při použití těchto metod se objevují jisté obtíže. Jednou z nich je navržení takového způsobu organizace podmnožin dané úlohy, aby mohly být tyto podmnožiny efektivně prohledávány. Dalším problémem je tvorba takové funkce, která dokáže vhodným způsobem ohodnotit kvalitu částečného řešení. Úkol rozdělit prohledávaný prostor do podmnožin, které mohou být dále zkoumány efektivněji než celý prostor je velice nelehký. Jednou z možností je rozdělení prostoru na strom řešení a postupně prohledávat jednotlivé větve tohoto stromu. Metoda, jež se zabývá touto problematikou, se nazývá *metoda větví a mezí* (Branch and Bound – B&B).

Metoda B&B se snaží nalézt celkové nejlepší řešení $\boldsymbol{x}^*$ pomocí rozdělování původní množiny problému na menší a menší podmnožiny a v každé takové podmnožině jsou stanoveny dolní meze hodnoty účelové funkce. Tyto podmnožiny jsou chápány jako množiny řešení, jež odpovídají podproblémům původního problému. Po rozdělení původního množiny daného problému na dvě nebo více disjunktních podmnožin může docházet k následnému dalšímu rozdělení podmnožin. Zobrazíme-li toto větvení graficky, připomíná nám strom. Na nulté úrovni stromu tento obsahuje jeden uzel symbolizující původní množinu a na dalších úrovních strom obsahuje uzly, které symbolizují jednotlivé podmnožiny daného problému. Z každého uzlu vedou hrany do dalších uzlů jeho podmnožin. Ze jména metody vyplývá, že se jedná o dvě různé procedury a to větvení množiny a určování mezí podmnožin. Určení mezí znamená stanovení dolních mezí hodnot účelové funkce v každé podmnožině,

která je vygenerována pomocí větvícího procesu, jež je definován jako prohledávání stromu řešení.

Tento typ metody je efektivní pro takové typy úloh, které vyžadují kompletní výpočet všech možných řešení za pomoci stromové struktury. S rostoucím rozsahem řešeného problému roste i náročnost metody, proto je dobře aplikovatelná do určitého malého rozsahu.

V literatuře se lze poměrně často setkat s pojmy a výroky o NP-úplnosti některých úloh, o tzv. NP-těžkých úlohách atp. Jistým, avšak poměrně vágním vyjádřením může být prohlášení, že NP-těžké úlohy jsou obtížně řešitelné. Cílem následujících odstavců je objasnit pojem NP-těžké úlohy.

Pro pochopení některých základních vlastností výše uvedených pojmů je třeba rozlišovat typ úlohy a instanci úlohy. *Typem úlohy* rozumíme, jakým způsobem je úloha zadána, tj. jaká data a v jakém uspořádání budou tvořit zadání úlohy, co má být výsledkem a jaký má být konečný vztah mezi výsledkem a zadáním. Konkrétním příkladem úlohy daného typu je pak *instance úlohy*. Jedním z konkrétních typů úloh jsou tzv. *rozhodovací úlohy*, kdy výsledkem je rozhodnutí buď „ano“ nebo „ne“. Některé rozhodovací úlohy jsou odvozeny z optimalizačních úloh, kdy se k zadání přidá konstanta K, a ptáme se, zda existuje přípustné řešení s hodnotou účelové funkce, která je lepší nebo rovna K.

Vstupem do algoritmu, který řeší konkrétní typ úlohy, jsou nějaká data popisující instanci daného typu úlohy – tato data se nazývají vstupní data. Velikost instance pak měříme jako objem těchto vstupních dat. Obecně se používá velikost dat v bitech, ale např. v grafových úlohách se velikost dat vyjadřuje počtem vrcholů a počtem hran.

Další měřitelnou jednotkou, která určuje kvalitu konkrétní algoritmu, je čas (resp. doba práce algoritmu). Pro teoretické úvahy však tato jednotka není příliš vhodná, jelikož různé počítače mívají různě rychlé procesory. Počet základních instrukcí[1], které je nutno při výpočtu vykonat je o něco vhodnější termín, ale i zde existuje závislost na rychlosti procesoru. Z těchto důvodů se tedy používá tzv. asymptotické vyjádření doby práce algoritmu pomocí symbolu O.

Nechť u a v jsou dvě nezáporné funkce reálné proměnné. Pokud existují reálné konstanty l, m takové, že pro všechna $x > m$ platí $v(x) \leq l\, u(x)$, pak se píše $v = O(u)$. Tento vztah je odhadem funkce v shora. Většinou pro malé hodnoty (pro $x \leq m$) argumentů funkcí

[1] Základní instrukcí rozumíme nějakou základní elementární činnost, která je nutná pro vykonání daného problému. Typickým příkladem je porovnání hodnot dvou položek v procesu třídění nějakého seznamu.

u a *v* nemá vliv na chování těchto funkcí a také nezávisí na násobení jedné z funkcí nějakou konstantou. Pokud řekneme, že nějaký algoritmus pracuje v čase *O(u(n))*, kde *n* je velikost instance úlohy, pak to znamená, že doba práce na jednotlivých instancích od určité velikosti nepřesahuje nějaký násobek funkce *u*. Vyjádření doby práce algoritmu symbolem *O*, se nezíská měřením skutečné doby práce, ale provede se teoretickým rozborem činnosti konkrétního algoritmu. Teoretický rozbor se dá provést buď pro paměťovou složitost (závislost paměťových nároků algoritmu na vstupních datech) nebo pro časovou složitost algoritmu (každé množině vstupních dat se přiřadí určitý počet operací vykonaných při výpočtu). Pro naše účely je důležitější časová složitost algoritmu, proto se jí budeme nadále věnovat. Časovou složitost je možno stanovit buď v závislosti na konkrétních datech, anebo na základě znalosti rozsahu těchto dat (stanovených např. v bitech). Pokud tedy známe rozsah dat, je možné jednoduše odvodit, jaký počet časových jednotek bude spotřebován pro tato data. Pokud víme, jakou dobu trvají elementární operace, lze pak určit dobu trvání pro určitý konkrétní počet vstupních hodnot. V případě, že se tedy podaří vyjádřit časovou složitost algoritmu jako funkci rozsahu vstupních dat, pak je důležité to, jak roste časová složitost algoritmu v závislosti na růstu rozsahu vstupních dat. Z tohoto hlediska nás pak zajímá právě asymptotická složitost, tedy to jakým způsobem se bude měnit chování algoritmu v závislosti na změně velikosti vstupních dat.

Polynomiálně řešitelné úlohy jsou úlohy, pro něž existuje algoritmus, jež tuto úlohu řeší a doba takového řešení je pro nejhorší případ omezena shora asymptoticky nějakým polynomem. Pro takový algoritmus pak platí, že doba práce je *O(p(n))*, kde *n* je velikost vstupních dat a *p* je nějaký polynom. Takový algoritmus řeší danou úlohy v *polynomiálním* čase nebo je prostě *polynomiální*. Třída úloh P je tvořena polynomiálně řešitelnými rozhodovacími úlohami, kdy za rozhodovací úlohu považujeme takovou, jejíž výsledek je buď „ano“, nebo „ne“. Úlohy tohoto typu jsou většinou koncipovány tak, že se ptáme, zda existuje přípustné řešení s hodnotou účelové funkce, která je lepší nebo aspoň rovna nějaké definované konstantě. Zvláštním typem rozhodovací verze je pak dotaz, zda vůbec přípustné řešení existuje. Toto se provede tím, že se nějakým vhodným způsobem zvolí nějaká konstanta, např. jako velmi malé (resp. velmi velké) číslo. Následně se ptáme, zda existuje takové řešení, které je větší (resp. menší) než zvolená konstanta. Polynomiálně řešitelné úlohy jsou většinou úlohy, jež lze snadno vyřešit.

Naopak, do třídy úloh NP (nedeterministicky polynomiální) patří rozhodovací úlohy, pro které existuje tzv. *ověřovací algoritmus*, který má následující vlastnosti:

- vstupem algoritmu jsou data *d* popisující instanci úlohy a tzv. *certifikát*, který můžeme popsat jako nějakou blíže nespecifikovanou část dat, jejíž velikost je shora omezena určitým pevně daným polynomem vzhledem k délce vstupních dat,
- algoritmus pracuje v polynomiálním čase vzhledem k velikosti vstupních dat *d* a výsledkem je vždy odpověď „ano“ nebo „nevím“,
- pro instanci dané úlohy, kdy je správná odpověď „ano“, existuje takový certifikát *c*, že ověřovací algoritmus dá odpověď „ano“,
- pro instanci dané úlohy, kdy je správná odpověď „ne“, existuje takový certifikát *c*, že ověřovací algoritmus dá odpověď „nevím“.

Jedním z příkladů NP úlohy je nalezení hamiltonovské kružnice, kdy certifikátem je zde kružnice, ověřovací algoritmus ověřuje, zda se opravdu jedná o kružnici a zda prochází všemi body. Je velmi důležité jakým způsobem je položena otázka, která se v úloze řeší a to z důvodu požadavku certifikátu a potvrzení pouze pro kladnou odpověď. Pokud otázku obrátíme, můžeme dostat zcela jinou úlohu a ta již do třídy NP patřit nemusí (nebo může být i těžší).

Každá úloha ze třídy P je zároveň ve třídě NP, neboť polynomiální algoritmus, který řeší úlohu je možno považovat i za algoritmus ověřovací, kdy u něj na certifikátu nezáleží. V současné době však není známo, zda platí rovnost, ale všeobecně se věří, že P ≠ NP [DEVL05].

NP-úplná úloha (NP-complete) je pak taková úloha, jež patří do NP a je ve třídě NP nejtěžší v tom smyslu, že jakákoliv jiná úloha je na ní polynomiálně redukovatelná. Polynomiální redukovatelnost úlohy *A* na jinou úlohu *B* znamená to, že existuje polynomiální algoritmus, který ze vstupních dat *A* vytvoří vstupní data pro úlohu *B* taková, že odpovědi na oba problémy jsou stejné. *NP-těžká úloha* (NP-hard) je potom taková úloha *B*, na kterou lze polynomiálně redukovat nějakou NP-úplnou úlohu *A*.

Nejjednodušší přístup, jak řešit optimalizační problémy ze třídy NP je ten, že se vytvoří seznam všech přípustných řešení, vypočítá se hodnota jejich účelové funkce a vybere se to nejlepší. Takový přístup kompletního výpočtu je po praktické stránce nepoužitelný. Důvodem je vysoký počet přípustných řešení i u problémů s rozumnou velikostí.

Vzhledem ke složitosti NP-úplných optimalizačních problémů se pak většina vědců soustředí na jinou složku technik, tzv. heuristické optimalizační techniky nebo jednoduše heuristiky. Tyto techniky se snaží nalézt optimální řešení (nebo alespoň body poblíž optimálního řešení) v rozumném výpočetním čase. Někdy se také uvádí, že heuristiky jsou takové metody, které nezaručují nalezení optimálního (nebo ani přípustného) řešení. Zpočátku vývoje operačního výzkumu byly heuristiky brány s jistou skepsí, avšak v současné době si tyto techniky zajistily přední místo mezi optimalizačními metodami a to díky teoretickému výzkumu ve výpočetní složitosti, který signalizoval vrozenou složitost NP-složitých problémů.

Heuristika je všeobecně charakterizována *přechodem (krokem)* od nějakého přípustného řešení k jinému a *lokálním kritériem,* s jehož pomocí je z množiny možných následujících řešení vybíráno to výsledné. Heuristiky lze dělit na *primární* (postup začínající přípustným řešením a přechází se vždy na další přípustné řešení) a *duální* (postup začínající nepřípustným řešením a přechází se na řešení s menší mírou nepřípustnosti tak, aby se lokální kritérium zvýšilo co nejméně [JANA02].

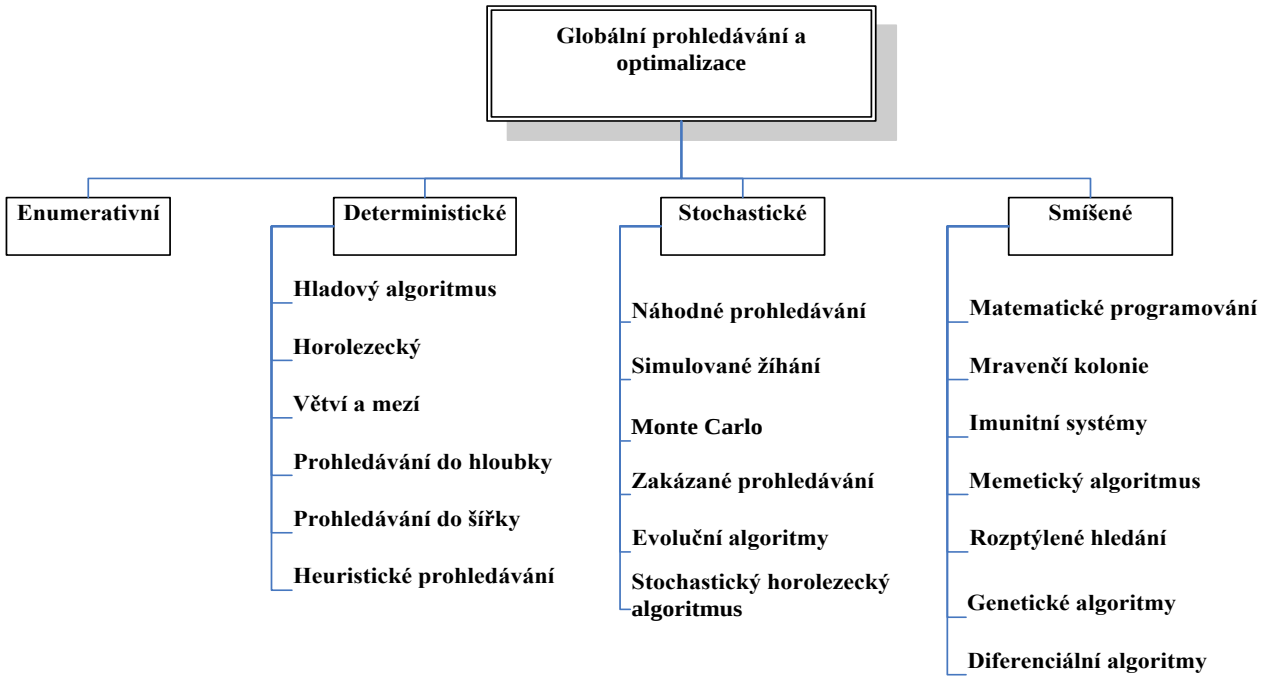
Metaheuristiky jsou takové heuristické postupy umožňující za jistých podmínek opustit lokálním optimum a přejít do jiných částí množiny přípustných řešení. Přechází se do takové oblasti, kde je určitá naděje nalezení řešení s lepší hodnotou účelové funkce, než bylo původně nalezené lokální optimum. Obdobně jako jiné heuristiky, ani metaheuristiky nezaručují nalezení optimálního řešení. Metaheuristiky přecházejí v jednotlivých krocích od přípustného řešení k jinému přípustnému řešení, které má lepší hodnotu účelové funkce, pomocí různých operací používaných minimalizačními heuristikami. Jako příklad takové operace v případě TSP může být výměna nebo inverze řetězce atp. Mezi známé metaheuristiky patří např. simulované žíhání, metoda zakázaného prohledávání, genetický algoritmus atp.

Moderní heuristické metody již dokáží nalézt vysoce kvalitní řešení i pro problémy, které jsou typu NP-úplných (např. alokace zdrojů, dopravní obslužnost, rozvrhování a mnoha jiných oblastí). V následující kapitole popíšeme některé obecné typy optimalizačních algoritmů a poté se budeme věnovat některým známým heuristickým metodám.

1.4 Prohledávací algoritmy

Optimalizační algoritmy, které slouží k nalezení minima dané účelové funkce pomocí hledání optimální numerické kombinace argumentů této funkce [LUEN84], lze rozdělit podle

principů jejich činnosti tak, jak je naznačeno na obrázku 1. Toto rozdělení není samozřejmě jediné možné, nicméně jej můžeme použít, neboť celkem dobře vystihuje současný stav a můžeme jej tedy brát i jako jeden z možných pohledů na klasické i moderní optimalizační metody.



Obrázek 1 - Rozdělení prohledávacích algoritmů – upraveno dle [ZELI02]

Jednotlivé třídy algoritmů představují obecně způsob řešení konkrétního problému pomocí metod s různým stupněm efektivity a složitosti a lze je charakterizovat následujícím způsobem [MICH00], [ZELI02]:

a) **enumerativní** – jedná se o výpočet všech možných kombinací argumentů daného problému. Tento přístup je vhodný pro problémy, u nichž jsou argumenty účelové funkce diskrétního charakteru a nabývají malého množství hodnot. Pokud by byl tento přístup použit obecně, zcela reálně by mohl potřebovat na úspěšné ukončení víc času, než je doba existence vesmíru.

b) **deterministické** – tato skupina algoritmů je postavena pouze na rigorózních metodách klasické matematiky. Algoritmy tohoto charakteru obvykle vyžadují předběžné předpoklady, jež umožní této metodě podávat efektivní výsledky. Jsou to obvykle následující předpoklady:

   i. problém je konvexní,

ii. prohledávaný prostor možných řešení je spojitý,

iii. účelová funkce má pokud možno pouze jeden extrém (je unimodální)

iv. problém je definován v analytickém tvaru.

c) **stochastické** – algoritmy tohoto typu jsou založeny na využití náhody. Jde v podstatě o čistě náhodné hledání hodnot argumentů účelové funkce s tím, že výsledkem je vždy to nejlepší řešení, které bylo nalezeno během celého náhodného hledání. Tyto typy algoritmů jsou většinou vhodné pro řešení úloh menšího rozsahu, avšak existují i jisté modifikace těchto algoritmů, které lze použít i pro úlohy většího rozsahu.

d) **smíšené** – tato třída algoritmů představuje směs metod deterministických a stochastických, které ve vzájemné spolupráci dosahují překvapivě dobrých výsledků. Poměrně silnou skupinou těchto algoritmů jsou evoluční algoritmy. Tyto smíšené algoritmy:

i. jsou robustní, což znamená, že nezávisle na počátečních podmínkách často naleznou kvalitní řešení,

ii. jsou efektivní a výkonné tzn., že jsou schopné nalézt kvalitní řešení během relativně malého počtu ohodnocení účelové funkce,

iii. jsou odlišné od čistě stochastických metod a to díky přítomnosti podmnožiny deterministických metod,

iv. mají minimální nebo žádné požadavky na předběžné informace,

v. jsou schopné pracovat s problémy typu „černá skříňka“, tzn., že nepotřebují ke své činnosti analytický popis problému,

vi. má-li účelová funkce globální extrém ve více argumentech, tyto algoritmy jsou schopny nalézt více než jeden takovýto argument.

Po shrnutí předešlých informací lze konstatovat, že:

a) enumerativní a deterministická optimalizace není vhodná na problémy, u nichž se prohledává rozlehlý prostor možných řešení,

b) stochastické algoritmy pracují dobře na problémech, u nichž se prohledává úzký prostor možných řešení, avšak po určitých modifikacích daných algoritmů jsou vhodné i pro použití na větší rozsahy možných řešení,

c) smíšená optimalizace je vhodná na problémy bez omezení velikosti jejich prostoru možných řešení.

1.5 Algoritmy Constraint Satisfaction

Standardní definice Constraint Satisfaction Problem (CSP) [PARD87], [SELM93a] je následující. Máme dánu množinu proměnných, z nichž každá nabývá konečný počet hodnot, a množinu omezení. Každé omezení je definováno přes určitou podmnožinu všech proměnných a omezuje povolené hodnoty jednotlivých proměnných. Cílem je pak nalézt jedno (nebo všechna) přiřazení hodnot proměnných tak, že tyto hodnoty vyhovují všem omezením. Jakýkoliv CSP problém s n-árními omezeními ($n>2$) lze transformovat na ekvivalentní s pouze binárními omezeními [PARD87]. V následující části se proto budeme věnovat pouze unárním a binárním omezením. CSP algoritmy se mohou např. využít v grafových úlohách. Proměnná je reprezentována vrcholem grafu a každá hrana reprezentuje omezení mezi proměnnými. Hrana, která vychází i končí ve stejném uzlu, reprezentuje unární omezení problému.

Způsobů jak řešit tyto problémy je několik, zde se budeme věnovat pouze dvěma z nich.

Jedním z přístupů je generování a testování. Postupně se generují všechna možná přiřazení hodnot proměnných a první přiřazení, které splňuje všechna omezení, je řešením CSP. Zde se jedná o analogii naivního třídění za využití permutací. Složitost tohoto problému je pak dána kartézským součinem všech oborů hodnot.

Druhým přístupem je metoda prohledávání s návratem (backtracking), kdy jsou postupně zkoumána řešení problému. Jakmile je vytvořena množina řešení, svázaných konkrétním omezením, je testována splnitelnost omezení. Pokud je testování neúspěšné, je změněna hodnota posledně přiřazené proměnné, jejíž hodnotu měnit lze.

Výpočetní složitost metody backtracking je pro úlohy většího rozsahu exponenciální, avšak podává lepší výsledky, než metoda generuj a testuj [PARD87].

Jedním z dalších známých problémů, na kterém byly např. prezentovány schopnosti splnitelnosti množiny výrokových formulí, je tzv. SAT problém (satisfiability problem).

Tento termín se obvykle nepřekládá, proto se bude v práci objevovat pod zkratkou SAT. Základem problému je hypotéza, že formule výrokové logiky α je formule sestavená z proměnných x_1, x_2,..., x_n a logických spojek negace, konjunkce a disjunkce. SAT-problém je obvykle definován jako rozhodnutí, zda je formule α splnitelná, tj. zda existuje takové ohodnocení proměnných v α, že hodnota α je „pravda“.

Přístup metody lokálního prohledávání považuje metody typu CSP jako optimalizační problematiku. Účelová funkce, která má být minimalizována, je pak dána počtem omezení, jež mají být splněny. Typická metoda lokálního prohledávání přiřadí náhodnou hodnotu každé proměnné, která se vyskytuje v CSP. Poté se za pomoci heuristiky zvané minimalizačně konfliktní heuristiky (min-conflict) [OSMA93], [OSMA95] opakovaně snižuje počet omezujících narušení a to opětovným přiřazením nových hodnot proměnným. Tento opakovaný proces vede k řešení či k nalezení lokálního optima, kdy jsou ještě nějaká omezení stále narušována, ale již žádná další zlepšení za pomoci záměny hodnot jakékoliv proměnné již v tomto bodě nejsou možná.

Úspěšným přístupem jak se dostat z lokálního optima, a který je navržen v rámci CSP, je přiřadit váhy [JIŘI02] daným omezením konkrétního problému (podmínka pro SAT) a zvyšovat tyto váhy v lokálním optimu pro označené omezení (nesplněné podmínky pro SAT). Snahou tohoto zvyšování je potom jakési vyplnění aktuálního lokálního minima do té doby, než se podaří z tohoto minima uniknout.

1.6 Metoda lokálního hledání

Nejjednodušší (ale nejméně efektivní) heuristickou metodou je tzv. metoda lokálního hledání (*local search method*) [BENT97], [BURK97], [CENE94], [LUEN84] někdy také uváděná jako metoda vyhledávání bezprostředního sousedství nebo horolezecký algoritmus (viz paragraf 3.3.1). Tato technika je základem mnoha heuristických metod pro kombinatorické optimalizační algoritmy. Metoda určí směr nejprudšího spádu kompletním prohledáním sousedství náhodně (nebo jiným způsobem) vybraného počátečního řešení. Jedná se o jednoduchou iterativní metodu, která má za cíl nalézt alespoň dobré přibližné řešení. Tato metoda však trpí základní nectností gradientových metod, tj. že nejspíše skončí v lokálním optimu a nedosáhne globálního optima.

Aby mohl být vysvětlen princip lokálního prohledávání, lze uvažovat lehce odlišnou definici optimalizačního problému danou modelem (1), (2), (3). Optimalizační problém

je definován jako dvojice (X, f), kde X je množina všech přípustných řešení, tj. řešení splňujících omezení daného problému a f je účelová funkce, která zobrazuje množinu X do množiny reálných čísel. Cílem je pak nalézt takové řešení $\boldsymbol{x}$ v množině X, které optimalizuje účelovou funkci f.

Množina $N(\boldsymbol{x})$ obsahující všechna řešení, která mohou být dosažena z bodu $\boldsymbol{x}$ jednoduchým posunem (aplikací nějakého jednoduchého operátoru na bod $\boldsymbol{x}$), se nazývá *sousední okolí (sousedství)* bodu $\boldsymbol{x}$.

Řešení $\boldsymbol{x}$ se nazývá lokálním minimem funkce f v případě bezprostředního okolí $N(\boldsymbol{x})$, jestliže platí:

$$f(\boldsymbol{x}) \leq f(\boldsymbol{y}), \ \forall \boldsymbol{y} \in N(\boldsymbol{x}). \tag{7}$$

Lokální hledání je pak procedura, která minimalizuje účelovou funkci f počtem úspěšných kroků, kdy je aktuální řešení $\boldsymbol{x}$ (7) nahrazeno takovým řešením, že:

$$f(\boldsymbol{y}) < f(\boldsymbol{x}), \boldsymbol{y} \in N(\boldsymbol{x}). \tag{8}$$

Základní vyhledávací algoritmus se spouští v jakémkoliv náhodném řešení a končí v lokálním minimu, kde žádné další zlepšení již není schopen nalézt. Mezi těmito dvěma stavy pak existuje mnoho rozličných způsobů jak lokální vyhledávání provádět. Například tzv. největší zlepšení (best improvement) lokálního prohledávání se aktuální řešení zamění s takovým řešením, kde je zlepšení největší (resp. kde je nejlepší zlepšení u účelové funkce) až po prohledání celého okolí výchozího bodu. Jiným příkladem je algoritmus tzv. prvního zlepšení (first improvement), kdy lokální prohledání funguje tak, že je akceptováno lepší řešení v momentu jeho nalezení. Výpočetní složitost lokálního prohledávání záleží na velikosti bezprostředního okolí daného řešení a také na počtu kroků, které jsou potřebné ke zhodnocení konkrétního tahu ve funkčním okolí. Obecně se dá říci, že čím větší okolí bodu, tím více kroků je potřeba k jeho prohledání a tím lépe se dá nalézt lokální minimum.

Velkým problémem lokálního prohledávání je právě lokální minimum. Ačkoliv nalezené řešení může mít dobrou kvalitu, ještě to neznamená, že je nutně optimální. Kromě toho, kdy se lokální prohledávání ocitne v lokálním minimu, neexistuje nějaký zřetelný způsob jak zajistit nalezení globálního minima. Metody založené na lokálním prohledávání, které se snaží odstranit tento problém, se někdy nazývají metaheuristickými metodami. Jedna

z jednodušších metod definovaných v této třídě je tzv. opakované lokální prohledávání, kdy je algoritmus spouštěn z nového, náhodně vygenerovaného řešení, až po dosažení lokálního minima. Tento proces se provádí do té doby, dokud není vyčerpán předem daný počet opakování. Nejlepší lokální minimum, které je nalezeno v průběhu výpočtu, je bráno jako přibližné globální minimum. Moderní metaheuristiky se snaží být důmyslnější než opakované lokální prohledávání. Tyto techniky sledují větší rozsah snah, jež přesahují jednoduché vyváznutí z lokálního minima. V následujících částech se budeme věnovat některým z těchto víceméně úspěšných moderních metaheuristických technik.

1.7 Metodika pro určení kvality řešení

V této kapitole budou popsány kritéria, která byla určující pro sledování průběhu vytvořeného hybridního algoritmu a která určují kvalitu tohoto algoritmu.

V rámci doktorské disertační práce byl empiricky testován hybridní algoritmus za využití různých statistických charakteristik. Tyto charakteristiky byly zaznamenávány v průběhu spuštěného algoritmu. V práci byly použity následující charakteristiky:

Tabulka 1 - Statistické hodnoty pro určení kvality navrženého algoritmu

Název hodnoty	Definice	Důvod použití
Počet iterací	Číslo vyjadřující, kolikrát byl daný algoritmus zopakován	Vyjadřuje výpočetní náročnost algoritmu.
Průměrné nalezené nejlepší řešení	Tato hodnota vyjadřuje průměrné nejlepší řešení nalezené v průběhu testování algoritmu.	Vyjadřuje kvalitu navštívených řešení v průběhu celého prohledávání.
Počet změn nejlepšího nalezeného řešení	Tato hodnota vyjadřuje průměrný počet změn u nalezeného řešení, které bylo lepší než to předcházející nalezené.	Vyjadřuje kvalitu nejlepších nalezených řešení v průběhu celého prohledávání.
Doba trvání průběhu algoritmu	Tato hodnota vyjadřuje celkovou dobu trvání algoritmu v sekundách.	Vyjadřuje časovou náročnost průběhu algoritmu.
Průměrná odchylka	Tato hodnota vyjadřuje procentuální rozdíl nalezeného řešení od nejlepšího známého řešení.	V případě TSP vyjadřuje kvalitu a robustnost navrženého algoritmu.

Výsledky nalezené pomocí vytvořeného hybridního algoritmu byly porovnány s již dříve naměřenými hodnotami jiných autorů a to u takových algoritmů, které mají podobné vlastnosti jako algoritmus navržený.

2. Cíle doktorské disertační práce

2.1 Hlavní cíl disertační práce

Hlavním cílem disertační práce je vytipování vhodných heuristických metod řešících optimalizační problémy typu obchodního cestujícího a vytvoření vlastního algoritmu, využívajícího tyto metody, který poskytuje minimálně stejné výsledky a v kratším čase než za použití původních metod. Součástí cíle je rovněž provedení nezbytných experimentů, které umožní zjistit kvalitu a charakteristiky algoritmu vycházejícího z vytvořené modifikace metod.

2.2 Další cíle disertační práce

- V disertační práci bude proveden rozbor a analýza vybraných metod globální optimalizace.
- Otestování navrženého algoritmu na několika třídách testovacích funkcí
- Prokázat, že navržená metoda bude využitelná pro řešení vybraných optimalizačních problémů.

3. Metaheuristiky na řešení optimalizačních problémů

Jak již bylo řečeno výše, algoritmus lokálního prohledávání nejprve vygeneruje počáteční řešení a dále se pomocí konkrétní heuristiky vybírá takové řešení, které určitým způsobem vylepší hodnotu účelové funkce. Lokální prohledávání má však jednu stinnou stránku a to, že hodnota účelové funkce nemůže být po určitém počtu přesunů do sousedního řešení dále zlepšována (v případě minimalizace) a algoritmus může uváznout v lokálním optimu[2].

Metaheuristiky jsou takové heuristické metody, které jsou vytvořeny, aby za jistých okolností umožnily opustit lokální minimum a přejít posloupností iteračních kroků do jiných částí množiny přípustných řešení, kde je již naděje nalézt řešení s lepší hodnotou účelové funkce, než bylo v nalezeném lokálním minimu.

3.1 Metaheuristiky založené na náhodě

3.1.1 Simulované žíhání

Simulované žíhání (*simulated annealing - SA*) [AART89], [DAVI87], [GASS04], [MORR93], [PANU05], [TANG92] je variantou horolezeckého algoritmu, v němž jsou heuristické kroky směřující k horšímu řešení řízeny určitou pravděpodobností [TAYL97]. Přístup SA je založen na simulování fyzikálních procesů probíhajících při odstraňování defektů krystalické mřížky. Krystal se zahřeje na určitou (vysokou) teplotu a potom se pomalu ochlazuje (žíhá). Defekty krystalické mřížky mají při vysoké teplotě vysokou pravděpodobnost zániku. Pomalé ochlazování systému zabezpečí, že pravděpodobnost vzniku nových defektů klesá. Při žíhání se soustava snaží dostat do takového stavu, ve kterém je její energie minimální – tj. krystal bez defektů. Existuje určitá analogie tohoto přírodního procesu s procesem řešení optimalizačních problémů.

V další části probereme podrobněji algoritmus SA. Upravená definice (1) pro SA je vyjádřena následovně:

$$minimalizuj \;\; f(\boldsymbol{x}) \; pro \; \boldsymbol{x} \in X \tag{9}$$

[2] Každý optimalizační problém lze definovat jako minimalizační (resp. maximalizační), proto dále v práci budeme místo optimum používat termín minimum.

kde $f(x)$ je účelová funkce a X je prohledávaný prostor tvořený množinou všech přípustných řešení určených (2) a (3).

Potom řešení x^* je globálním minimem v případě, kdy platí, že

$$f(x^*) \leq f(x), \text{ pro všechna } x \in S. \tag{10}$$

Aktuální řešení x je přeměněno náhodnou transformací (sedmý řádek algoritmu) na nové řešení x' z okolí $N(x)$. Při SA se neprohledává celé sousedství aktuálního řešení (jako u jiných metod) a proto může být sousedství definováno ve větším rozsahu. Tato vlastnost je dána právě procesem náhodné transformace. Původní řešení se nahradí novým x' v následném procesu SA s pravděpodobností dle Metropolisova (11) vzorce:

$$P_r(x, x') = \exp\left(-\frac{(f(x') - f(x))}{Tep}\right). \tag{11}$$

Kde koeficient *Tep* značí hodnotu teploty, jež je v algoritmu dále upravována. Jestliže funkční hodnota nového řešení x' je stejná nebo lepší než funkční hodnota původního řešení x, tedy $f(\mathbf{x}') \leq f(\mathbf{x})$, položíme pravděpodobnost nahrazení rovnu jedné. V tomto případě je nové řešení automaticky akceptováno do dalšího procesu SA. V případě, že funkční hodnota nového řešení x' není lepší než funkční hodnota původního řešení x, $f(x') > f(x)$, pravděpodobnost akceptování je menší než jedna, ale i v tomto případě má nové řešení šanci pokračovat v SA.

Algoritmus simulovaného žíhání má následující jednoduchou formu – podle [KIRK83]:

```
x=náhodně vygenerované řešení
Tep=Tmax, x*=x, k=1
while (Tep>Tmin and k>0) do
begin t=0, k=0
    while(t<tmax and k<kmax) do
    begin t=t+1
        x'= transformace(x)
```

```
        if f(x′)≤f(x) then Pr=1 else Pr= e^(-(f(x′)-f(x))/Tep)
        if random<Pr then
        begin  x=x′
               k=k+1
               if f(x)<f(x*) then x*=x
        end
    end
    Tep=α*Tep
end
```

kde:

- x^* je nejlepší nalezené řešení v průběhu běhu algoritmu,
- Tep je koeficient teploty
- α je koeficient ochlazování

Teplota *Tep* je ohraničena minimální a maximální hodnotou $T_{min} \leq Tep \leq T_{max}$, snižování teploty je realizováno po každém provedení vnějšího cyklu. Způsob snižování teploty určuje tzv. plán chlazení. Nejjednodušeji je možné snižovat teplotu pomocí vynásobení koeficientem α. Zkušenosti ukazují, že nejlepší hodnoty koeficientu α jsou mezi 0.8 a 0.99. Ve složitějších implementacích algoritmu může být rychlost snižování teploty závislá na procentu úspěšných pokusů o překlopení do jiného stavu při aktuální teplotě.

Celočíselné proměnné t a k jsou počitadla pro vnější resp. vnitřní while-cyklus. Proměnná t zaznamenává celkový počet pokusů SA pro danou teplotu *Tep*, zatímco proměnná k zaznamenává počet úspěšných pokusů, které byly akceptovány Metropolisovým vzorcem. Pro volbu maximálních hodnot t_{max} a k_{max} neexistuje všeobecný předpis, t_{max} a k_{max} jsou obvykle ve vztahu k velikosti množiny sousedů, hodnota k_{max} je volena od několika set do několika tisíc a $t_{max} = 10 * k_{max}$. Volba jednotlivých parametrů metody je obvykle uskutečněna na základě zkušeností po mnoha experimentech.

Podmínka ukončení algoritmu je splněna buď dosažením minimální teploty nebo tzv. zmrazením krystalu, tj. neuskutečněním při dané teplotě ani jednoho úspěšného pokusu z celkového počtu t_{max} pokusů o „vykmitnutí z pevné pozice v krystalické mřížce“.

Reálná proměnná *random* je náhodně generované číslo z intervalu (0,1). Proměnná $\boldsymbol{x}^*$ zaznamenává nejlepší řešení v průběhu provádění celého algoritmu. Ve všeobecnosti proměnná $\boldsymbol{x}$ nemusí po skončení SA obsahovat nejlepší řešení.

Výkon SA silně závisí na typu snižování teploty. Nikoliv překvapivě bylo navrženo mnoho druhů snižování teploty. Například v [DOWS93] a [LUND86] jsou uvedeny tři způsoby snižování teploty:

- *Postupná redukce teploty:* V tomto případě je teplota konstantou pro jistý počet iterací (např. výběr náhodných tahů po dobu akceptace konkrétní testovací hodnoty) až do doby, než je hodnota teploty upravena nějakým pravidlem. Tímto pravidlem obyčejně bývá geometrická redukční funkce, která hodnotu teploty snižuje tímto způsobem: $\alpha(Tep) = \alpha * Tep$, kde $\alpha < 1$. Často pak bývá tento typ snižování teploty nazýván geometrické ochlazování. Jak již bylo uvedeno, nejlepších výsledků bývá dosaženo za využití hodnoty α v rozmezí $0.8 \leq \alpha \leq 0.99$ [LUND86]. Počet iterací pro každou hodnotu teploty záleží na velikosti prohledávaného prostoru, a u různých hodnot teploty se může měnit.

- *Spojitá redukce teploty*. V tomto případě je teplota snižována po každé provedené iteraci. Redukce teploty je velice pomalá a je vyjádřena vztahem $\alpha(t) = Tep/(1+b*Tep)$, kde b je velice malá hodnota.

- *Nemonotónní redukce teploty*. Teplota je měněna po každé provedené iteraci, avšak mohou nastat i případy, kdy je za určitých podmínek teplota zvýšena. O tomto typu redukce je také pojednáno v [MORR93].

V literatuře [KIRK83], [MORR93], [RUSS03] je popsána podrobná teorie SA. Byly dokonce dokázány existenční teorémy, za jakých podmínek SA poskytuje globální minimum funkce $f(\boldsymbol{x})$ v definičním oboru $\boldsymbol{x}$. Často se používá rozšíření SA směrem ke genetickému algoritmu (GA) [CHAM95]. Místo jednoho řešení se současně optimalizuje SA malá

populace řešení, které si vždy po určitém počtu kroků s malou pravděpodobností vymění informaci operací totožnou s křížením z GA. V literatuře [OSMA95], [REEV93] jsou popsány různé modifikace metody.

3.1.2 Metoda Simulated jumping

Varianta SA nazvaná simulované skákání (Simulated Jumping - SJ) [AMIN99] je relativně novou metaheuristickou metodou. Metoda je založená na předpokladu, který se využívá ve fyzice. Některé materiály obsahující feromagnetické i antiferomagnetické složky, jež mohou mít mnoho nestabilních stavů. Pro tyto typy složek je pak mnohem těžší nalézt základní stav (nízkoenergetický stav, resp. stav odstranění defektu krystalické mřížky materiálu) pouze samotným ochlazováním a proto se využívá jiný proces, kdy je tento materiál prudce zahříván a poté prudce ochlazován. Tím pádem se dosáhne nízkoenergetického stavu lépe. SJ se snaží využívat této teorie i v oblasti kombinatorických optimalizačních problémů. SJ, narozdíl od SA (kdy se postupně snižuje pravděpodobnost přijetí rostoucí tendence – pro minimalizaci), zvyšuje a snižuje tuto pravděpodobnost v průběhu běhu algoritmu.

Pseudokód je uveden níže. Ochlazovací a zahřívací postupy jsou navrženy v [AMIN99] a mohou být různě adaptovány pro různé typy problémů. Algoritmus je podobný jako algoritmus SA až na tu výjimku, že pokud není uskutečněn žádný přesun, teplota je zvětšena. Snížení teploty se provádí pouze po jistém počtu přesunů (resp. po jistém počtu zvýšení teploty). SJ bylo s úspěchem aplikováno na problém asymetrického obchodního cestujícího, problémy mobilních radiových sítí atp.

Algoritmus simulovaného skákání má následující jednoduchou formu – podle [AMIN99]:

```
//typické parametry pro algoritmus (převzato z [AMIN99])
//T0 = 0.001, γ = [0.001, 0.2], R = 0.15, MaxCykly = 300
     begin
     x=náhodně vygenerované řešení
     x*=x
     do
        for i = 1 to MaxCykly
         náhodný výběr bodu y ze sousedství N(x)
         Δf = f(y) – f (x)
         if (Δf < 0 )
```

```
      x = y
    else
      if (r < e^(-Δf/Tep))
        x = y                          //Akceptována změna
      else
        Tep = Tep+R/i                  //Zahřátí systému
    Tep = γ*Tep                        //Ochlazení systému
     if (f(x) < f(x*))
        x*=x
  end
while not ukončovací podmínka
end
```

kde:

- x^* je nejlepší nalezené řešení v průběhu běhu algoritmu,
- r je náhodné číslo z rozsahu 0,1,
- R je koeficient zahřátí systému,
- γ je parametr sloužící k ochlazení systému.

3.1.3 Iterativní lokální prohledávání

Jednou z nejjednodušších metaheuristik pro lokální prohledávání je opakované náhodné generování počátečního řešení, ačkoliv to znamená, že se všechny předešlé informace, dosažené v průběhu prohledávání, ztratí. Sofistikovanější verze tohoto přístupu využívá informací, které byly zaznamenány v předchozím kroku prohledávacího algoritmu a nazývá se iterativní lokální prohledávání (Iterated Local Search) [GASS04], [STÜT99]. Hlavní myšlenkou tohoto přístupu je využití předcházejícího nalezeného lokálního minima a záměny (ve většině případů se využije nejlepší nalezené řešení) nebo modifikace řešení (obvykle se provede určitý počet náhodných přesunů v prostoru) pro vytvoření nového počátečního řešení, čímž se zvyšuje pravděpodobnost navštívení slibnějších oblastí prohledávaného prostoru.

Modifikace řešení na základě znalostí předešlých lokálních optim se nazývají „kick-moves“ [STÜT99] a pomáhají zajistit únik z lokálních optim. Rozdíl mezi tímto

přístupem a pouhým náhodným prohledávání je v tom, že předešlé nalezené lokální optimum je využito pro vygenerování nové počáteční pozice.

Jednoduchý základní pseudokód pro iterativní lokální prohledávání je – podle [STÜT99]:

```
Iterativní_Lokální_Prohledávání
begin
     x = Vygenerované_Počáteční_Řešení
     x = Lokální_Prohledávání N(x)
     do
          x = Modifikace (x, historie)
          y = Lokální_Prohledávání N(x)
          x = Akceptační_Kritérium (x, y, historie)
     while (ukončovací podmínka)
end
```

Nejdříve se vygeneruje počáteční řešení (obvykle náhodně). Na tomto řešení se poté aplikuje procedura lokálního prohledávání. Funkce modifikace zajistí změnu řešení *x* za pomoci historie prohledávání (porovnají se výsledky zaznamenané v průběhu chodu algoritmu) a funkce vrátí nové řešení *x*. Toto nové řešení je poté vylepšeno lokálním prohledáváním a pak vrátí řešení *y*, které je novým lokálním minimem. Toto nové řešení je následně porovnáno s řešením *x* a spolu s informacemi, které byly zachyceny v průběhu historie algoritmu je rozhodnuto, zda se toto nové řešení zamění se starým nebo ne. Pokud akceptační kritérium schválí toto nové řešení, pak je prohlášeno za nové řešení *x* a tento proces se opakuje až do té doby než se naplní ukončovací podmínka.

Velká nevýhoda tohoto přístupu je v tom, že pokud lokální minimum nebo předešlé nejlepší řešení se nenachází dostatečně blízko (z hlediska minimálního počtu přesunů mezi dvěmi řešeními) ke globálnímu minimu, pak tato metoda nebude pravděpodobně lepší (může být i horší) než lokální prohledávání založené na náhodném výběru řešení.

3.2 Algoritmy založené na populaci

V této kapitole budou vysvětleny vybrané metaheuristické algoritmy, které si zachovávají populaci řešení. Pojem populace řešení znamená určitou skupinu individuálních řešení, které si algoritmus pamatuje a dále s ní pracuje.

3.2.1 Optimalizace mravenčí kolonie

Optimalizace mravenčí kolonie (Ant Colony Optimization - ACO) [DORI96], [ENGE05], [GAMB99], [MOSC93], [STÜT97] je algoritmus založený na chování mravenců v kolonii. Princip je následující. Zdrojem mravenců je jejich mraveniště a cílem je nalezení potravy. Mravenci po určité době naleznou optimální (nejkratší) cestu k cíli a po té se pohybují. Tento efekt je dán faktem, že mravenci si cestu značkují feromonem. Jeho intenzita pak ovlivňuje mravencovo rozhodování, kterým směrem se vydá. Pokud například dojde první mravenec k rozcestí, nejdříve se rozhodne náhodně. Při procházení po cestě zanechá stopu a při návratu touto cestou ji označkuje podruhé, jak se zvyšuje počet projití cestou, zvyšuje se i intenzita feromonu. Pokud prochází kratší cestou, trvá mu to i kratší dobu a tím pádem se začne zvyšovat i intenzita feromonu. Po určité době je intenzita feromonu tak silná, že se přidávají i ostatní mravenci a zesilování tak probíhá i nadále.

Pro účely implementace optimalizačních algoritmů je vytvářen umělý mravenec, jehož vlastnosti oproti skutečným mravencům jsou upraveny tak, aby došlo ke zlepšení výsledků algoritmů, které řeší konkrétní problémy. Podobnosti se skutečnými mravenci jsou především v koloniích spolupracujících mravenců, v použití feromonové stopy, v nepřímé komunikaci mravenců pomocí feromonové stopy a v pravděpodobnostním rozhodování. U umělých mravenců se však vyskytují i rozdíly oproti skutečným. Algoritmus si např. pamatuje vnitřní stavy (jakási osobní paměť, která zaznamenává doposud vykonané akce), umělí mravenci nejsou zcela slepí. Jiným rozdílem je to, že množství zanechaného feromonu je funkcí kvality nalezeného řešení.

Mravenci použití v koloniích fungují jako stochastické procedury vytvářející nová řešení interaktivním přidáváním komponent (feromonová stopa) do částečného řešení. Mravenci při výběru každé další komponenty zvažují informaci o řešeném problému, snaží se využít takovou informaci, která vede ke slibnému řešení problému. Jeden konkrétní mravenec bere v potaz zkušenosti získané v průběhu výpočtu ostatními mravenci; tyto informace jsou reprezentovány feromonovou stopou a neustále se vyvíjí.

Feromon je tedy vyjádřen vahou, která je přiřazena dané cestě vedoucí k cíli. Tato váha je aditivní, což umožňuje přidávat další feromony od dalších mravenců. V tomto algoritmu je také zohledněn fakt, že se feromony vypařují, pokud cesta není využívána a tím pádem se i snižuje hodnota vah u jednotlivých spojů. Tento fakt pak zvyšuje robustnost optimalizačního algoritmu pro nalezení globálního extrému.

Pseudokód pro algoritmus mravenčí kolonie následuje níže – podle [DORI96]:

```
Mravenčí algoritmus(γ, α1, α2, q, R, S) //např. γ=100, α1=0.1,
α2=0.1, q=0.9, R=n/3
begin
     foreach ant i
          x_anti= Vygenerováno_Počáteční_Řešení
          x_anti= Lokální_Prohledávání(x_anti)

     foreach feromonová dráha j, T_j = 1 / (γ* g(x*))
     zesílení = true
     do
          foreach ant i = 1 to n
               for k = 1 to r
                    s_pravděpodobností (q)
                         //exploatace
                         vyber sousední řešení x_anti^k z N(x_anti)
                         částečně náhodně, částečně takové, že
                         velikost feromonů je maximalizována

                    else
                          //explorace
                         vyber sousední řešení x_anti^k z N(x_anti)
                         částečně náhodně a částečně pomocí vah
                         směřujíc k takovým řešením, které mají
                         vysokou hodnotu feromonů

               end for
               x_anti^(r+1)= Lokální_Prohledávání (x_anti^r)
               if zesílení = true
                    x_anti = best x_anti^k pro k = 1 to r+1
               else
                    x_anti = x_anti^k
          if žádné zlepšení v jakémkoliv x_anti
               zesílení = false
          if existuje takové x_anti, že g(x_anti) < g(x*)
               x* = x_anti
               zesílení = true

          //Vypařování
          foreach feromonová dráha T_j = T_j *(1-α1)
          //Zesilování
          foreach feromonová dráha T_b přítomné v řešení x*
               T_b = T_b + α2/g(x*)
          if proběhlo S iterací aniž by se x* zlepšilo
               //Proveď diverzifikaci
          foreach ant i, x_anti = Vygenerováno_Počáteční_Řešení
               foreach feromonová dráha j, T_j = 1 / (γ * g(x*))
               x_anti = x*
```

```
while ukončovací podmínka
end
```

Algoritmus začne tím, že každému mravenci přiřadí náhodné řešení a poté jej vylepší za pomoci lokálního prohledávání. Na základě ceny nejlepšího nalezeného řešení se inicializuje feromonová dráha (tedy matice vah) a pak každý mravenec provede určitý počet kroků a to buď exploatací (ke zlepšení hodnoty aktuálního řešení daného mravence) nebo explorací (pro modifikaci aktuálního řešení daného mravence – částečně náhodně, částečně pomocí vah k řešení, které obsahuje vysoké hodnoty feromonů). Výsledné řešení získané modifikací je pak vylepšeno pomocí lokálního prohledávání. Pokud se algoritmus nachází ve fázích zesilování, pak je nejlepší řešení, které je nalezeno v průběhu modifikace a po provedení lokálního prohledávání označeno jako aktuální řešení daného mravence [WANG06]. Toto se opakuje pro každého mravence. Pokud již není provedeno žádné zlepšení, fáze zesilování se vypne (prohledávaný prostor v této fázi nebude příznivým pro nalezení optimálního řešení). Pokud je však nejlepší řešení dále zlepšováno, fáze zesilování se naopak zapne (tedy prostor bude zřejmě slibný – fáze zesilování má analogii ve značkování cesty mravenci pomocí feromonů). Nyní má každý element feromonové dráhy svou hodnotu zredukovánu (dochází k simulaci vypařování feromonů na skutečné dráze). V dalším kroku se těm elementům, které jsou obsaženy v nejlepším nalezeném řešení, zvyšuje hodnota jejich feromonů (což vede k zesilování vhodných argumentů řešení). Pokud po určitém počtu iterací již neexistuje žádné další zlepšení nejlepšího nalezeného řešení, pak se algoritmus diverzifikuje přenastavením datové struktury feromonové dráhy a nastavením všech mravenčích řešení (až na jedno) na nové náhodné počáteční řešení. Zbývající mravenec si drží nejlepší nalezené řešení. Tento proces pokračuje do té doby, než není uspokojena nějaká ukončovací podmínka. Jako obvykle, toto je pouze jeden z možných příkladů mravenčího algoritmu a v literatuře se objevuje mnoho úprav algoritmu.

3.2.2 Evoluční strategie

Evoluční strategie patří historicky mezi první úspěšné stochastické algoritmy. Byla navržena už počátkem 60-tých let Rechenbergem a Schwefelem [BÄCK91], [HOLL75], [HOLL92], [MERZ99], [SEIF02]. Vychází ze všeobecných představ přirozeného výběru, ovšem o mnoho vágnějších než například u GA. Pro neuronové sítě může být tato metoda použita pouze pro optimalizaci vah nebo jiných proměnných, nikoli pro optimalizaci topologie sítě.

Základem evoluční strategie je následující předpis, který "mutuje" aktuální řešení $\boldsymbol{x}$ na nové řešení $\boldsymbol{x}'$,

$$\boldsymbol{x'=x+r(0,\sigma)}, \tag{12}$$

kde $\mathbf{r}(0,\sigma)$ je vektor náhodných čísel s nulovou střední hodnotou a směrodatnou odchylkou σ.

Pokud platí, že $f(\boldsymbol{x}') < f(\boldsymbol{x})$, pak je nové řešení $\boldsymbol{x}'$ akceptováno jako lepší řešení. Směrodatná odchylka σ se v průběhu strategie mění podle (13). Hodnota $\varphi(k)$ vyjadřuje koeficient úspěšnosti, který je definovaný jako poměr počtu úspěšných mutací v průběhu posledních k iterací vzhledem k celkovému počtu iterací.

$$\sigma' = \begin{cases} c_d.\sigma & \text{když } \varphi(k) < 1/5 \quad c_d < 1 \\ c_i.\sigma & \text{když } \varphi(k) > 1/5 \quad c_i > 1 \\ \sigma & \text{když } \varphi(k) = 1/5 \end{cases} \tag{13}$$

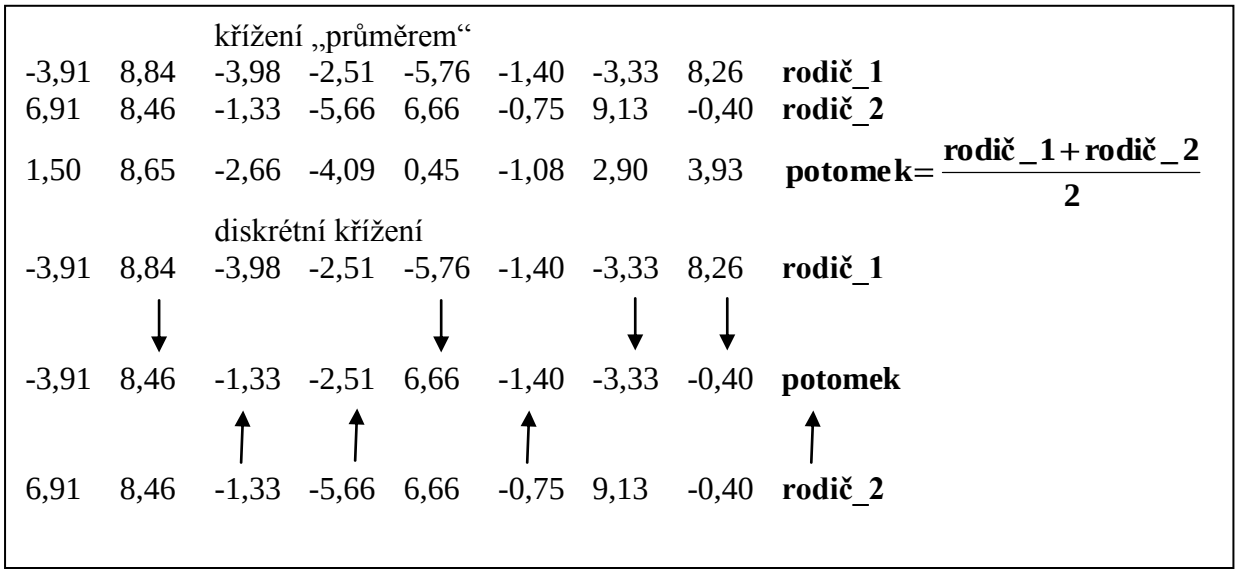
Hodnoty parametrů c_i a c_d řídí zvětšování resp. zmenšování směrodatné odchylky, v literatuře [SELM93a] jsou tyto koeficienty specifikované $c_d = 0.82$ a $c_i = 1/c_d = 1.22$. Algoritmus evoluční strategie v pseudokódu má tento tvar – podle [KVAS96]:

```
x = náhodně generovaný vektor reálných proměnných
t = 0, σ = σini, x* = x
while t < tmax do
begin i = 0, k = 0
     while i < imax do
     begin i = i+1, x' = x+r(0,σ)
          if f(x') < f(x) then
          begin k = k+1, x = x'
          if f(x) < f(x*) then x* = x
          end
     end
```

```
        if k/i_max<0.2 then σ = c_d·σ else if  k/i_max>0.2
        then σ = c_i·σ
    end
```

Proměnná *t* zaznamenává celkový počet iterací. Směrodatná odchylka je ve 2. řádku inicializována hodnotou σ_{ini} a pro dané σ je opakován elementární krok evoluční strategie i_{max}-krát. Proměnná *k* zaznamenává úspěšnost mutací v prvním (vnitřním) cyklu algoritmu. Druhý (vnější) cyklus aplikuje pro různé hodnoty σ evoluční strategii t_{max}-krát a dále obsahuje proměnnou *t* jako počitadlo iterací. V 6. řádku algoritmu se provede modifikace řešení ***x*** pomocí generátoru náhodných čísel, jehož střední hodnota je nulová a směrodatná odchylka je σ. Volba hodnot základních parametrů evoluční strategie (t_{max} , σ_{ini} , i_{max} a k_{max}) již vyžaduje určité experimentování, avšak obvykle se hodnota σ_{ini} blíží jedničce, a hodnoty i_{max}, t_{max}, k_{max} se rovnají řádově tisícům.

Podobně, jako pro SA, bylo dokázáno i pro evoluční strategii [SELM93a], že potenciálně poskytuje globální extrém optimalizované funkce ***f(x)***. Schwefelem se spolupracovníky [SCHW81] byly navrhnuty další sofistikovanější verze evoluční strategie, takže v současnosti je možné už mluvit o celé třídě evolučních strategií. Pracuje se zde poté s celým souborem vektorů ***x***. Kromě mutace se také používá křížení, tedy částečná výměna informací mezi vektory reálných čísel (viz obr. 2). Nejlepší jedinci se poté vyberou z takto navrhnutých vektorů a z původních "rodičovských" vektorů.



Obrázek 2 - Dva z mnoha druhů křížení používaných u evoluční strategie. Křížení průměrem vezme dva vektory, sečte hodnoty jejich prvků na odpovídajících místech a vydělí celý vektor dvěma. Křížení diskrétní přebírá hodnoty na odpovídajících místech náhodným výběrem z jednoho nebo druhého vektoru rodičovských jedinců - převzato z [KVAS00]

Kromě vlastní hodnoty proměnné ve vektoru může být každá proměnná charakterizována i vektorem "strategických" proměnných [KVAS00]. Totiž, některé proměnné mají větší vliv na hodnotu funkce než jiné proměnné, a proto by i jejich změny měly mít jiné měřítko. To může být zabezpečeno tím, že každá proměnná má svůj vlastní rozptyl. Pro dvě proměnné potom vrstevnice pravděpodobnosti umístění mutovaného vektoru nepředstavují kružnici, ale elipsu. Tato elipsa je však orientována ve směru souřadnicových os. Můžeme si však představit, že optimum není umístěno ve směru delší osy elipsy, ale někde našikmo. Ideální by pak bylo, kdybychom mohli tuto elipsu pravděpodobností umístění mutovaného vektoru natočit tak, aby hlavní osa elipsy směřovala k optimu. Tak by mutovaný vektor měl největší šanci co nejvíce se přiblížit k optimu. Toto natočení lze zajistit kovariancemi. Vektor "strategických" proměnných tedy může pro *n*-rozměrný vektor $\boldsymbol{x}$ zahrnovat *n* rozptylů $c_{ii} = \sigma_i^2$ stejně jako $n(n-1)/2$ kovariancí c_{ij} *n*-rozměrného normálního rozdělení pravděpodobnosti daného hustotou pravděpodobnosti vektoru mutací $\boldsymbol{z}$

$$p(\boldsymbol{z}) = \sqrt{\frac{1}{(2\pi)^n \det \boldsymbol{A}}} \exp\left(-\frac{1}{2} z^T \boldsymbol{A} \boldsymbol{z}\right) \qquad (14)$$

V literatuře se pak objevuje mnoho modifikací této metody.

3.2.3 Genetické algoritmy

Genetické algoritmy (genetic algorithms - GA) [FOX93], [GOLD89], [HARP94], [HAYK94], [MICH92], [MITC98], [OLEJ03], [PANU07C] jsou inspirovány Darwinovými zákony přirozeného výběru. V evolučním vývoji nebo při šlechtění rostlin či živočichů se prosazují jedinci, kteří mají jisté žádoucí charakteristiky, které jsou na genetické úrovni determinovány kombinací rodičovských chromozómů. U zrodu GA stála myšlenka, že při hledání lepších řešení složitých problémů by bylo možno obdobným způsobem kombinovat části existujících řešení.

Ačkoli genetické algoritmy nemají již prakticky s biologií nic společného, udržují si biologickou terminologii. Evolucí jsou míněny postupné změny řešení vedoucí k nalezení extrému funkce. Konkrétní řešení $\boldsymbol{x}$ tvoří jedince (někdy je formálně jedinec nazýván chromozómem). V případě TSP bude jedincem jedna vygenerovaná hamiltonovská kružnice. Hodnota zdatnosti (*fitness*) jedince odpovídá hodnotě účelové funkce $f(\boldsymbol{x})$ v tomto řešení. Není zde však tak důležitý samotný jedinec, jako postupný vývoj, kooperace a fungování populace

- souboru jedinců [MAŘÍ01a], [MAŘÍ01b]. Neúspěšní jedinci vymírají, úspěšní přežívají a množí se. Při aplikaci GA na problémy operačního výzkumu každý jedinec v algoritmu nějakým způsobem kóduje jedno řešení ***x*** problému.

Pro manipulace s chromozómy se používají genetické operátory selekce (*selection*), křížení (*crossover*) a mutace (*mutation*). Při selekci se jedná o výběr jedinců z celkové populace, kteří se stanou rodiči. Důležitým hlediskem, jež se přímo či nepřímo uplatňuje při výběru alespoň jednoho z rodičů, je právě jeho zdatnost. Hybnou silou změn jsou křížení (výměna genetické informace mezi jedinci) a mutace (velmi málo pravděpodobné provedení náhodné změny chromozómu, zabraňující zdegenerování populace).

Genetické algoritmy pracují tím způsobem, že se nejprve vytvoří počáteční populace o velikosti *p* jedinců a pak se tato populace mění pomocí genetických operátorů tak dlouho, dokud není splněna nějaká podmínka ukončení.

Volně lze kostru GA definovat takto – podle [MITC98]:

```
P:=počáteční_populace_chromozómů(p)
repeat
     R:=selekce(P)
     Q:=křížení(R)
     Q:=mutace(Q)
     P:=vytvořit_novou_populaci(P,Q);
until (podmínka_ukončení);
```

Počáteční populace se většinou získá náhodným generováním. Byly provedeny také pokusy nasadit do počáteční populace kvalitní řešení, získaná jinými heuristickými metodami, ale při tomto způsobu se ukázalo nebezpečí předčasné konvergence do nějakého ještě ne dost dobrého lokálního optima. Pokud jde o velikost *p* populace, je jasné, že příliš malá populace může zavinit špatné pokrytí prostoru řešení, zatímco velká populace zvyšuje výpočetní náročnost. Zkušenosti ukazují, že pro většinu problémů je dostačující populace o velikosti 50 až 200 jedinců.

Selekcí je třeba z populace vybrat zdatné jedince (s dobrou hodnotou *fitness*[3]), kteří se potom stanou rodiči. Rodiče se vybírají pseudonáhodně. Zdatnější jedinci mají větší šanci být vybráni než méně zdatní jedinci. Čím je jedinec zdatnější (čím má lepší hodnotu *fitness*), tím je pravděpodobnost jeho výběru ke křížení větší. Pokud se výběr uskutečňuje v souladu s rozdělením pravděpodobnosti, počítaným jako podíl hodnoty fitness chromozómu k celkové hodnotě fitness populace, označujeme tento způsob výběru jako „ruleta“. Jiným způsobem selekce je například soutěžení (*tournament*). Soutěže se účastní jen část populace a její vítěz, nejzdatnější účastník soutěže, se pak stane rodičem.

Křížení se pro vybranou skupinu rodičovských chromozómů může uskutečňovat několika způsoby. Například pro případ, kdy je jedinec reprezentován sekvencí čísel úloh, představuje nejjednodušší přístup tzv. jednobodové křížení, při němž se zvolí náhodně nějaký bod, dělící oba rodičovské chromozómy na dvě části. Jeden potomek pak zdědí levou část z prvního rodiče a na zbývající pozice se mu doplní chybějící prvky v tom pořadí, v jakém se vyskytují ve druhém rodiči, druhý potomek vznikne analogicky s obráceným pořadím rodičů. Např. jestliže řetězce BACDEF a DCABEF jsou rodičovské chromozómy a dělící bod se nachází za druhou pozicí, pak dostaneme potomky BADCEF a DCBAEF.

Mutace je v obecnosti malá náhodná změna jedné či několika proměnných (prvků chromozómu), která ovlivní řešení, ať už kladně nebo záporně. Pravděpodobnost uskutečnění mutace je nízká (obvykle menší než 1%). Mutace je nutná k tomu, aby se zamezilo přílišné specializaci – zapadnutí celé populace do jednoho lokálního minima; aby vždy byla možnost vytvoření zásadně nových chromozómů odpovídajících lepšímu řešení. Mutace přinášejí do chromozómů novou genetickou informaci. Pro mutaci mohou být použity stejné operátory, jakými lze generovat sousední řešení v předchozích popsaných metodách.

Jednou z možností změny *p*-členné populace je vygenerovat pomocí křížení a mutace novou generaci *p* potomků a nahradit jí rodičovskou generaci en bloc. Jiné způsoby umožňují nějaké překrývání populace rodičů a potomků a populaci tedy mění inkrementálně. Např. vygenerovaný potomek nahrazuje pseudonáhodně vybraného slabého příslušníka aktuální populace.

V literatuře [ROJA96], [WHIT90] jsou popsány různé modifikace GA spolu s neuronovými sítěmi.

[3] V přechodu do nové generace je pro každého jedince spočítána tzv. fitness funkce, jež vyjadřuje kvalitu řešení reprezentovaného tímto jedincem. Podle této kvality jsou vybíráni jedinci, kteří jsou dále modifikováni (pomocí mutace a křížení). Tím pak vznikne nová populace.

3.2.4 Memetické algoritmy

Memetické algoritmy (MA) [GRIG06], [HANS98], [MERZ99], [MOSC93] mají více obecný koncept než GA, protože kombinují myšlenky GA a lokálního prohledávání. MA mohou pracovat s mnoha různými typy algoritmů a heuristik.

Následující pseudokód (podle [MOSC89]) je jednoduchým příkladem MA.

```
Memetický_Algoritmus()
     begin
     populace = Vygenerovaná_Počáteční_Populace
     foreach x v populaci
          x = Lokální_Prohledávání(x)
     do
          for i = 1 to #rekombinací

               vyber náhodně dva rodiče p1, p2 z populace
               x = rekombinace(p1,p2)
               x = Lokální_Prohledávání(x)
               populace = Přidej_Do_Populace(x)

          end for

          populace = Vyber(populace)

          if Konvergence(populace)

               foreach x v populaci \ { nejlepší } do
                    x = Lokální_Prohledávání(Mutace(x))


     while (not ukončovací podmínka)
     end
```

Algoritmus na začátku vygeneruje množinu náhodných počátečních řešení. Poté je aplikován algoritmus lokálního prohledávání na každé toto řešení a také ho vylepší. Potom z této množiny náhodně vybere dva rodiče a zkombinuje je pomocí rekombinačního operátoru. Poté je znovu aplikován algoritmus lokálního prohledávání na výsledných chromozomech, které jsou přidány do populace. Tento proces se opakuje pro požadovaný počet rekombinací. Poté je vybráno nejlepší řešení a to je uchováno a všechna horší řešení jsou odstraněna. Pokud se populace po určitý počet iterací (obvykla cca 30) nezmění pak je zřejmě populace zkonvergována. Pokud se tak stane, operátor mutace je aplikován na každé řešení následován lokálním prohledáváním tak, aby znovu začalo prohledávání. Prohledávání pokračuje, dokud není uspokojeno nějaké ukončovací pravidlo. MA byl úspěšně využit např. pro řešení

kvadratického přiřazovacího problému (Quadratic Assignment Problem - QAP) [BURK97], [MOSC89].

Velmi podobným přístupem jako jsou MA je genetický hybridní algoritmus [FLEU94], který kombinuje GA s jinými negenetickými metodami optimalizace (např. použitím SA). Dalším podobným přístupem je algoritmus TS s elitně opakovaným spouštěním řešení. V tomto algoritmu je uchováván seznam tzv. elitních řešení, jejichž pomocí se následně generují nová počáteční řešení. Existuje celá řada dalších hybridních přístupů pro MA [FLEU94].

3.3 Algoritmy založené na prohledávání sousedství

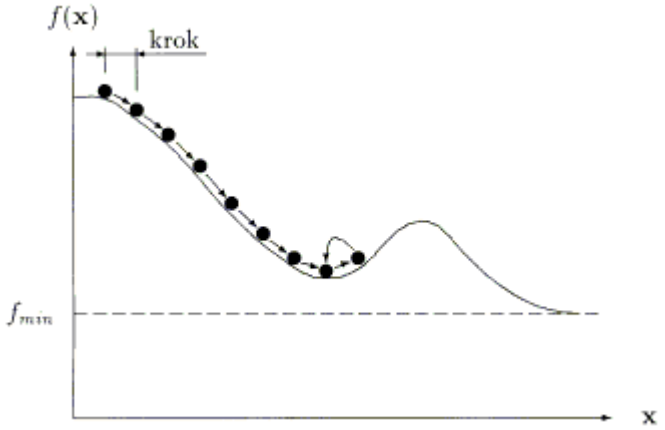
3.3.1 Horolezecký algoritmus

Horolezecký algoritmus (*hill climbing*) [MLAD97], [MORR93] patří do skupiny algoritmů, ve kterých jsou dovoleny i kroky směřující ke zhoršení aktuálního řešení. Podobně jako v metodě lokálního hledání v každém kroku vybíráme nejlepší řešení ze sousedních řešení. Rozdíl je v tom, že podmínkou ukončení algoritmu není nenalezení zlepšujícího řešení mezi sousedními řešeními, ale provedení určitého počtu iterací. Princip horolezeckého algoritmu je následující – podle [KVAS00]:

1) Zvolíme náhodně počáteční přípustné řešení.
2) Pro aktuální řešení vygenerujeme všechna sousední řešení, a vypočteme pro ně hodnoty účelové funkce.
3) Ze sousedních řešení vybereme takové, které má nejnižší hodnotu účelové funkce a zvolíme ho jako nové aktuální řešení.
4) Dokud není proveden předepsaný počet iterací, pokračujeme bodem 2.

V průběhu celé historie algoritmu zaznamenáváme nejlepší dosažené řešení, které slouží jako výsledné optimální řešení.

Základní nevýhodou horolezeckého algoritmu je, že se po určitém počtu iteračních kroků vrací k lokálně optimálnímu řešení, které se již vyskytlo v některém předcházejícím kroku (problém zacyklení – viz. obr. 3).



Obrázek 3 - Zacyklení u horolezeckého algoritmu – převzato z [KVAS00]

Tento problém se částečně řeší tak, že se algoritmus spustí několikrát z různých náhodně vygenerovaných počátečních řešení a za výsledek se bere nejlepší řešení z několika průběhů.

Obecně lze říci, že je hledáno řešení $\boldsymbol{x}^*$ *v* určité oblasti *X* pro které platí:

$$f(\boldsymbol{x}^*) = \min_{\boldsymbol{x} \in X} f(\boldsymbol{x}). \tag{15}$$

Dále je definována množina všech přípustných transformací *T*, kde transformace $t \in T$ (může být charakterizována nějakou jednoduchou operací např. překlopením všech možných bitů řetězce v případě hledání konkrétní hodnoty osmibitového řetězce) zobrazuje vektor $\boldsymbol{x} \in X$ na další vektor $\boldsymbol{x}' \in X$, kde $\boldsymbol{x}' \neq \boldsymbol{x}, t : X \rightarrow X$ pro $\forall\, t \in T$. Pro každou transformaci pak existuje transformace k ní inverzní. Sousedství $N(\boldsymbol{x})$ obsahuje obrazy $\boldsymbol{x}$ vytvořené transformacemi $t \in T$.

V tomto okamžiku již existuje aparát potřebný k formulaci horolezeckého algoritmu. Pro náhodně vygenerovaný vektor $\boldsymbol{x}$ se hledá minimum funkce $f(\boldsymbol{x})$ v sousedství $N(\boldsymbol{x})$,

$$f(\boldsymbol{x}^*) = \min_{\boldsymbol{x}' \in N(\boldsymbol{x})} f(\boldsymbol{x}'). \tag{16}$$

Získané řešení $\boldsymbol{x}^*$ je použito v následující iteraci algoritmu jako jakýsi střed nového sousedství $N(\boldsymbol{x}')$ a tento proces je *n*-krát opakován. V průběhu celého algoritmu jsou zaznamenávány hodnoty nejlepšího nalezeného řešení. Hlavním omezením je problém cyklických řešení. Po určitém konečném počtu iteračních kroků se algoritmus vrátí k řešení,

které se již vyskytovalo a bylo označeno jako lokální řešení v předcházejícím iteračním kroku.

Jednoduchou modifikaci standardního horolezeckého algoritmu provedl Kvasnička s kolegy [KVAS96]. Základním konceptem jejich pojetí je tzv. pravděpodobnostní vektor [TAYL97], jehož hodnoty určují pravděpodobnost výskytu hodnot 1 v *n*-bitovém vektoru. Tato modifikace je nazvána horolezecký algoritmus s učením (HCwL).

Účelová funkce *f* je definována jako zobrazení

$$f : \langle 0,1 \rangle^n \rightarrow R, \tag{17}$$

jež přiřadí každému *n*-bitovému vektoru $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_n)$ reálné číslo $y \in R$, formálně $y=f(\boldsymbol{\alpha})$. Úkolem je pak nalézt takový vektor $\boldsymbol{\alpha}_{opt} \in \langle 0,1 \rangle^n$, který odpovídá globálnímu minimu funkce *f* v prostoru $\{0,1\}^n$

$$f(\boldsymbol{\alpha}_{opt}) = \min_{\alpha \in \langle 0,1 \rangle^n} f(\boldsymbol{\alpha}) \tag{18}$$

Prohledávaný prostor $X = \{0,1\}^n$ je pro tento typ optimalizace složen z 2^n bodů (*n*-bitových vektorů). Z toho tedy plyne, že pro větší hodnoty *n* není možné použít přístup založený na kompletním prohledání prostoru *X*, požadovaný čas pro výkon algoritmu roste exponenciálně. Symbolem (17) je pak míněna množina všech *n*-bitových vektorů, jejichž souřadnice jsou pouze 0 nebo 1.

Kvasnička [KVAS96] nejprve zavádí takzvanou mutaci *n*-bitového vektoru $\boldsymbol{\alpha} \in \langle 0,1 \rangle^n$ na další vektor $\boldsymbol{\alpha}' \in \langle 0,1 \rangle^n$ jehož prvky jsou definovány následovně:

$$\alpha_i' = \begin{cases} 1-\alpha_i & \text{jestliže } random < P \\ \alpha_i & (\text{jinak}) \end{cases} \tag{19}$$

pro *i=1,2,...,n,* kde *P* je pravděpodobnost překlopení jednoho bitu a *random* je náhodné číslo s rovnoměrným rozložením z intervalu $\langle 0,1 \rangle$. Jinými slovy, mutace z $\boldsymbol{\alpha}$ do $\boldsymbol{\alpha}'$ je sekvenční operace, která změní na základě pravděpodobnosti *P* stochasticky každý bit. Mutace, pak může být považována za funkci O_{mut}:

$$\boldsymbol{\alpha}' = O_{mut}(\boldsymbol{\alpha}; P), \tag{20}$$

kde pravděpodobnost P je v pozici parametru této funkce. Sousedství $N(\boldsymbol{\alpha},P)$ (předepsané pravděpodobností P) vektoru $\boldsymbol{\alpha}$, se skládá z n-bitových vektorů, které jsou vytvořeny mutací vektoru $\boldsymbol{\alpha}$:

$$N(\boldsymbol{\alpha};P) = \boldsymbol{\alpha}' = O_{mut}(\boldsymbol{\alpha};P). \tag{21}$$

Jednoduchá forma adaptivního horolezeckého algoritmu může být implementována následujícím způsobem. Na začátku je náhodně vygenerován vektor $\boldsymbol{\alpha}$ a pravděpodobnost P je nastavena na nějakou maximální hodnotu P_{max} (např. 0.1). Dalším krokem je vytvoření sousedství $N(\boldsymbol{\alpha},P)$. Pomocí lokálního prohledávání je v tomto sousedství nalezeno nejlepší řešení a to je využito v následujícím kroku jako nový vektor $\boldsymbol{\alpha}$ pro konstrukci nového sousedství. Toto nové sousedství použije o něco nižší hodnotu pravděpodobnosti P. Celý proces je opakován dokud hodnota pravděpodobnosti neklesne pod předdefinovanou hodnotu P_{min}. Poslední hodnota vektoru $\boldsymbol{\alpha}$ je potom výstupem a vyjadřuje suboptimum řešení optimalizační úlohy (18).

Zajímavým rozšířením tohoto obecného konceptu je pak použití pravděpodobnostního vektoru $\boldsymbol{w}=(w_1, w_2, \ldots, w_n)$, jehož prvky jsou v rozmezí $0 \leq w_i \leq 1$ a jež stanovují pravděpodobnost výskytu hodnot 1 na daných pozicích. Pokud $w_i = 0$, pak $\alpha_i=0$. Pro každý prvek w_i je odpovídající hodnota prvku α_i stanovena tímto způsobem:

$$\alpha_i = \begin{cases} 1 & pokud\ r < w_i \\ 0 & jinak \end{cases} \tag{22}$$

Hodnota r je náhodně vygenerované číslo v rozmezí $0 \leq r \leq 1$. Takovou tvorbu n-bitových vektorů s ohledem na pravděpodobnostní vektor $\boldsymbol{w}$ lze vyjádřit jako funkci R

$$\boldsymbol{\alpha} = R(\mathbf{w}) \tag{23}$$

Sousedství $N(\boldsymbol{w})$ složené z náhodně vygenerovaných n-bitových vektorů s ohledem na pravděpodobnostní vektor $\boldsymbol{w}$ je vyjádřeno následujícím vzorcem:

$$N(\boldsymbol{w}) = \boldsymbol{\alpha} = R(\boldsymbol{w}) \tag{24}$$

Pokud jsou prvky vektoru $\boldsymbol{w}$ lehce nad hodnotou 0 nebo těsně pod hodnotou 1, pak je velikost sousedství $N(\boldsymbol{w})$ velmi malá. V porovnání s předcházející verzí horolezeckého algoritmu tato situace odpovídá tomu, kdy je pravděpodobnost mutace P velmi malé číslo. Na

druhou stranu, jestliže se pravděpodobnost w_i blíží hodnotě 0.5, pak vektory konstruované podle (22) a (23) vytvoří relativně rozlehlou oblast se deterministicky vytvořeným středem takovým, že $\alpha_i = 0$ (jestliže $w_i < 0.5$) a $\alpha_i = 1$ (jestliže $w_i > 0.5$).

Tento koncept je možné zavést do klasického horolezeckého algoritmu tak, že v každém novém řešení, které je generováno v průběhu algoritmu, je zohledněn i vliv pravděpodobnostního vektoru $\boldsymbol{w}$ (23).

V [KVAS96] je na příkladu testovací funkce porovnána tato metoda s GA. Z tab. 2 je patrné, že na této testovací funkci pro vyšší počet proměnných dává HCwL lepší výsledky než GA. Tyto výsledky však neznamenají, že by HCwL obecně významně překonával algoritmus GA. To se stalo pouze v tomto případě, kdy byla nastavena velikost populace na hodnotu 1000. Jelikož má testovací funkce záludný průběh pro vyšší hodnoty proměnných, je vhodné i zvýšit velikost populace s rostoucí hodnotou proměnných [KVAS96].

Tabulka 2 - Správné výsledky po 10 opakováních (v procentech) – převzato z [KVAS96]

p	HCwL	GA
1	100%	100%
2	100%	100%
3	100%	80%
4	100%	80%
5	100%	60%
6	90%	50%

Možností jak upravit horolezecký algoritmus je mnoho. Nevýhodou základního algoritmu je možnost uváznutí v lokálním optimu. Další jeho slabostí je neexistence informace o tom, zda se po ukončení běhu algoritmus dostal do globálního nebo pouze do lokálního optima. Samotné optimum, kterého bylo dosaženo, závisí na počáteční inicializaci a obecně není možné stanovit nějakou horní hranici pro výpočetní čas algoritmu. Na druhou stranu je velice jednoduché aplikovat tento algoritmus na celou řadu problémů.

3.3.2 Nastavitelné prohledávání sousedství

Nastavitelné prohledávání sousedství (Variable Neighbourhood Search - VNS) [MLAD97] je metoda, která je schopna do procesu lokálního prohledávání zahrnout několik sousedství, které mohou měnit svou velikost. Začne se v nějakém počátečním bodu (obvykle vybraném pomocí náhody). Poté si algoritmus náhodně vybírá různá sousední okolí z nejmenších sousedství a na ně aplikuje lokální prohledávání do té doby, než je získáno lokální minimum. Pokud není nalezené řešení vylepšeno, algoritmus navštíví další větší

sousedství a prohledá jej tím samým způsobem. Jakmile je získáno zlepšující řešení, je toto zapamatováno, avšak v dalších krocích jsou znovu prozkoumávána nějaká větší sousedství. Toto se děje do doby než je prozkoumána nějaká maximální velikost různých sousedství. Následuje pseudokód pro tento algoritmus – podle [MLAD97].

```
VariableNeighbourhoodSearch(N1(),N2(),..,Nkmax())

     begin
     x = Generování_Počátečního_Řešení()
     k = 1
     do

          y = náhodné řešení vybrané z Nk(x)
          z = Lokální_Prohledávání(y)

          if (f(z) < f(x))
               x = z
               k = 1 //zlepšení řešení použij nejmenší N(x)
          else if (k < kmax) //žádné zlepšení řešení nebylo
                               nalezeno
               k = k + 1 //použij další největší sousedství

     while (not ukončovací podmínka)
     end
```

Metoda VNS byla aplikována na celou řadu problémů a to ať kombinatorických tak i pro problémy spojité optimalizace. První aplikace VNS byly také aplikovány na problémy kvadratického programování [MLAD03], kdy při samotném výpočtu se jednotlivá sousedství tvoří jako neustále se zvětšující n-rozměrné rovnoběžnostěny. Jiné implementace byly prováděny na alokační problémy. Zde je sousedství definováno podle jednotlivých řešených problémů (např. přiřazení materiálu jednotlivým zákazníkům, umístění ještě nepřipsaného materiálu zákazníkovi apod.).

V literatuře jsou předvedeny výsledky této metody, které dávají dobré výsledky i v porovnání s jinými metodami.

3.3.3 Metoda zakázaného prohledávání

Metoda zakázaného prohledávání (tabu search - TS) [AMBE00], [BATT93], [BATT95], [GLOV89], [GLOV90], [GLOV93] vychází z horolezeckého algoritmu, kde se snaží odstranit problém zacyklení. Do horolezeckého algoritmu je zavedená tzv. krátkodobá paměť, která si po určitý krátký interval předcházející historie algoritmu pamatuje inverzní

transformace k lokálně optimálním transformacím řešení, použitým k získání nových aktuálních řešení. Tyto inverzní transformace jsou pak zakázány (*tabu*) při tvorbě nového okolí pro dané aktuální řešení a tudíž je zamezeno využití již jednou navštíveného řešení. Tímto jednoduchým způsobem je možné podstatně omezit výskyt zacyklení při pádu do lokálního minima. Takto modifikovaný horolezecký algoritmus postupně prohledává oblast, ve které je hledáno globální minimum funkce.

Hlavní myšlenka heuristiky je realizována pomocí zakázaného seznamu *TL* (*tabu list*), který dočasně obsahuje inverzní transformace k transformacím použitým v předcházejících iteracích. Zakázaný seznam transformací *TL* je omezen svou maximální velikostí *kmax*. Zakázaný seznam *TL* je obnovován v průběhu chodu celého algoritmu. Jestliže transformace *t* patří do zakázaného seznamu, pak se nemůže používat v lokální minimalizaci v rámci okolí aktuálního řešení. Při inicializaci algoritmu je zakázaný seznam prázdný, po každé iteraci se do zakázaného seznamu přidá transformace inverzní k transformaci, která poskytla nejlepší řešení v rámci sousedství v předcházejícím kroku. (např. se do zakázaného seznamu zapíší pořadová čísla vyměněných úloh). Zakázaný seznam musí být kratší, než je počet možných transformací v rámci sousedství. Po *kmax* iteracích zakázaný seznam už obsahuje *kmax* transformací a každé další přidání nové transformace je doprovázeno vyloučením momentálně nejstarší transformace $t^{\wedge} v$ tabu seznamu. Říkáme, že zakázaný seznam se cyklicky obnovuje.

Základní verze metody zakázaného prohledávání se řídí následujícím předpisem: – podle [GLOV97]

```
Tabu search
     begin
     x = náhodně vygenerované řešení
     time = 0, f_min = ∞, T = {}

     while (time < time_max) do
     begin time = time+1, f_loc-min = ∞
          for t∈X do
          begin x' = x
               if f(x') < f_loc-min and (t∉TL or f(x') < f_min) then
                    x* =x', t* = t, f_loc-min = f(x')
          end
          if f_loc-min<f_min then
```

```
            f_min = f_loc-min, x_min = x*
        x = x*
        if |TL| < kmax then
            TL = TL ∪{t*}
        else TL = (TL ∪{t*})\{t^}
end
```

Numerické zkušenosti s algoritmem zakázaného prohledávání ukazují, že velikost *kmax* zakázaného seznamu je důležitým parametrem ovlivňujícím možnost vymanit se z lokálních minim. Jestliže je parametr *kmax* příliš malý, pak se může vyskytnout zacyklení algoritmu, stejně jako u klasického horolezeckého algoritmu, ale například po více krocích. Naopak je-li parametr *kmax* příliš velký, potom s velkou pravděpodobností přeskočíme nadějná hluboká údolí funkce.

Zakázaný seznam se používá ke konstrukci modifikovaného sousedství aktuálního řešení. Toto sousedství neobsahuje řešení, která by vznikla transformacemi obsaženými v zakázaném seznamu. Lokální minimalizace se vykonává v modifikovaném sousedství s výjimkou tzv. aspiračního kritéria (AK). Toto kritérium dovoluje použití zakázané transformace v případě, že vzniklé sousední řešení by poskytlo lepší hodnotu účelové funkce, než má dočasné nejlepší řešení. Toto aspirační kritérium bude jedním z nosných témat této disertační práce a bude součástí autorem vytvořeného a zkoumaného hybridního algoritmu (viz paragraf 4.7)

Jiným přístupem, který je založený na koncepci dlouhodobé paměti, využívá prostředky intenzifikace a diverzifikace algoritmu TS k nalezení globálního optima. Využívá možnosti pokutování transformací, které se v dosavadním procesu hledání vyskytovaly nejčastěji. Případně do dlouhodobé paměti zaznamenáváme frekvenci výskytu určitých atributů dosud získaných řešení. Intenzifikační strategie podporuje transformace vedoucí k řešením s „dobrými“ vlastnostmi. Naproti tomu diverzifikační strategie znamená podporu řešení, obsahujících významně odlišné atributy od těch, které se vyskytly v průběhu dosavadního hledání.

Technika zakázaného prohledávání může být použita jak v klasickém spojení s horolezeckým algoritmem, tak i v kombinaci s jinými algoritmy. U ostatních metod však není natolik efektivní, neboť tyto metody neprohledávají celé sousedství aktuálního řešení a pravděpodobnost zapůsobení zakázaného seznamu je tedy dost malá.

3.3.4 Robustní zakázané prohledávání

Robustní zakázané prohledávání (Robust Tabu Search - RTS) [TAIL91], [TAIL95] je rozšířenou verzí základního zakázaného prohledávacího schématu, kdy se využívá náhodné délky tabu seznamu a určité délky podmínkové paměti a také paralelizace výpočtu. RTS bylo s úspěchem aplikováno na problém typu QAP [TAIL91]. Pokud je nastaven např. počet iterací na N^2 a maximální délka tabu seznamu kolísá okolo hodnoty 10% velikosti celé řešené úlohy, je možné dosáhnout velice kvalitních výsledků. Toto vše závisí spíše na typu problému než na jeho velikosti (pro $N \geq 20$ [TAIL91]). Podmínková paměť nutí vracet řešení zpět do takového řešení, kde nebyl proveden jistý počet přesunů. Toto se provádí tak dlouho, dokud nějaký přesun neukáže žádný atribut z tabu seznamu nebo do té doby, kdy je aplikováno nejvíce zlepšující aspirační kritérium.

Implementace této metody snižuje výpočetní složitost metody TS pro QAP a to díky rozdělení úlohy na menší části a následné paralelizace výpočtu. V literatuře je dokázané, že metoda TS pracuje pro QAP problémy do velikosti 30 velmi dobře za použití jednoho tabu seznamu a jednoduchého aspiračního kritéria. Pro větší problémy je pak vhodná metoda RTS, kdy je navržena lepší aspirační funkce a kdy se přidávají další parametry pro rychlejší výpočty. Toto je pak doporučeno pro velikost QAP okolo 64 jednotek. Pro větší problémy je pak doporučeno používat např. sofistikovanější dlouhodobou paměť.

3.3.5 Reagující zakázané prohledávání

Reagující zakázané prohledávání (Reactive Tabu Search – ReTS) je dalším významným rozšířením metody zakázaného prohledávání. Toto schéma je trochu komplikovanější a není až tak důležité vysvětlovat jeho činnost v této práci. Hlavní myšlenka spočívá v rostoucí velikosti tabu seznamu v případě, že algoritmus navštíví větší množství řešení znovu navštíveno a naopak se seznam zmenšuje v případě, že je množství znovu navštívených řešení menšího počtu. Algoritmus si tedy udržuje určitou velikost seznamu, která se hodí na konkrétní problém a na prohledávaný prostor. Druhou důležitou věcí, kterou algoritmus používá, je jistá sekvence náhodných přesunů v případě, kdy se algoritmus ocitne v prostoru hledání, ze kterého se z nějakého důvodu nemůže dostat. Tato část ReTS vychází z myšlenky iterativního lokálního prohledávání, kde se používá podobná metody úniku z lokálního minima.

Výhodou metody je využití flexibilních paměťových struktur typu strom v prohledávacím procesu. V procesu je zavedena hašovací technika (hashing), kdy je ke

každému záznamu veden i klíč, který je zajištěn v procesu ukládání nějakého dosaženého výsledku. Další technikou, která je zavedena v procesu prohledávání, je datová struktura známá jako binární strom, což je orientovaný graf s jedním kořenem. Z tohoto kořene pak existuje cesta do všech vrcholů grafu, přičemž každý vrchol může mít maximálně dva orientované potomky. Obě tyto techniky jsou využity pro rozdělení úlohy a následné zpracování jednotlivých vygenerovaných řešení. ReTS nepotřebuje pro svoji činnost mít a priori nastavenou velikost seznamu pro ukládání zakázaných přesunů, což svým způsobem urychluje práci.

Podrobněji je o této problematice pojednáno v [BATT93], [BATT95].

3.3.6 Rychlé lokální prohledávání

Rychlé lokální prohledávání (Fast Local Search - FLS) [LOAP05] je zobecněním dřívějšího schématu, které bylo vytvořeno pro zrychlení algoritmu známého jako první zlepšení (first improvement). Obě tyto heuristiky byly s úspěchem využity na problémy obchodního cestujícího, problémy částečného omezení nebo problémy typu QAP.

Jedním z faktorů, který velmi ovlivňuje výkonnost algoritmu lokálního prohledávání je velikost prohledávaného okolí aktuálního řešení. Pokud se bere v úvahu příliš velký počet okolních bodů, které je potřeba prohledat, pak samotné prohledávání může být velice nákladné. Toto platí zejména v případech, kdy prohledávání potřebuje velký počet kroků k nalezení lokálního optima a každý výpočet účelové funkce vyžaduje významný počet výpočtů. Bentley [BENT97] představil přibližnou 2-Opt metodu (*approximate 2-Opt*), která omezuje velikost sousedství v problému obchodního cestujícího. Jednoduchým principem této metody je ignorovat taková okolí zkoumaného řešení, jejichž zkoumání nevede ke zlepšení prohledávání konkrétního prostoru.

Okolí, které se algoritmus rozhodne prohledat je rozděleno do několika menších prostorů a každému z toho prostoru je přiřazena aktivační jednotka (*bit*). Prohledávají se pouze ty prostory, jejichž aktivační bit je nastaven na hodnotu 1. Tyto prostory pak nazýváme aktivními prostory. Prostory, jejichž bit je nastaven na hodnotu 0, nazýváme neaktivní prostory a tyto pak jsou algoritmem ignorovány, tzn., že nejsou zahrnuty do prohledávání. Proces prohledávání pokračuje do té doby, než je naplněna podmínka ukončení procesu, nikoliv do doby, kdy je nalezeno lepší řešení. Proces probíhá v určeném pořadí. Toto pořadí může být statické nebo dynamické (např. se mění podle výsledků provedených v jednotlivých přesunech).

Na počátku prohledávání jsou všechna rozdělená okolí nastavena jako aktivní. Jestliže je tento prostor prohledán a je zjištěno, že neobsahuje žádné zlepšení nebo neobsahuje žádný zlepšující přesun, stane se neaktivním. V opačném případě zůstane aktivní a je proveden zlepšující přesun (tedy přesun do bodu, který vykazuje zlepšení). Podle provedených přesunů jsou pak aktivovány i příslušné prostory prohledávání. Jak probíhá celý prohledávací proces, a jsou nacházena lepší řešení, zmenšuje se i počet prostorů, které jsou aktivní. Toto se děje do doby, kdy již nejsou žádné prostory aktivní, tedy hodnota jejich bitu je nastavena na nulu. Řešení se poté ocitá alespoň v přibližném lokálním minimu. Tato celková procedura může být mnohokrát rychlejší než konvenční algoritmy lokálního prohledávání.

3.4 Algoritmy založené na penalizacích a váhách

3.4.1 GENET a jiné váhové metody

Tato metoda modifikuje hodnoty argumentů účelové funkce pomocí vah tak, aby algoritmus umožnil uniknout z lokálního optima. Technika rozšiřuje obecné metody optimalizační problematiky o tzv. uspokojení daných omezení [BISH95], [BOSE96], [DAVE94], [TANG92].

V posledních letech byly vyvinuty mnohé algoritmy, které jsou právě na bázi tohoto schématu a lze je aplikovat na CSP nebo na problémy typu SAT. Jsou mezi nimi metoda GENET [DAVE94], [MINT92], vážený GSAT problém [FRAN96], [SELM93a], [SELM93b] a také útěková metoda (Breakout Method) [MORR93].

GENET je jedním z přístupů jak splnit omezení daného problému pomocí základních operací, které se podobají tzv. minimalizačně konfliktní heuristice. V zásadě se jedná o to, že CSP je reprezentována sítí, kde každý uzel představuje možné přiřazení hodnoty k proměnné, a okraje představují omezení problému. Jednou z inovací, která se vyskytuje v této metodě, je využití zpracování váhových hodnot, které se přiřadí jednotlivým okrajům hledaného prostoru (constraint). Váhy se přiřazují těm omezením, které byly porušeny (tyto hodnoty mohou být jednoduše nastaveny na 1 pro omezení porušené a 0 pro uspokojené omezení, pokud jsou však používány takové podmínky, kdy je třeba měnit hodnotu více proměnných, pak lze použít funkci, která postupně snižuje hodnotu vah, tak jak se omezení blíží stavu, kdy bude uspokojeno). GENET využívá tuto funkci tak, jak se pokouší minimalizovat sumu vah této funkce pro všechna omezení v problému.

3.4.2 Diskrétní Lagrangeův Multiplikátor

Diskrétní Lagrangeův multiplikátor (Discrete Langrangian Multiplier - DLM) [BERT82] vychází z modifikace matematické teorie spojité optimalizace. DLM spojuje Lagrangeův multiplikátor se schématem zvyšováním hodnoty omezení problému vždy, když algoritmus dosáhne lokálního optima. Tento postup je téměř totožný s algoritmem GENET; ve skutečnosti GENET byl ukázán jako jeden z algoritmů z třídy Lagrangeovských prohledávacích problematik, což bylo prokázáno v [CHOI88]. Teorie je potom taková, že algoritmus zvyšuje Lagrangeův multiplikátor (vykonáváním výstupu v Lagrangeově multiplikátorovém prostoru) do té doby, kdy je účelová funkce minimalizována. Tato problematika je více popsána např. v [CHOI88] nebo v [JANÁ99].

4. Penalizační lokální prohledávání rozšířené o aspirační kritérium

Hlavním cílem této práce tkví v návrhu hybridního[4] algoritmu vytvořeného za využití některých metod z oblasti umělé inteligence pro problémy typu obchodního cestujícího a pro další optimalizační úlohy. Pro řešení těchto úloh bylo zjištěno, že dobrých výsledků se dosahuje u algoritmů SA a TS a z jejich některých částí byl vytvořen algoritmus penalizačního lokálního prohledávání (PLP). Tento algoritmus se stal jádrem disertační doktorské práce. PLP bude následně rozšířen o aspirační kritérium, které je součástí TS (viz odstavec. 3.3.3) a jež je vhodným rozšířením algoritmu PLP.

4.1 Vznik algoritmu

Částečné CSP jsou takové problémy, kde neexistuje takové řešení, které by vyhovovalo všem omezením. Při řešení těchto úloh je zájem o minimalizaci počtu omezení, která jsou narušena či o minimalizaci dalších kritérií, která jsou na aplikaci závislá.

Jedním z problémů, který je i částečným CSP, se nazývá Radio Link Frequency Assignment Problem (RLFAP) [TSAN93], [WANG91a]. Tento problém řeší problematiku udělování frekvencí různým stanicím (rádia, vysílače atp.), tak aby nedocházelo k rušení signálu mezi těmito stanicemi. Každá taková stanice je reprezentována proměnnou, jejíž doména je množina všech frekvencí, které má daná stanice k dispozici. Základním omezením zahrnujícím dvě proměnné F_1 a F_2 je:

$$\left| F_1 - F_2 \right| > k_{12}. \qquad (25)$$

Tyto dvě proměnné představují dvě frekvence, které jsou od sebe vzdáleny dostatečně daleko. Hodnota konstanty k_{12} je odvislá od pozice těchto dvou spojení a také závisí na různém fyzikálním prostředí, ve kterém se dané spoje nachází. Tato konstanta nemusí být vždy nutně nastavena na správnou hodnotu (ve skutečnosti minimální rozdíl ve frekvenci těchto dvou bodů může být odlišný od hodnoty konstanty k_{12}). Proto se tato hodnota často nadhodnocuje, aby bylo dostatečně garantováno, že nedojde k rušení signálu. Problém RLFAP je typu NP-úplný [GARE79] a také částečným CSP a vyžaduje minimalizaci porušených omezení, která je kombinovaná s doménovými optimalizačními kritérii.

[4] Hybridní v této práci bude myšleno jako vzniklého kombinací vybraných vlastností různých algoritmů.

V literatuře se objevuje možnost řešení tohoto problému pomocí rozšířené metody GENET, jehož použití je pro problém typu RLFAP příliš složité. GENET byl částečnou inspirací pro PLP s tím, že nepoužívá rozšířený model neuronové sítě GENETu.

Další metodou, ze které bylo čerpáno pro tvorbu PLP je metoda SA (viz odst. 3.1.1), a to především ta část týkající se měnící hodnoty teploty během průběhu prohledávání prostoru. Teplota je postupně ochlazována a samotná hodnota teploty má vliv na to, zda bude nové řešení přijato či nikoliv podle (11). Bude vysvětleno dále.

Třetí a poslední metodou, jež byla inspirací pro tvorbu PLP je metoda TS (viz odst. 3.3.3). Z této metody bylo vybráno aspirační kritérium, které vymezuje, zda se má algoritmus vyhnout aktuálně nalezenému řešení, či právě naopak pokračovat v dalším prohledávání.

Všechny tyto tři metody mají velký potenciál při řešení rozličných optimalizačních úloh a zdá se být patrné, že spojením vybraných částí může vzniknout funkční a efektivní hybridní algoritmus využitelný v celé řadě optimalizačních problémů. V následující části bude rozebrán samotný princip algoritmu a jeho dílčí modifikace.

4.2 Principy algoritmu

PLP je optimalizační technika využitelná pro kombinatorické optimalizační problémy. Účelová funkce konkrétního problému je navýšena pomocí nově vytvořené množiny pokut[5]. Lokální vyhledávání je omezováno touto množinou pokut a během hledání optimálního (resp. suboptimálního) řešení je pak možné se soustředit na slibné oblasti prohledávaného prostoru. Algoritmus se provádí opakovaně a prohledává lokální okolí určeného bodu. Jakmile se algoritmus zastaví v lokálním minimu, množina pokut, jež je na začátku prázdná (hodnoty penalizací jsou nastaveny na nulu), je určitým způsobem modifikována a algoritmus dále optimalizuje upravenou účelovou funkci.

Jak se penalizační faktory dále mění, jsou jednotlivá nalezená řešení upřesňována a postupně jsou lokalizována nová lokální minima. Lokální vyhledávání je pak generováno podle znalostí, jež jsou dány buď na počátku prohledávání, nebo se postupně objevují v průběhu prohledávání. Tyto znalosti se týkají především umístění a hodnoty lokálních minim a především se týkají hodnoty a umístění globálního minima. Chování tohoto

[5] Pokutou v našem případě budeme rozumět hodnotu, která bude přiřazena k jistému argumentu účelové funkce a ovlivní tak další prohledávání v lokálním prostoru. Pokuta může být také nazývána penalizací nebo penalizačním faktorem.

algoritmu může být postaveno na podobném principu jako je metoda špatně postavené úlohy (ill-posed) [HANS98], [TIKH77], kdy se využívají znalosti předcházející řešení a slouží k vyřešení aproximačních problémů. Tyto konkrétní znalosti pak představují určitá omezení, která dále definují problém, tak že se redukuje počet všech možných kandidátů na vyřešení daného problému. Algoritmus také využívá dalších znalostí, které se naučí během prohledávání a to využitím různých omezení, jež se objeví na základě prohledávání daného prostoru. V podstatě se dá říci, že PLP je založeno na meta-heuristice lokálního prohledávání. V následujících kapitolách budou rozebrány jednotlivé komponenty daného algoritmu, kterými jsou lokální prohledávání, penalizace účelové funkce a aspirační kritérium.

4.3 Lokální prohledávání

Lokální prohledávání je základem mnoha heuristických metod pro kombinatorické optimalizační problémy. V kap. 1.6 byl představen princip metody lokálního prohledávání. V této práci bylo využito mnoho procedur, které využívají lokálního prohledávání. Pro účely popisu navrženého algoritmu v obecné rovině lze lokální prohledávání definovat jako:

$$\boldsymbol{x_2} \leftarrow \textit{procedura LokálníProhledávání (}\boldsymbol{x_1}\textit{, f),} \qquad (26)$$

kde $\boldsymbol{x_1}$ *je* původní řešení a $\boldsymbol{x_2}$ je konečné řešení (lokální minimum) a f je účelová funkce, která má být minimalizována.

Na rozdíl od ostatních obecných meta heuristických metod jako je například SA nebo TS, tento algoritmus neupravuje vnitřní mechanismy lokálního prohledávání. Místo toho provádí opakovaná volání v proceduře lokálního prohledávání a upravuje hodnoty argumentů upravené účelové funkce mezi jednotlivými kroky výpočtu, které po sobě následují. Před každým opakovaným voláním funkce jsou hodnoty argumentů účelové funkce problému navýšeny tak, že je zahrnuta množina penalizačních podmínek, které nám umožní dynamicky vytvořit kvalitnější řešení. Toto navýšení pomocí penalizačních podmínek je vysvětleno v následující části.

4.4 Charakteristiky řešení

Argumenty účelové funkce chápeme jako charakteristiky (vlastnosti) daného problému a podle těchto vlastností se dají dále určit nástroje nebo techniky, jejichž pomocí lze problém vyřešit.

Omezení na těchto charakteristikách určují průběh lokálního prohledávání. Znalosti, které se vztahují k danému problému, lze nazvat ohodnocení charakteristik. Tato ohodnocení mají přímý i nepřímý vliv na odpovídající vlastnosti a ohodnocení daného řešení. Ohodnocení charakteristik pak může být buď konstantní, nebo měnitelné. Znalosti o průběhu prohledávacího procesu lze získat z dosud navštívených řešení a z konkrétních lokálních minim. Znamená to tedy, že pokud se konkrétní ohodnocení argumentu nachází v aktuálním zkoumaném řešení, lze jej ohodnotit a tudíž dále využít pro další prohledávání. Konkrétní hodnota argumentu U_i je pak reprezentována následujícím vzorcem

$$U_i(\boldsymbol{x}) = \begin{cases} 1, & \text{řešení } s \text{ má vlastnost } i \\ 0, & \text{jinak} \end{cases}, \boldsymbol{x} \in X . \qquad (27)$$

4.5 Penalizace účelové funkce

Omezení na argumentech funkce je možno vytvořit zvýšením účelové funkce *f* problému a to započítáním množinou penalizačních faktorů. Nová účelová funkce je nazvána zvýšenou účelovou funkcí a je definována následovně

$$g(\boldsymbol{x}) = f(\boldsymbol{x}) + \lambda \sum_{i=1}^{M} p_i \, U_i(\boldsymbol{x}), \qquad (28)$$

kde M je počet argumentů stanovených před samotným řešením problému, p_i je penalizační parametr odpovídající hodnotě argumentu U_i a λ je regulující parametr dané funkce. Penalizační parametr p_i předává hodnoty do těch ohodnocení argumentů účelové funkce U_i, které jsou zahrnuty v aktuálním řešení. Regulační parametr λ reprezentuje poměrovou část penalizací a ovlivňuje výslednou velikost hodnoty účelové funkce. Jeho význam tkví v možnosti kontroly vlivu znalostí na proces prohledávání prostoru. Algoritmus opakovaně používá lokálního prohledávání a jednoduše modifikuje penalizační vektor ***p***:

$$\boldsymbol{p} = (p_1, p_2, ..., p_M). \tag{29}$$

Vektor se změní vždy, když se prohledávání usadí v lokálním minimu. Zpočátku jsou všechny penalizační parametry nastaveny na hodnotu 0 (žádný argument funkce není omezením) a algoritmus je spuštěn, aby nalezl lokální minimum účelové funkce. Po nalezení prvního lokálního minima (i po nalezení dalších lokálních minim) provede algoritmus úpravu penalizačního vektoru a současně i upraví účelovou funkci a ta je změněna na zvýšenou účelovou funkci. Upravený postup prohledávání pak jednoduše zvýší hodnotu penalizačního faktoru o hodnotu 1. Tento postup je proveden buď pro jeden, nebo pro více argumentů účelové funkce, jež se nachází v lokálním minimu. Počáteční znalosti jsou pak postupně vkládány do rozšířené účelové funkce a to postupným výběrem těch penalizačních parametrů, které mají být zvýšeny.

Zdrojem znalostí je pak hodnota koeficientů účelové funkce a samotné lokální minimum. Předpokladem je, že každý koeficient účelové funkce definovaný na řešení je určený konkrétní hodnotou c_i[6]. Tato hodnota může být v průběhu prohledávání konstantní nebo proměnlivá. Abychom mohli zjednodušit naši analýzu, předpokládejme, že tento koeficient účelové funkce je konstantní a je dán vektorem

$$\boldsymbol{c} = (c_1, c_2, ..., c_M), \tag{30}$$

který obsahuje kladné nebo nulové hodnoty. Hodnota indikátoru (27) je pak nastavena na 1 ($U_i(\boldsymbol{x}^*) = 1$) pro to řešení, které se nachází v lokálním minimu.

4.6 Modifikace penalizačních faktorů

V lokálním minimu $\boldsymbol{x}^*$ jsou penalizační parametry zvýšeny o hodnotu 1 pro všechny argumenty U_i takové, které maximalizují následující výraz:

$$\text{užitek}(\boldsymbol{x}^*, f) = U_i(\boldsymbol{x}^*) \frac{c_i}{1 + p_i}. \tag{31}$$

[6] Někdy se také uvádí cena řešení, resp. cena argumentu účelové funkce.

Jinými slovy, zvýšení penalizačního parametru argumentu x_i je vyjádřeno postupem daným rovnicí (31). V lokálním minimu je následně vybrán ten postup, který má maximální užitek (maximální hodnota) a ten je i vykonán. Penalizační parametr p_i je začleněn ve vztahu (31) z důvodu zamezení zkreslování funkce užitku v momentě, kdy jsou penalizační faktory neúměrně zvyšovány. Úloha penalizačního faktoru v (31) je jakési počítadlo, které počítá kolikrát je argument funkce penalizován. Pokud je konkrétní argument účelové funkce penalizován častěji než je počet iterací, pak výraz $\frac{c_i}{1+p_i}$ v (31) zvýší hodnotu argumentu. Poté následuje možnost penalizovat další hodnotu argumentu funkce. Politika penalizace spočívá v přičtení nějaké hodnoty k hodnotě argumentu účelové funkce, tzn. že argumenty, které způsobují zvýšení účelové funkce jsou penalizovány častěji než ty, jež tak nečiní; důvodem je snaha o únik z lokálního minima. Úsilí o vyhledání nejmenší hodnoty účelové funkce je použito pouze v argumentech účelové funkce právě navštívených lokálních minim, a proto jsou penalizovány pouze hodnoty lokálních minim.

Základ algoritmu je popsán níže:

```
begin
     k = 0
     x0 = náhodně vygenerované řešení prostoru
     for i = 1 until M do // penalizace jsou nastaveny na 0
          pi = 0
     while Ukončující Kritérium do
     begin
          g = f + λ*Σpi*Ui
          xk+1 = LokálníProhledání(xk,g)
          for i = 1 until M do
               užitekᵢ = Ui(xk+1) * ci/ (1+pi)
          for each i takové že užitekᵢ je maximum do
               pi = pi + 1
          k = k+1
     end
     x* = nejlepší řešení nalezené vzhledem k účelové funkci f
     return x*
end
```

kde X je prohledávaný prostor, f je účelová funkce, g je změněná účelová funkce, λ je upravující parametr, U_i je funkce pro argument i, c_i je cenou pro argument i, M je počet argumentů dané funkce, p_i je penalizace pro argument i.

Tento jednoduchý princip může být aplikován s různými modifikacemi na mnoho optimalizačních problémů. Jeho aplikace na konkrétní problém obvykle vyžaduje definování argumentů účelové funkce, přiřazení hodnot k těmto argumentům a hlavně dosazení procedury (*LokálníProhledání*) do výše vyjádřeného algoritmu. Tato procedura je jednou z často používaných heuristických technik.

Podobný princip, který je založen na dalším ovlivňování průběhu hledání optima na základě předchozích znalostí, lze nalézt ve třídě metod tzv. teorie optimálního vyhledávání (*optimal search theory)* [KOOP57], [STON83]. Obdobná schémata využitá v algoritmu se používají i při bádání v metodách známých jako *reinforcement learning* [THRU92], [WANG91b].

4.7 Aspirační kritérium pro PLP

Důležitým prvkem, který bude zkoumán v této práci a zároveň bude i přidán do algoritmu PLP, je prvek aspiračního kritéria. Aspirační kritéria se běžně využívají u algoritmu zakázaného prohledávání [BADE97], [GASS04], [GLOV93]. V této kapitole bude popsán způsob a motivace přidání aspiračního kritéria do algoritmu a bude ukázáno, jak tento algoritmus může být rozšířen právě o tento prvek.

4.7.1 Důvod využití aspiračního kritéria

PLP využívá penalizačních faktorů tak, že v momentě, kdy se prohledávání ocitne v lokálním minimu, je účelová funkce penalizována a to do té doby, než je algoritmu umožněno uniknout z lokálního minima. Následně může algoritmus pokračovat dál v prohledávání. Někdy se může stát, že argumenty účelové funkce jsou penalizovány příliš brzy a později, v průběhu prohledávání se tato penalizace může stát méně významnou (nebo i matoucí). Někdy se také může stát, že se hodnota penalizace může v průběhu algoritmu změnit nebo jindy je penalizační hodnota bariérou při dalším prohledávání prostoru. Což může být částečně kontraproduktivní v momentě, kdy algoritmus neprohledá prostor, kde existuje možnost nalezení nového lepšího řešení. Čím větší bývá hodnota parametru λ, tím častěji se stává, že je řešení blokováno. Ačkoliv nižší hodnota parametru λ zvětšuje šanci, že se dostaneme do lepšího řešení, na druhou stranu redukuje vliv penalizace na algoritmus, tzn., že musí být proveden větší počet iterací, abychom se dostali z lokálního minima. V případě, že by nějaká metoda nemusela zohledňovat vliv hodnoty λ, pak by penalizované lokální

prohledávání mohlo být méně citlivé právě na změny parametru λ. Současně s tím by se mohly lépe řešit problémy, kde se hodnoty argumentů často mění v průběhu prohledávání. Jedním z možných přístupů je využití aspiračního kritéria. Algoritmus pak nemusí prohledávat takové prostory, jejichž zkoumání je vzhledem k jejich charakteru zbytečné. A z tohoto důvodu je vhodné zavést myšlenku aspiračního kritéria.

4.7.2 Definice aspiračního kritéria

Jak již bylo naznačeno, myšlenka aspiračního kritéria pochází z metody zakázaného prohledávání. V této metodě aspirační kritérium povoluje použití zakázané transformace v případě, že vzniklé sousední řešení poskytuje lepší hodnotu účelové funkce, než má dočasné nejlepší řešení a to i v momentě, že je toto řešení obsaženo v seznamu zakázaných transformací (*tabu list)*. Toto kritérium je jedno z nejužívanějších forem aspiračního kritéria a je nazýváno vylepšené nejlepší aspirační kritérium [GLOV97]. Ve skutečnosti existuje mnoho forem tohoto kritéria, jak je uvedeno v [GLOV97], avšak v této práci bude bráno v potaz pouze vylepšené nejlepší aspirační kritérium.

Pro penalizované lokální prohledávání byla pro vyřešení problému využita politika penalizací. Jednou z možností by bylo i využití tabu seznamu jednotlivých řešení nebo tabu seznamu přesunů v prostoru; tomuto problému se však věnuje metoda TS a její další modifikace a tudíž se tímto nebudeme dále zabývat. Zlepšujícím nejlepším aspiračním přesunem v algoritmu je pak takový přesun, který vygeneruje nejlepší nově nalezené řešení a zároveň tento přesun není vybrán lokálním hledáním za využití zvýšené účelové funkce. Nyní následuje pseudokód pro lokální hledání za využití aspiračního kritéria.

```
Lokální_Prohledání_S_Aspiračním_Kritériem (x, f, g, N)
begin
     Do
     If (Přesun_Pomocí_Aspirace (x, f, g, N, z))
       x = z
     else
          begin
            y = y v N(x) takové, že g(y) je minimalizováno
            Δh = g(y) – g(x)
            If (Δg <= 0) then x = y
            If (Δg > 0) then  iterace = iterace + 1
            Else              iterace = 0
          end
     If (f(x) < f(x*)) then x* = x
     While (Δg <= 0) a (iterace < 2)
     Return x
end

Přesun_Pomocí_Aspirace (x, f, g, N, z)
begin
     z = z v N(x) takové, že f(z) je minimalizováno
     if (f(z) < f(x*) a ((g(z) – g(x)) > 0 ))
          return true
     else
          return false
end
```

kde:

- *x, y* a z jsou hodnoty řešení,
- *f()* vrací cenu řešení vzhledem k originální účelové funkci,
- *g()* vrací zvýšenou cenu řešení,
- x^* je nejlepší nalezené řešení v průběhu běhu algoritmu,
- *N(x)* je funkce sousedství, která dá konkrétní řešení tohoto sousedství *x*.

Tento postup může být jednoduše implementován, pokud se uvažuje nějaký přesun v sousedství, který přinese nové nejlepší řešení shodné s účelovou funkcí *f*. Sousední okolí tohoto řešení je poté prozkoumáno za použití zvýšené účelové funkce *g*, a to z toho důvodu, aby se poznalo, zda existuje jiný přesun, který může zredukovat zvýšenou cenou dalšího řešení. Jestliže nejlepší přesun v sousedství přinese nové lepší řešení takové, že cena originální účelové funkce je menší než ta cena, která by byla vybrána s ohledem na zvýšenou účelovou funkci, pak vybereme ten přesun, který vygeneruje nejnižší cenu nového nejlepšího

řešení (což se dá považovat za aspirační přesun). Jinak vybereme takový přesun s nejnižší hodnotou zvýšené účelové funkce.

4.8 Provedené experimenty

Pro určení kvality algoritmu s aspiračním kritériem byl tento algoritmus testován na vybraných testovacích funkcích (viz kap. 5) a na problému obchodního cestujícího (viz kap. 6). Ve všech těchto případech byl porovnán jak základní algoritmu penalizovaného prohledávání bez aspiračního kritéria tak i upraveného algoritmu s aspiračním kritériem. Experimenty byly spouštěny na osobním počítači P4 CPU 2,80 GHz a navržený algoritmus byl implementován v programovacím jazyku Visual Basic.NET.

Ve všech těchto případech byl také zkoumán vliv velikosti parametru λ [PANU06a], [PANU07b] na kvalitu řešení daného problému. Velikost tohoto parametru se měnila a hodnoty parametru jsou vyjádřeny v (32).

$$\lambda = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\} \quad (32)$$

Z celé rodiny testovacích funkcí [HEDA06] byl vybrán vzorek 16 funkcí a to od těch triviálních až po ty, se kterými i mnohé jiné algoritmy mají problémy [ZELI02] a kdy pro jejich komplikovanost trvá delší dobu nalezení optimálního řešení. V průběhu měření byly zaznamenány různé hodnoty, jež vyjadřují kvalitu algoritmu a také vyjadřují, co se děje v průběhu prohledávání. Hodnoty, které byly měřeny pro testovací funkce, jsou popsány v následující kapitole, jakož i detailnější rozbor tohoto problému.

Pro testování algoritmu na TSP byly vybrány některé příklady z knihovny TSPLIB [REIN91], jež obsahuje celou řadu různě velikých problémů. Pro TSP nás zajímaly následující měřené hodnoty:

a) časová náročnost,

b) průměrná odchylka od nejlepšího známého řešení.

Podrobněji je k této tématice referováno v kapitole 6.

5. Přehled testovacích funkcí

Pro testování optimalizačních algoritmů se používají dva rozdílné postupy. V prvním z nich se obvykle vychází z již dříve řešených příkladů, avšak pomocí jiných algoritmů. Výsledky z právě testovaného algoritmu lze pak porovnat s výsledky, které již existují. Druhý způsob spočívá ve využití množiny testovacích funkcí [HEDA06], které v sobě zahrnují různé vlastnosti, jako je nelinearita, různé patologie typu rovina okolo extrému apod. V této kapitole bude představeno několik funkcí, na kterých byly upravené algoritmy testovány.

Tabulka 3 představuje jednotlivé funkce, které byly vybrány z [HEDA06] a [ZELI02] a spolu s nimi je představen předpis jednotlivých funkcí, spolu s hodnotami rozsahu veličiny x_i, která byla na začátku náhodně vygenerována, a v tomto rozsahu se poté dále s algoritmem pracovalo. Abychom se vyhnuli nedorozumění, jsou používány originální názvy funkcí. Vzhledem k tomu, že jsou známy analytické vztahy, je u většiny z nich velmi jednoduché vypočítat hodnotu a pozici globálního extrému.

Tabulka 3 - Předpisy testovacích funkcí vybraných pro zkoumání vlastností algoritmu PLP

Název funkce
1st De Jong
$\sum_{i=1}^{n} x_i^2, \quad -5.12 \le x_i \le 5.12$
3rd De Jong
$\sum_{i=1}^{n} \text{abs}\left(x_i\right), \quad -2 \le x_i \le 2$
Power function
$\sum_{i}^{n} \lvert x_i \rvert^{i+1}, \quad -1 \le x_i \le 1$
Schwefel function
$\sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right), \quad -512 \le x_i \le 512$
Griewangk´s function
$-\prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + \sum_{i=1}^{n} \frac{x_i^2}{4000} + 1, \quad -600 \le x_i \le 600$

Sine envelope sine wave function
$-\sum_{i=1}^{n-1}\left(\frac{\sin^2\left(\sqrt{x_{i+1}^2+x_i^2}-0.5\right)}{0.001\left(x_{i+1}^2+x_i^2+1\right)^2}+0.5\right), \quad -100 \le x_i \le 100$
Pathological function
$\sum_{i=1}^{n}\left(\frac{\sin^2\left(\sqrt{x_{i+1}^2+100x_i^2}-0.5\right)}{0.001\left(x_{i+1}^2-2x_{i+1}x_i+x_i^2\right)^2+1.0}+0.5\right), \quad -100 \le x_i \le 100$
Egg holder function
$\sum_{i=1}^{n}\left(-x_i \sin\sqrt{\text{abs}\left(x_i-\left(x_{i+1}+47\right)\right)}-\left(x_{i+1}+47\right)\sin\left[\sqrt{\text{abs}\left[x_{i+1}+47+\frac{x_i}{2}\right]}\right]\right), \quad -500 \le x_i \le 500$
Ackley´s function
$\sum_{i=1}^{n-1}\left(20+e^{-20}e^{-0.2\sqrt{0.5\left(x_{i+1}^2+x_i^2\right)}}-e^{0.5\left(\cos 2\pi x_{i+1}+\cos 2\pi x_i\right)}\right), \quad -30 \le x_i \le 30$
Moved axis parallel hyper-ellipsoid function
$\sum_{i=1}^{n} 5i * x_i^2, \quad -5.12 \le x_i \le 5.12$
Master cosine wave function
$-\sum_{i=1}^{n-1} e^{-\frac{1}{8}\left(x_{i+1}^2+0.5x_i x_{i+1}+x_i^2\right)}\cos 4\sqrt{x_{i+1}^2+0.5x_i x_{i+1}+x_i^2}, \quad -5 \le x_i \le 5$
Rastrigin function
$a*10\left(\sum_{i=1}^{Dim}\left(x_i^2-10\cos 2\pi x_i\right)\right), \quad -5.12 \le x_i \le 5.12$
Rosenbrock´s saddle function
$\sum_{i=1}^{Dim-1}\left(100\left(x_i^2-x_{i+1}\right)^2+\left(1-x_i\right)^2\right), \quad -2 \le x_i \le 2$
Ackley path function
$-a*e^{-b*\sqrt{\frac{\sum_{i=1}^{n}x^2}{n}}}-e^{\frac{\sum_{i=1}^{n}\cos c*x_i}{n}}+a+e^1,$ $a=20; b=0.2; c=2*\pi; i=1\ldots n, -32.768 \le x_i \le 32.768$

V další části této kapitoly jsou uvedeny výsledky, které byly získány na vybraných testovacích funkcích, které jsou vhodné k otestování robustnosti navrženého algoritmu. U

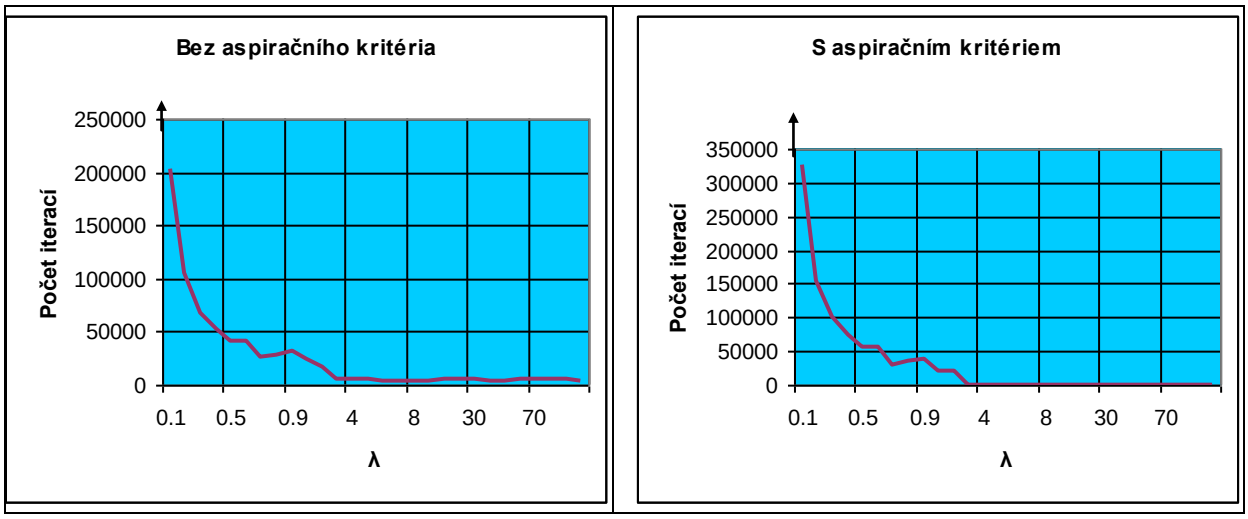
každé funkce bylo v určitém rozmezí, které je obecně známo a používáno, hledán globální extrém a byly sledovány následující ukazatele:

- počet iterací nutných k dosažení nejlepšího nalezeného řešení,
- nejlepší nalezené řešení,
- počet kolikrát se změnilo nejlepší nalezené řešení v průběhu prohledávání,
- celkový čas nutný k nalezení nejlepšího nalezeného řešení.

Bylo celkem spuštěno 100 testů a každá funkce byla definována pro 100D, tedy každá funkce měla 100 argumentů. Naměřené hodnoty pak byly zprůměrovány. V následujících obrázcích je potom u každé funkce znázorněn název funkce a jednotlivé naměřené výsledky. V přílohách A – D je znázorněna vizualizace čtyř vybraných funkcí.

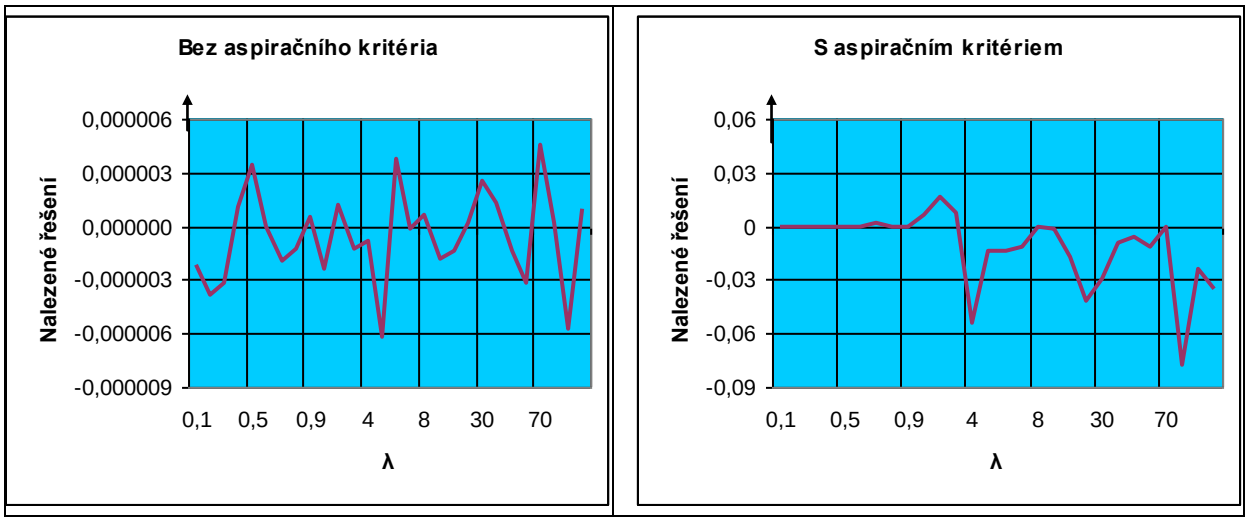
Z naměřených hodnot je patrné, že algoritmus PLP je silným nástrojem při hledání globálního extrému. Experimenty dále naznačují, že algoritmus dává lepší výsledky, pokud je spojený s aspiračním kritériem. PLP s aspiračním kritériem dává obecně stabilnější výsledky u nalezeného globálního extrému, hodnoty nalezeného extrému nejeví známky vysoké rozkolísanosti (jednotlivé hodnoty se pohybují blíže k optimu než hodnoty u PLP bez použití aspiračního kritéria). Jistou nevýhodou je delší časová náročnost pro výpočet, což je důsledkem většího počtu iterací, který je zapříčiněn kontrolou, zda se řešení nachází v aspiračním kritériu či nikoliv. Odměnou za to je však přesnější výsledek. Taktéž počet nahrazených nových lepších řešení je obecně nižší u PLP s aspiračním kritériem.

Další problematikou řešenou v rámci této práce je nalezení takového parametru λ, který by mohl dávat dobré výsledky v rozumném čase. Opět z obrázků, které jsou zobrazeny dále v této kapitole (obr. 4 – obr. 12) je patrné, že dobré výsledky dává parametr, pokud se nachází v rozmezí $\langle 0.4, 0.9 \rangle$, kdy jak časová náročnost je klesající, tak i hodnoty nalezených globálních extrémů se velmi blíží k hodnotám, jež jsou pro dané testovací funkce známé. Z toho hlediska lze uvažovat pro další výzkum a výpočty tento rozsah hodnot za přijatelný. Hodnoty, které jsou v rozmezí $\langle 1, 100 \rangle$ dávají také uspokojující řešení, ale pouze z hlediska časové náročnosti.



Obrázek 4- Porovnání počtu iterací u 1st de Jong funkce

Pro analýzu chování algoritmu na testovacích funkcích byly vybrány pouze některé a ty budou dále popsány. Všechny naměřené hodnoty u zbylých funkcí jsou v příloze E. Do první rodiny funkcí patří ty jednodušší jako je např. funkce 1st de Jongova. Jak je patrné z obrázku 5, algoritmus PLP bez aspiračního kritéria nalezne suboptimální řešení za pomoci menšího počtu iterací, než PLP s aspiračním kritériem. S hodnotami počtu iterací samozřejmě souvisí i celkový čas, který v této části analýzy nebude brán v potaz, avšak u celkového porovnání výsledku v příloze je uveden.

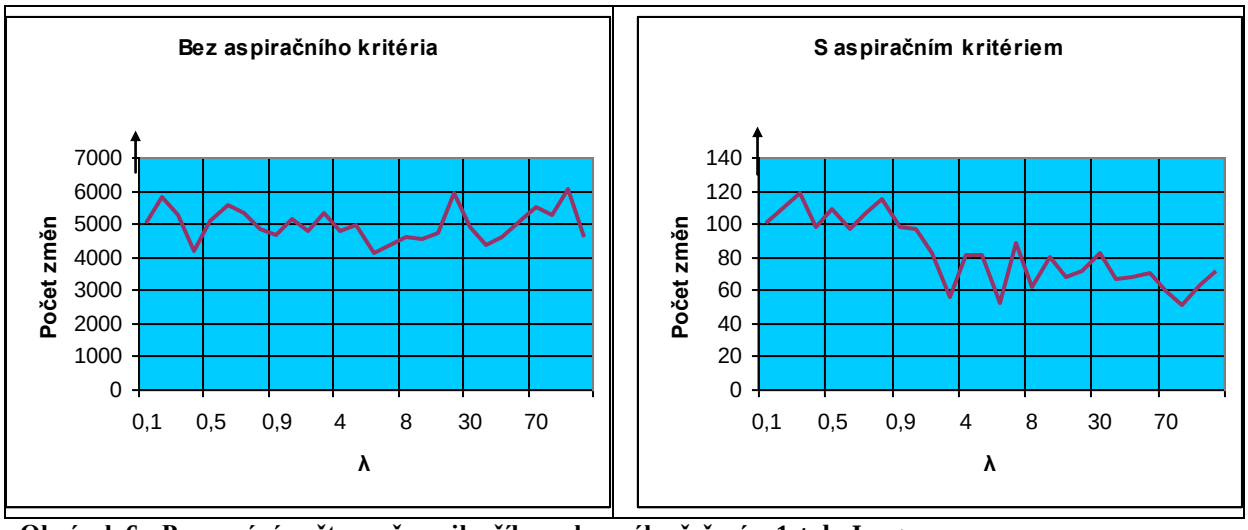


Obrázek 5 - Porovnání nejlepších nalezených řešení u funkce 1st de Jong

Pokud se týká nejlepšího nalezeného řešení v průběhu algoritmu, je patrné, že PLP s aspiračním kritériem dává lepší výsledky a to především u hodnot parametru λ v rozmezí $\langle 0.1, 0.9 \rangle$. Stanovení takové hodnoty parametru λ bude určující pro testování PLP v problémech TSP. Z obrázku 6 je patrné, že naměřené hodnoty u PLP s aspiračním kritériem

nevykazují takovou rozkolísanost jako u PLP bez aspiračního kritéria, tzn. že hodnoty se vyskytují velice blízko optimální hodnotě pro danou funkci. Vzhledem k tomu, že je parametr λ právě tak nízký, pak se hodnota penalizace o tolik nezvyšuje a nalezené řešení je tudíž kvalitnější než u větších hodnot parametru λ. U hodnot parametru v rozmezí $\langle 4,100 \rangle$ je rychlejší tendence se rychle vymanit z lokálního minima a existuje tudíž i reálná možnost nenalezení kvalitního řešení.

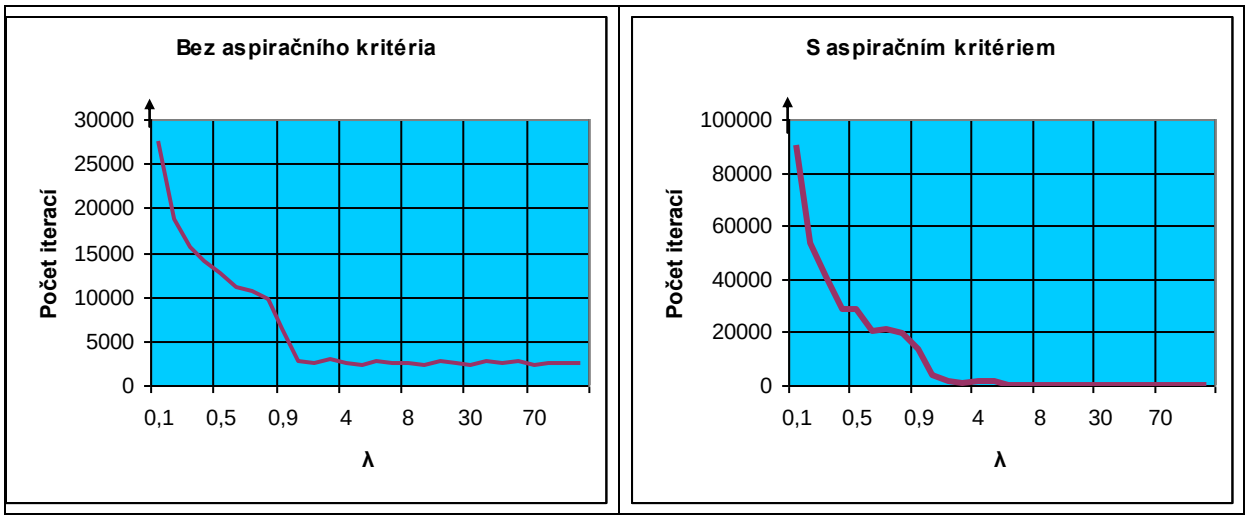
Pro určení kvality PLP s aspiračním kritériem je také vhodné porovnat počet změn nejlepšího nalezeného řešení. V průběhu algoritmu je zaznamenána hodnota nejlepšího nalezeného řešení, v momentu, kdy je nalezeno další jiné lepší řešení je to původní nahrazeno tímto novým a současně je navýšena o jednotku proměnná, která zaznamenává tuto hodnotu. Tato hodnota je zaznamenávána z toho důvodu, že je možné v momentu náhrady nejlepšího možného řešení provádět ještě jiné operace (jako je např. porovnávání hodnot s jinými naměřenými hodnotami a běh algoritmu pak dále upravovat), čímž se ušetří jak časová tak i paměťová náročnost algoritmu. Obrázek 6 zobrazuje porovnání těchto naměřených hodnot u obou typů porovnávaných algoritmů. Opět je patrné, že PLP s aspiračním kritériem dává lepší výsledky, a proto je možné jej označit za kvalitnější a paměťově méně náročnou variantu lokálního prohledávání.



Obrázek 6 - Porovnání počtu změn nejlepšího nalezeného řešení u 1st de Jong

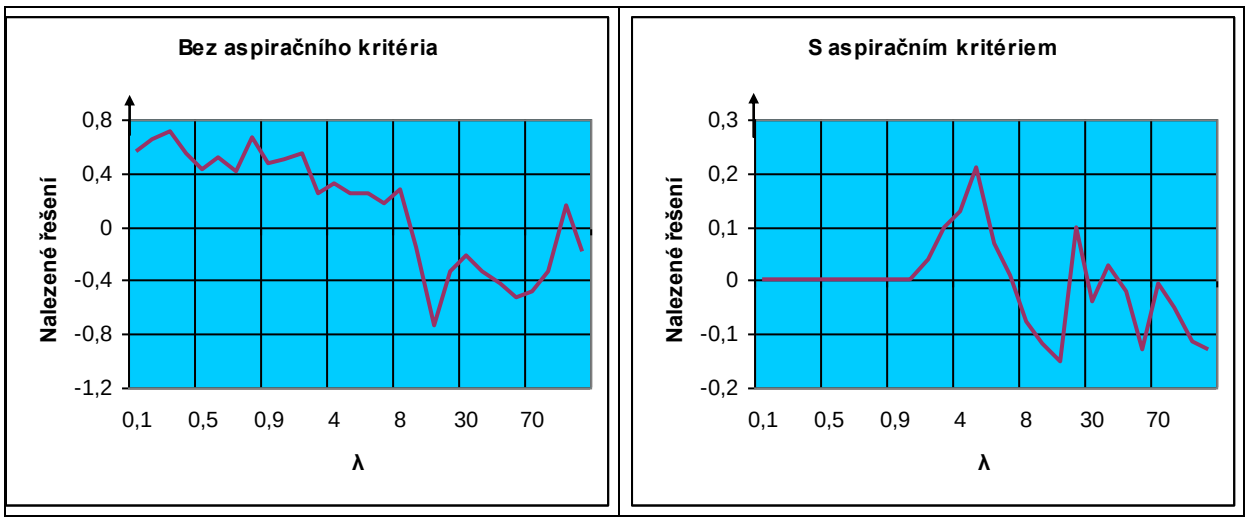
Dalším typem funkcí, které byly prozkoumávány, již nejsou základní mocninné funkce, ale funkce z rodiny goniometrických. Jsou uvedeny v příloze E. Chování algoritmu PLP bude ukázáno na Rastriginově funkci, jež v sobě zahrnuje funkci cosinus. V průběhu

algoritmu byl opět spočítán větší počet iterací u PLP s aspiračním algoritmem než u algoritmu bez použití aspiračního kritéria, jak je patrné z obrázku 8. U hodnot parametru λ vyšších než 1 se snižuje počet iterací pouze u algoritmu s použitím aspiračního kritéria.



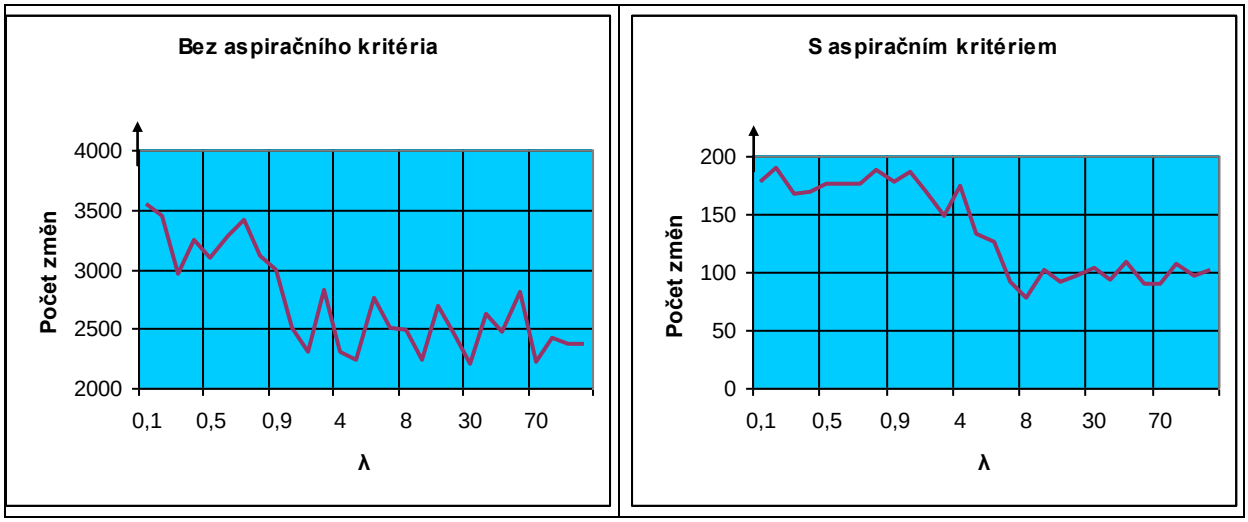
Obrázek 7 - Porovnání počtu iterací u Rastriginovy funkce

V případě hodnot nejlepšího nalezeného řešení v průběhu algoritmu i u této funkce je vidět, že PLP s aspiračním kritériem poskytuje výsledky, které se nachází v blízkosti globálního extrému u dané funkce a to především pro hodnoty parametru $\lambda \in \langle 0.1, 0.9 \rangle$. Pokud bychom měli porovnat hodnoty počtu iterací a nejlepšího nalezeného řešení, pak i zde nejlépe vychází hodnota parametru λ přibližně v mezích $\langle 0.4, 0.8 \rangle$, protože počet iterací je pro tyto hodnoty v rozmezí 28320 pro hodnotu $\lambda = 0.4$ a 19245 pro $\lambda = 0.8$ a hodnoty naměřených nejlepších řešení se dají považovat za blízké optimálnímu řešení.



Obrázek 8 - Porovnání nejlepších nalezených řešení u Rastriginovy funkce

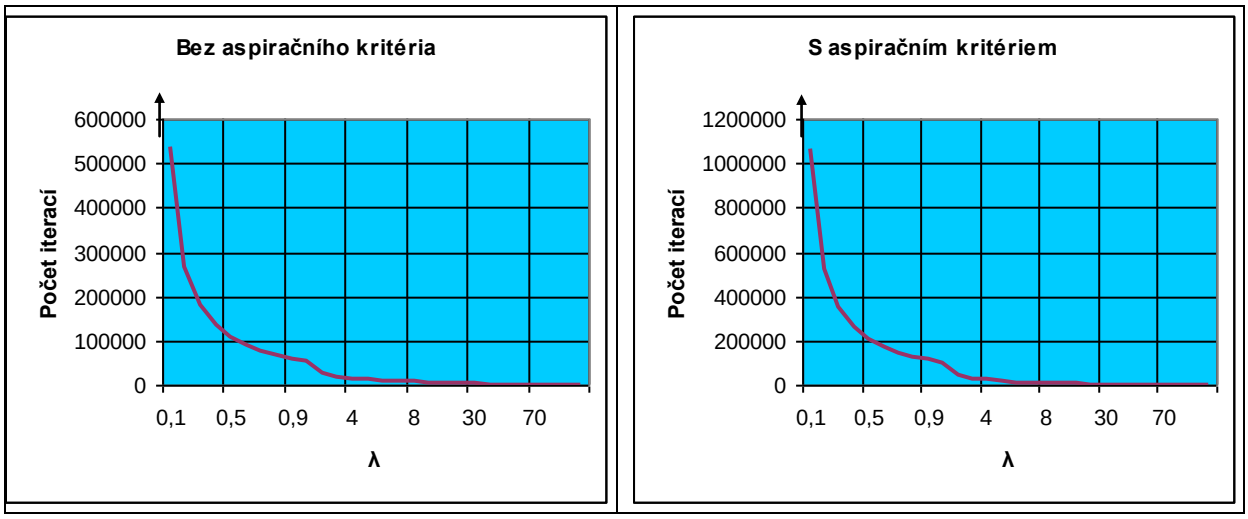
V případě počtu změn nejlepšího nalezeného řešení u této funkce (viz obr. 9) je opět patrný rozdíl mezi oběmi porovnávanými algoritmy a tento rozdíl je několikanásobný. Pokud by bylo vhodné nějakým způsobem upravovat, nebo ještě provádět jiné různé operace v momentě nahrazení nejlepšího možného řešení, je vhodné použít algoritmus PLP s aspiračním kritériem. I v tomto případě je vidět, že aspirační kritérium je vhodný nástroj pro rozšíření algoritmu PLP nebo i jiného algoritmu, který je založen na lokálním prohledávání.



Obrázek 9 - Porovnání hodnot změn nejlepšího nalezeného řešení u Rastriginovy funkce

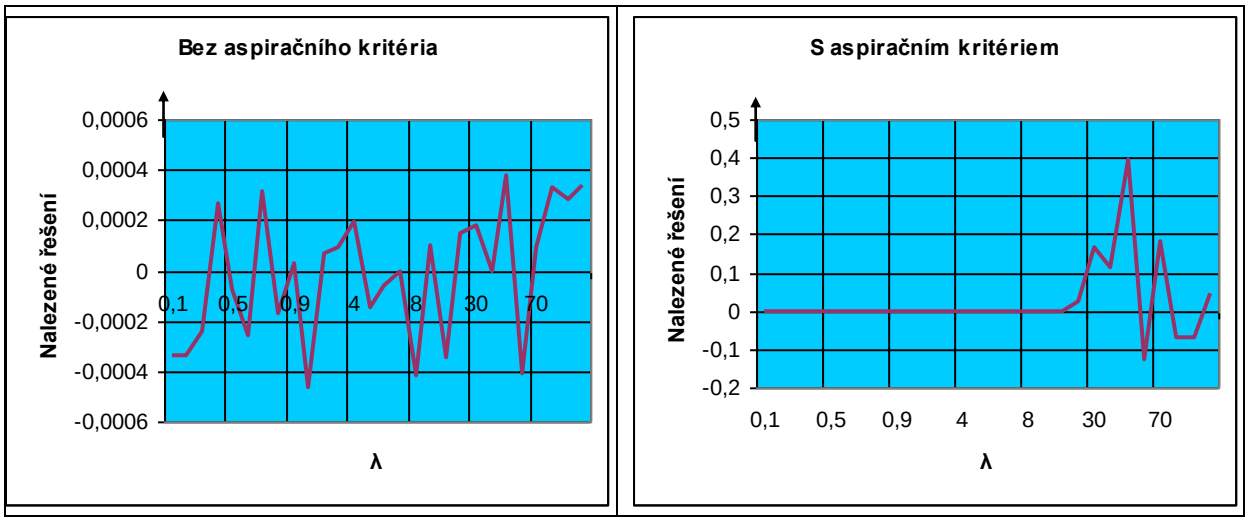
Dalším typem funkce, který byl v této kapitole analyzován, je typ exponenciální funkce. Konkrétně se zaměříme na funkci Ackley path. Grafy ostatních funkcí, které byly zkoumány, jsou uvedeny v příloze E a je zde uvedena i rovnice funkce spolu s rozsahem, v jehož rámci probíhalo zkoumání této funkce.

Počet iterací pro nalezení řešení této funkce (viz obr. 10) je o mnoho vyšší než u předcházejících typů funkcí, což je zapříčiněno relativní složitostí dané funkce. Ale i v tomto případě si jak algoritmus PLP bez aspiračního kritéria, tak i PLP s aspiračním kritériem s touto funkcí poradil a v průběhu hledání nalezl kvalitní výsledky.



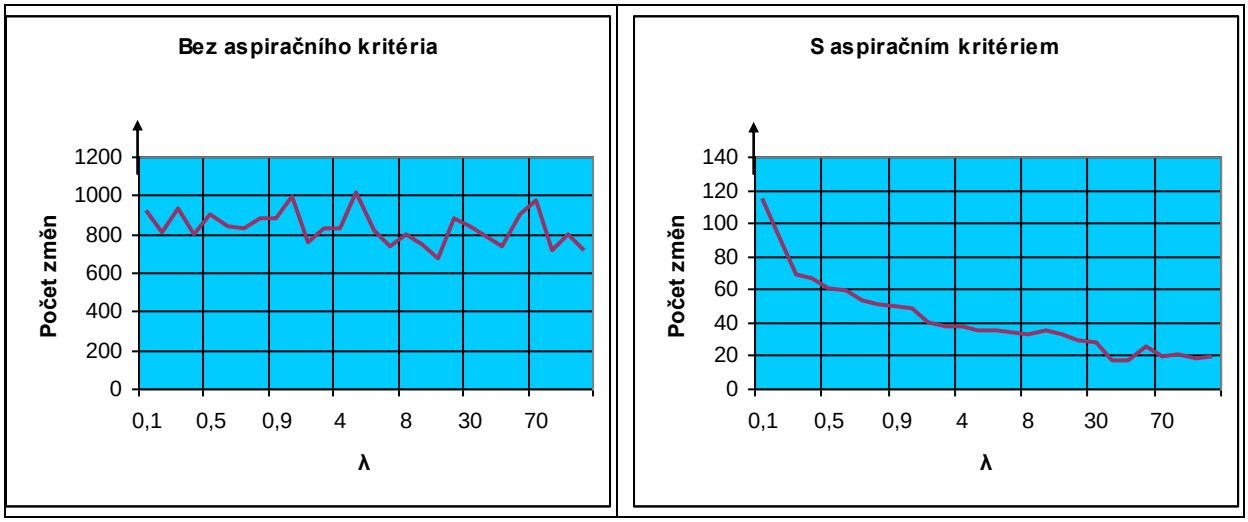
Obrázek 10 - Porovnání počtu iterací u Ackley's Path funkce

V případě nejlepšího nalezeného řešení algoritmus PLP s aspiračním kritériem (viz obr. 11) pro tento typu funkce nalezl kvalitní řešení pro hodnoty z vyššího rozsahu než v případech použití algoritmu bez aspiračního kritéria a tudíž se dá usuzovat, že algoritmus je vhodným nástrojem pro hledání suboptimálního řešení. Rozsah hodnot parametru λ, kdy bylo nalezeno kvalitní řešení je v rozmezí $\langle 0.1,9 \rangle$. Pokud porovnáme tuto hodnotu z hlediska počtu iterací, pak lze usoudit, že parametr λ může být nastaven v rozmezí $\langle 0.5,9 \rangle$ a PLP s aspiračním kritériem pak bude podávat kvalitní řešení. U PLP bez aspiračního kritéria je patrná jistá rozkolísanost u nejlepších nalezených řešení, což může být dáno právě absencí aspiračního kritéria, kdy si algoritmus nepamatuje rozsah již navštívených řešení, a proto se do některých vrací znovu a nevykazuje je z propočtu jako zakázaná řešení.



Obrázek 11 - Porovnání nejlepšího nalezeného řešení u Ackley's Path funkce

Počet změn nejlepšího nalezeného řešení (viz obr. 12) u Ackley's Path funkce vykazuje opět mnohonásobně nižší rozdíly u PLP s aspiračním kritériem a proto lze opět říci, že aspirační kritérium je vhodným nástrojem pro implementaci do algoritmů typu lokálního prohledávání.



Obrázek 12 - Porovnání hodnot počtu změn nejlepšího nalezeného řešení u Ackley path funkce

V předchozí části proběhla podrobnější analýza využití algoritmu PLP na testovacích funkcích. Nyní bude tento algoritmus porovnán s jinými známými algoritmy.

Aby mohl být algoritmus PLP s aspiračním kritériem porovnán i s jinými algoritmy, byla zvolena hodnota parametrem $\lambda = 0.5$ [PANU07b]. Tato hodnota vyplynula z analýzy chování algoritmu na vybraných testovacích funkcích. Dalším algoritmem, který sloužil k porovnání s PLP, byl zvolen algoritmus SA. Pro tento algoritmus byly zvoleny dvě hodnoty parametru *k* a to konkrétně 1000 iterací a 10000 iterací. Pro větší hodnotu iterací platí, že SA má delší dobu trvání, ale může podávat lepší výsledky, což je patrné právě v tab. 4. Pro srovnání byly vybrány 4 testovací funkce a to 1 de Jong, Ackley's Path, Griewangk a Schwefell.

Tabulka 4 - Porovnání PLP s aspiračním kritériem s jinými metodami

Funkce	Penalizované lokální prohledávání s aspirací(λ=0.5)		SA pro k=1000		SA pro k=10000	
	Průměrná odchylka (%)	Průměrný čas (s)	Průměrná odchylka (%)	Průměrný čas (s)	Průměrná odchylka (%)	Průměrný čas (s)
1deJong	0.05	11.50	0.85	0.10	0.22	18.11
AckleyP	0.05	94.65	0.35	5.13	0.26	20.53
Grie	0.25	213.44	0.37	3.97	0.34	38.80
Schw	0.01	44.34	1.53	6.10	0.92	61.54

Na všech těchto vybraných funkcích je patrné, že PLP dává lepší výsledky než SA u obou nastaveních hodnot *k*. V případě PLP je čas nutný pro výpočet suboptimálního řešení delší než u SA s $k = 1000$, ale na druhou stranu, nalezené řešení se více blíží optimálnímu řešení. Z tohoto hlediska lze tedy považovat PLP za algoritmus, který vykazuje lepší výsledky než algoritmus SA, a to i s nastavením $k = 10000$.

6. Problém obchodního cestujícího

Problém obchodního cestujícího [CROE58], [JANÁ02], [JOHN90] je jedním z nejznámějších problémů řešených v kombinatorické optimalizaci. V této kapitole se budeme věnovat tomu, jakým způsobem může být penalizované lokální prohledávání aplikováno na tento problém. Také bylo provedeno srovnání s některými heuristickými algoritmy, které se využívají při řešení TSP.

6.1 Problematika TSP

Problém obchodního cestujícího má ve skutečnosti mnoho variant [LIN73]. V této práci se budeme věnovat klasickému symetrickému TSP. Problém je určen počtem N měst a symetrickou maticí vzdáleností $\boldsymbol{D}=[d_{ij}]$, která určuje vzdálenosti mezi jakýmikoliv dvěma městy i a j. Cílem TSP je pak nalézt takovou posloupnost měst začínající a končící stejným městem, ve které se každé město nalézá právě jednou (kromě prvního). Podmínkou je, aby součet vzdáleností po sobě následujících měst byl minimální. Takováto posloupnost se v literatuře nazývá také minimální Hamiltonovská kružnice. Posloupnost lze popsat jako cyklickou permutací π N měst, ve které $\pi(i)$ je index města, které bylo navštíveno právě po městě s indexem i, $i=1,...,N$. Cena této permutace je pak definována jako:

$$f(\pi)=\sum_{i} d_{i\pi(i)} \tag{33}$$

a tím je i dána účelová funkce TSP.

Touto problematikou se zabývá celá řada autorů [JOHN03], [KNOX94], [LARR99], [LAPO92], [REIN91]. V současné době problém TSP hraje velmi důležitou roli ve vývoji a v testování nových optimalizačních technik. V následujících kapitolách bude ukázáno, jak penalizované lokální prohledávání může být aplikováno na tento typ problému.

6.2 Heuristiky lokálního prohledávání pro TSP

Pro TSP existuje celá řada algoritmů založených na lokálním prohledávání. Jedním z těch nejjednodušších je tzv. algoritmus *2-záměny (2-opt)* [CROE58]. Algoritmus nejdřív začne s náhodnou permutací všech měst, kterou nazveme trasa *Tr*. Sousedství trasy *Tr* je

definováno jako množina všech tras, které mohou být získány záměnou dvou nesousedních uzlů v trase *Tr*. Takový přesun, který to umožní, se nazývá *2-záměna*.

Pokud je nalezena taková trasa *Tr'*, která má lepší řešení[7] než předcházející trasa *Tr*, je touto nahrazena. Jestliže v sousedství neexistuje žádná trasa *Tr*, která by dosahovala kratších vzdáleností než právě tato trasa, pak je trasa *Tr* nazvána *2-optimální* [MICH00] a algoritmus je ukončen. Algoritmus může být samozřejmě spouštěn opakovaně z různých náhodných permutací.

Algoritmus *2-záměny* může být jednoduše zobecněn na jakoukoli proceduru *k-záměny*, tak že je vybráno *k* uzlů pro vytvoření sousedního řešení trasy *Tr*. Jestliže je *k* malé číslo, celý prostor řešení může být prohledán velmi rychle, avšak pro takové číslo roste pravděpodobnost nalezení pouze suboptimálního řešení. Na druhou stranu, pro větší hodnoty čísla *k* se enormně zvětšuje počet nalezených sousedních řešení – počet podmnožin roste exponenciálně s velikostí *k*. Z tohoto důvodu se zřídkakdy pro *k-záměnu* používá $k > 3$ [MICH00].

Algoritmus, jenž je v literatuře [MICH00] znám jako Lin-Kernighanův (LK) algoritmus, je považován za „nesporného krále“ [REEV96] heuristik pro lokální prohledávání v problému obchodního cestujícího. [MICH00] jej uvádí jako „nejznámější proceduru pro TSP“. Lin a Kerninghan vylepšili výměnu hran tak, že počet *k* hran může být variabilní [LIN73].

6.3 Aplikace penalizovaného lokálního prohledávání pro TSP

6.3.1 Zvýšená účelová funkce a argumenty řešení

Prvním krokem aplikace algoritmu je náhodné vygenerování posloupnosti měst daného problému a výpočet prvotních nákladů řešení. Pak je dána matice vzdáleností $\boldsymbol{D} = [d_{ij}]$ mezi jednotlivými městy. Souběžně s touto maticí se vygeneruje pomocná matice, která zaznamená, zda byla daná hrana použita v řešení, či nikoliv. Množina argumentů může být definována s ohledem na neorientované hrany e_{ij} $(i = 1...N, j = i+1...N, i \neq j)$, jež se mohou objevit v trase s určitou cenou, která je dána délkou hrany d_{ij}. Každá hrana e_{ij} spojující město *i* a město *j* si s sebou nese i penalizaci, která je na počátku nastavená na nulovou hodnotu a v průběhu běhu algoritmu se postupně zvyšují. Tyto hranové penalizace jsou uspořádány v symetrické matici $\boldsymbol{P} = [p_{ij}]$. Penalizace jsou postupně kombinovány s účelovou funkcí

[7] Původní řešení je nahrazeno novým v případě nalezení prvního zlepšení (first improvement). Jinými slovy lze říci, že prohledávání sousedství trasy *T* je ukončeno v momentu nalezení prvního zlepšení.

problému, tak že vznikne zvýšená účelová funkce, která je lokálním prohledáváním minimalizována. Toto se děje na základě pomocné matice vzdáleností

$$\boldsymbol{D}' = \boldsymbol{D} + \lambda \boldsymbol{P} \tag{34}$$

Lokální prohledávání využívá matici $\boldsymbol{D}'$ místo původní matice $\boldsymbol{D}$ pro evaluaci přesunů. Algoritmus si upraví $\boldsymbol{P}$ a $\boldsymbol{D}'$, vždy když je dosaženo lokálního minima. Hrany, které jsou penalizovány v lokálním minimu, jsou vybrány s ohledem na funkci využitelnosti (31), která je pro problém obchodního cestujícího definována následujícím vztahem:

$$využitelnost\ (trasa, e_{ij}) = Ie_{ij}(trasa)\ \ (d_{ij}/(1+p_{ij})), \tag{35}$$

kde Ie_{ij}(trasa) = 1, pokud e_{ij}= trasa, jinak 0.

6.3.2 Metoda penalizovaného prohledávání v TSP

Algoritmus penalizovaného lokálního prohledávání nepředpokládá žádný vnitřní mechanizmus lokálního prohledávání (sám v sobě neobsahuje žádný takový algoritmus) a tudíž může být lehce kombinován s různými prohledávacími algoritmy a to i bez ohledu jak je algoritmus komplexní.

Prohledávací procedury, které jsou kombinovány s algoritmem penalizovaného lokálního prohledávání, jsou pak implementovány jako procedury, jež poskytnou počáteční trasu a vrátí lokálně optimální trasu s ohledem na sousedství prohledávaného prostoru. Matice vzdáleností použitá lokálním prohledáváním je pomocná matice $\boldsymbol{D}'$, jež byla popsána v předcházejícím odstavci. V momentě, kdy je vykonán další přesun a je navštíveno nové řešení, je stále potřebný odkaz na matici $\boldsymbol{D}$, aby bylo možno porovnat toto nové řešení s původním.

Nyní následuje popis, jakým způsobem pracuje algoritmus penalizovaného lokálního prohledávání na TSP. Algoritmus začne v libovolném počátečním řešení, lokální prohledání poté nalezne nějaké lokální minimum. Penalizované lokální prohledávání penalizuje pomocí funkce využitelnosti (35) jednu nebo více hran, jež se objevily v lokálním minimu. Poté, co jsou penalizace zvýšeny, metoda je znovu spuštěna z posledního lokálního minima, aby mohla nalézt další nové lokální minimum. Jakmile je nové lokální minimum nalezeno nebo metoda uvázne v současném lokálním minimu, jsou opět zvýšeny penalizace a tak metoda pokračuje dál.

Algoritmus penalizovaného lokální prohledávání se neustále pokouší odstranit ty hrany, které se objevují v lokálním minimu tím, že jim dává větší penalizaci. Snaha, která je vložena do metody, odstranit konkrétní hranu, závisí na délce hrany. Čím delší je hrana, tím větší úsilí je vloženo do metody. Tento efekt závisí na velikosti regulačního parametru λ. Pokud je parametr λ větší číslo, pak algoritmus penalizovaného lokálního prohledávání může v některých případech opominout určité informace o lokálním gradientu, zatímco pokud je parametr menší číslo, algoritmus se pak dostane z lokálního minima za pomocí většího počtu penalizačních cyklů. Nicméně existuje jistý rozsah hodnot parametru λ, kdy se vybere takový přesun, jež zlepší současné řešení (s ohledem na gradient) a zároveň odstraní penalizované hrany (s ohledem na rozhodnutí algoritmu). Pokud se i delší hrany stále vyskytují v nalezeném řešení navzdory penalizacím, metoda bude diverzifikovat svůj výběr a bude se snažit odstranit i kratší hrany.

Penalizace postupně přidává ohodnocení hranám tak, jak se objevují v lokálním minimu a pak algoritmus prohledává nové oblasti celého prostoru přidáváním těch hran, které ještě nebyly penalizovány. Rychlost tohoto procesu závisí na parametru λ. Čím je nižší tento parametr, tím pomalejší je tento proces, protože algoritmus stráví více času v aktuální oblasti, předtím než je přinucen tuto oblast opustit a začít hledat v jiné oblasti. A naopak, čím větší je parametr, tím je rychlejší opuštění tohoto prostoru.

6.4 Vyhodnocení algoritmu penalizovaného prohledávání na TSP

Aby bylo zjištěno, jak algoritmus pracuje, byla provedena celá řada experimentů. Experimenty byly zkoušeny pomocí nejjednodušší TSP heuristiky, tedy 2-záměny. Výhody patrné z použití algoritmu penalizovaného prohledávání oproti SA a TS jsou patrné dále v textu na vybraných příkladech (viz. tab. 6). Výhoda algoritmu se převážně projevuje na problémech malých a středních velikostí.

V provedených experimentech je použita knihovna veřejných problémů TSP, jež se nazývá TSPLIB [REIN91]. Většina z příkladů zahrnutých v knihovně TSPLIB již někdy dříve byla použita pro řešení optimalizace a tyto příklady byly také použity a citovány v mnohé literatuře o TSP.

Každý uvedený případ byl spuštěn desetkrát a pokaždé bylo na počátku vygenerováno jiné náhodné počáteční řešení. Pokaždé byly naměřeny různé hodnoty, které charakterizovaly dané řešení (doba běhu algoritmu, kvalita řešení atp.). Kvalita řešení byla měřena procentuální

odchylkou od nejlepšího známého řešení (nebo pokud je známo, tak optimálního řešení) a tato odchylka je dán následujícím vzorcem:

$$\text{odchylka} = \frac{\text{nalezené řešení} - \text{nejlepší známé řešení}}{\text{nejlepší známé řešení}} * 100 \qquad (36)$$

Jediný parametr, který v algoritmu vyžaduje optimální nastavení je parametr λ. Některé experimenty ukázaly, že algoritmus penalizovaného prohledávání je jistým způsobem tolerantní k nastavení parametru λ a to do té doby, kdy je tento parametr roven podílu průměrné délky jednotlivých hran v nalezeném řešení, které se blíží (resp. se nachází) lokálnímu optimu a celkovým počtem měst [VOUD93]. Tyto poznatky pak byly vyjádřeny v následujícím vzorci, pro výpočet parametru λ:

$$\lambda = a \frac{f(\text{lokální optimum})}{N} \qquad (37)$$

kde f (lokální optimum) je hodnota lokálního minima trasy, jež je vypočítáno v průběhu lokálního prohledávání (např. první lokální minimum předtím, než jsou aplikovány penalizace) a číslo N je počet měst, jež jsou zahrnuta v daném případu. Rovnice (37) pak zavádí parametr a, jehož hodnoty se nachází ve snadno ovladatelném rozsahu $\langle 0,1 \rangle$. Pomocí různých experimentů [LAWL85] bylo dokázáno, že tento parametr nezávisí pouze na daném případu, ale také na konkrétní prohledávací heuristice, která je použita. Obecně se dá říci, že existuje nepřímý vztah mezi efektivitou lokálního prohledávání a parametrem a. 2-záměnová heuristika, která není tolik efektivní, vyžaduje vyšší hodnotu parametru a než efektivnější heuristika 3-záměny nebo LK. Je to zapříčiněno tím, že rozsah penalizací potřebný k úniku z lokálního minima roste tím, jak se zvyšuje účinnost dané heuristiky, a proto tedy musí být použita nižší hodnota parametru a k tomu, aby lokální gradient mohl působit na rozhodování algoritmu. Pro jednotlivé typy heuristik a pro dané problémy z knihovny TSPLIB pak byly navrhnuty rozsahy parametru a jak je uvedeno v tab. 7.

Tabulka 5 - Navržené rozsahy parametru a pro algoritmus penalizovaného prohledávání podle různých TSP heuristik[LAPO92]

Heuristika	Navržený rozsah parametru *a*
2-záměna	$1/8 \leq a \leq 1/2$
3-záměna	$1/10 \leq a \leq 1/4$
LK	$1/12 \leq a \leq 1/6$

Nižší hranice těchto intervalů, představuje typické hodnoty pro parametr *a*, které umožní algoritmu uniknout z lokálního minima ve snesitelné míře. Pokud jsou použity hodnoty menší, než jsou tyto uvedené v tab. 5, pak algoritmus vyžaduje příliš mnoho penalizačních cyklů k úniku z lokálního minima.

6.5 Penalizované lokální prohledávání a heuristika 2-záměny

V tomto paragrafu jsou prezentovány výsledky dosažené pomocí penalizovaného lokálního prohledávání s různou velikostí parametru λ. Množina problémů byla sestavena z několika příkladů z knihovny TSPLIB malé a střední velikosti od 48 do 318 vrcholů. Limit pro zastavení algoritmu byl nastaven na hodnotu 200 000 iterací. Jelikož pouze parametr λ je možné v algoritmu penalizovaného prohledávání měnit, byla použita množina hodnot v rozsahu $\langle 0.1,100 \rangle$.

Jako velmi dobrá hodnota parametru pro hledání řešení v TSP byla stanovena hodnota 0.3. V každém z vybraných problémů bylo spuštěno 10 běhů a pokaždé bylo nalezeno jiné počáteční náhodné řešení a po ukončení měření byly zprůměrovány jednotlivé naměřené hodnoty (potřebný čas k dosažení nejlepšího řešení, odchylka od nejlepšího známého řešení atp.). Výsledky jsou uvedeny v tab. 6.

6.5.1 Porovnání s dalšími obecnými metodami pro TSP

Vzhledem k tomu, že algoritmus PLP používá heuristiku *2-záměny*, není primárně vhodný pro hledání řešení problému TSP větších rozsahů. Při prohledávání literatury, která se týká tohoto tématu, nebyla nalezena jiná metoda, která by vykazovala dobré výsledky (nějaké optimální řešení v krátkém čase) pro úlohy s více jak přibližně 320 vrcholy za použití právě heuristiky *2-záměny*. Z toho důvodu byly vybrány pouze problémy uvedené v tab. 6. Iterační Lin-Kernighanův algoritmus a jeho varianty [JOHN89], [JOHN90] dosahovaly podobných výsledků při hledání optimálních řešení. Pro srovnání algoritmu PLP s jinými metodami byly vybrány algoritmy SA a TS, které používají stejnou heuristiku.

6.5.2 Simulované žíhání

Algoritmus simulovaného žíhání, který byl implementován pro problém TSP navrhl Johnson v [JOHN89] a použil geometrické ochlazovací schéma (viz. odst. 3.1.1). Tento algoritmus generuje nové spojení v problému pomocí náhodných *2-záměn*. Jestliže takový nový přesun vyvolá snížení nákladů v trase je okamžitě přijat. Přesuny, které nezlepší celkové náklady, jsou přijaty s pravděpodobností (38), kde Δ je rozdíl v nákladech v důsledku nové záměny a *Tep* je současná teplota v algoritmu. Na začátku běhu algoritmu je teplota relativně vysoká, a tudíž přibližně 50% nových přesunů je akceptováno.

$$e^{\frac{-\Delta}{Tep}} \tag{38}$$

V každém kroku nastavení teploty je algoritmu umožněno provést v každém kroku stejný počet experimentů, tak aby dosáhl vyváženého stavu. Jakmile bylo tohoto vyváženého stavu dosaženo, teplota byla následně vynásobena určitým ochlazovacím parametrem a, jenž je obvykle nastaven na vyšší hodnotu (a = 0.9). K zastavení algoritmu bylo použito schéma navržené a popsané v [JOHN89].

6.5.3 Zakázané prohledávání

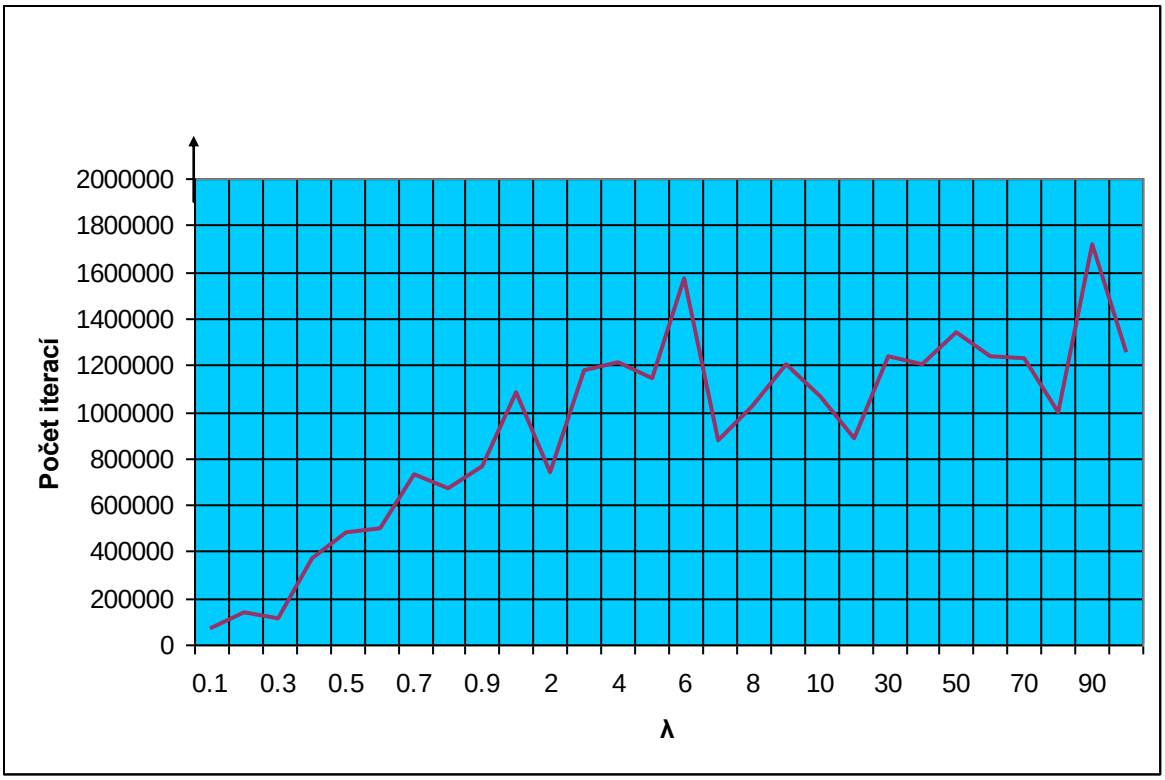
Varianta zakázaného prohledávání popsaná Knoxem [KNOX94] využívá kombinaci zakázaného seznamu a aspiračního kritéria.

Metoda provede zlepšení v lokálním prohledávání tím, že vybere nejlepší přesun v sousedství vybraného úseku, avšak takového, který není zahrnut v tabu seznamu. Určení tohoto seznamu je velice důležité v dané metodě a je klíčem k úspěchu v průběhu algoritmu. V této variantě je jakákoliv výměna hran klasifikována jako tabu jen v případě, kdy jsou oba dva uzly současně v tabu seznamu. Jakmile aspoň jeden uzel není v tabu seznamu, pak tam není přidána ani daná výměna. Aktualizace tabu seznamu pak spočívá v tom, že se do něj zahrnou ty uzly, které byly pomocí 2-záměny zaměněny. Jakmile se seznam zaplní, jsou z něj odstraněny nejstarší položky a zaměněny za ty, které byly vymazány.

Tabulka 6 - Porovnání penalizovaného lokálního prohledávání, simulovaného žíhání, zakázaného prohledávání na vybraných příkladech TSPLIB

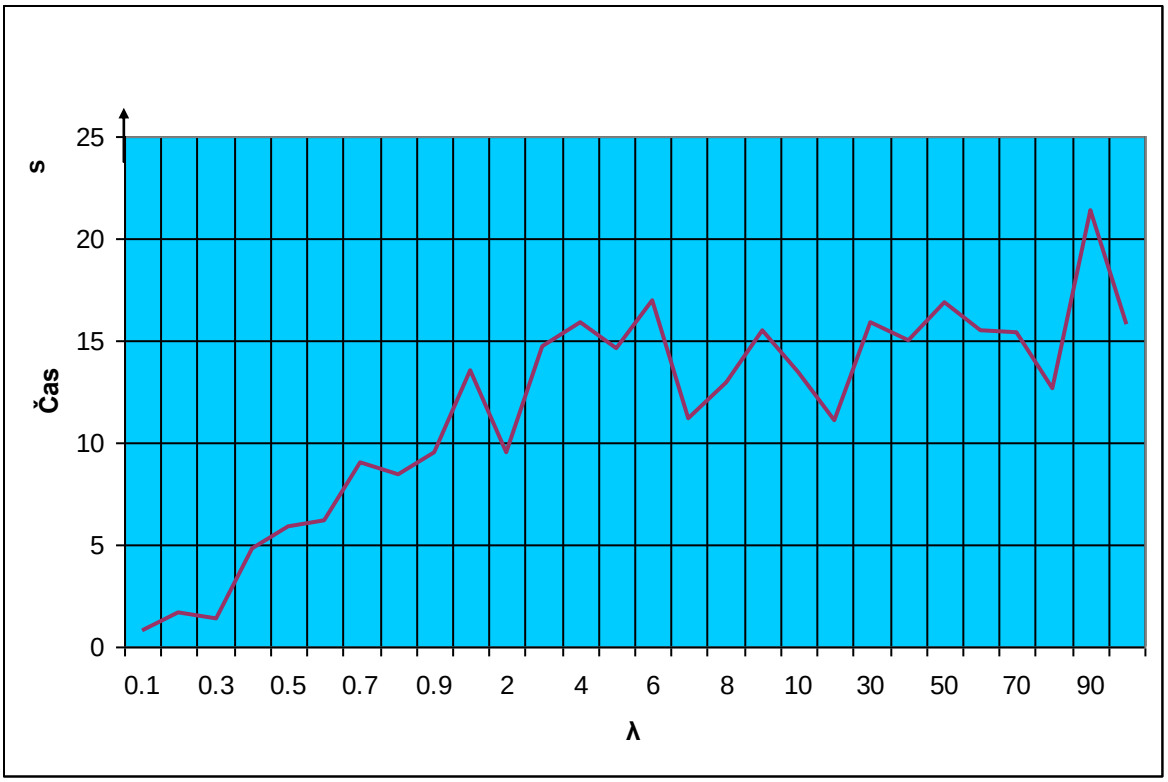
Problém	Penalizované lokální prohledávání s aspirací		Simulované žíhání		Tabu prohledávání		Opakované největší zlepšení	
	Prům. odchylka (%)	Prům. čas (s)	Prům. odchylka (%)	Prům.. čas (s)	Prům. odchylka (%)	Prům. čas (s)	Prům. odchylka (%)	Prům. čas (s)
eil51	0.00	10.46	0.73	6.34	0.00	1.14	0.23	42.40
eil76	0.00	10.97	1.21	18.00	0.00	5.24	1.85	153.45
kroA100	0.00	12.37	0.42	37.36	0.00	21.34	3.97	319.15
kroC100	0.00	11.25	0.80	36.58	0.25	4.80	0.34	706.35
eil101	0.00	10.74	1.76	33.29	0.00	61.41	0.33	1201.98
kroA150	0.00	17.03	1.86	103.32	0.03	413.06	1.41	3290.95
kroA200	0.00	150.16	1.04	229.38	0.72	776.93	1.70	731.10
lin318	0.005	346.44	1.34	829.46	1.31	2672.80	3.110	9771.28

Simulované žíhání a zakázané prohledávání bylo testováno na 8 typech TSP problémů. PLP dává lepší výsledky než ostatní algoritmy, tab. 6 ukazuje výsledky pro SA, TS a PLP (kde byl vybrán parametr λ o velikosti 0.3 – protože u testovacích funkcí dával nejlepší výsledky) a posledním algoritmem, který byl porovnán se základní 2-záměnovou heuristikou opakovaného největšího zlepšení. Tento algoritmus byl vybrán jako kontrast k výše zmíněným a pokaždé byla náhodně vybrána jedna trasa a bylo provedeno 200 000 iterací. Z výše zmíněných výsledků je patrné, že PLP podává lepší výsledky než všechny zmíněné algoritmy. Zakázané prohledávání nalezne optimální řešení snadným způsobem pro malé instance problémů a pro větší typy problému je označeno jako ucházející algoritmus, kdy nenalezne optimální řešení, ale nalezne řešení s malou odchylkou. Simulované žíhání také nalezne dobré řešení, avšak několikrát se stalo, že propadlo a nenalezlo kvalitní řešení. Oba tyto algoritmy podávaly horší výsledky, co se týká časové náročnosti než algoritmus PLP. Všechny zmíněné algoritmy využívaly 2-záměnové heuristiky.



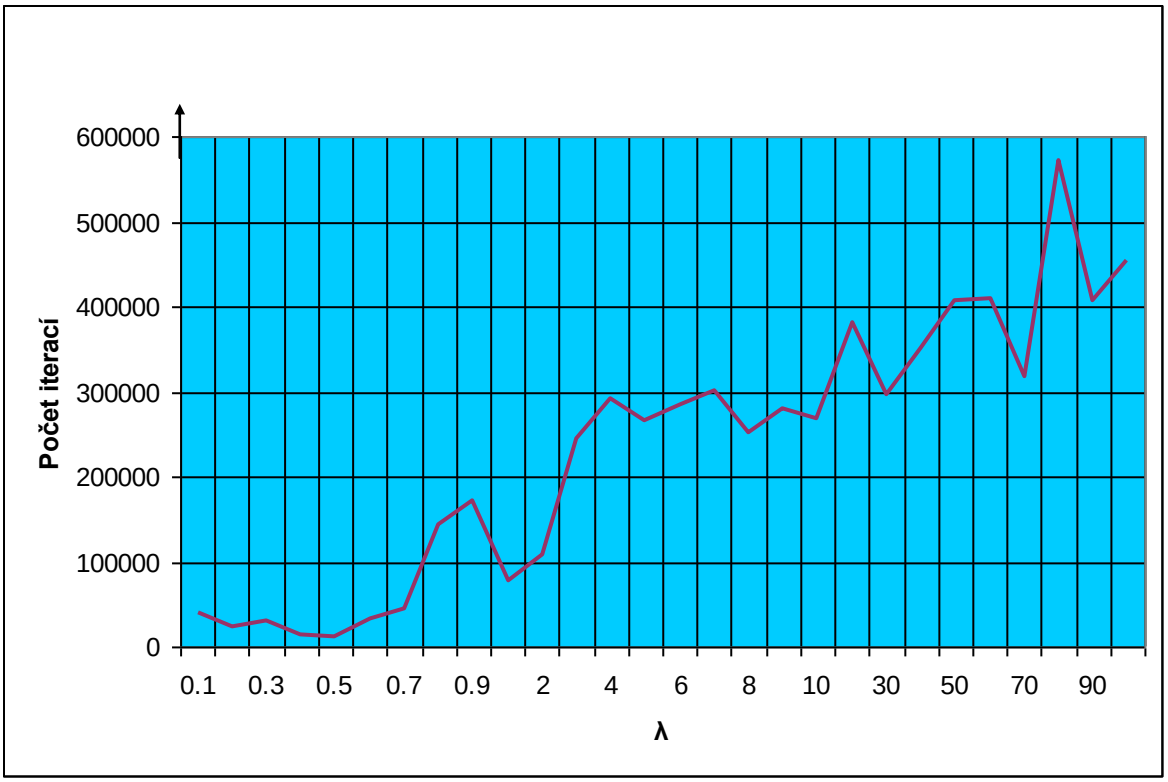
Obrázek 13 - Průměrný počet iterací pro různé hodnoty parametru λ v TSP kroA200

Pro stanovení parametru λ takového, že by mohla být jeho hodnota stanovena za kvalitní pro řešení úloh TSP, byly vybrány následující dva typy problémů a to konkrétně kroA150 a kroA200 z knihovny vybraných TSP úloh [LIN65]. U obou těchto problémů byly měřeny hodnoty doby trvání běhu algoritmu a počet iterací nutných k dosažení optimálního řešení. Zastavující kritérium pro algoritmus PLP bylo stanoveno na okamžik nalezení nejlepšího řešení. Algoritmus byl spuštěn desetkrát a výsledné hodnoty byly zprůměrovány. Jak je vidět z obr. 13 – 16, algoritmus se pro TSP chová odlišně než pro testovací funkce, tzn. s rostoucí hodnotou parametru λ, roste i časová a výpočtová náročnost algoritmu. Kdežto u testovacích funkcí s rostoucí hodnotou parametru, časová náročnost výpočtu spíše klesala. Tento jev lze přisoudit právě nastavení zastavení výpočtu v algoritmu. U testovacích funkcí byla hledána optimální (sub-optimální) hodnota testovaných funkcí. Algoritmus si po dobu svého běhu pamatoval nejlepší nalezené hodnoty a v momentě dosažení nějakého kritéria (obvykle počet iterací, jež má algoritmus provést) je běh algoritmu zastaven. Je zaznamenán celkový čas, počet iterací a počet zlepšujících iterací nutných k dosažení nalezeného řešení.

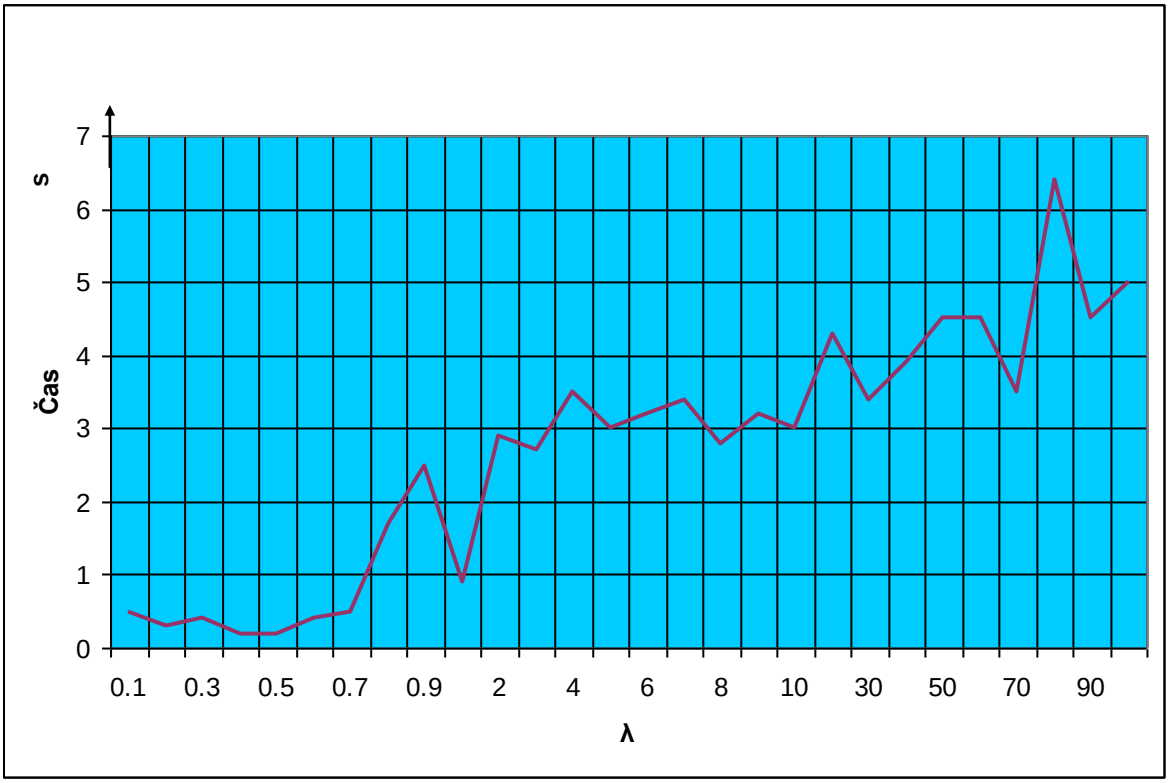


Obrázek 14 - Doba trvání běhu algoritmu v sekundách pro různé hodnoty parametru λ v TSP kroA200

U testování v případě TSP byl algoritmus nastaven tak, že nebyl ukončen dříve, než nalezl optimální řešení. Pokud je tedy nastavena vyšší hodnota parametru λ, pak algoritmus nastavuje vyšší hodnoty penalizací a tím pádem i nastane dřívější opuštění lokálního minima s tím rizikem, že se neprozkoumá toto lokální minimum dostatečně kvalitně a je možný únik aniž by mohlo být označeno za minimum globální. V případě malých hodnot parametru, je pak lokální prozkoumávání jemnější a lépe nachází globální minimum. Při testování algoritmu PLP s aspiračním kritériem (tento typ algoritmu jevil lepší výsledky při testování na testovacích funkcích a z tohoto důvodu byl zvolen i pro zkoumání problému TSP) bylo nutné zjistit, která hodnota parametru λ podává nejlepší výsledky pro TSP. Jak je patrné z obr. 13-16, velmi kvalitních výsledků bylo dosaženo u parametru v rozmezí $\langle 0.1, 0.3 \rangle$.



Obrázek 15 - Průměrný počet iterací pro různé hodnoty parametru λ v TSP kroA150



Obrázek 16 - Doba trvání běhu algoritmu v sekundách pro různé hodnoty parametru λ v TSP kroA150

Závěr

Rekapitulace

V práci byl představen obecný koncept lokálního prohledávání spolu se základními definicemi, které jsou potřebné pro porozumění problematiky lokálního prohledávání. Byla také navržena kritéria, která byla sledována v průběhu spuštěného algoritmu. Jejich sledování pomohou lépe porozumět chování algoritmu a také jakého efektu bylo dosaženo v průběhu prohledávání.

Dále je navržen algoritmus penalizovaného lokálního prohledávání a tento je rozšířen o aspirační kritérium, které je převzato z algoritmu zakázaného prohledávání. Tím je odstraněna ta část prohledávání, kdy je nalezeno nějaké řešení, avšak není známa velikost tohoto řešení a tudíž ani není znám vliv tohoto nalezeného řešení na celkový průběh algoritmu. Aspirační kritérium je vhodným nástrojem pro zjištění, zda se nově nalezené lepší řešení nachází v aktuálním zkoumaném prostoru řešení či nikoliv. Testování zda se řešení nachází v aspiračním kritériu, předchází samotnému algoritmu penalizovaného lokálního prohledávání.

Na vybraných testovacích funkcích a problémech obchodního cestujícího je poté ukázáno, jaký je rozdíl mezi navrženým algoritmem a dalšími vybranými algoritmy z oblasti lokálního prohledávání. Aspirační kritérium je využito v momentě, kdy v prohledávaném okolí se nachází nějaké lepší řešení, než již dříve nalezené. V takovém případě je toto řešení přijato a pracuje se s ním i dále. Tento postup pak dává lepší výsledky než jiné metody lokálního prohledávání. K tomuto závěru se dospělo na základě několika kontrolních experimentů, které zahrnovaly upravený algoritmus lokálního prohledávání.

V průběhu tvorby hybridního algoritmu bylo také zjištěno, že nemalý vliv na kvalitu nalezeného řešení má parametr λ. Pro zjištění jaká velikost tohoto parametru má nejlepší účinek pro nalezení optimálního řešení byla jeho hodnota měněna v mezích od 0.1 do 100 a to v jistých krocích. Ze zobrazených výsledků je patrné, že nejlepších výsledků dosahoval algoritmus v momentech, kdy hodnota tohoto parametru byla nastavena v rozmezí $\langle 0.4, 1.0 \rangle$. Velikost tohoto parametru má vliv na hladký průběh algoritmu, kdy při nižších hodnotách algoritmus nevykazuje velké výkyvy, tzn. je dosahováno poměrně stabilních výsledků. Daní za to je však vyšší časová náročnost. U vyšších hodnot parametru, je dosahováno již horších výsledků, ale v mnoha případech i tyto výsledky jsou kvalitní.

Na proběhnutých experimentech a to převážně z oblasti testovacích funkcích je také patrné, že lepších výsledků dosahuje algoritmus s aspiračním kritériem než bez něj. Z naměřených hodnot je také patrné, že úspěch aspiračního přesunu není jednoduše spojený pouze s penalizačními podmínkami (jež jsou základem penalizovaného lokálního prohledávání) v rozšířené účelové funkci, ale velký vliv na výsledky má také hodnota parametru λ.

Splnění cíle

Hlavním cílem této doktorské práce byl návrh hybridního algoritmu z oblasti evoluční optimalizace. Tento cíl lze považovat za splněný a to navržením úpravy algoritmu penalizovaného prohledávání o prvek aspiračního kritéria. Tento algoritmus byl dále zkoumán spolu s měnícími se hodnotami parametru λ. V části, která se věnovala testování tohoto algoritmu je patrné, že algoritmus je použitelný v celé řadě optimalizačních úloh.

Kapitole 3 byly popsány různé metody z oblasti globální optimalizace a proveden rozbor současného stavu. Tudíž lze říci, že byl splněn první dílčí cíl. Druhý dílčí cíl otestování navrženého algoritmu na několika třídách testovacích funkcí byl splněn v kapitole 5. Třetí dílčí cíl byl splněn aplikací na problém obchodního cestujícího v kapitole 6.

Přínos doktorské disertační práce

Teoretickým přínosem doktorské disertační práce je vytvoření nového algoritmu penalizačního lokálního prohledávání z oblasti globální optimalizace, která spojuje výhody metod lokálního prohledávání, simulovaného žíhání a zakázaného prohledávání.

Přínosem pro společenskou praxi je aplikovatelnost tohoto algoritmu na různé reálné problémy spočívající v kombinatorické optimalizaci, která svoji povahou bývá NP-těžká. Aplikovatelnost algoritmu byla dokumentována na problému obchodního cestujícího. Také bylo ukázáno, že navržený algoritmus není složité adaptovat na některé typy problémů, kde je vhodné využít lokálního prohledávání.

Možnosti pokračování

Tato doktorská práce se zabývala využitím metody penalizovaného lokálního prohledávání, která sama o sobě nabízí mnoho témat k dalšímu rozpracování. Jednou z možných cest je implementace záporných penalizací, jež by mohly zajímavým způsobem

podpořit další chování algoritmu. Další možnou úpravou algoritmu by mohlo být automatické nastavování parametru λ, protože jak vyplývá z výsledků, různá velikost tohoto parametru přináší i různé výsledky v různých částech průběhu hledání optima. Další možností je automatické nastavování omezujícího kritéria, které zastavuje algoritmus a výzkum by mohl být věnován i funkci využitelnosti, na jejímž základě se vybírá ten argument účelové funkce, který bude penalizován.

Další možností je zdokonalení aspiračního kritéria. Jedna z perspektivních možností úpravy aspiračního kritéria je, že by hodnoty penalizací byly ignorovány v momentě, kdy by bylo nalezeno takové řešení, jež by mělo lepší kvalitu než nejhorší řešení z nejlepších nalezených. Aspirační přesun do množiny aktuálně nejlepších řešení by tímto nebyl umožněn, čímž by se zamezilo prozkoumávání již jednou zkoumaných řešení a v průběhu prozkoumávání celého problému by bylo možné měnit počet aspiračních přesunů. Tak by bylo možné zjistit, zda různá velikost aspiračních přesunů nemá vliv na celkovou kvalitu nalezených řešení a zda by tím mohlo být dosaženo lepších výsledků. Pokud by tato zlepšení měla nějaký efekt, pak bychom mohli očekávat od algoritmu penalizovaného lokálního prohledávání kvalitnější prozkoumání ve zkoumaném terénu daného problému.

Seznam použité literatury

[AART89] AARTS, E., KORST, J. *Simulated Annealing and Boltzmann Machines.* Chichester: J. Wiley and Sons, 1989. 272 s. ISBN 0-471-92146-7.

[AMBE00] AMBERG, A., DOMSCHKE, W., VOSS, S. Multiple Center Capacitated Arc Routing Problems: a Tabu Search Algorithm using Capacitated Trees. In *European Journal of Operational Research*, Elsevier, 2000. s. 360-376. ISSN 0377-2217.

[AMIN99] AMIN, A. Simulated Jumping. In *Annals of Operation Research*, volume 86, number 1, Springer. 1999. s. 23 – 38.

[BADE97] BADEAU, P., et al. *A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows.* In *Transportation Research - C 5*, 1997. s. 109-122.

[BÄCK91] BÄCK, T., HOFFMEISTER, F., SCHWEFEL, H. P. A Survey of Evolution Strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo (CA): Morgan Kaufmann, 1991. s. 2-9.

[BATT93] BATTITI, R., TECCHIOLLI, G. The Reactive Tabu Search. In *ORSA Journal on Computing 6(2)* 1993. s.126-140.

[BATT95] BATTITI, R., TECCHIOLLI, G. Local search with memory: Benchmarking RTS. In *Operations Research Spectrum*,. Berlin: Springer, 1995. s. 67-86, ISSN 0171-6468.

[BENT97] BENTLEY, J.J. Fast algorithms for geometric traveling salesman problems. In *ORSA Journal on Computing 4*, 1997, s. 387-411.

[BERT82] BERTSEKAS, D. P. *Constrained Optimization and Lagrange Multiplier Methods.* Londýn: Academic Press, 1982. 410 s. ISBN 1-886529-04-03

[BISH95] BISHOP, C. M. *Neural Networks for Pattern Recognition.* Oxford University Press, New York, 1995. 504 s. ISBN 0-19-853849-9.

[BOSE96] BOSE N. K., LIANG P. Neural Network Fundamentals with Graphs, Algorithms and Applications. In *Electrical and Computer Engineering*, 1996. ISBN 0-07-006618-3.

[BURK97] BURKARD, R.E., KARISCH, S.E. QAPLIB – A Quadratic Assignment Problem Library. In *Journal of Global Optimization 10*, 1997 s. 391-403.

[CENE94] CENEK, P., KLÍMA, V., JANÁČEK, J. *Optimalizace dopravních a spojových procesů*. Vysoká škola dopravy a spojov v Žilině, Žilina, 1994. ISBN 80-7100-197-X.

[CROE58] CROES, A. A Method for Solving Travelling-Salesman Problems. In *Operations Research 5.* 1958, s. 791-812.

[CRES95] CRESCENZI P., KANN. V. *A compendium of NP optimization problems*. Technical report SI/RR-95/02, Dipartimento di Scienze dell´Informazione, Universita di Roma „La SapienzaL, 1995.

[DAVE94] DAVENPORT, A., TSANG, E., WANG, C. J., ZHU, K. GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *Proceeding of AAAI-94*, 1994. s. 325-330.

[DAVI87] DAVIS L., STEENSTRUP M. *Genetic Algorithms and Simulated Annealing.* Los Altos, CA: Morgan Kaufmann Publishers, 1987.

[DEVL03] DEVLIN, K. *Jazyk matematiky: Jak zviditelnit neviditelné*. Praha: Dokořán, 2003. ISBN: 80-86569-09-8.

[DEVL05] DEVLIN, K. *Problémy pro třetí tisíciletí: Sedm největších nevyřešených otázek matematiky.* 1. vyd. Praha: Dokořán, 2005. ISBN 80-7363-016-8.

[DOWS93] DOWSLAND, K. A. Simulated Annealing. In *Reeves, C. R. (ed.), Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientic Publishing, 1993. s. 20-69.

[DORI96] DORIGO, M., MANIEZZO, V. AND COLORNI, A. The Ant System: Optimization by a colony of cooperating agents. In *IEEE Transactions on Systems and Cybernetics - Part B*, Vol. 26, No. 1, 1996. s. 1-13.

[ENGE05] ENGELBRECHT, A. P. *Fundamentals of Computational Swarm Intelligence.* Chichecster: John Wiley & Sons Ltd, 2005. ISBN 0-470-09191-6

[FLEU94] FLEURENT, C., FERLAND, J.A. Genetic Hybrids for the Quadratic Assignment Problem. In *Quadratic Assignment and Related Problems,* Vol. 16, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1994. s. 173-187.

[FOX93] FOX, B.L. Integrating and accelerating tabu search, simulated annealing and genetic algorithms. In *Annals of Operations Research 41*, 1993. s. 47-67.

[FRAN96] FRANK, J. Weighting for Gobot: Learning Heuristic for GSAT. In *Proceeding of AAAI-96,* Vol. 1, 1996. s. 338-343.

[GAMB99] GAMBARDELLA, L.M., TAILLARD,E.D., DORIGO,M. Ant Colonies for the QAP. In *The Journal of the Operational Research Society,* vol 50, 1999. s. 167-176.

[GARE79] GAREY, M. R., JOHNSON, D. S. *Computers and Intractability: a guide to the Theory of NP-Completeness.* W. H. Freeman, New York, 1997. ISBN 0-7167-1044-7.

[GASS04] GASS, S.I., HARRIS, C.M. *Encyclopedia of Operations Research and Management Science centennial edition*, druhé vydání, Dodrecht, Kluwer AP, 2004. ISBN 0-7923-7827-X

[GLOV89] GLOVER, F. Tabu search Part I. In *Journal on Computing,* Vol. 1, Operations Research Society of America (ORSA), 1989, s. 109-206.

[GLOV90] GLOVER, F. Tabu search Part II. In *Journal on Computing*, Vol. 2, Operations Research Society of America (ORSA), 1990, s . 4 - 32,

[GLOV93] GLOVER, F., TAILLARD, E., DE WERRA, D. A user's guide to tabu search. In *Annals of Operations Research 41,* 1993 s. 3 – 28.

[GLOV97] GLOVER, F., LAGUNA, M. *Tabu Search.* Norwelll, MA: Kluwer Academic Publishers, 1997. 382 s. ISBN 0-7923-8187-4.

[GOLD89] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA: Addison – Wesley, 1989. 432 s.ISBN 0-2011-5767-5.

[GRIG06] GRIGORENKO, I. *Optimal Control and Forecasting of Complex Dynamical Systems.* Singapore: World Scientific Publishing, 2006. ISBN 981-256-660-0.

[HANS98] HANSEN, P.C. *Rank-deficient and discrete ill-posed problems: Numerical aspects of linear inversion*, Philadelphia, PA: SIAM, 1998. ISBN 0-8987-1403-6.

[HARP94] HARP, S. A., SAMAD, T. Genetic Optimization of Self-Organizing Feature Maps. In *Proc. Int. Conf. on Neural Networks*, s. 341 – 346, 1994.

[HAYK94] HAYKIN, S. Neural Networks. 2. vyd. New York: Prentice-Hall, 1994. ISBN 0-13-273350-1.

[HEDA06] HEDAR, Abdel-Rahman. *Test Functions for Unconstrained Global Optimization* [online]. [2006] [cit. 2007-09-10]. Dostupný z WWW: <http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm>.

[HOLL75] HOLLAND, J.H. *Adaptation in Natural and Artificial Systems.* Ann Arbor (MI): The University of Michigan Press, 1975. ISBN 0-2625-8111-6.

[HOLL92] HOLLAND, J.H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan: The MIT Press, 1992. 228 s. ISBN 0-262-58111-6

[CHAMB95] CHAMBERS L. *Practical Handbook of Genetic Algorithms: New Frontiers*. CRC Press, New York, 1995. ISBN 0-8493-2519-6

[CHOI88] CHOI, K.M.F., LEE, J.H.M., STUCKEY, P.J. A Lagrangian Reconstruction of a Class of Local Search Methods. In *Proceedings of 10th IEEE International Conference on Tools with Artificial Intelligence*, IEEE Press, Taipei, Taiwan, ROC, s. 166-175, 1988.

[JANA99] JANÁČEK, J. *Matematické programování.* Žilina: Žilinská univerzita v Žilině, 1999. ISBN 80-7100-573-8.

[JANA02] JANÁČEK, J. *Optimalizace na dopravních sítích.* Žilina: Žilinská univerzita v Žilině, 2002. ISBN 80-8070-031-1.

[JIŘI02] JIŘINA, M. *Preprocessing of Initial Weights in the SOM*. Neural Network World, 2002. ISBN 80-01-01814-8

[JOHN89] JOHNSON, D., ARAGON, C., MCGEOCH, L., SCHEVON, C. Optimisation by simulated annealing: an experimental evaluation, part I, graph partitioning. In *Operations Research,* Vol 37, 1989. s. 865-892.

[JOHN90] JOHNSON, D. Local Optimisation and the Travelling Salesman Problem. In *Proceedings of the 17th Colloquium on Automata Languages and Programming*, Lecture Notes in Computer Science, 443, Springer-Verlag, 1990. s. 446-461.

[JOHN03] JOHNSON, D. S., MCGEOCH, L.A. The Travelling Salesman Problem: A Case Study in Local Optimisation. In *Local Search in Optimisation*, Chichester: Wiley, 2003.

[KIRK83] KIRKPATRICK, S., GELATT, C. D., VECCHI, M. P. Optimisation by Simulated Annealing. In *Science,* Vol. 220, 1983. s. 671-680.

[KNOX94] KNOX, J. Tabu Search Performance on the Symmetric Travelling Salesman Problem. In *Computers Ops Res.,* Vol. 21, No. 8, 1994. s. 867-876.

[KOOP57] KOOPMAN, B. O. The Theory of Search, Part III. The Optimum Distribution of Searching Effort. In *Operations Research*, 5, 1957. s. 613-626

[KVAS96] KVASNICKA, V, PELIKAN, M., POSPICHAL J. Hill Climbing with Learning /An abstraction of Genetic Algorithm, In *Neural Network World*, 5, 1996. s. 773-796.

[KVAS00] KVASNIČKA V., POSPÍCHAL J., TIŇO P. *Evolučné algoritmy*. Bratislava: STU Bratislava, 2000. ISBN 80-227-1377-5.

[LARR99] LARRANAGA, P., et al. Genetic Algorithms for the Traveling Salesman Problem. In *Artificial Intelligence Review,* 13, 1999, s. 129-170.

[LAPO92] LAPORTE, G. The Travelling Salesman Problem: An overview of exact and approximate algorithms. In *European Journal of Operational Research,* 59, 1992. s. 231-247.

[LAŠČ90] LAŠČIAK, A. *Optimálne programovanie.* Bratislava: Alfa, vydavatelstvo technickej a ekonomickej literatúry, 1990. 2. vyd. ISBN 80-05-00663-2.

[LAWL89] LAWLER, E.L., et al. *The Travelling Salesman Problem: a guided tour in combinatorial optimisation.* John Wiley & Sons, 1985. ISBN 978-0-471-90413-7.

[LIN65] LIN, S. Computer Solutions of the Travelling-Salesman Problem. In *Bell Systems Technical Journal 44,* 1965. s. 2245-2269.

[LIN73] LIN, S., KERNIGHAN, B. W. An effective heuristic algorithm for the travelling salesman problem. In *Operations Research 21, 1973.* s. 498-516.

[LIND08] LINDA, Bohdan. *Lineární programování*. Pardubice : Univerzita Pardubice, v tisku. 200 s. ISBN 978-80-7395-038-5

[LOAP05] LOAP, H. S., COELHO, L. S. Particle Swarm Optimization with Fast Local Search for the Blind Traveling Salesman Problem. In *Hybrid Intelligent System: proceeding of the fifth International Conference 6 – 9 November,* Rio de Janeiro, Brazil, *2005,* s. 245-250.

[LUEN84] LUENBERGER, D. G. *Linear and Nonlinear Programming.* Reading, MA: Addison-Wesley Publishing Company, 1984. ISBN 1-4020-7593-6

[LUND86] LUNDY, M., MEES, A. Convergence of an annealing algorithm. In *Mathematical Programming*, 34, 1986, s. 111-124.

[MAŘÍ01a] MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J. a kol. *Umělá inteligence (3).* Praha: Academia, 2001. ISBN 80-200-0472-6

[MAŘÍ01b] MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J. a kol. *Umělá inteligence (4).* Praha: Academia, 2001. ISBN 80-200-1044-0.

[MERZ99] MERZ, P. and FREISLEBEN, B. *A Comparison of Memetic Algorithms, Tabu Search, and Ant Colonies for the Quadratic Assignment Problem*. In *Proceedings of the 1999 International Congress of Evolutionary Computation (CEC'99)*, IEEE Press, 1999. s. 2063-2070.

[MICH92] MICHALEWICZ, Z. *A Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer Verlag, 1992. 354 s. ISBN 3-540-58090-5

[MICH00] MICHALEWICZ, Z, Fogel, D. B. *How to Solve It: Modern Heuristics*. Springer-Verlag, 2000. ISBN 3-540-66061-5.

[MINT92] MINTON, S., et al. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. In *Artificial Intelligence*, 58, 1992. s. 161-205.

[MISH06] MISHRA, S. Some new test functions for global optimization and performance of repulsive particle swarm method. In *MPRA Paper No. 2718,* Munich: University Library of Munich, 2006.

[MITC98] MITCHELL, M. *An introduction to genetic algorithms.* Cambridge (Massachusetts): The MIT Press, 1998. 221 s. ISBN 0-262-63185-7.

[MLAD97] MLADENOVIC, N. & HANSEN, P. Variable Neighborhood Search. In *Computers in Operations Research*, Vol. 24, No. 11, 1997. s. 1097-1100.

[MLAD03] Mladenovic, N., Petrovic, Kovacevic-Vujcic, V., Cangalovic, M. Solving a spread-spectrum rada polyphase code design problem by tabu search and variable neighbourhood search. In *European journal of Operations research* , Vol. 151, 2003. s. 389-399.

[MORR93] MORRIS, P. The breakout method for escaping from local minima. In *Proceedings of AAAI-93,* 1993. s. 40-45.

[MOSC89] MOSCATO, P. *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms.* C3P Report 826, Caltech Concurrent Computation Program, Caltech, California, USA, 1989.

[MOSC93] MOSCATO, P. An introduction to population approaches for optimization and hierarchical objective functions: a discussion on the role of tabu search. In *Annals of Operations Research* 41, 1993. s. 85-121.

[OLEJ03] OLEJ, V. *Modelovanie ekonomických procesov na báze výpočtovej inteligencie.* Hradec Králové: M&V, 2003. 160 s. ISBN 80–90324–9-1

[OSMA93] OSMAN, I. H. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems. In *Annals of Operations Research*, 41, 1993. s. 421-451.

[OSMA95] OSMAN, I. H. An Introduction to Meta-Heuristics. M. Lawrence and C. Wilson (eds.), In *Operational Research Tutorial Papers,* Operational Research Society Press, Birmingham, UK, 1995. s. 92-122.

[PANU05] PANUŠ, J., ŠIMONOVÁ, S. Pre-Computiation of Regional Data for Optimization Analysis. In *Eurocon 2005: IEEE Catalog Number 05EX1255C*. 1st edition. Beograd, Serbia and Montenegro: School of Electrical Engineering, Beograd, 2005. s. 1–4. ISBN 1–4224–0050–3.

[PANU06a] PANUŠ, J., ŠIMONOVÁ, S. Measurability of evaluative criteria used by optimization methods. In *Proceedings of Conference on Computional, Intelligence, Man-Machine Systems and Cybernetics*. 1st edition. Dallas, Texas: Wseas Press, 2007. s. 451-455. ISBN 960-8457-55-6.

[PANU07a] PANUŠ, J. Komparace vybraných stochastických optimalizačních algoritmů. In *Aktuální otázky rozvoje regionů 2007*. 1. vyd. Seč u Chrudimi: Univerzita Pardubice, 2007. ISBN 978–80–7194–9.

[PANU07b] PANUŠ, J. Vliv parametru lambda na chod algoritmu penalizačního lokálního prohledávání. In *Scientific Papers: Sborník vědeckých prací Univerzity Pardubice series D*. 1. vyd. Pardubice : Univerzita Pardubice, 2007. s. 150-157. ISSN 1211-555X.0

[PANU07c] PANUŠ, J., ŠIMONOVÁ, S. Genetic algorithms for optimization of thematic clusters. In *Eurocon 2007 - IEEE : Catalog Number: 07EX1617C*. 1st edition. Warzsawa : Warsaw University of Technology, 2007. s. 1–6. ISBN 1–4244–0813-X.

[PANU07d] PANUŠ, J., ŠIMONOVÁ, S. Stochastic optimization methods for the investment strategy. *WSEAS Transactions on Mathematics*. 2007, vol. 6, is. 1, s. 111–118.

[PAPA82] PAPADIMITRIOU, C. H., STIEGLITZ, K. *Combinatorial Optimisation: Algorithms and Complexity.* Prentice – Hall, 1982. ISBN 0-486-40258-4.

[PARD87] PARDALOS, P. M., ROSEN, J. B. Constrained Global Optimization: Algoritms and Applications, volume 268 of Lecture Notes in *Computer Science*. Berlin: Springer-Verlag, 1987.

[REEV93] REEVES, C.R., BEASLY, J.E. Chapter 1: Introduction. In Reeves, C. (ed.), *Modern Heuristic Techniques for Combinatorial Problems.* Blackwell Scientific Publishing, 1993. s. 1-19.

[REEV96] REEVES, C. R. Modern Heuristic Techniques. In Rayward-Smith, V. J., Osman, I. H., Reeves, C. R. and Smith, G. D. (ed.), John Wiley & Sons. *Modern Heuristic Search Methods*, 1996. s. 1-26.

[REIN91] REINELT, G. A Travelling Salesman Problem Library. In *ORSA Journal on Computing*, 3, 1991. s. 376-384.

[ROJA96] ROJAS, R. *Neural Networks: A Systematic Introduction.* Berlín: Springer, 1996. 522 s. ISBN 3-5406-0505-3

[RUSS03] RUSSELL, S., NORVIG, P. *Artificial Intelligence. A Modern Approach.* Second Edition. New Jersey: Prentice Hall, 2003. ISBN 0-13-790395-2.

[SEIF02] SEIFFERT U., LAKHMI, C. J. *Self-Organizing Neural Networks Recent Advances and Applications.* Physica – Verlag, New York, Heidelberg, 2002. ISBN 3-7908-1417-2.

[SELM93a] SELMAN, B., KAUTZ, H. Domain independent versions of GSAT solving large structured satisfiability problems. In *Proceeding of IJCAI-93*, 1993. s. 290-295.

[SELM93b] SELMAN, B., KAUTZ, H. An Empirical Study of Greedy Local Search for Satisfiability Testing. In *Proceeding AAAI-93*, 1993. s. 46-51.

[SCHW81] SCHWEFEL, H. P. *Numerical Optimization for Computer Models.* Chichester (UK): Wiley, 1981. 398 s. ISBN 0–4710–9988–0.

[SCHW95] SCHWEFEL, H.P. *Evolution and Optimum seeking.* New York: Wiley & Sons, 1995. ISBN 0–471–57148–2.

[SJOB94] Sjöberg, J., et. al. *Neural Network in System Identification.* Report LiTH-ISY-R-1622, 1994.

[STON83] STONE, L. D. The Process of Search Planning: Current Approaches and Continuing Problems. In *Operations Research, 31*, 1983, s. 207-233.

[STÜT97] STÜTZLE T. *MAX-MIN Ant System for Quadratic Assignment Problems*. ResearchReport AIDA-97-04, Department of Computer Science, Darmstadt University of Technology, Germany, 1997.

[STÜT99] STÜTZLE T. *Iterated Local Search for the Quadratic Assignment Problem*. Research Report AIDA-99-03, Department of Computer Science, Darmstadt University of Technology, Germany, 1999.

[TAIL91] TAILLARD, E.D. Robust Tabu Search for the Quadratic Assignment Problem. In *Parallel Computing, 17*, 1991 s. 443–455.

[TAIL95] TAILLARD, E.D. Comparison of Iterative Searches for the Quadratic Assignment Problem. In *Location Science 3*, 1995, s. 87-105.

[TANG92] TANG, E.P.K., WANG, C.J. A Generic Neural Network Approach for Constraing Satsifaction Problems. In *Neural network applications.* Springer-Verlag, 1992, s.12-22.

[TAUF98] TAUFER, I., DRÁBEK, O. Umělé neuronové sítě jako prostředek pro modelování nelineárních soustav. In *Acta Montanistica Slovaca.* 1998, Volume 8, Number 4. s. 489-494.

[TAYL97] TAYLOR, J.C. *An Introduction to Measure and Probability.* New York: Springer-Verlag, 1997. 320 s. ISBN 0–3879–4830–9

[TIKH77] TIKHONOV, A. N., ARSENIN, V. Y. *Solutions of Ill-Posed Problems.* Washington: Winston & Sons, 1977. 224 s. ISBN 0–4709–9124–0.

[THRU92] THRUN, S. B. *Efficient exploration in reinforcement learning*. Technical report CMU-CS-92-102, School of Computer Science, Carnegie-Mellon University, 1992.

[TSAN93] TSANG, E. *Foundations of Constraint Satisfaction.* Londýn: Academic Press, 1993. 300 s. ISBN 0-1270-1610-8.

[VOUD98] VOUDOURIS, C., TSANG, E. Guided Local Search and Its Application to the Traveling Salesman Problem. In *European Journal of Operational Research, 1998, s. 469-499*

[WANG91a] WANG, C. J., TSANG, E. Solving constraint satisfaction problems using neural networks. In *Proceeding of IEE Second International Conference on Artificial Neural Networks*, 1991, s. 295-299.

[WANG91b] WANG, T. *Global Optimization for Constrained Nonlinear Programming.* Thesis, Zhejiang University, 1991.

[WANG06] WANG, F.Y., LIU, D. *Advances in Computational Intelligence- theory and applications.* Singapore: WSP, 2006. ISBN 981-256-734-8

[WHIT90] WHITLEY D., STARKWEATHER T., BOGART C. Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity. In *Parallel Computing, No. 14*, 1990, s. 347 – 361.

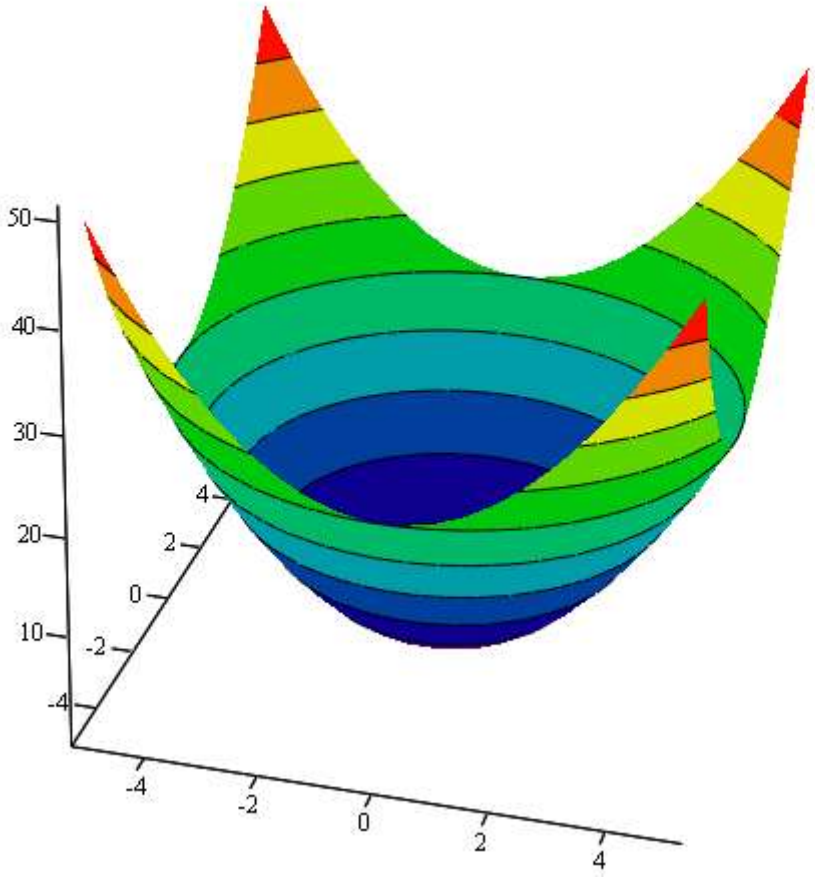
[ZELI02] ZELINKA, I. *Umělá inteligence v problémech globální optimalizace.* Praha: BEN – technická literatura, 2002. ISBN 80-7300-069-5.

Seznam tabulek

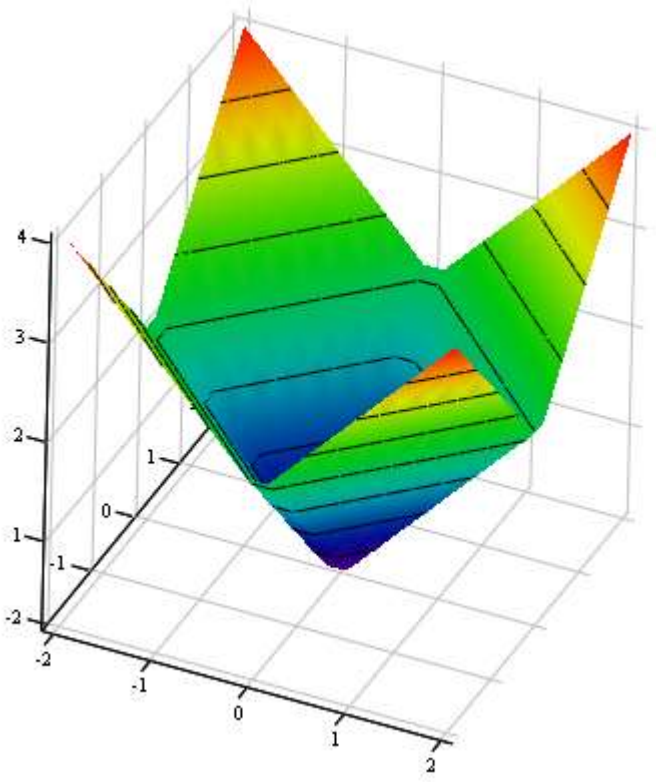
Seznam obrázků

Přílohy

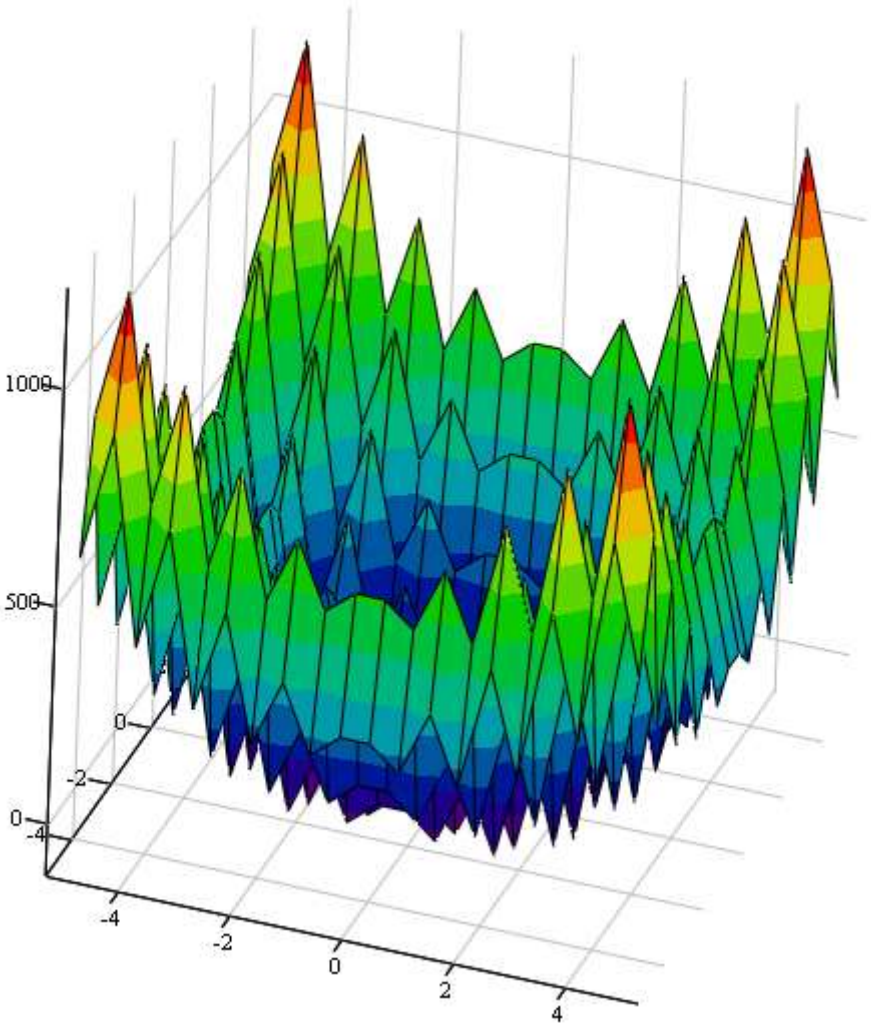
Příloha A – Vizualizace funkce Sphere model – 1st De Jong Function



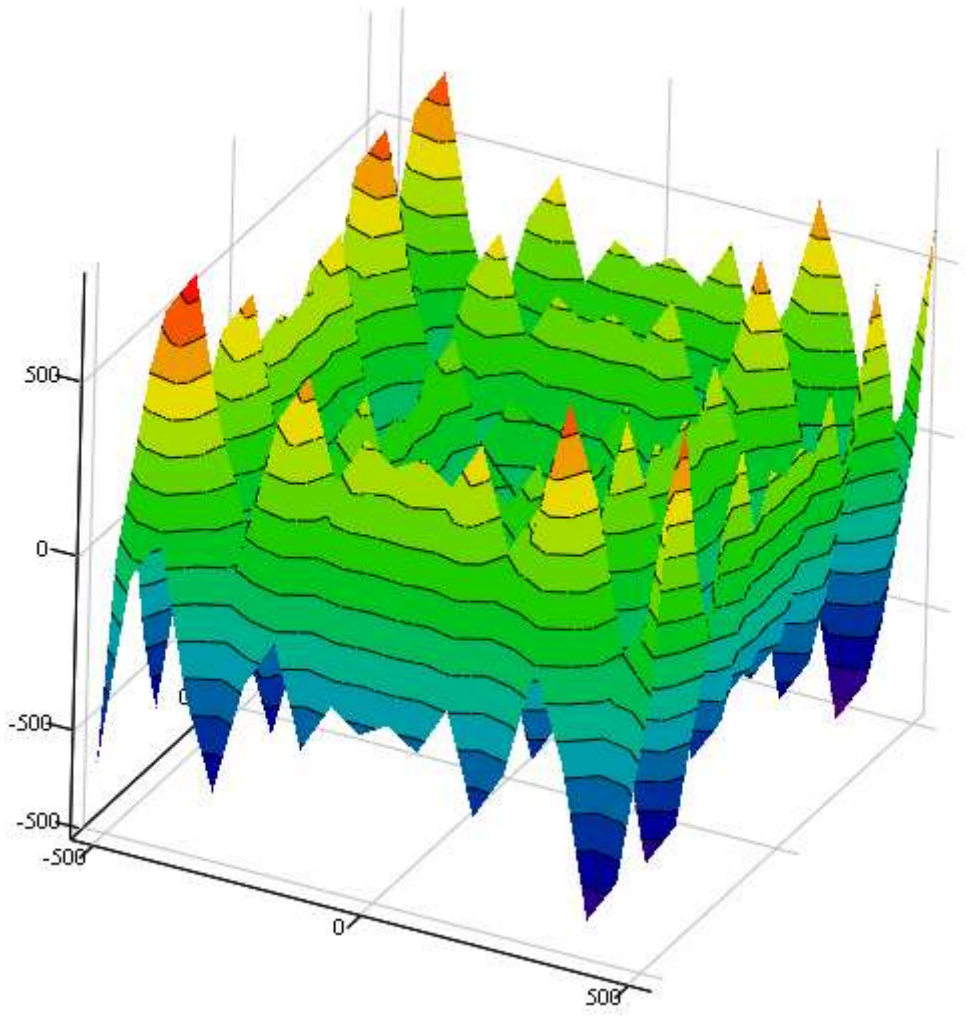
Příloha B – Vizualizace funkce 3rd De Jong Function



Příloha C – Vizualizace funkce Rastrigin



Příloha D – Vizualizace funkce Schwefel

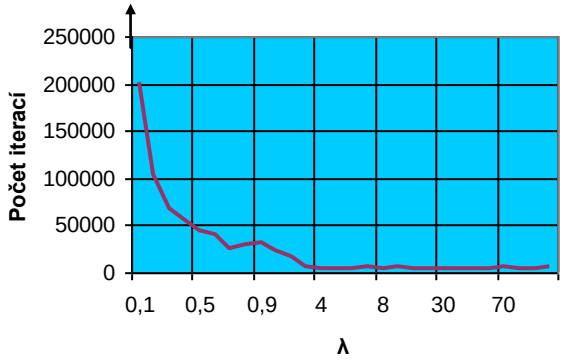


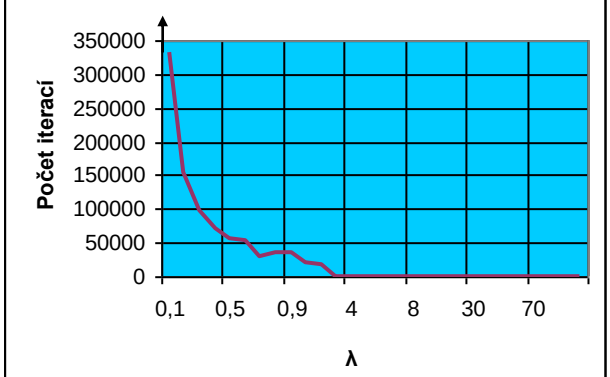
Příloha E – Naměřené hodnoty pro různé testovací funkce

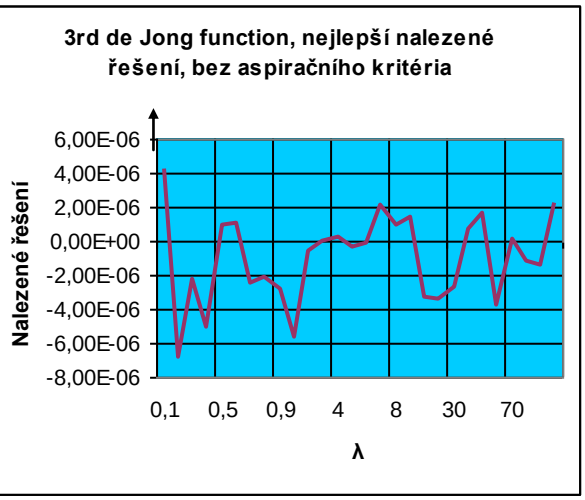
3rd De Jong

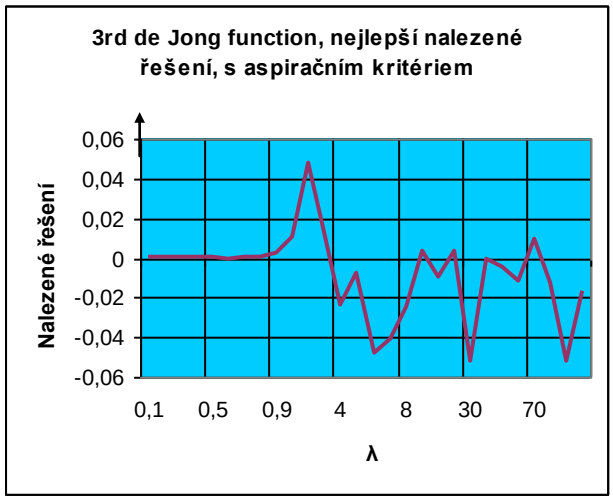
$$\sum_{i=1}^{n} Abs\left(x_i\right) \quad -2 \le x_i \le 2$$

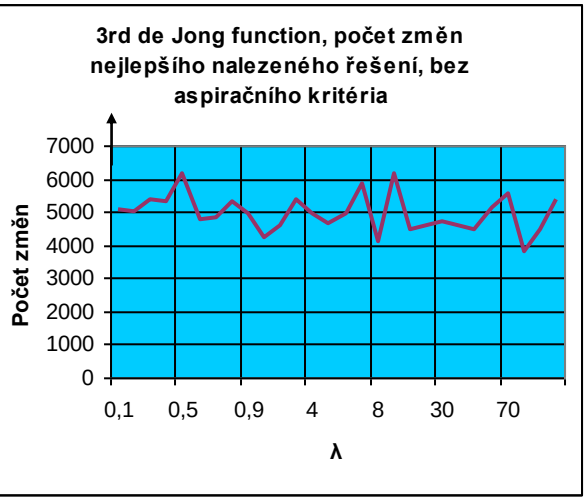
Globální minimum $f(x) = 0, x(i) = 0; i = 1...n$

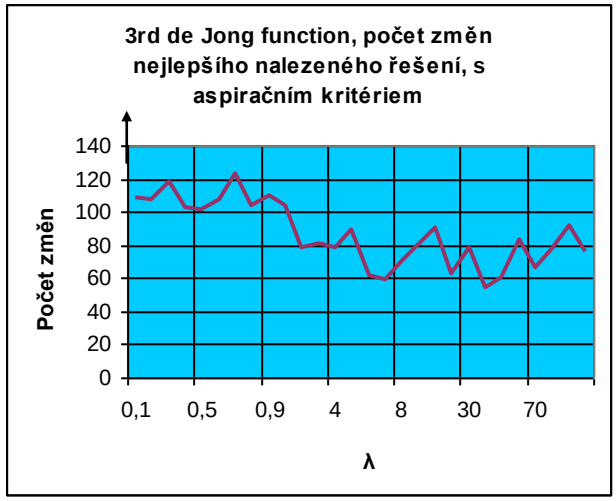


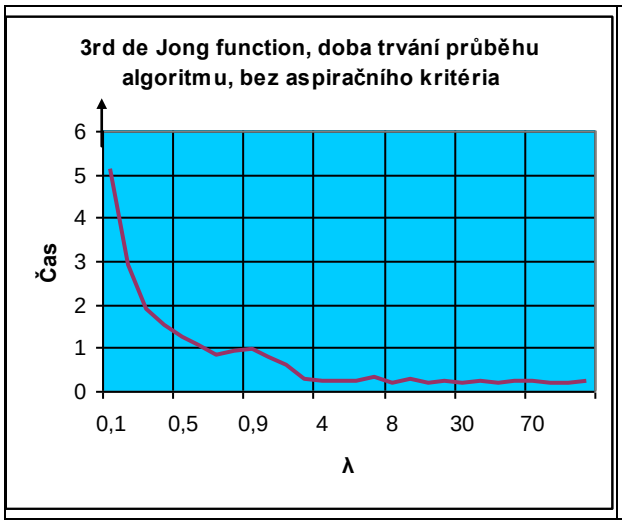


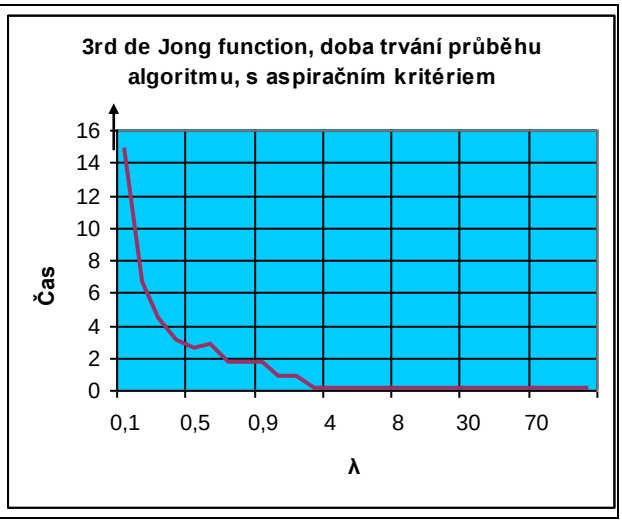






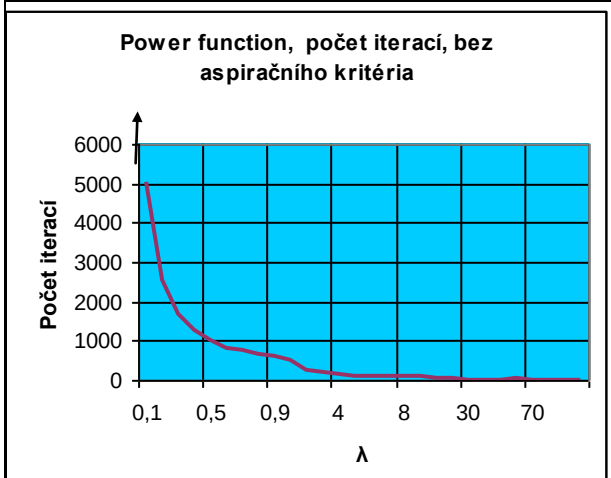


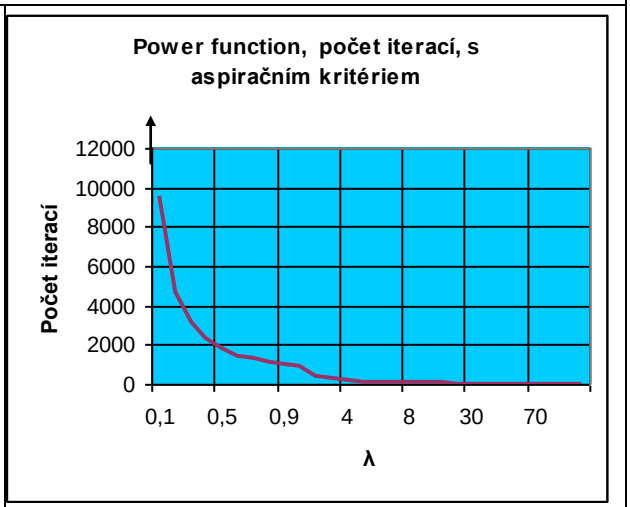


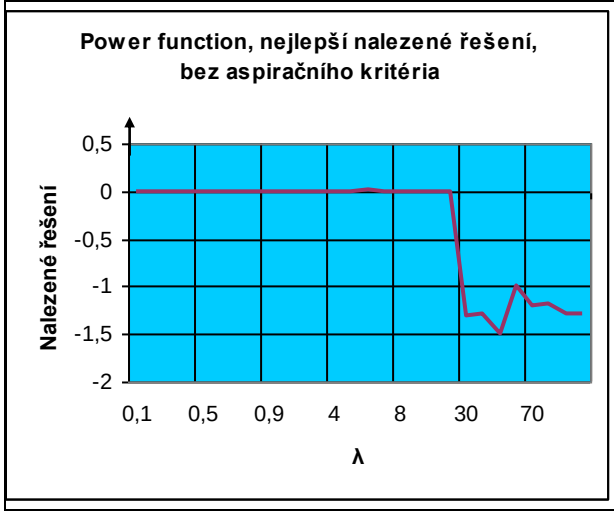


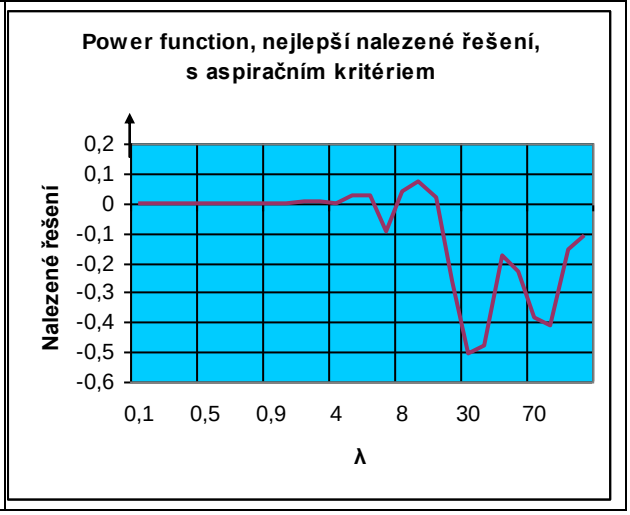
Power function

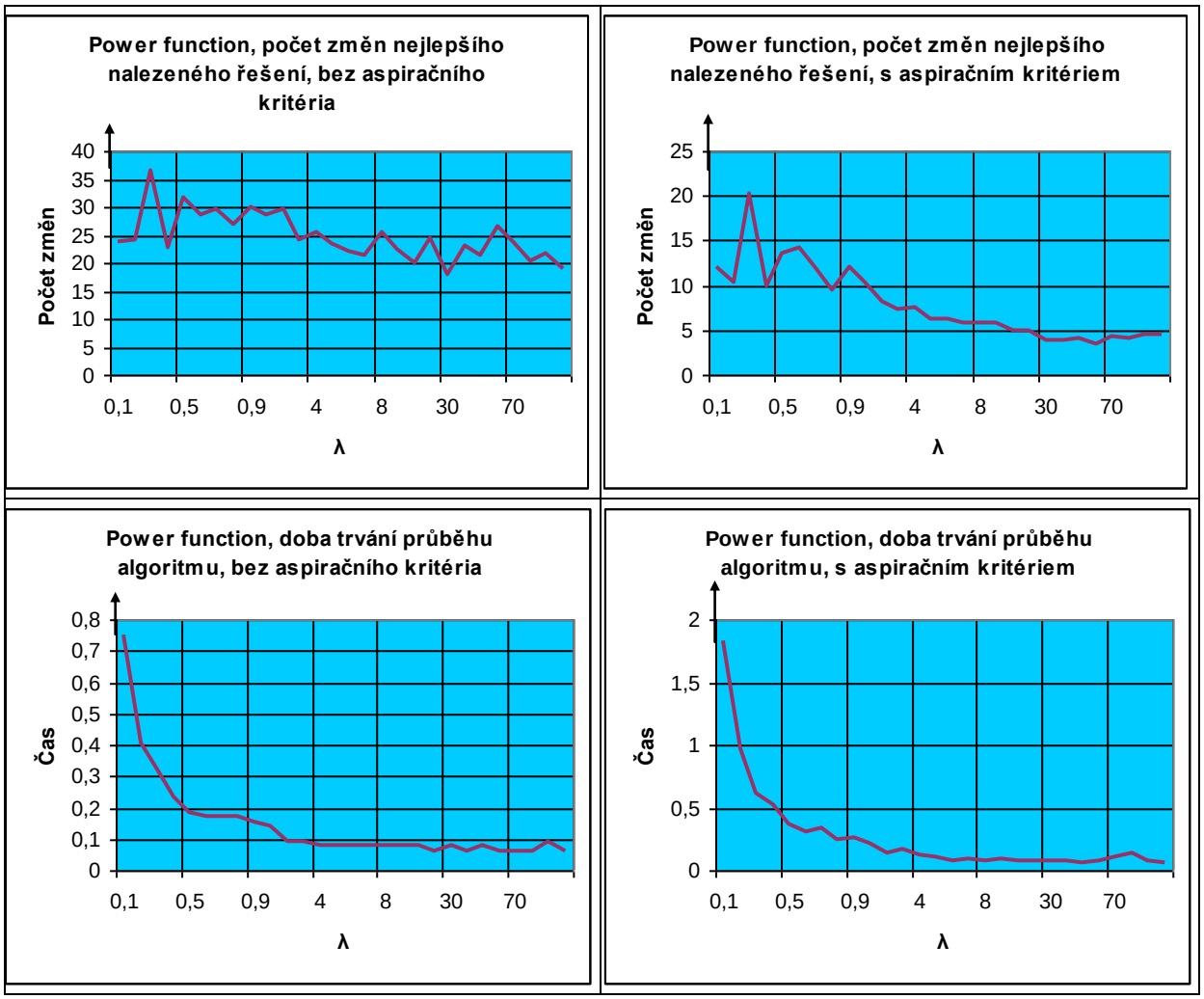
$$\sum_{i}^{n} \left| x_i \right|^{i+1}; -1 \le x_i \le 1$$







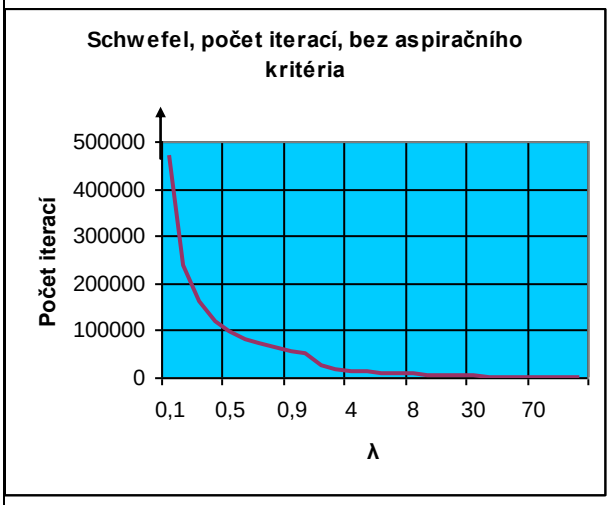


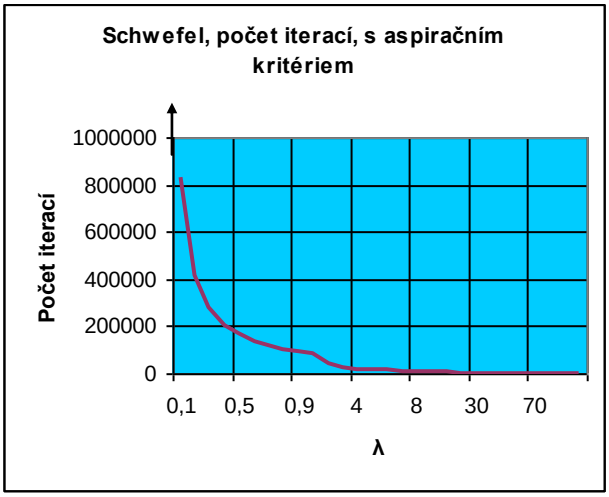

Power function, počet změn nejlepšího nalezeného řešení, bez aspiračního kritéria
Počet změn
40
35
30
25
20
15
10
5
0
0,1
0,5
0,9
4
8
30
70
λ
Power function, počet změn nejlepšího nalezeného řešení, s aspiračním kritériem
Počet změn
25
20
15
10
5
0
0,1
0,5
0,9
4
8
30
70
λ
Power function, doba trvání průběhu algoritmu, bez aspiračního kritéria
Čas
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
0,1
0,5
0,9
4
8
30
70
λ
Power function, doba trvání průběhu algoritmu, s aspiračním kritériem
Čas
2
1,5
1
0,5
0
0,1
0,5
0,9
4
8
30
70
λ

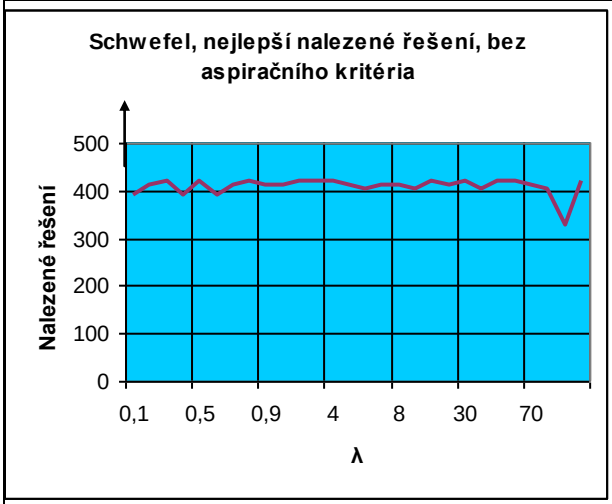
Schwefel function

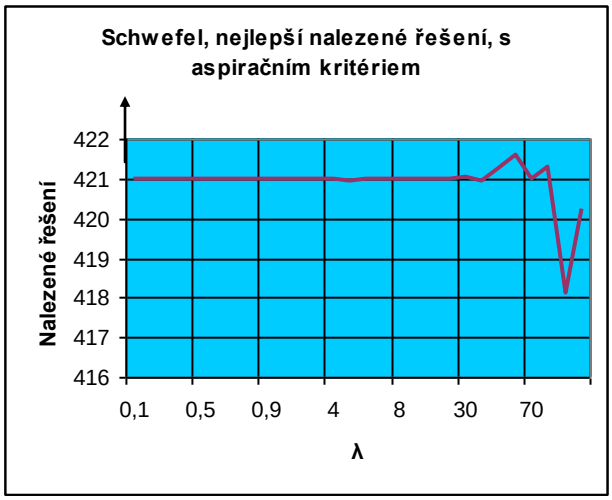
$$\sum_{i=1}^{n} -x_i \sin\left(\sqrt{|x_i|}\right), \quad -512 \le x_i \le 512$$

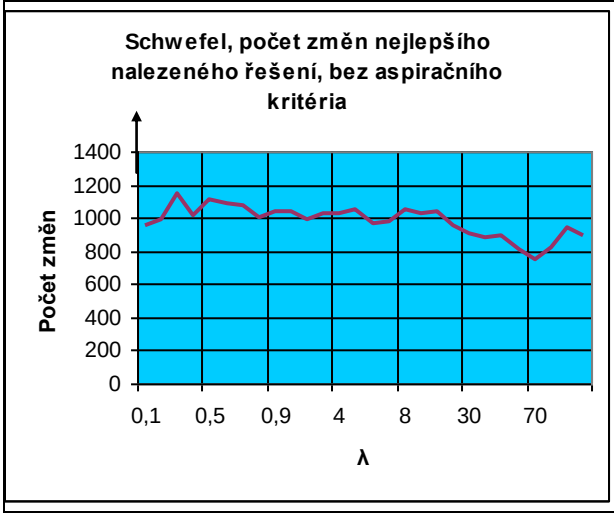
Globální minimum *f(x) = -n*418,9829, x(i)= 420,9687; i = 1...n*

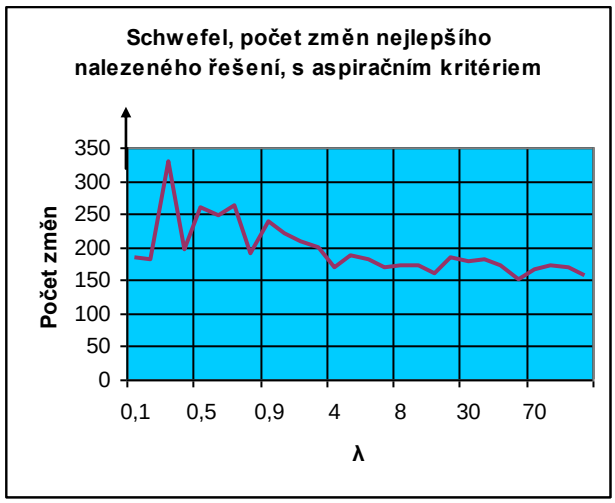


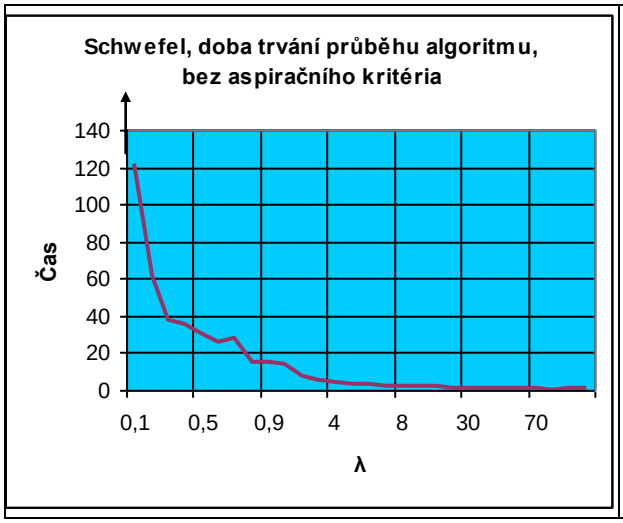


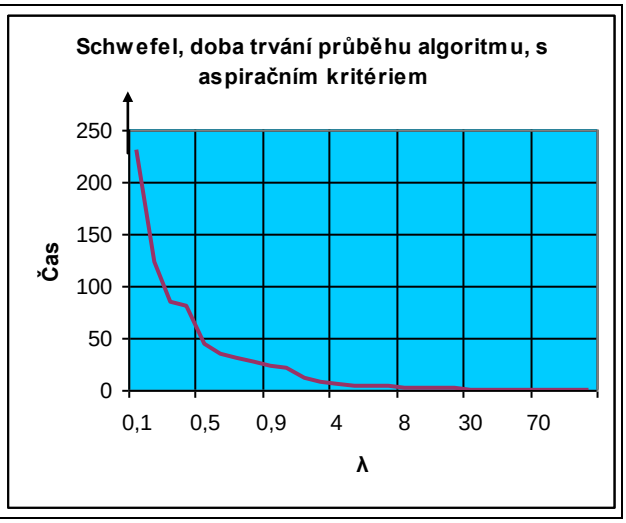








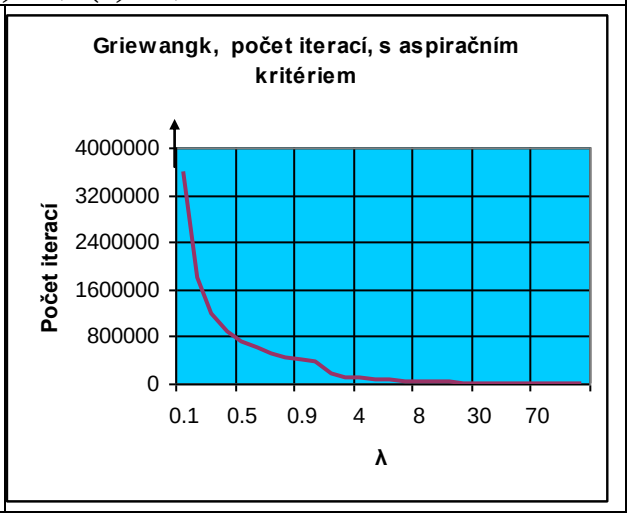


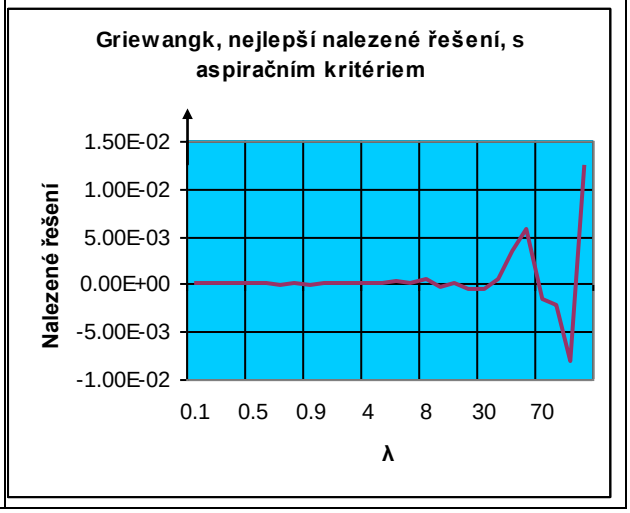


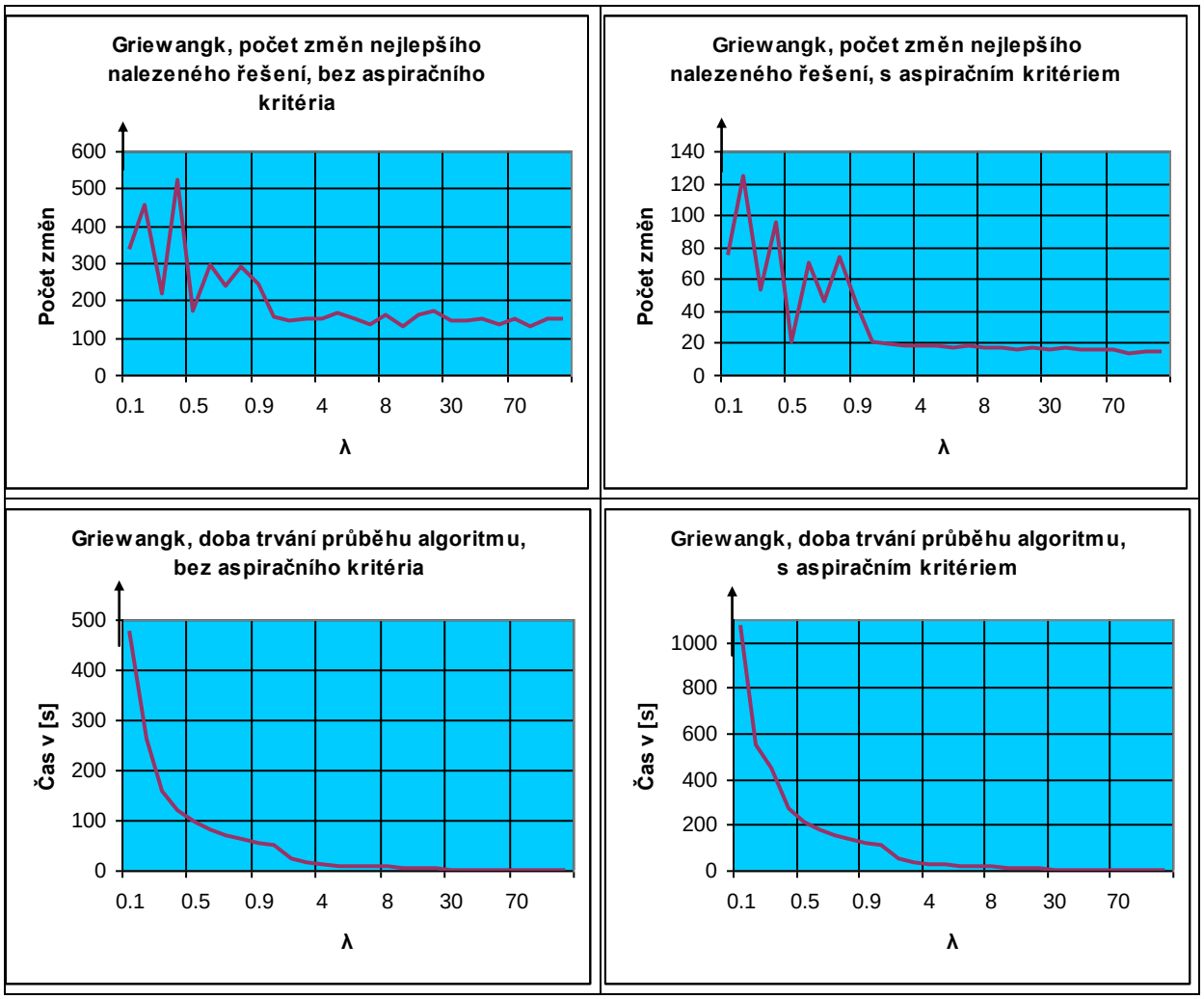
Griewangk's function

$$-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+\sum_{i=1}^{n}\frac{x_i^2}{4000}+1, \quad -600 \le x_i \le 600$$

Globální minimum *f(x) =0, x(i)= 0; i = 1...n*

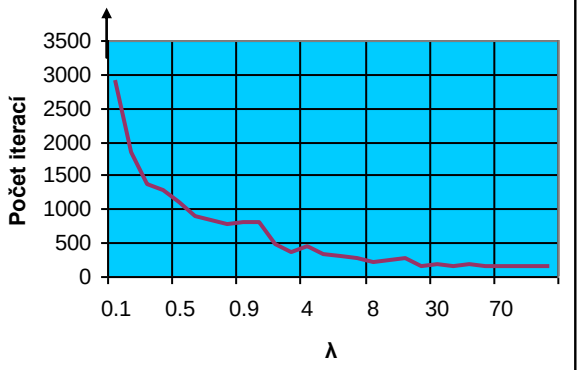


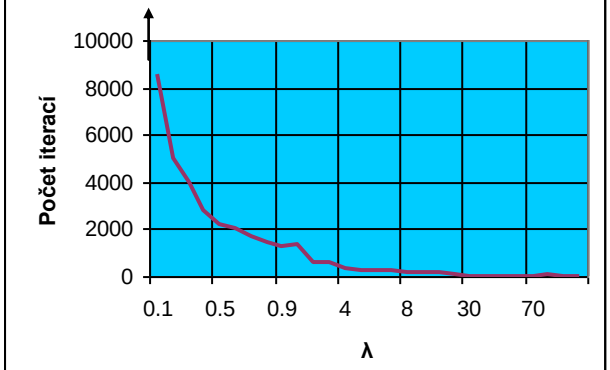


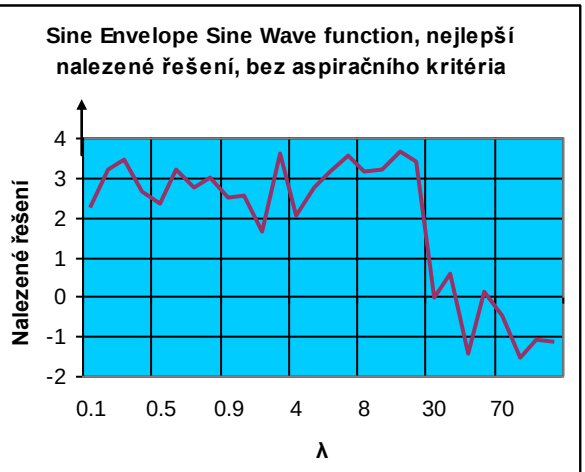

Griewangk, počet změn nejlepšího nalezeného řešení, bez aspiračního kritéria
Počet změn
600
500
400
300
200
100
0
0.1
0.5
0.9
4
8
30
70
λ
Griewangk, počet změn nejlepšího nalezeného řešení, s aspiračním kritériem
Počet změn
140
120
100
80
60
40
20
0
0.1
0.5
0.9
4
8
30
70
λ
Griewangk, doba trvání průběhu algoritmu, bez aspiračního kritéria
Čas v [s]
500
400
300
200
100
0
0.1
0.5
0.9
4
8
30
70
λ
Griewangk, doba trvání průběhu algoritmu, s aspiračním kritériem
Čas v [s]
1000
800
600
400
200
0
0.1
0.5
0.9
4
8
30
70
λ

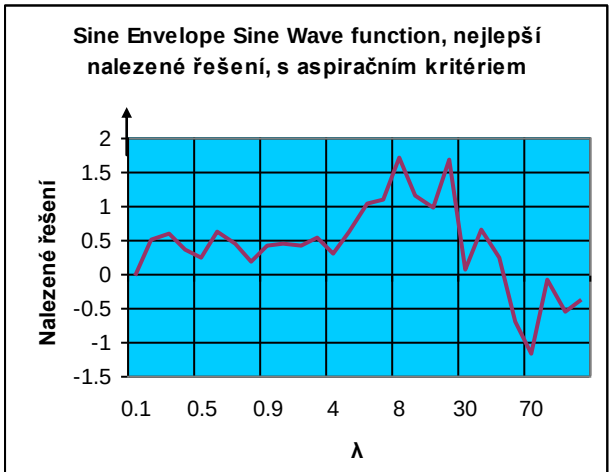
Sine envelope sine wave function

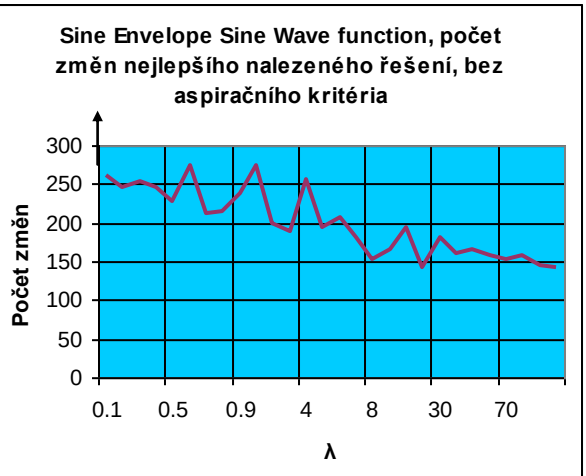
$$-\sum_{i=1}^{n-1}\left(\frac{\sin^2\left(\sqrt{x_{i+1}^2+x_i^2}-0.5\right)}{\left(0.001\left(x_{i+1}^2+x_i^2\right)+1\right)^2}+0.5\right), \quad -100 \le x_i \le 100$$

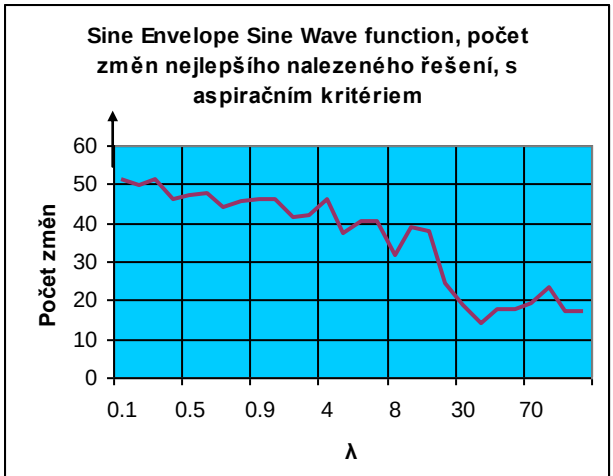


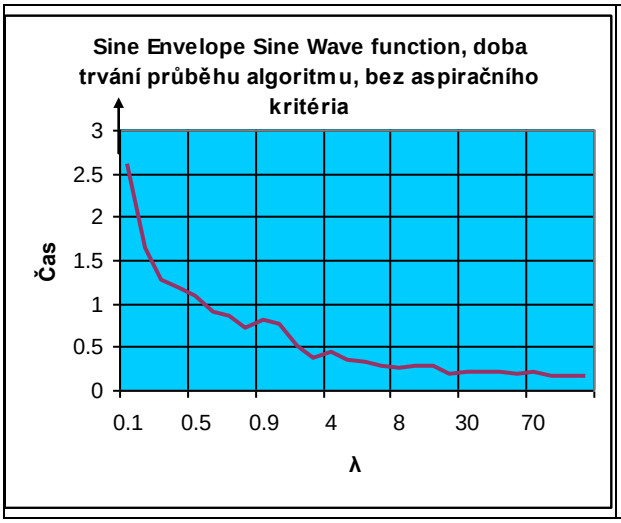


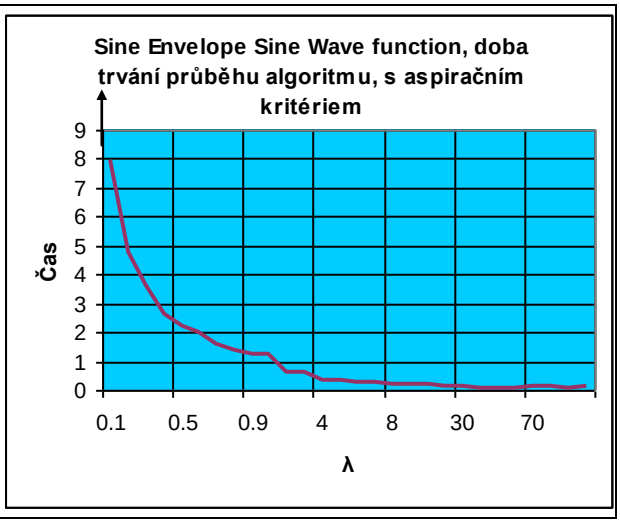






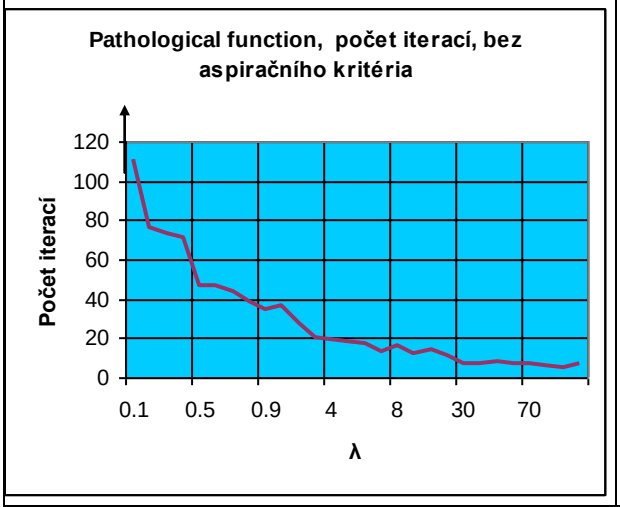


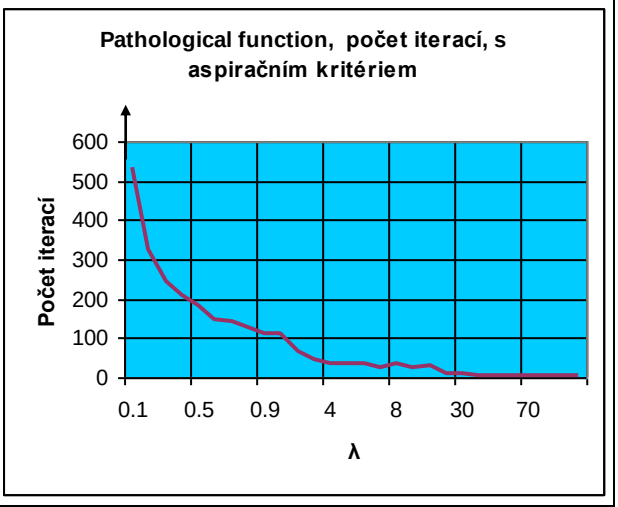


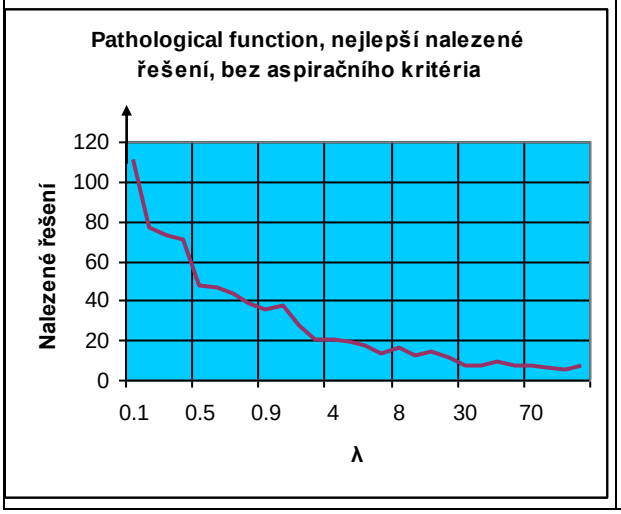


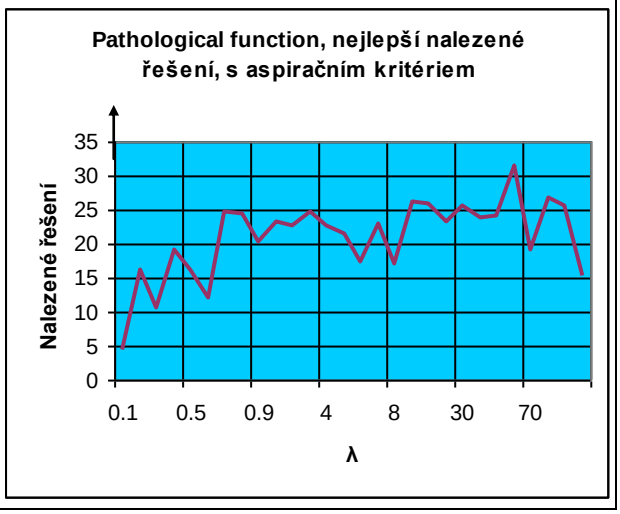
Pathological function

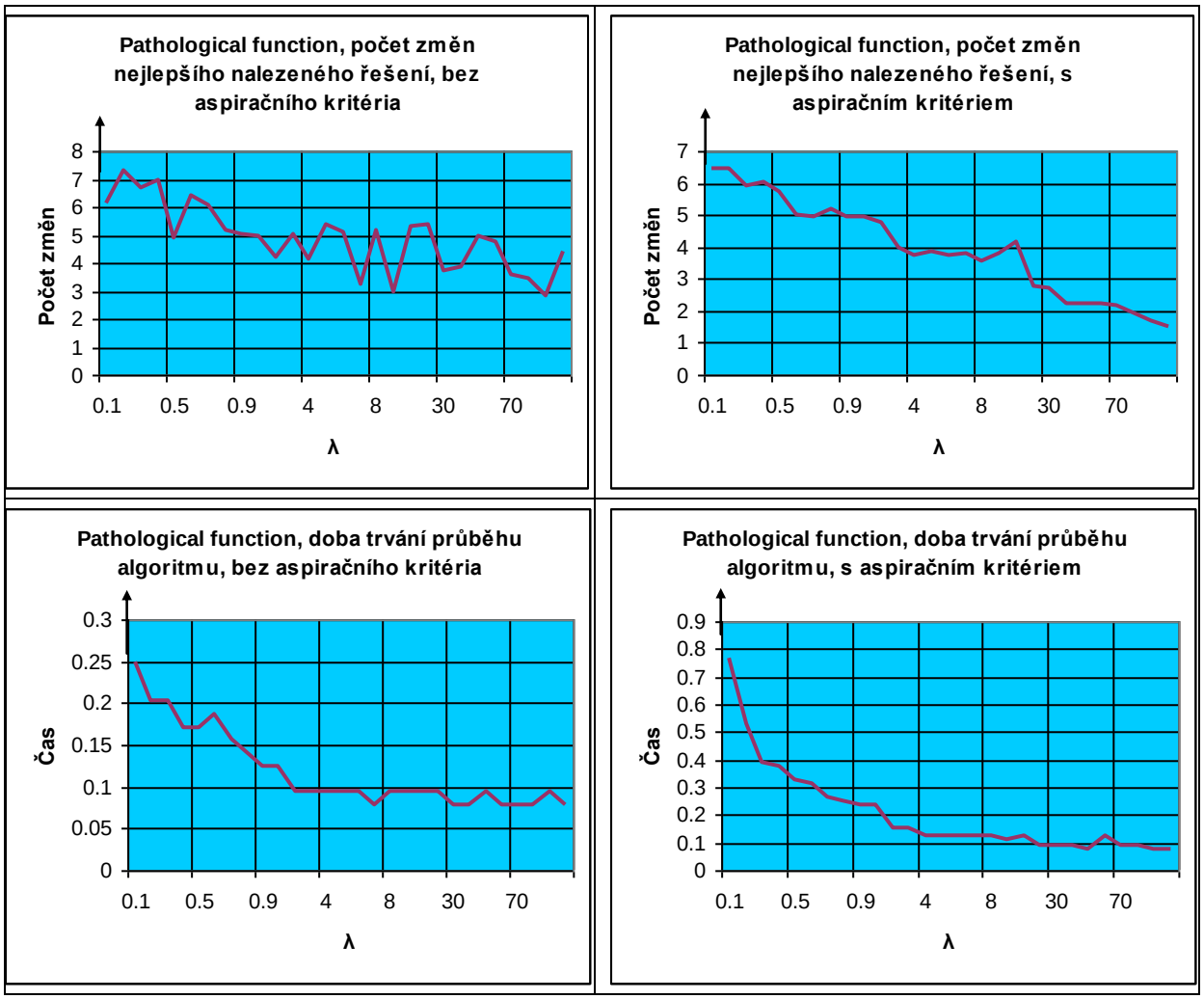
$$\sum_{i=1}^{n}\left(\frac{\sin^2\sqrt{x_{i+1}^2+100x_i^2}-0.5}{0.001\left(x_{i+1}^2-2x_{i+1}x_i+x_i^2\right)^2+1.0}+0.5\right), \quad -100 \le x_i \le 100$$





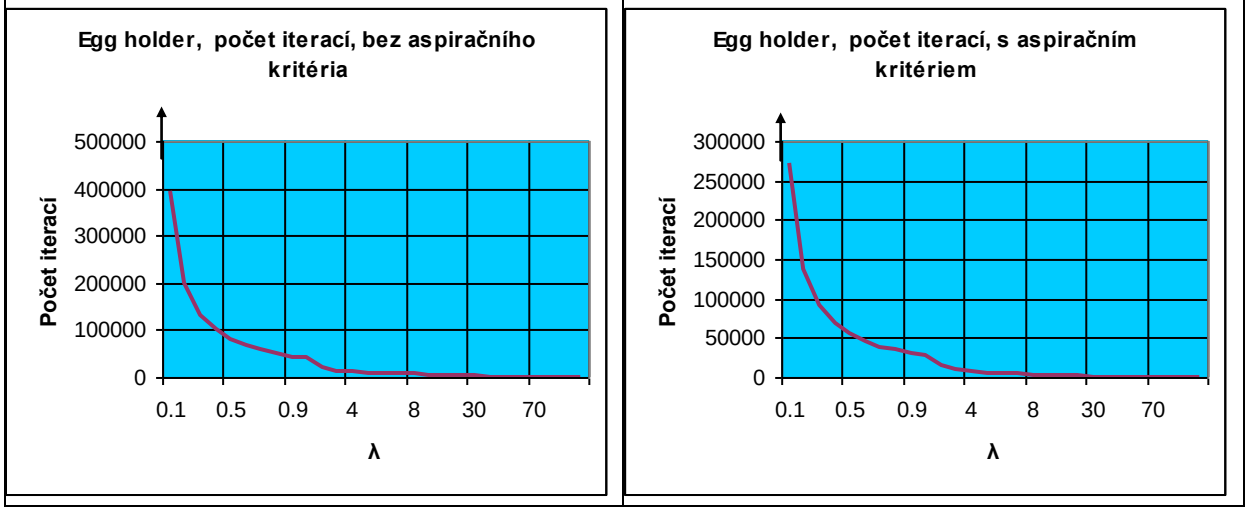


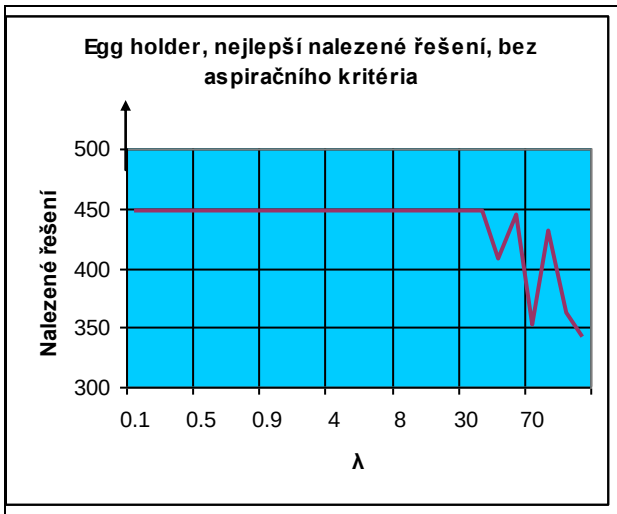


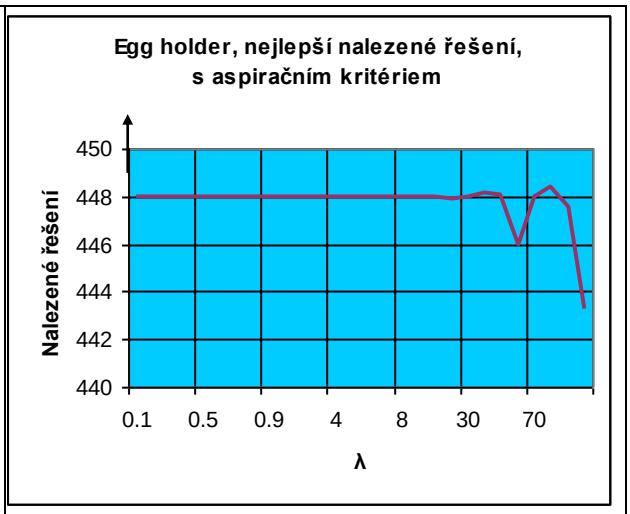


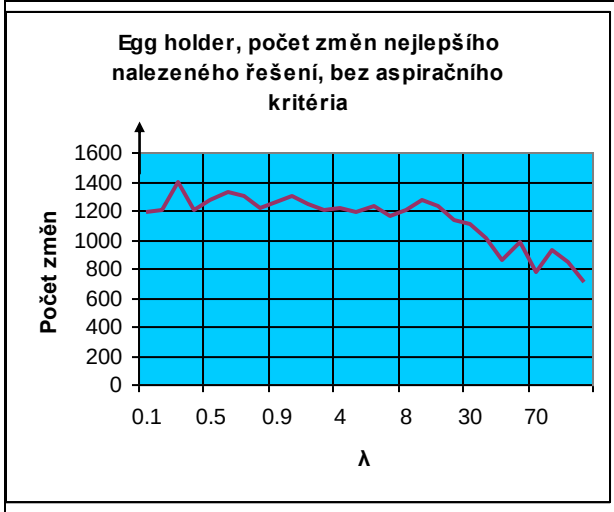
Egg holder function

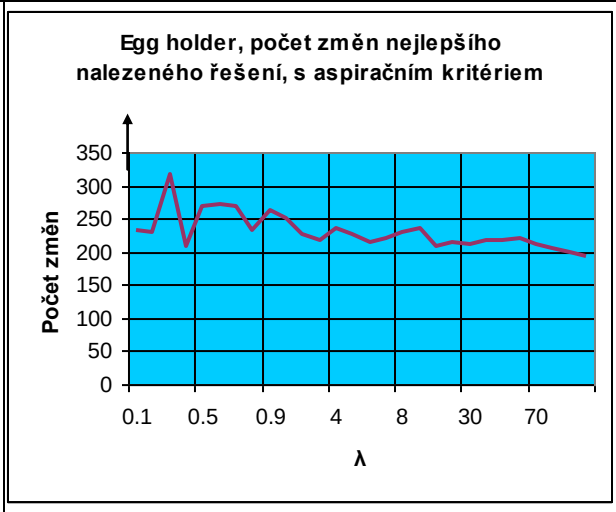
$$\sum_{i=1}^{n}\left(-x_i Sin\sqrt{Abs\left(x_i - x_{i+1} + 47\right)} - x_{i+1} + 47\ Sin\left[\sqrt{Abs\left[x_{i+1} + 47 + \frac{x_i}{2}\right]}\right]\right); -500 \le x_i \le 500$$

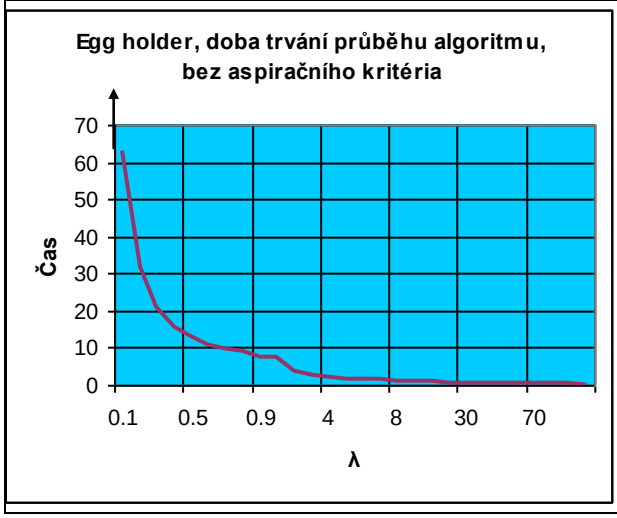


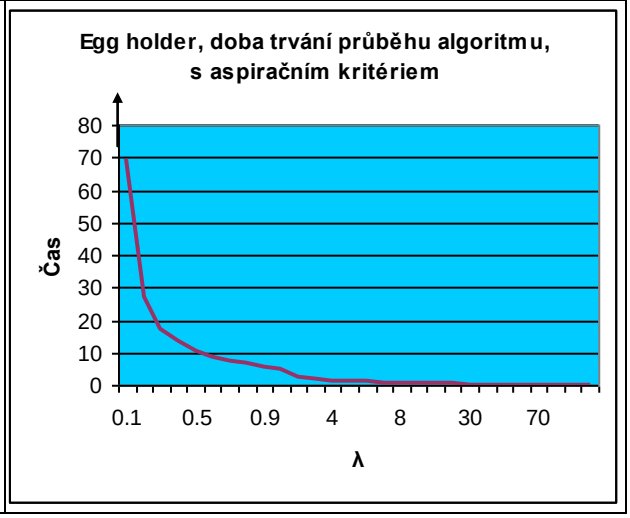

Egg holder, nejlepší nalezené řešení, bez aspiračního kritéria
Nalezené řešení
500
450
400
350
300
0.1
0.5
0.9
4
8
30
70
λ


Egg holder, nejlepší nalezené řešení, s aspiračním kritériem
Nalezené řešení
450
448
446
444
442
440
0.1
0.5
0.9
4
8
30
70
λ


Egg holder, počet změn nejlepšího nalezeného řešení, bez aspiračního kritéria
Počet změn
1600
1400
1200
1000
800
600
400
200
0
0.1
0.5
0.9
4
8
30
70
λ


Egg holder, počet změn nejlepšího nalezeného řešení, s aspiračním kritériem
Počet změn
350
300
250
200
150
100
50
0
0.1
0.5
0.9
4
8
30
70
λ

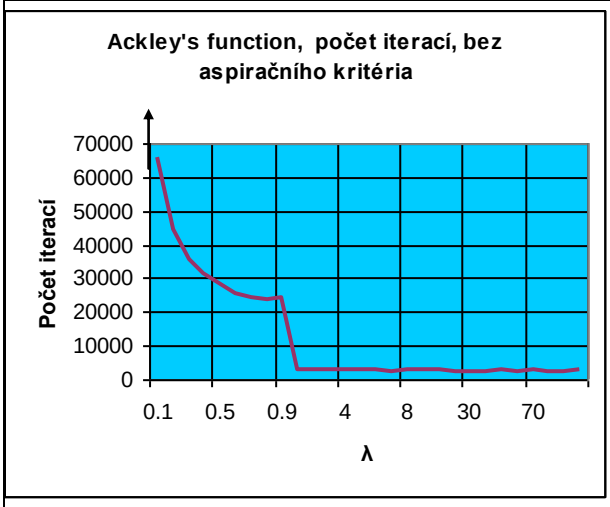

Egg holder, doba trvání průběhu algoritmu, bez aspiračního kritéria
Čas
70
60
50
40
30
20
10
0
0.1
0.5
0.9
4
8
30
70
λ

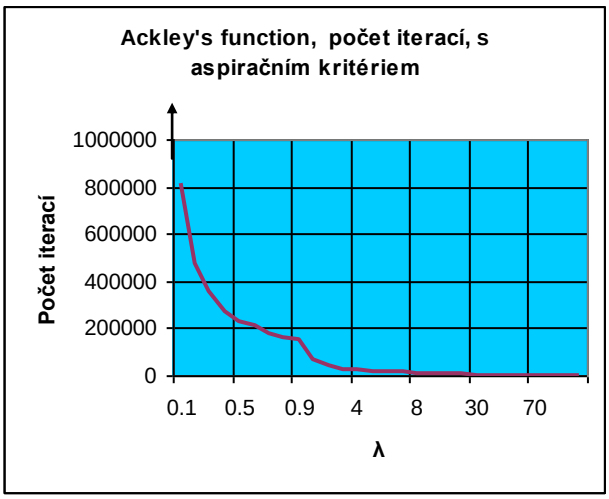

Egg holder, doba trvání průběhu algoritmu, s aspiračním kritériem
Čas
80
70
60
50
40
30
20
10
0
0.1
0.5
0.9
4
8
30
70
λ

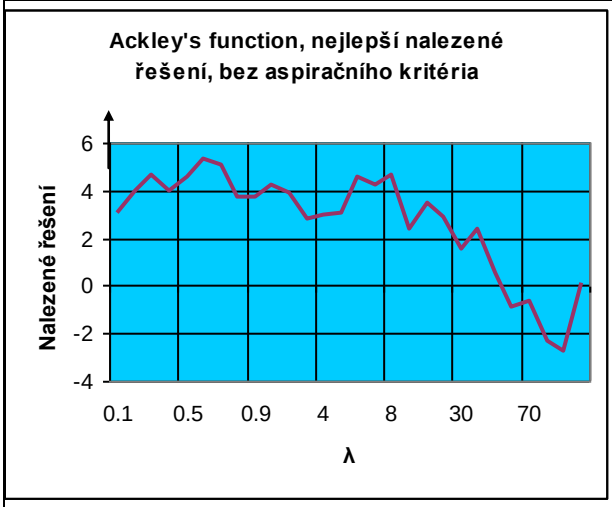
Ackley's function

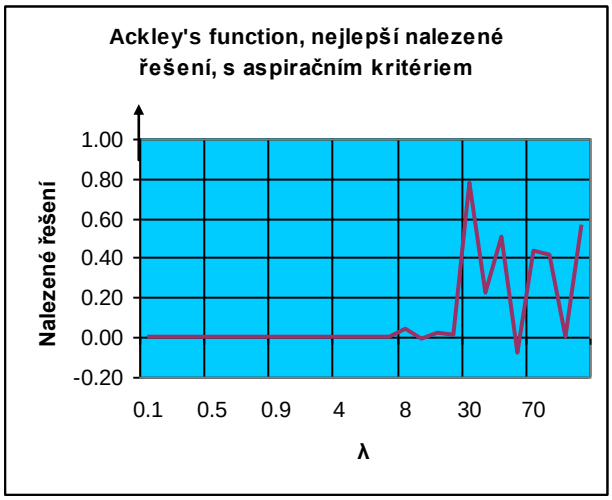
$$\sum_{i=1}^{n-1}\left(20+e^{-20}e^{-0.2\sqrt{0.5\left(x_{i+1}^2+x_i^2\right)}}-e^{0.5\left(\cos 2\pi x_{i+1}+\cos 2\pi x_i\right)}\right), \quad -30 \le x_i \le 30$$

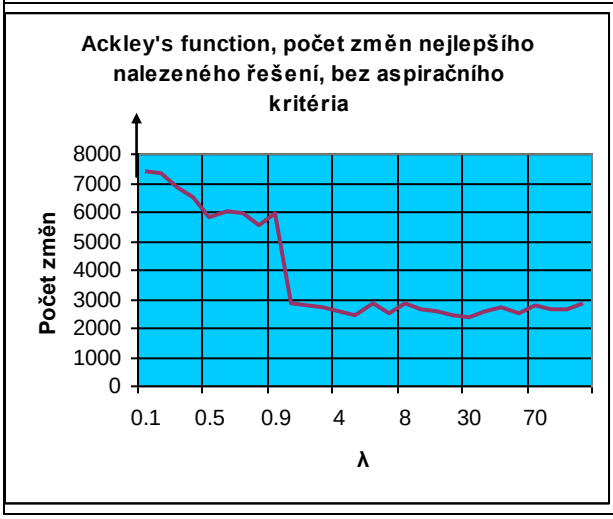
Globální minimum $f(x) = 0, x(i) = 0; i = 1...n$

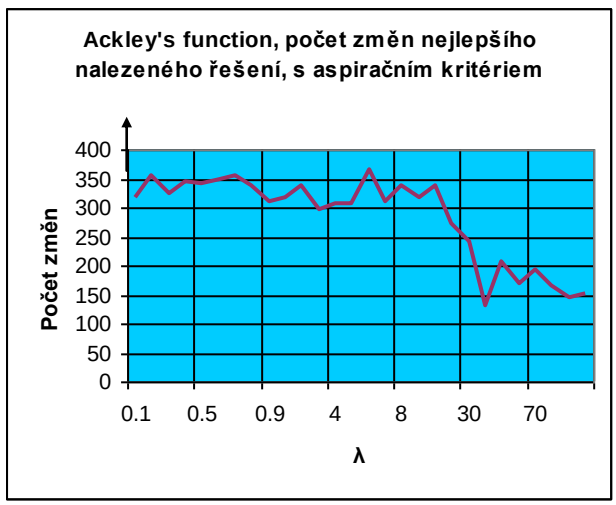


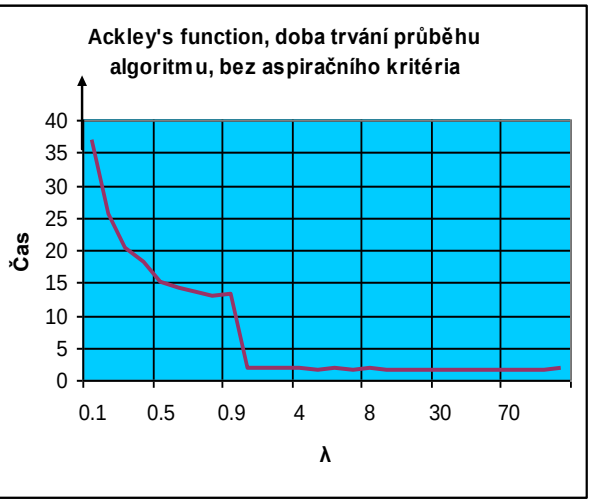


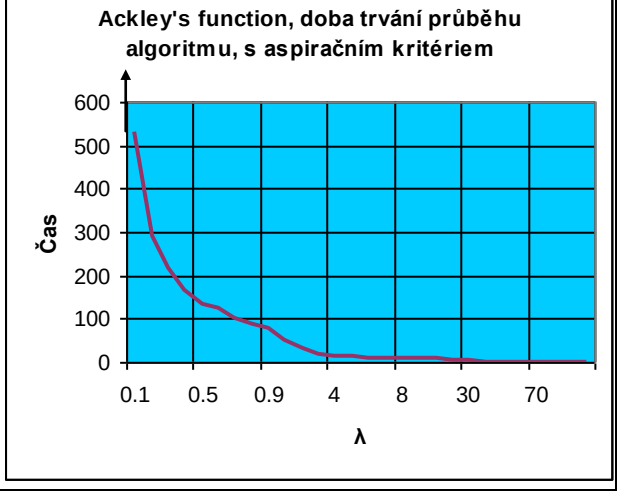






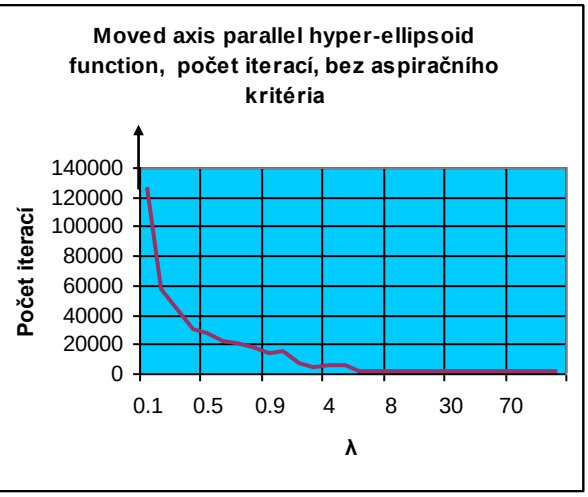


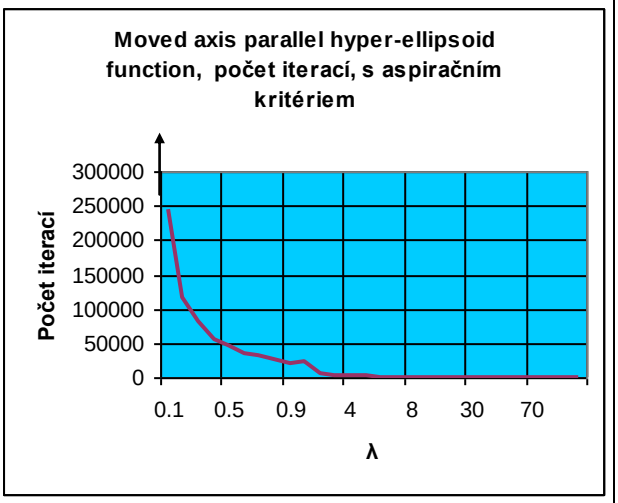


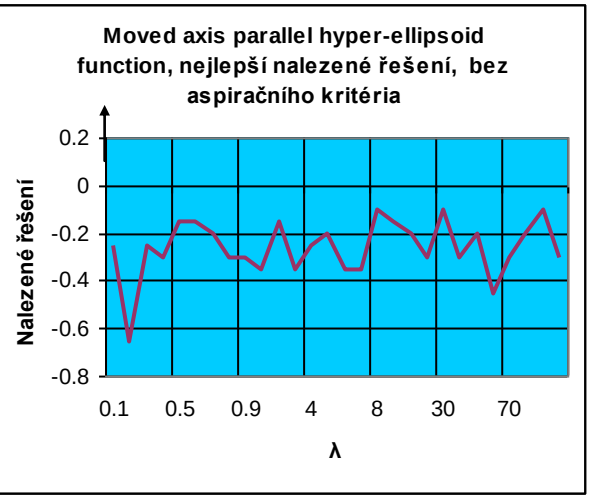


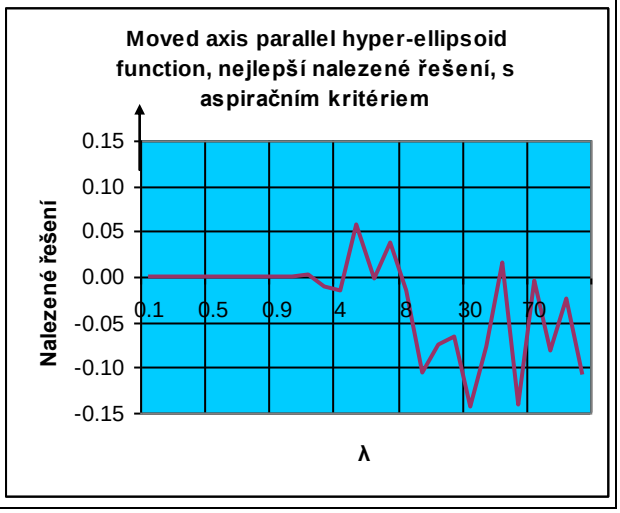
Moved axis parallel hyper-ellipsoid function

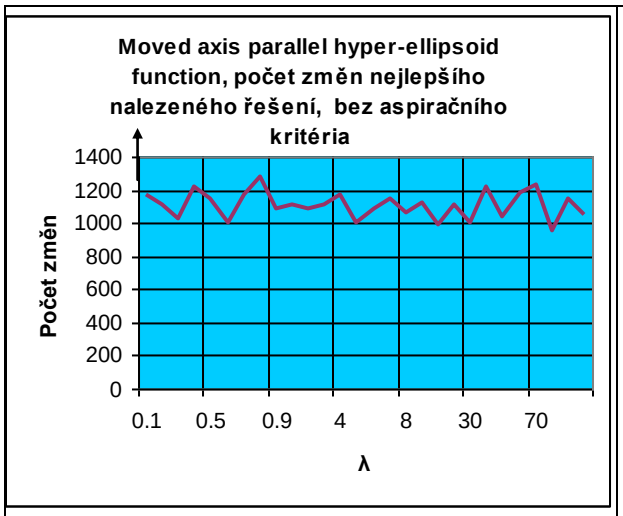
$$\sum_{i=1}^{n} 5i * x_i^2 ; -5{,}12 \le x_i \le 5{,}12$$

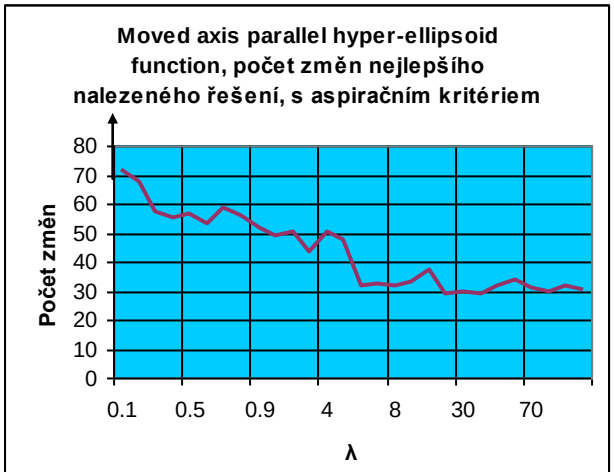


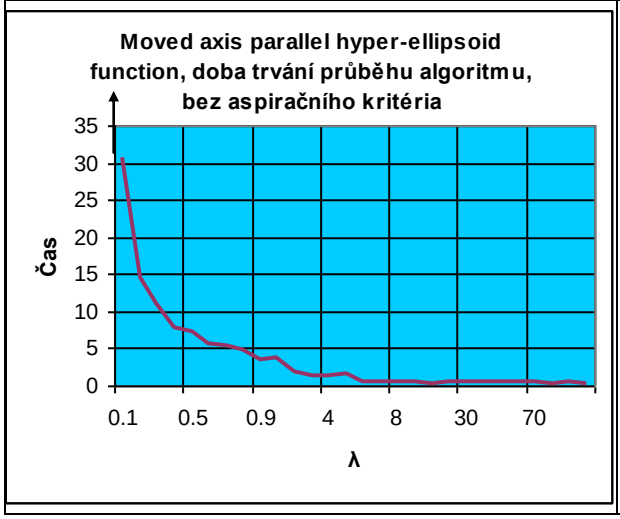


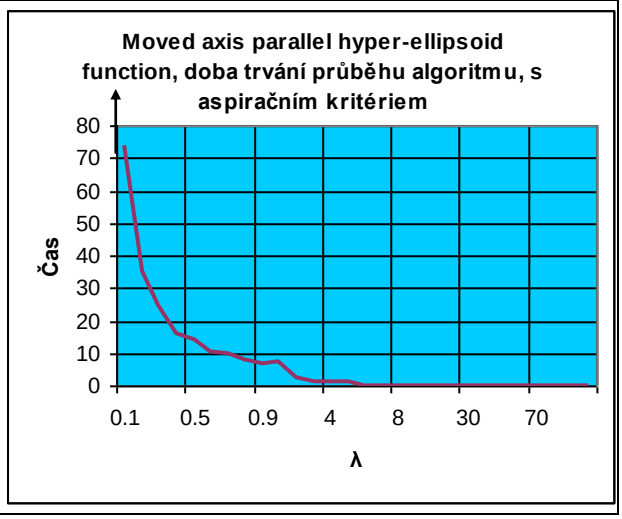





Moved axis parallel hyper-ellipsoid function, počet změn nejlepšího nalezeného řešení, bez aspiračního kritéria
Počet změn
1400
1200
1000
800
600
400
200
0
0.1
0.5
0.9
4
8
30
70
λ

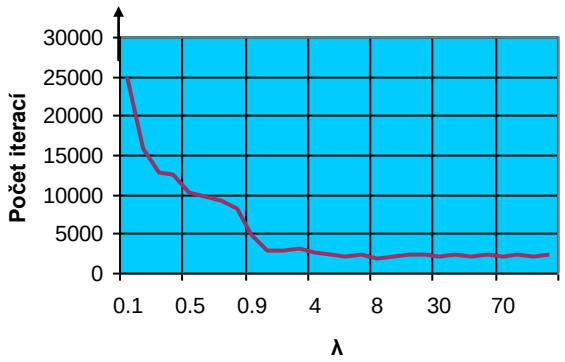

Moved axis parallel hyper-ellipsoid function, počet změn nejlepšího nalezeného řešení, s aspiračním kritériem
Počet změn
80
70
60
50
40
30
20
10
0
0.1
0.5
0.9
4
8
30
70
λ

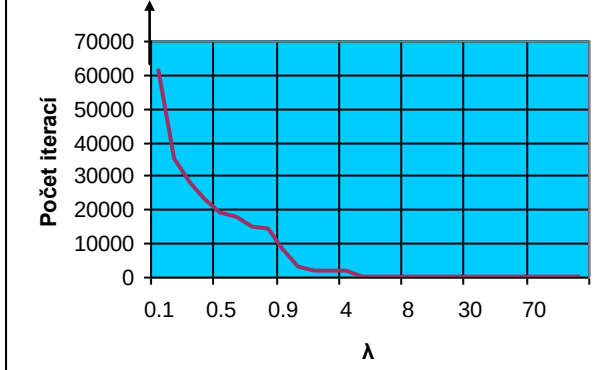

Moved axis parallel hyper-ellipsoid function, doba trvání průběhu algoritmu, bez aspiračního kritéria
Čas
35
30
25
20
15
10
5
0
0.1
0.5
0.9
4
8
30
70
λ

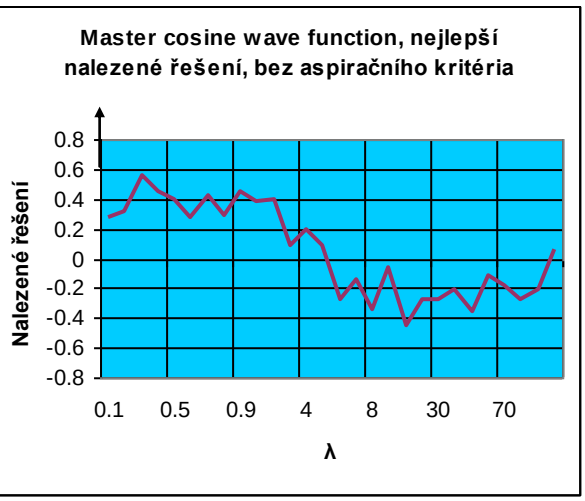

Moved axis parallel hyper-ellipsoid function, doba trvání průběhu algoritmu, s aspiračním kritériem
Čas
80
70
60
50
40
30
20
10
0
0.1
0.5
0.9
4
8
30
70
λ

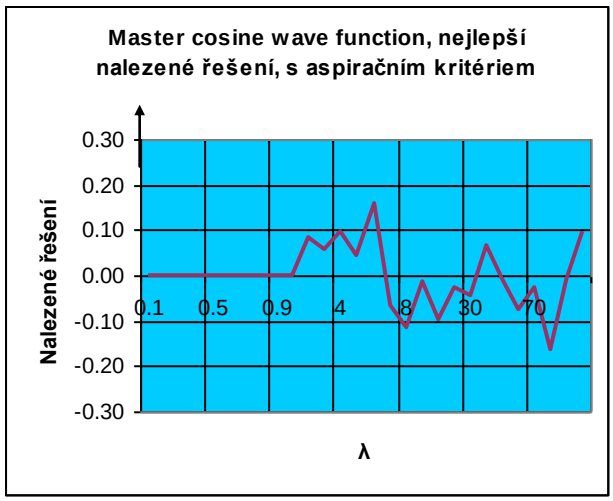
Master cosine wave function

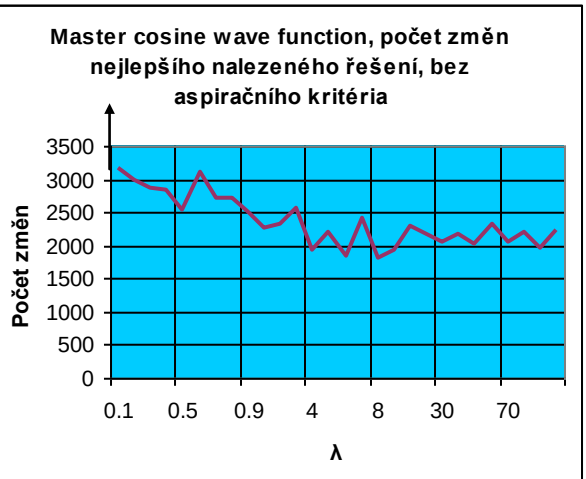
$$-\sum_{i=1}^{n-1} e^{-\frac{1}{8}\left(x_{i+1}^2+0.5x_ix_{i+1}+x_i^2\right)} \cos\left(4\sqrt{x_{i+1}^2+0.5x_ix_{i+1}+x_i^2}\right), \quad -5 \le x_i \le 5$$

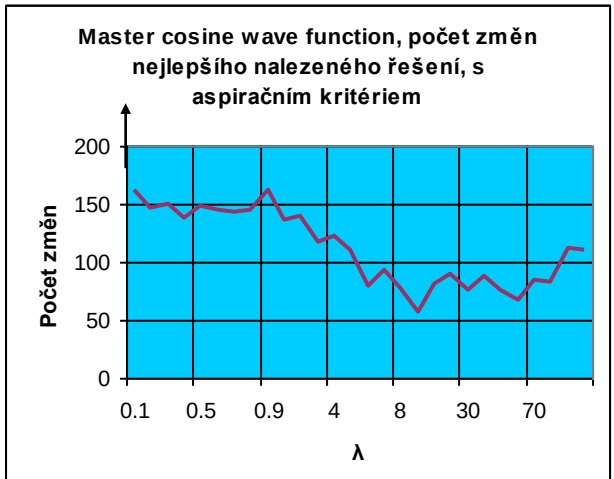


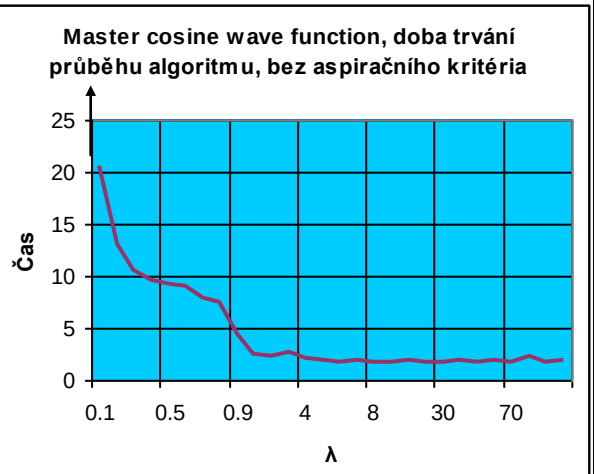


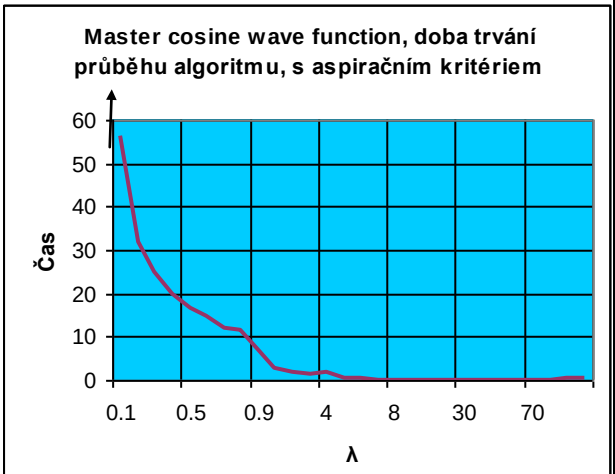








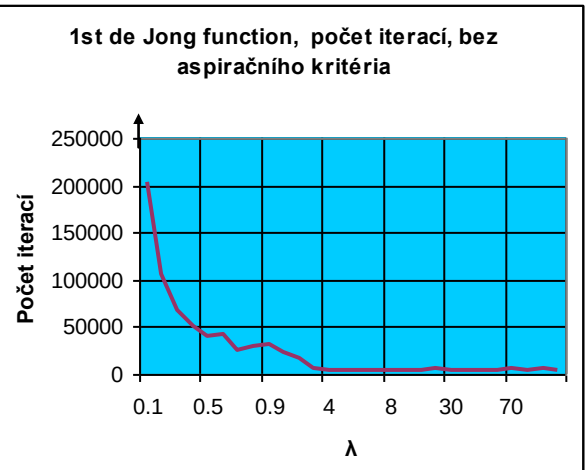


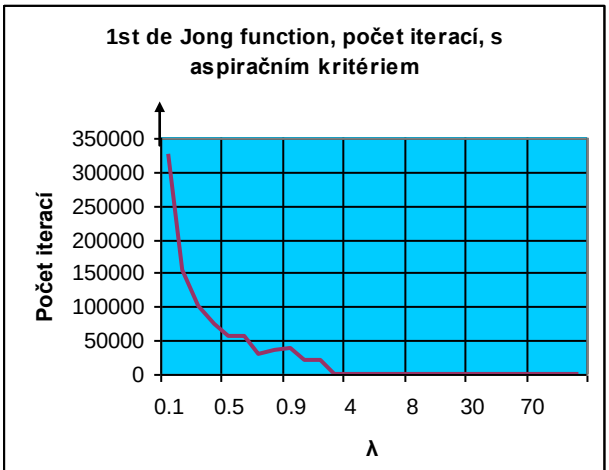


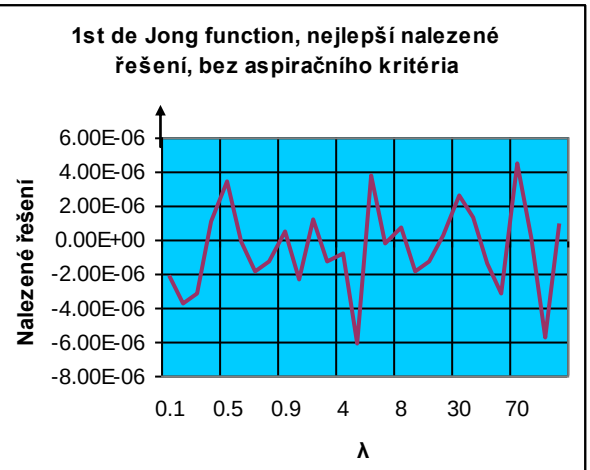
Sphere model – 1st De Jong Function

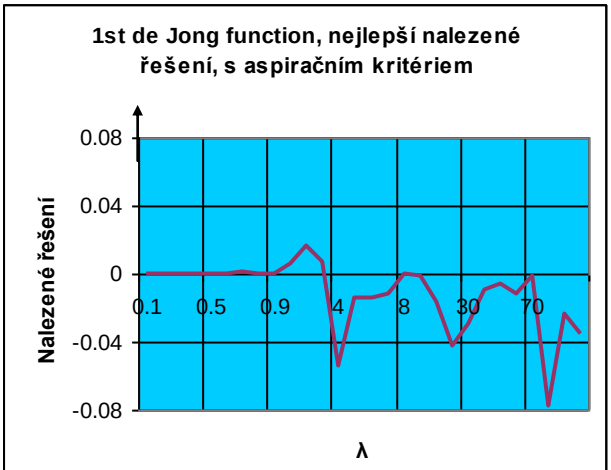
$$\sum_{i=1}^{n} x_i^2 \; ; -5{,}12 \le x_i \le 5{,}12$$

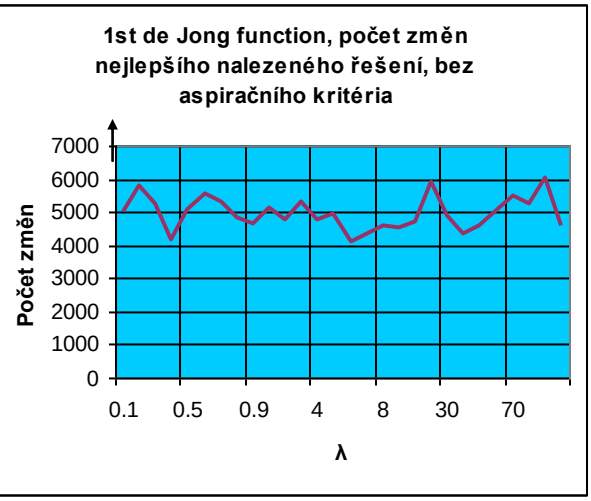
Globální minimum $f(x) = 0, x(i) = 0; i = 1...n$

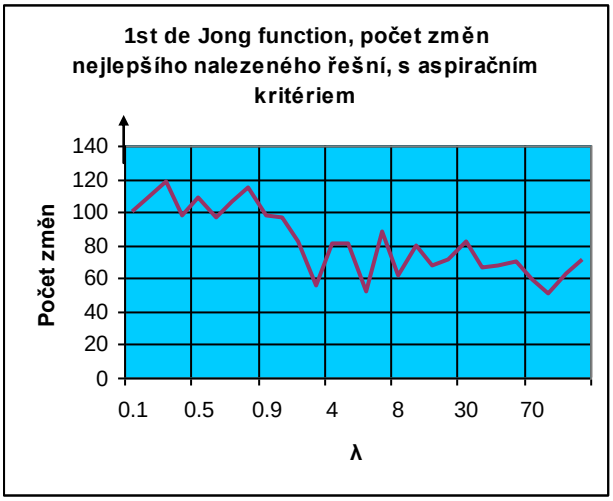


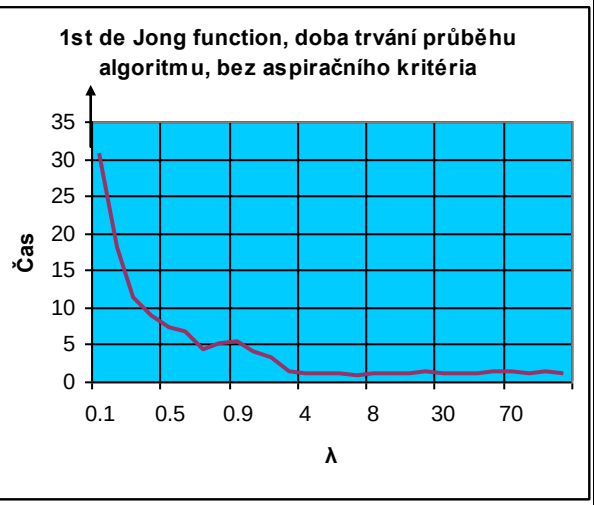


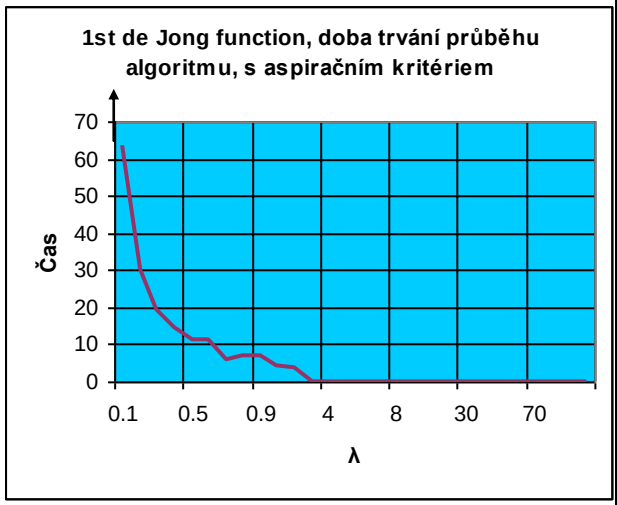








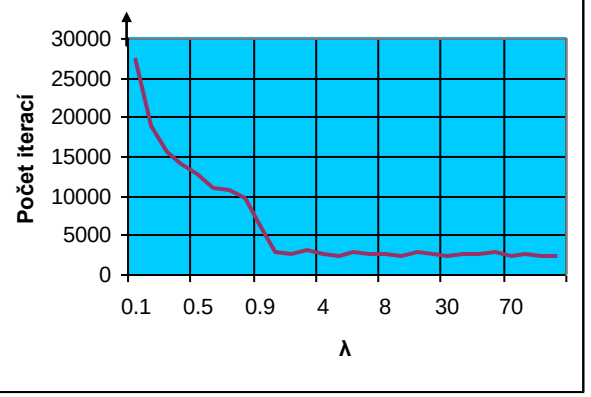


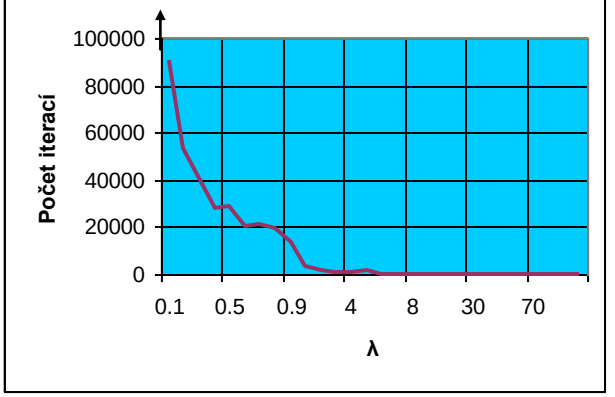


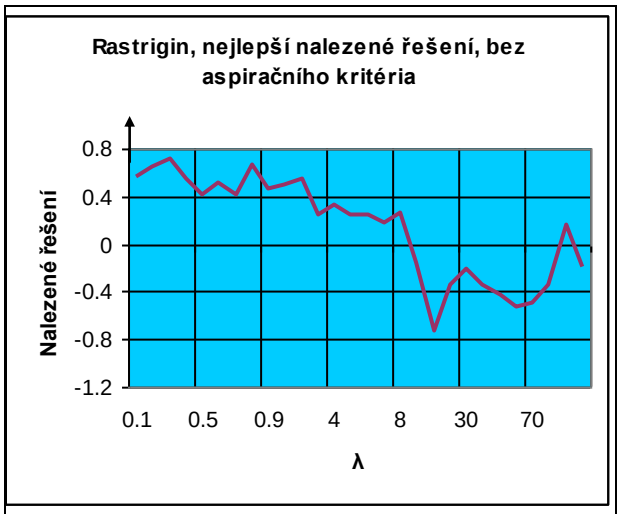
Rastrigin Function

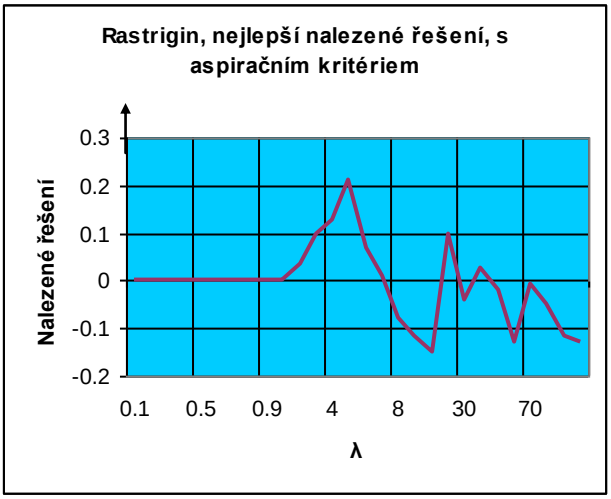
$$f(x) = 10 * Dim + \sum_{i=1}^{Dim} \left(x_i^2 - 10 \cos(2 Pi\, x_i) \right); -5{,}12 \le x_i \le 5{,}12$$

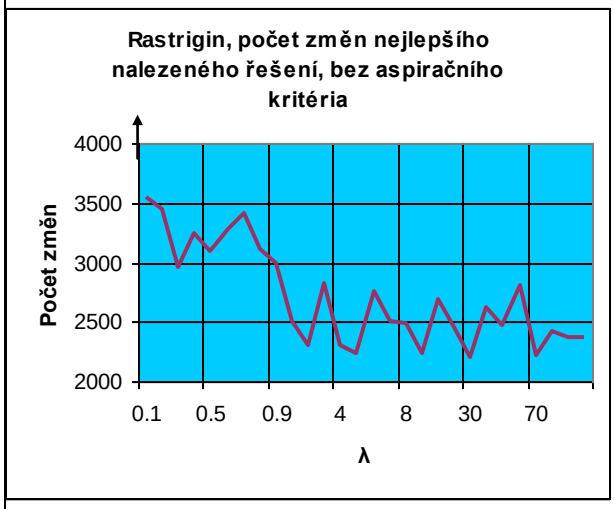
Globální minimum $f(x) = 0, x(i) = 0; i = 1...n$

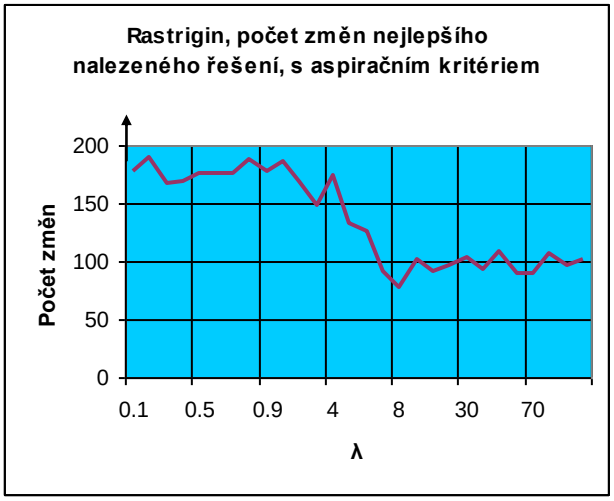


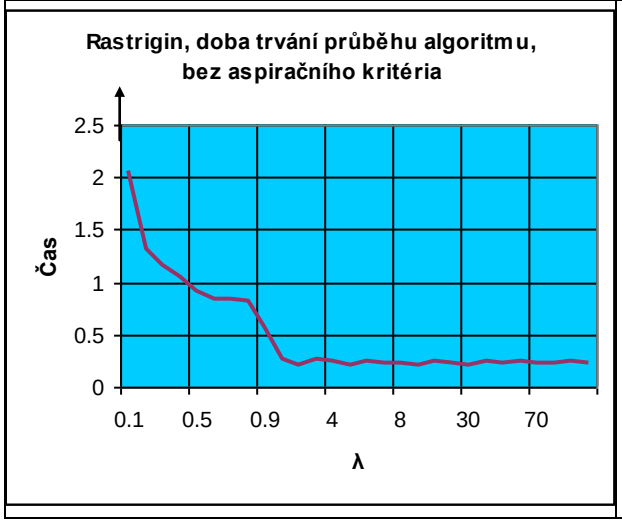


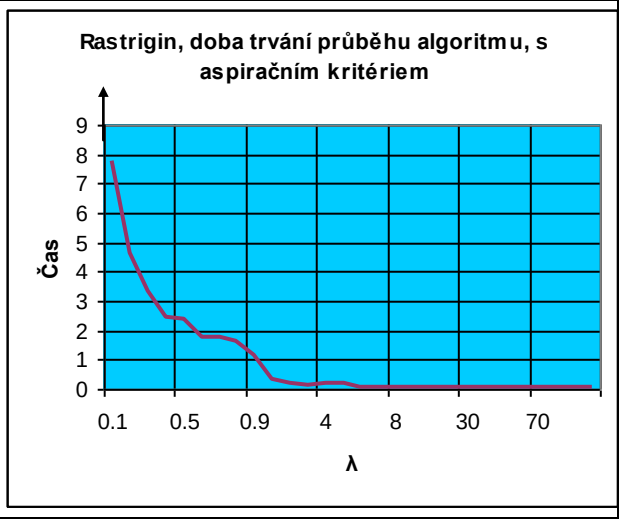

Rastrigin, nejlepší nalezené řešení, bez aspiračního kritéria
Nalezené řešení
0.8
0.4
0
-0.4
-0.8
-1.2
0.1 0.5 0.9 4 8 30 70
λ


Rastrigin, nejlepší nalezené řešení, s aspiračním kritériem
Nalezené řešení
0.3
0.2
0.1
0
-0.1
-0.2
0.1 0.5 0.9 4 8 30 70
λ


Rastrigin, počet změn nejlepšího nalezeného řešení, bez aspiračního kritéria
Počet změn
4000
3500
3000
2500
2000
0.1 0.5 0.9 4 8 30 70
λ

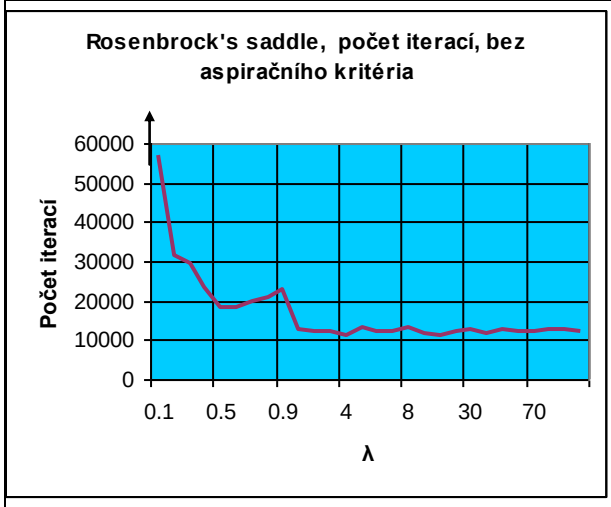

Rastrigin, počet změn nejlepšího nalezeného řešení, s aspiračním kritériem
Počet změn
200
150
100
50
0
0.1 0.5 0.9 4 8 30 70
λ

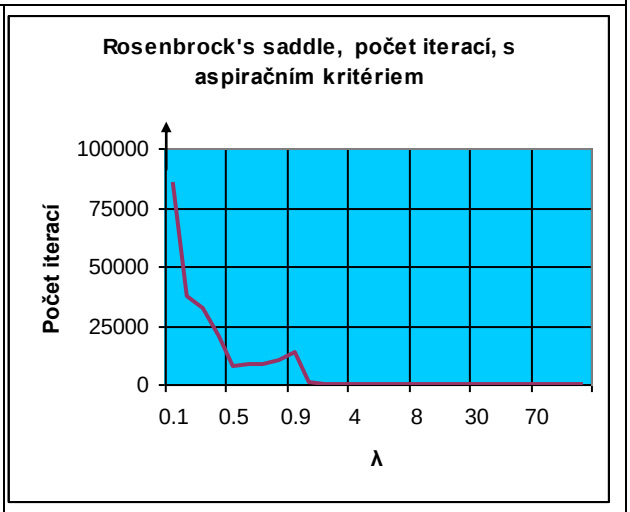

Rastrigin, doba trvání průběhu algoritmu, bez aspiračního kritéria
Čas
2.5
2
1.5
1
0.5
0
0.1 0.5 0.9 4 8 30 70
λ

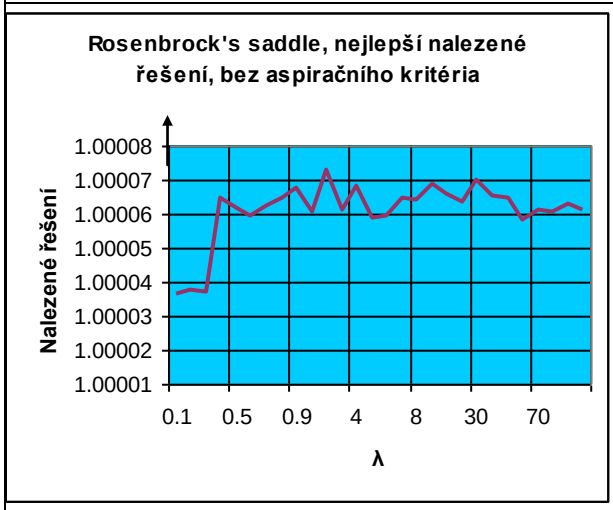

Rastrigin, doba trvání průběhu algoritmu, s aspiračním kritériem
Čas
9
8
7
6
5
4
3
2
1
0
0.1 0.5 0.9 4 8 30 70
λ

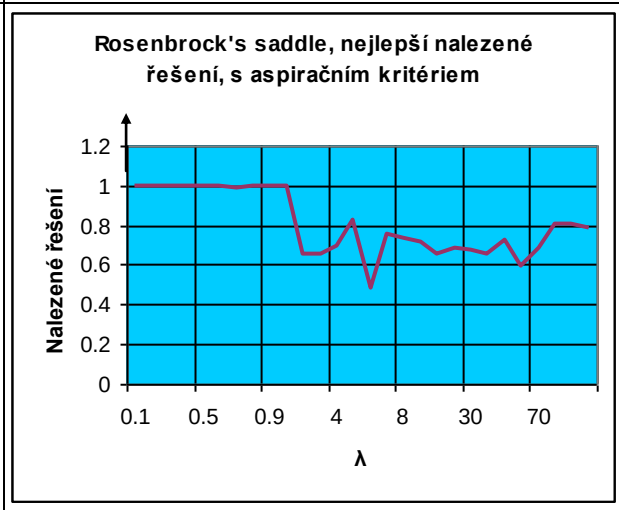
Rosenbrock's saddle function

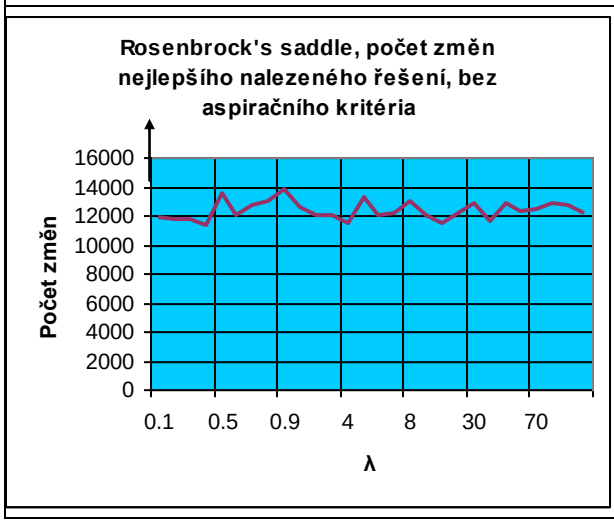
$$\sum_{i=1}^{Dim-1} \left[100 \left(x_i^2 - x_{i+1} \right)^2 + \left(1 - x_i \right)^2 \right]; -2 \le x_i \le 2$$

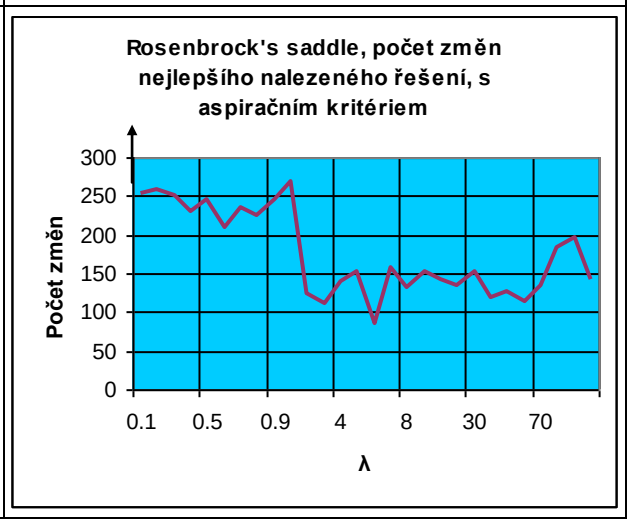


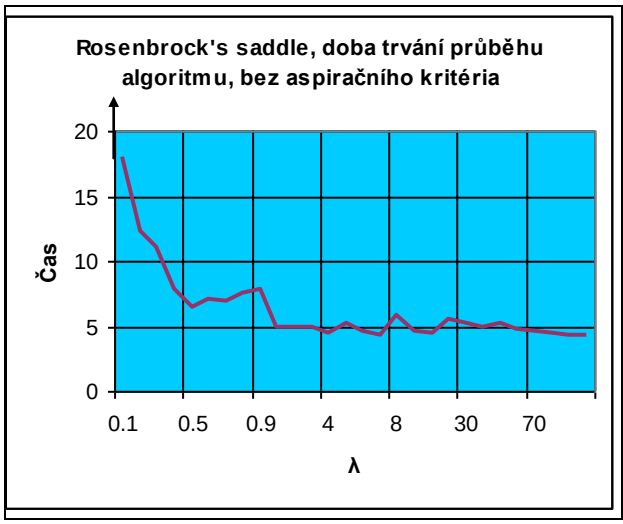


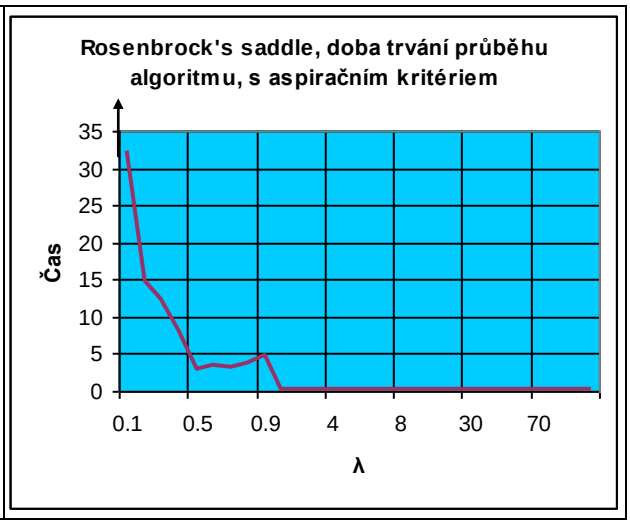








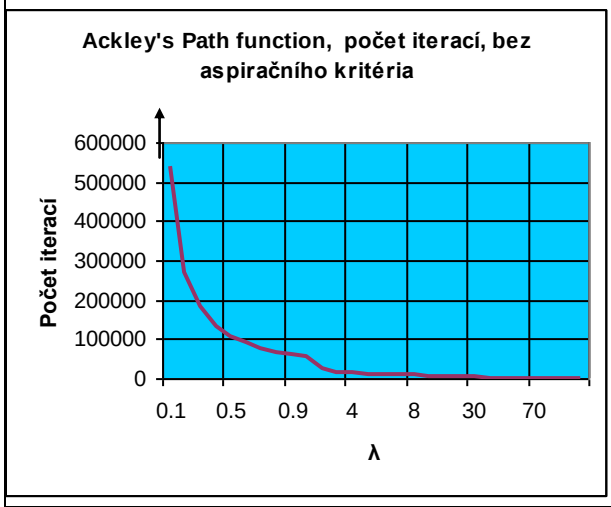


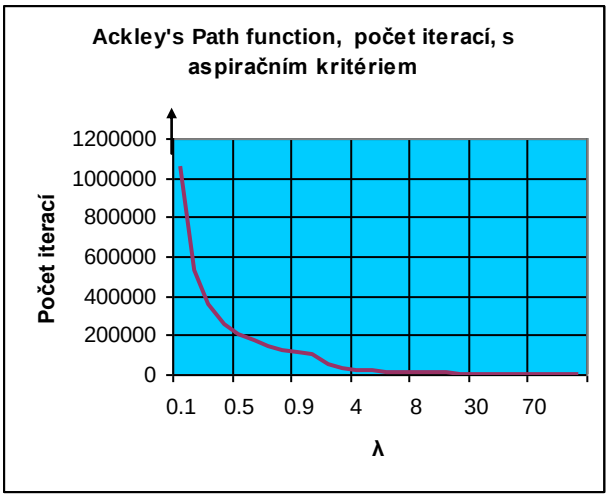


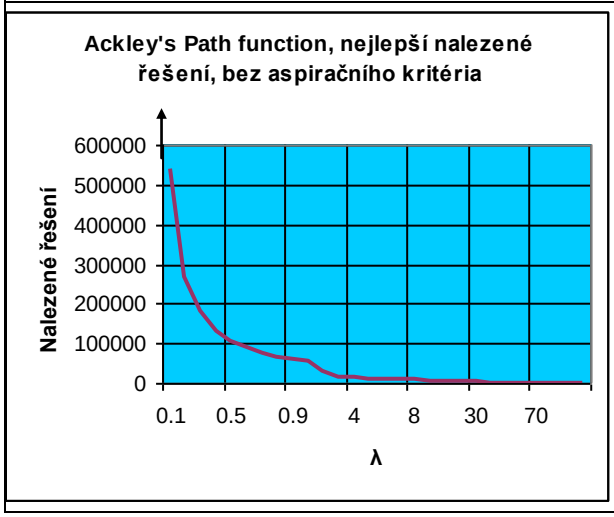
Ackley path function

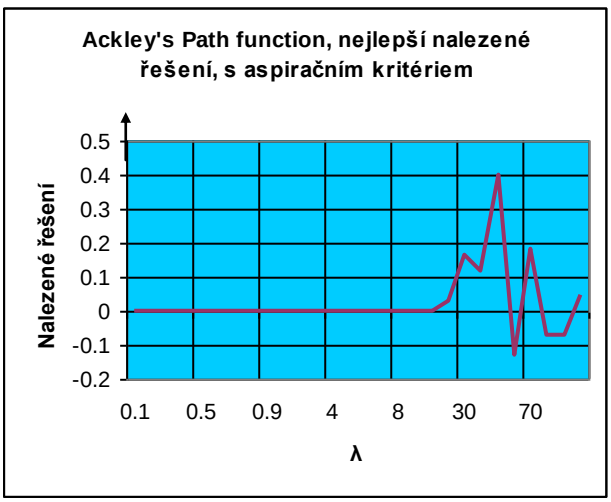
$$-a*e^{-b*\sqrt{\frac{\sum_{i=1}^{n}x^2}{n}}}-e^{\frac{\sum_{i=1}^{n}\cos c*x_i}{n}}+a+e^1$$

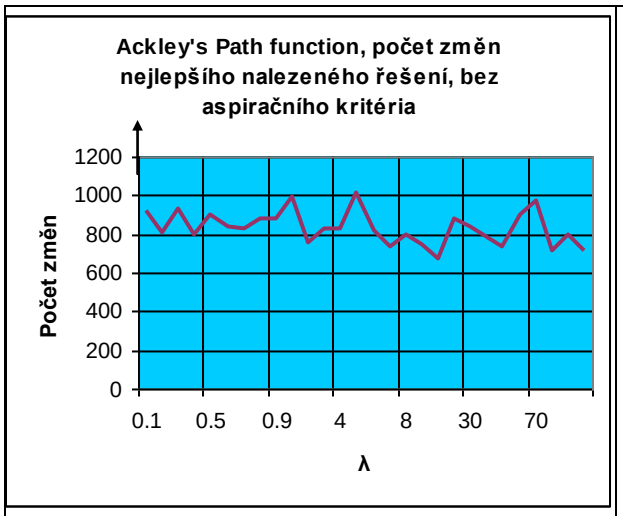
$$a=20; b=0{,}2; c=2*\pi; i=1\ldots n; -32{,}768 \le x_i \le 32{,}768$$

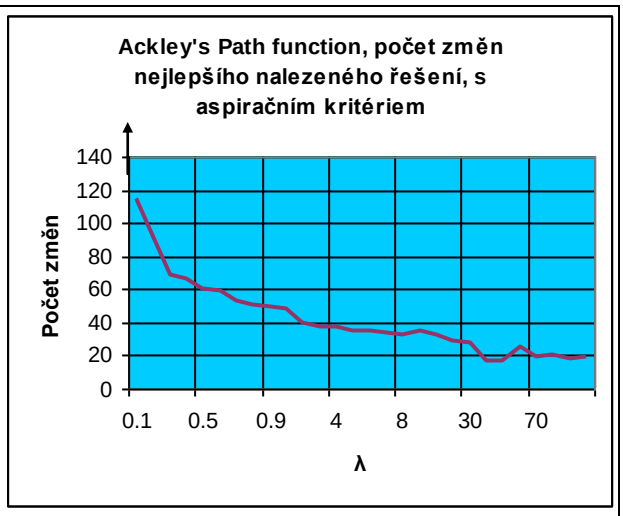


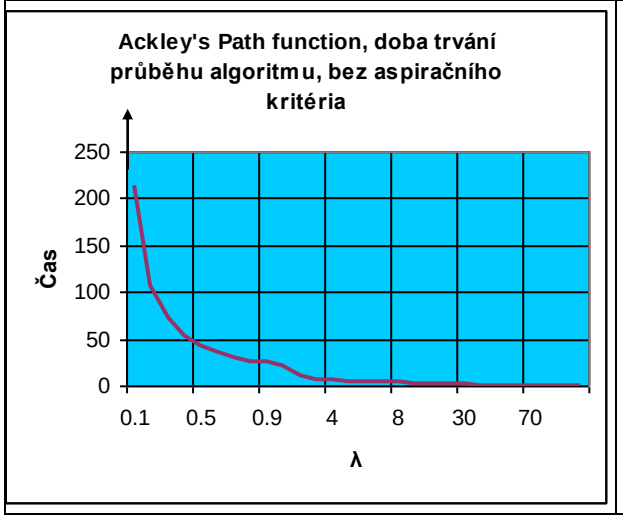


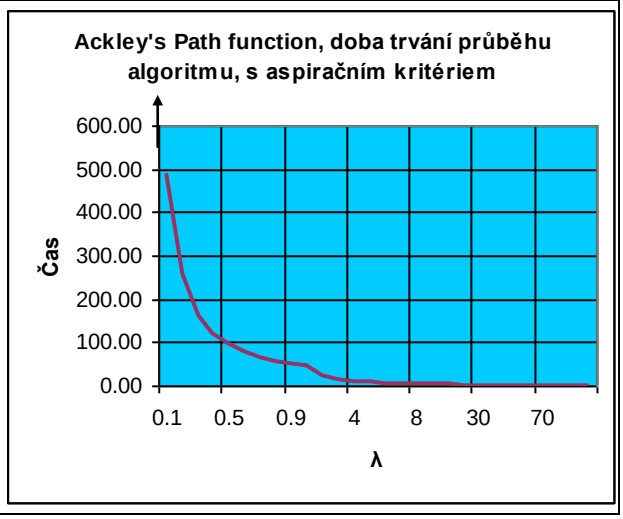





Ackley's Path function, počet změn nejlepšího nalezeného řešení, bez aspiračního kritéria
Počet změn
1200
1000
800
600
400
200
0
0.1
0.5
0.9
4
8
30
70
λ

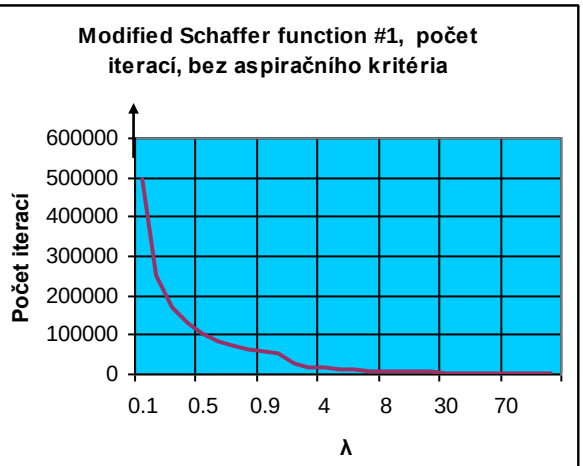

Ackley's Path function, počet změn nejlepšího nalezeného řešení, s aspiračním kritériem
Počet změn
140
120
100
80
60
40
20
0
0.1
0.5
0.9
4
8
30
70
λ

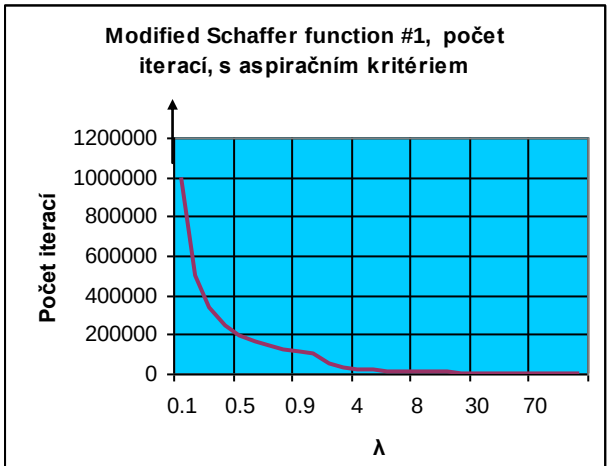

Ackley's Path function, doba trvání průběhu algoritmu, bez aspiračního kritéria
Čas
250
200
150
100
50
0
0.1
0.5
0.9
4
8
30
70
λ

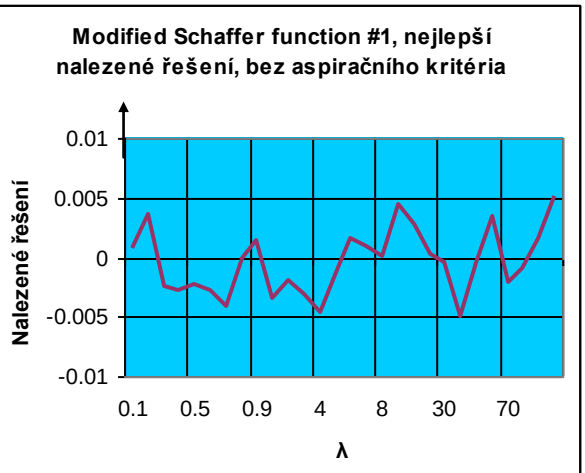

Ackley's Path function, doba trvání průběhu algoritmu, s aspiračním kritériem
Čas
600.00
500.00
400.00
300.00
200.00
100.00
0.00
0.1
0.5
0.9
4
8
30
70
λ

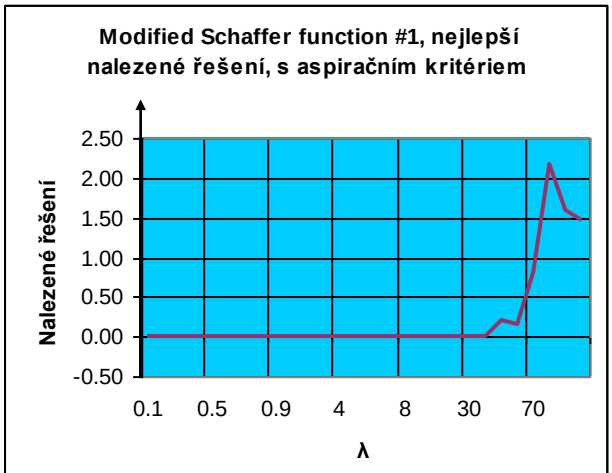
Modified Schaffer function #1

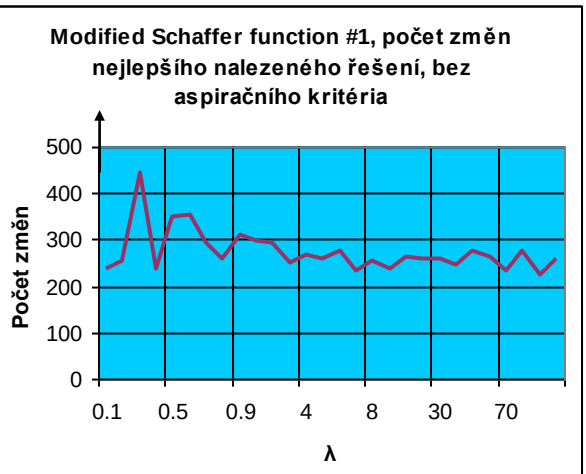
$$0.5 + \frac{\sin^2\left(x_1^2 + x_2^2\right) - 0.5}{\left(1 + 0.001\left(x_1^2 + x_2^2\right)\right)^2} \quad x_1, x_2 \in [-100, 100]$$

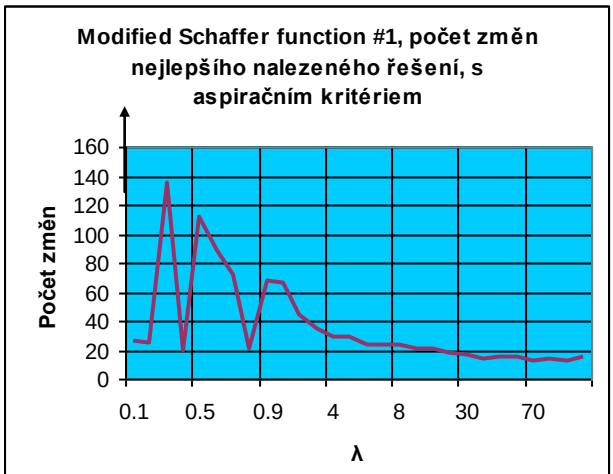


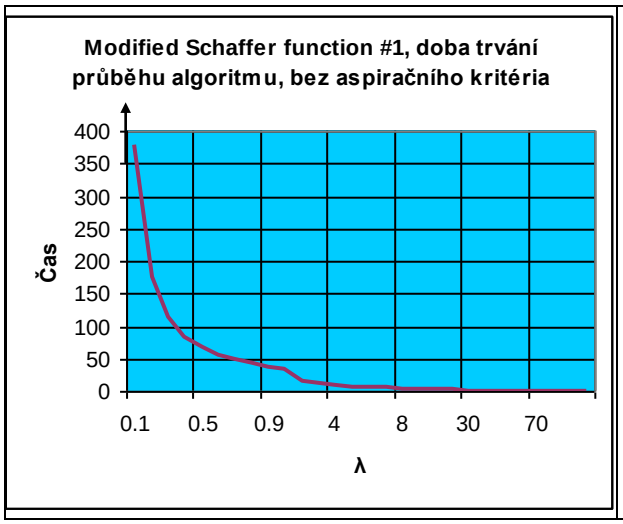


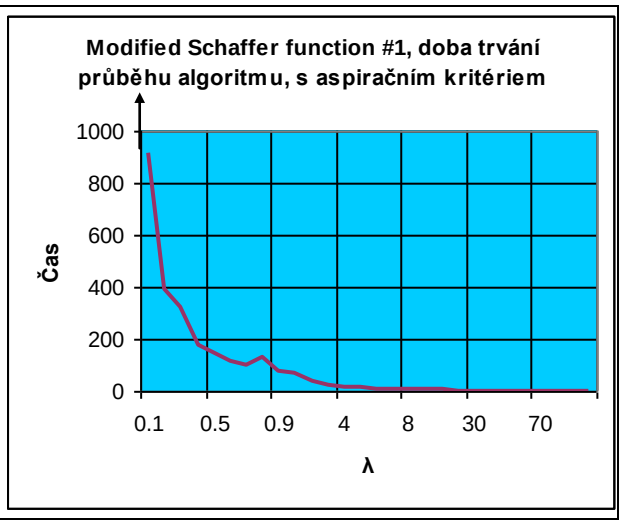






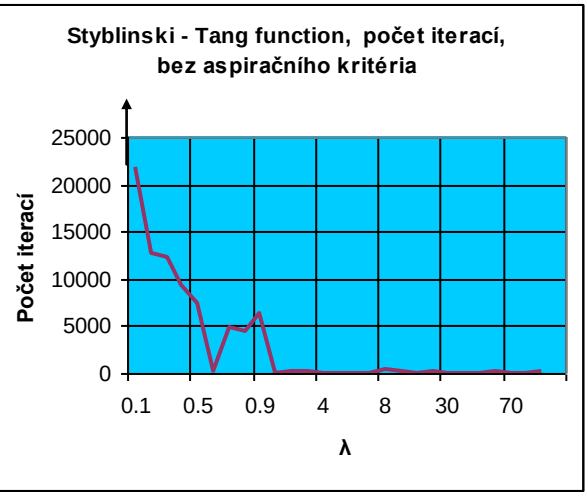


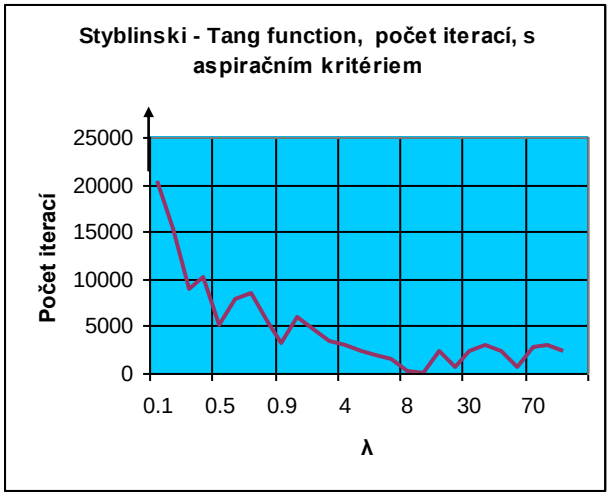

Modified Schaffer function #1, doba trvání průběhu algoritmu, bez aspiračního kritéria
Čas
400
350
300
250
200
150
100
50
0
0.1
0.5
0.9
4
8
30
70
λ

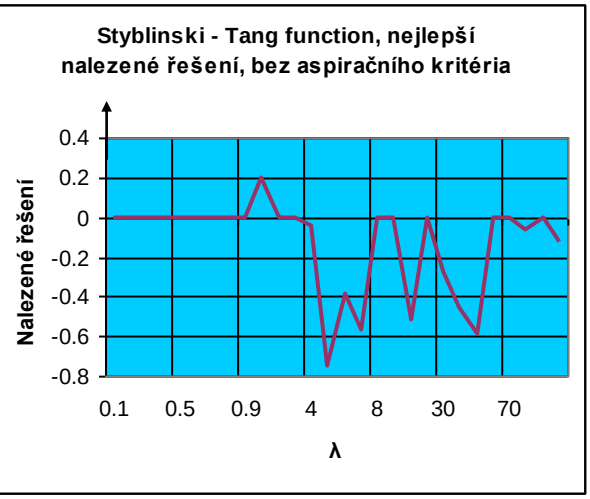

Modified Schaffer function #1, doba trvání průběhu algoritmu, s aspiračním kritériem
Čas
1000
800
600
400
200
0
0.1
0.5
0.9
4
8
30
70
λ

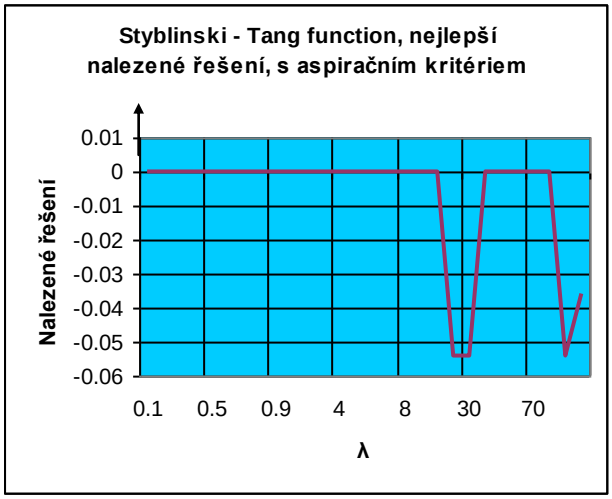
Styblinski – Tang function

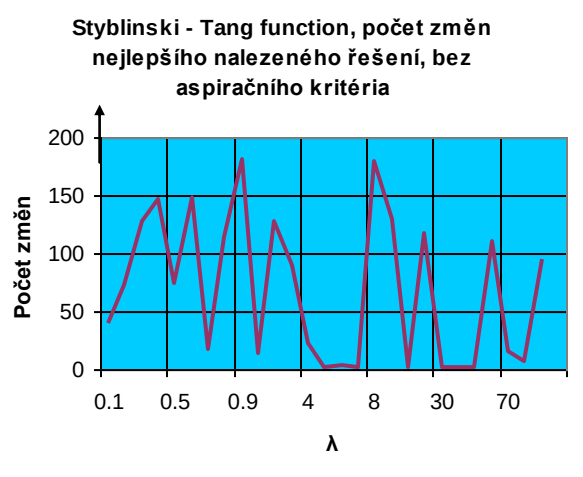
$$\frac{1}{2}\sum_{i=1}^{2}\left(x_i^4 - 16x_i^2 + 5x_i\right) \quad x_1, x_2 \in \langle -5,5 \rangle$$

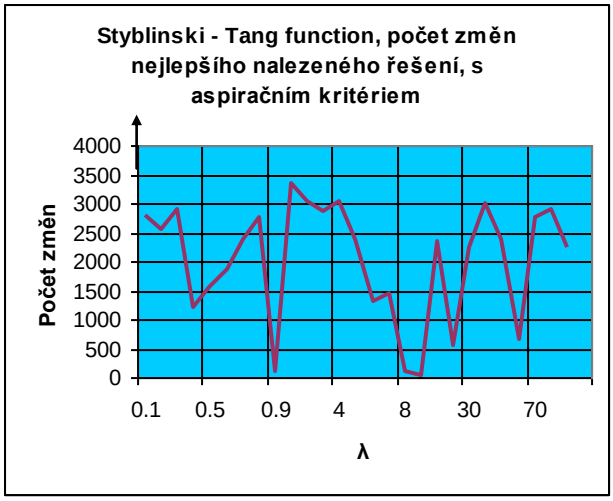


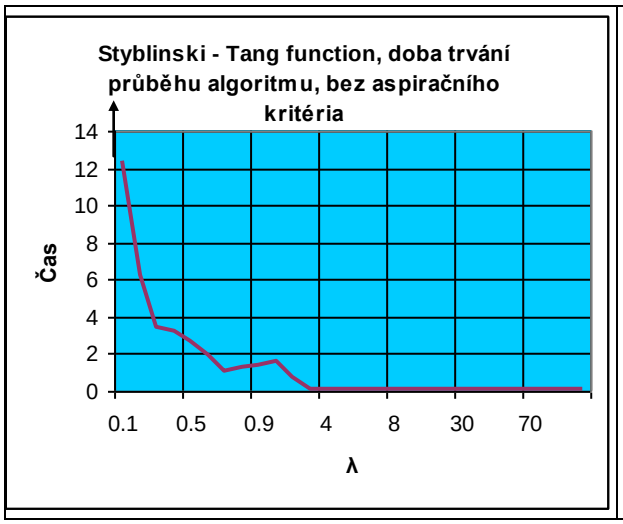


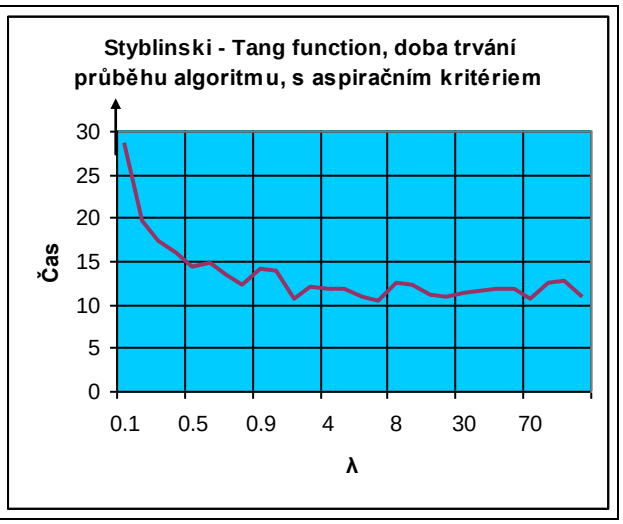









Styblinski - Tang function, doba trvání průběhu algoritmu, bez aspiračního kritéria
Čas
14
12
10
8
6
4
2
0
0.1
0.5
0.9
4
8
30
70
λ


Styblinski - Tang function, doba trvání průběhu algoritmu, s aspiračním kritériem
Čas
30
25
20
15
10
5
0
0.1
0.5
0.9
4
8
30
70
λ

ÚDAJE PRO KNIHOVNICKOU DATABÁZI

Název práce	Evoluční algoritmy v optimalizačních problémech veřejné správy
Autor práce	Ing. Jan Panuš
Obor	Systémové inženýrství a informatika
Rok obhajoby	2008
Vedoucí práce	Doc. RNDr. Bohdan Linda, CSc.
Anotace	V této doktorské disertační práci je předvedena úprava známých algoritmů z oblasti lokálního prohledávání. Do navrženého algoritmu penalizačního lokálního prohledávání byl přidán penalizační faktor spolu s aspiračním kritériem, které by mělo urychlit vyhledávání optimálního (nebo alespoň suboptimálního) řešení ve vybraných problémech. Následně je předvedena funkcionalita algoritmu na vybraných úlohách z oblasti problematiky obchodního cestujícího a také funkcionalita na vybraných testovacích funkcích. Pro charakteristiku předvedeného algoritmu byly naměřeny různé experimentální hodnoty, které jsou porovnávány, a na jejich základě je zkoumáno, zda je algoritmus úspěšný či nikoliv.
Klíčová slova	Optimalizace, stochastické optimalizační algoritmy, penalizační lokální prohledávání, zakázané prohledávání, aspirační kritérium, problém obchodního cestujícího.