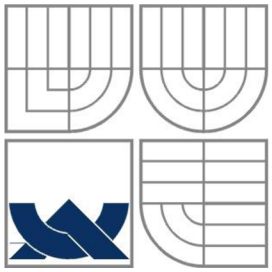
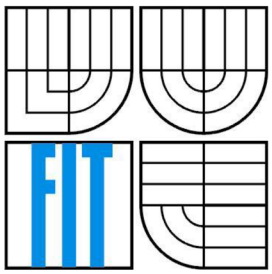


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ŘEŠENÍ OPTIMALIZAČNÍCH ÚLOH ALGORITMY PSO

SOLVING OPTIMIZATION TASKS BY PSO ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MAREK GONZÁLEZ
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. FRANTIŠEK ZBOŘIL, CSc.
SUPERVISOR

BRNO 2012

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2011/2012

Zadání bakalářské práce

Řešitel: **González Marek**
Obor: Informační technologie
Téma: **Řešení optimalizačních úloh algoritmy PSO**
Solving Optimization Tasks by PSO Algorithms
Kategorie: Umělá inteligence

Pokyny:

1. Prostudujte teorii základních optimalizačních algoritmů PSO (Particle Swarm Optimization).
2. Vyberte alespoň dvě úlohy, vhodné pro demonstraci činnosti těchto algoritmů.
3. Pro řešení vybraných úloh navrhněte a implementujte příslušné programy/applety.
4. Proveďte experimenty s cílem zjištění vlivu nastavitelných parametrů.
5. Zhodnoťte získané výsledky.

Literatura:

- Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence, Oxford University Press, 1999.
- Sivanandam, S. N.,Deepa, S. N.: Introduction to Genetic Algorithms, Springer, 2008

Při obhajobě semestrální části projektu je požadováno:

- První dva body zasdání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zbořil František V., doc. Ing., CSc.**, UITS FIT VUT
Datum zadání: 1. listopadu 2011
Datum odevzdání: 16. května 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Práce se zabývá popisem algoritmu particle swarm optimization (PSO) a demonstrací jeho činnosti na vybraných optimalizačních úlohách. PSO byl převážně navržen pro spojitou optimalizaci a řadí se mezi algoritmy hromadné inteligence. Práce obsahuje úvod do problematiky optimalizace a teoretický popis algoritmu. Po teoretické části následuje část praktická, která se věnuje implementaci algoritmu a hledání vhodného nastavení jeho parametrů. Řešené úlohy jsou shlukování, problém obchodního cestujícího a hledání minima vícerozměrných funkcí.

Abstract

In this document we describe the Particle Swarm Optimization (PSO) and discuss its performance in solving optimization tasks. PSO is stochastic population-based computational method mainly focused on continuous optimization. We give an introduction to the field of optimization and provide a theoretical description of the PSO method. We have implemented the method in C/C++ and investigated the best working parameter set. The implementation is evaluated on clustering, travelling salesman problem, and function minimization case studies.

Klíčová slova

Optimalizace rojem částic, hromadná inteligence, optimalizace, shlukování, problém obchodního cestujícího, minimalizace funkcí.

Keywords

Particle swarm optimization, swarm intelligence, optimization, clustering, travelling salesman problem, function minimization.

Citace

GONZÁLEZ, Marek. *Řešení optimalizačních úloh algoritmy PSO*. Brno, 2012. Bakalářská práce. FIT VUT v Brně. Vedoucí práce Doc. Ing. František Vítězslav Zbořil, CSc.

Řešení optimalizačních úloh algoritmy PSO

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Doc. Ing. Františka V. Zbořila, CSc. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……

Marek González

15. 5. 2012

Poděkování

Rád bych poděkoval svému vedoucímu Doc. Ing. Františku V. Zbořilovi, CSc. za všechny podněty a odbornou pomoc, které mi při psaní této práce poskytl.

Obsah

1 Úvod

Optimalizace je dnes důležitým procesem pro mnoho oborů lidské činnosti. Celou řadu problémů z nejrůznějších odvětví, od průmyslu po ekonomii, lze formálně převést na optimalizační úlohy řešitelné výpočetní technikou. Úkolem optimalizace je pak danou činnost v co největší možné míře zefektivnit. Optimalizačních metod a přístupů existuje velmi mnoho. Některé metody jsou založeny na fyzikálních principech, některé se opírají o silný matematický aparát a některé jsou inspirovány přírodou. Jednou z takových metod je i particle swarm optimization[1] (zkráceně PSO), jehož popis, implementace a následné experimentování s parametry algoritmu jsou cílem této práce.

Práce je rozdělena na část teoretickou a na část praktickou. První kapitola se zabývá pojmy z oblasti optimalizace. Oblast optimalizace je, co se týče využívaných metod a technik, velmi rozsáhlá a tato kapitola si neklade za cíl podat jejich hlubší rozbor. Jejím úkolem je podat toliko nezbytný teoretický základ pro popis principu PSO. Názvy některých zde uvedených pojmů se často zdroj od zdroje liší. Protože PSO spadá do umělé inteligence, práce se drží názvosloví právě z tohoto oboru.

Druhá teoretická kapitola je věnována samotnému PSO. Přestože variant a modifikací tohoto algoritmu existuje celá řada, tato kapitola se jimi zcela záměrně příliš nezabývá a diskutuje pouze jednu, v literatuře nejčastěji popisovanou, variantu. Protože však popisovaná varianta nemá všechny prvky striktně dány a umožňuje řešit různá úskalí více způsoby, kapitola nabízí na řešení některých aspektů spojených s PSO více pohledů. Mimo tyto aspekty, které je nutno při implementaci zohlednit, má na samotnou činnost algoritmu zásadní vliv hned několik parametrů. I těmto parametrům a jejich vlivům na běh PSO je zde věnován široký prostor.

Poslední teoretická kapitola se věnuje definicím vybraných optimalizačních úloh. Vybrané úlohy pro demonstraci PSO jsou shlukování, problém obchodního cestujícího a problém hledaní minima více rozměrných funkcí. Zejména pro problém obchodního cestujícího, který je diskrétním problémem, bylo nutno popsat všechny změny oproti PSO z předchozí kapitoly, protože PSO byl původně navržen pro problémy spojité.

[1] Český termín pro algoritmus je optimalizace rojem částic, leč v dalším textu je používán termín anglický.

První kapitola praktické části je zaměřena na popis implementačního výstupu této práce. Zde je uvedena struktura aplikací, jejich popis a problémy spojené s implementací. Výsledky experimentů s parametry implementovaného algoritmu jsou pak prezentovány v poslední kapitole.

2 Optimalizace

Optimalizace je matematická disciplína zabývající se hledáním nejlepšího možného řešení dané úlohy. To znamená, že cílem optimalizace je najít takové parametry úlohy, které dávají nejlepší výstup. Například z hlediska statiky je dobré, aby konstrukce měly minimální hmotnost, poddajnost a napětí při maximální tuhosti, přičemž některé z těchto kritérií jsou navzájem v rozporu. Optimalizace takovéto úlohy bude spočívat v nalezení vyhovující kombinace výše zmíněných parametrů [11]. Z matematického pohledu se hledají takové hodnoty definičního oboru funkce, kterým odpovídají optimální hodnoty z oboru hodnot.

2.1 Pojmy

2.1.1 Optimum

Nejlepší řešení úlohy je posuzováno vzhledem k zadaným kritériím $F = \{f_1, f_2, f_3, \dots, f_n\}$. Kritérium f je funkce a pro jednokriteriální optimalizaci $|F| = 1$ je problém vyjádřen jako hledání minima či maxima (obecně optima) této funkce. Hledání minima funkce f je ekvivalentní k hledání maxima funkce $-f$ [1].

Protože u úloh, kterými se tato práce zabývá, je za optimum považováno globální minimum, pojem optimalizace se bude v tomto textu vztahovat k hledání globálního minima. Optimalizační problém lze formálně zadat následovně: Je dána funkce $f : X \rightarrow Y$. Najdi takové $\tilde{x} \in X$, pro které platí $f(\tilde{x}) \leq f(x)\ \forall x \in X$.

Je-li $|F| > 1$, hovoří se o vícekriteriální optimalizaci. Pro účely této práce bude použito zjednodušení takové, že $F(x) = \omega_1 f_1(x) + \omega_2 f_2(x) + \omega_3 f_3(x) + \cdots + \omega_n f_n(x)$. Tedy že výsledná funkční hodnota je dána váženým součtem funkčních hodnot všech kritérií. Potom pro hledané minimum $\tilde{x} \in X$ platí $F(\tilde{x}) \leq F(x)\ \forall x \in X$.

2.1.2 Objektivní funkce

Funkce f se nazývá objektivní funkce. Její definiční obor tvoří N-dimenzionální prostor všech možných řešení, tzv. stavový prostor. Objektivní funkce nemusí být vždy matematický výraz. Může jít o algoritmus provádějící komplexní simulace apod. [2]. Cílem výpočtu je zobrazení stavového prostoru do prostoru funkčních hodnot $f : \mathbb{R}^n \rightarrow \mathbb{R}$ tak, aby každému prvku stavového prostoru byla přiřazena právě jedna hodnota tzv. cena. Cena pak udává optimálnost řešení [1].

2.1.3 Stavový prostor

Na stavový prostor objektivní funkce mohou být kladeny dodatečné omezující podmínky plynoucí z logiky řešené úlohy nebo z praktických důvodů. Prvky stavového prostoru splňující omezující podmínky se nazývají kandidátní řešení. N-dimenzionální stavový prostor obsahuje kandidátní řešení v podobě vektorů o N prvcích. Jinými slovy je kandidátní řešení kombinace N parametrů a cílem optimalizace je najít takovou kombinaci parametrů, která vyhovuje podmínkám optimality.

Stavový prostor lze rozdělit na spojitý a diskrétní. Diskrétní stavový prostor se vyznačuje konečnou množinou kandidátních řešení. Navzdory tomu cena těchto kandidátních řešení není omezena pouze na celočíselné hodnoty, ale může nabývat i hodnot reálných. Spojitý stavový prostor se skládá z nekonečné množiny kandidátních řešení.

2.2 Optimalizační algoritmy a jejich klasifikace

Obecně lze optimalizační algoritmy rozdělit do dvou základních tříd, a to na algoritmy deterministické a algoritmy stochastické.

2.2.1 Deterministické algoritmy

Problémy, u kterých je předem známá souvislost objektivní funkce s kandidátními řešeními, jsou řešitelné pomocí deterministických algoritmů. Tyto algoritmy dávají pro daný problém vždy stejné optimální řešení. Avšak pro problémy, jejichž stavový prostor je příliš velký či objektivní funkce není příliš známá, přestávají být efektivní.

2.2.2 Stochastické algoritmy

Výše zmíněná úskalí do značné míry řeší druhá skupina algoritmů. Zde hraje určitou roli nedeterminismus. Tyto algoritmy se snaží vlivem náhody a heuristických znalostí o problému vybrat co nejmenší počet kandidátních řešení k prozkoumání. Právě díky tomuto omezení stavového prostoru se daří nalézt řešení v kratším čase. Nalezené řešení může být pro každý běh algoritmu jiné a nemusí být vždy nejlepší. Může se takovému řešení pouze blížit. Nicméně pro různé problémy může být i málo optimální řešení vyhovující, protože bývá přednější nalézt dostatečně optimální řešení v krátkém čase, než čekat na nejlepší výsledek dlouhou dobu [2].

Stochastické algoritmy jsou dále děleny na další podskupiny algoritmů. Jednou z těchto skupin je skupina heuristik využívající kolektivní neboli hromadnou inteligenci. Právě do této skupiny patří i particle swarm optimization, o kterém bude vedena diskuze v kapitole č. 3.

2.2.3 Hromadná inteligence

Kolektivní inteligence je jedna z disciplín umělé inteligence. Zaměřuje se na multi-agentní systémy, které se vyznačují tím, že nemají centrální řízení. Celkové chování systému je určováno spoluprací jednoduchých entit. Ta probíhá na základě vzorů, které byly v přírodě vypozorovány z chování různých sociálních zvířat a hmyzu (např. mravenců, včel, ryb, ptáků, …) [3].

3 Particle Swarm Optimization

Jde o stochastický populačně zaměřený algoritmus pro globální optimalizaci inspirovaný sociálním chováním ptáku a ryb. Algoritmus byl vyvinut v roce 1995 a jeho autory jsou Dr. Eberhart a Dr. Kennedy. Particle swarm optimization (dále jen PSO) se řadí mezi algoritmy hromadné inteligence.

V průběhu let byl PSO úspěšně aplikován v mnoha oblastech. Například v robotice pro plánování tras, ve strojovém učení, v meteorologii pro předpověď počasí, v počítačové grafice pro zpracování obrazu a videa, apod. I přesto, že původní algoritmus byl definován pro problémy se spojitým stavovým prostorem, se PSO uplatňuje i při řešení kombinatorických, tedy diskrétních problémů, jakými jsou například problém obchodního cestujícího nebo problém plánování tras vozidel [12]. Důvodů úspěšnosti PSO je několik. Algoritmus je odolný vůči velikosti a komplexnosti řešených problémů. Dokáže konvergovat a najít optimální řešení i tam, kde jiné metody selhávají.

3.1 Princip

Jak bylo řečeno v nadřazené kapitole, PSO je založeno na chování hejna ptáků nebo ryb. Při pozorování sociálního chování ptáků v hejnu bylo zjištěno, že pokud jeden člen hejna objeví dobrou cestu (např. za jídlem, ochranou, atd.), všichni další jedinci jsou schopni ho rychle následovat i přes to, že se vyskytují na opačném konci hejna. Nicméně není žádoucí, aby ho celé hejno následovalo po stejné stopě. Aby byl prostor prohledáván efektivněji, jedinci se pohybují s určitou mírou vlastní vůle a nezávislosti. Jejich směr letu tak není ovlivněn pouze hejnem [4].

V terminologii PSO se jedinci označují jako částice. Každá částice se pohybuje po stavovém prostoru a vždy se nachází na jednom kandidátním řešení. Algoritmus pracuje cyklicky a uchovává v paměti v jednom čase všechny částice, které byly na začátku náhodně rozmístěny po prostoru. Každá částice si udržuje informace o své rychlosti, o své dosud dosažené nejlepší pozici, o nejlepší pozici dosažené jinou částicí v rámci sousedních částic nebo v rámci celého hejna a o své aktuální pozici. Pozice je reprezentována kandidátním řešením. Podle cen kandidátních řešení jsou určovány nejlepší

pozice jednotlivce i celého hejna. Všechny tyto informace se podílí na výpočtu nové pozice. Protože mohou mít obecně různou důležitost, jsou jim pomocí konstant přiřazeny váhy [1].

Během každé iterace jsou pomocí objektivní funkce vyhodnocena všechna kandidátní řešení, nad kterými se nachází částice. Podle výsledků se změní směr pohybu částic. Protože algoritmus nemá žádné informace o průběhu objektivní funkce mimo částice a kromě zapamatovaných nejlepších pozic nemá žádné informace o předešlých výpočtech, celý cyklus vyhodnocení se opakuje pro nové pozice částic i v případě, že již byla pozice vyhodnocována dříve pro jinou částici.

Algoritmus popsaný v této práci připouští možnost výskytu dvou a více částic na stejném kandidátním řešení. V takovém případě je jedno kandidátní řešení vyhodnoceno v dané iteraci vícekrát, což zároveň snižuje prohledávací schopnost hejna. Jednou z příčin této události může být náhodné setkání. V takové situaci se částice od sebe v dalších iteracích pravděpodobně rozlétnou. Zajímavější případ může nastat, když je jednou z částic nalezeno velmi dobré kandidátní řešení, přičemž ostatní částice nejsou schopny po delší dobu nalézt lepší. Hejno se postupem času začne slétávat právě k této pozici, až na ní všechny částice skončí. Potom se hovoří o tzv. konvergenci. Kandidátní řešení, ke kterému algoritmus takto dospěl (dokonvergoval), lze z důvodu úplné ztráty jakékoliv další prohledávací schopnosti považovat za výsledek optimalizace. Je-li toto řešení pouze lokálním optimem, jedná se o předčasnou konvergenci. Opačným jevem ke konvergenci je tzv. exploze roje, tedy jev, při kterém částice získávají stále větší rychlost a nekontrolovatelně se od sebe rozletí.

3.2 Formální model

3.2.1 Pohyb částic

Matematické vyjádření pohybu částic ukazují rovnice (3-1) a (3-2). První rovnice ukazuje výpočet nové polohy částice, druhá výpočet nové rychlosti. Rovnice (3-3) znázorňuje výpočet nejlepší osobní pozice. Výpočet proměnné $\vec{p}_{(t)}^{g}$ v rovnici (3-2) je dán zvoleným topologickým modelem. O topologiích a rovnicích pro jejich výpočet pojednává kapitola 3.2.2. Geometrický význam rovnic (3-1) a (3-2) ve dvourozměrném prostoru je na obrázku 3-1.

$$\vec{x}^i_{(t+1)} = \vec{x}^i_{(t)} + \vec{v}^i_{(t)} \quad \textbf{(3-1)}$$

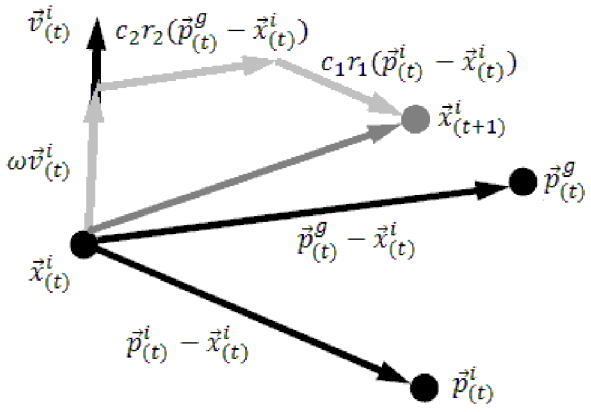
$$\vec{v}^i_{(t+1)} = \omega\vec{v}^i_{(t)} + c_1 r_1(\vec{p}^i_{(t)} - \vec{x}^i_{(t)}) + c_2 r_2(\vec{p}^g_{(t)} - \vec{x}^i_{(t)}) \quad \textbf{(3-2)}$$

$$\vec{p}^i_{(t)} \in \{\vec{x}^i_{(0)}, \vec{x}^i_{(1)}, \dots, \vec{x}^i_{(t-1)}\} | f(\vec{p}^i_{(t)}) = min(f(\vec{x}^i_{(0)}), f(\vec{x}^i_{(1)}), \dots, f(\vec{x}^i_{(t-1)})) \quad \textbf{(3-3)}$$

Pro všechny rovnice obsažené v této kapitole platí, že horní index i označuje i-tou částici a spodní index t vyjadřuje časovou posloupnost, tedy počet dosud provedených iterací cyklu.

V rovnici (3-1) je pozice částice dána vektorem $\vec{x}$. Vektor obsahuje souřadnice kandidátního řešení ve stavovém prostoru. Vektor $\vec{v}$ reprezentuje rychlost a udává změnu pozice.

Rovnice (3-2) je rozdělena na tři části, které ovlivňují chování částice. (Z hlediska vektorového součtu se jedná o tři vektory, jejichž součtem vznikne výsledná rychlost tak, jak je ukázáno na obrázku 3-1.). První část $\omega\vec{v}$ je setrvačnost, kde ω je váha setrvačnosti. Setrvačnost reprezentuje snahu částice pohybovat se v nezměněném směru. Část $c_1 r_1(\vec{p} - \vec{x})$ vyjadřuje individuální paměť, která uchovává pozici, kde částice nalezla nejlepší řešení p ze všech pozic, které navštívila. Částice se snaží k této pozici vracet. Poslední část $c_2 r_2(\vec{p}^g - \vec{x})$ udává sociální vliv okolí. $\vec{p}^g$ je nejlepší ze všech pozic kdy navštívených ostatními částicemi v okolí částice nebo v rámci celého hejna. I k této pozici je částice přitahována. Okolím částic se zabývá kapitola 3.2.2. Konstanty c_1 a c_2 se nazývají kognitivní a sociální váha a udávají míru důležitosti individuální paměti a sociálního vlivu. r_1 a r_2 jsou náhodné hodnoty rovnoměrného rozdělení pravděpodobnosti z intervalu (0,1) [5].



Obr. 3-1: Geometrická interpretace pohybových rovnic.

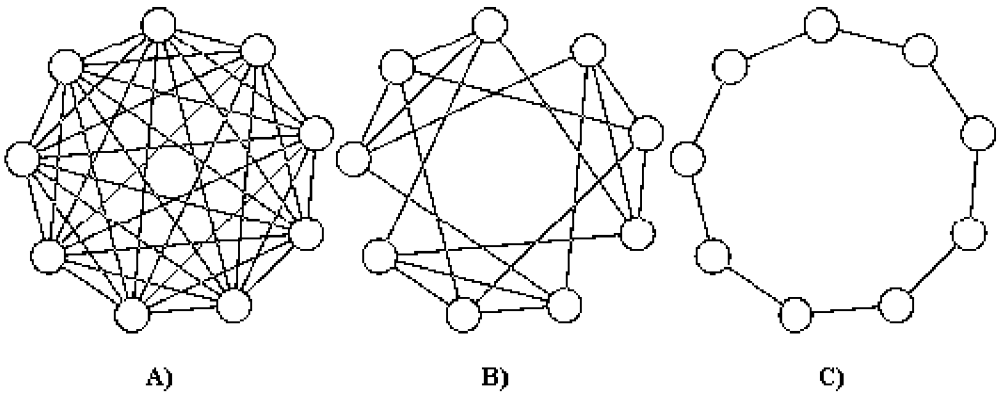
3.2.2 Topologie

Topologie neboli sousedství částic je další faktor, který se významně podílí na výsledku algoritmu. Nejlepší ze všech pozic sousedních částic ovlivňuje pohyb částice, proto je důležité jej vhodně zvolit. Obecně existují dva modely topologie. Oba modely jsou vyjádřeny v rovnicích (3-4) a (3-5). Rovnice představují výpočet nejlepší sociální pozice. Platí, že během výpočtu je použita pouze jedna z nich tedy buďto rovnice (3-4) pro výpočet nejlepší globální pozice, anebo rovnice (3-5) pro výpočet nejlepší pozice v rámci lokálního okolí částice, a to v závislosti na zvoleném topologickém modelu.

$$\vec{p}^{g}_{(t)} \in \{\vec{x}^{0}_{(t)}, \vec{x}^{1}_{(t)}, \dots, \vec{x}^{n}_{(t)}\} | f(\vec{p}^{g}_{(t)}) = min(f(\vec{x}^{0}_{(t)}), f(\vec{x}^{1}_{(t)}), \dots, f(\vec{x}^{n}_{(t)})) \quad \textbf{(3-4)}$$

$$\vec{p}^{li}_{(t)} \in \{\vec{x}^{i-l}_{(t)}, \vec{x}^{i-l+1}_{(t)}, \dots, \vec{x}^{i+l}_{(t)}\} | f(\vec{p}^{g}_{(t)}) = min(f(\vec{x}^{i-l}_{(t)}), f(\vec{x}^{i-l+1}_{(t)}), \dots, f(\vec{x}^{i+l}_{(t)})) \quad \textbf{(3-5)}$$

Rovnice (3-4) odpovídá globálnímu sousedství tzv. *gbest* modelu. V tomto modelu jsou propojeny všechny částice navzájem a jsou přitahovány ke stejné nejlepší pozici. Všechny částice tedy sdílejí stejnou informaci. Výhodou je menší výpočetní náročnost, protože nejlepší sousední pozice nemusí být počítána pro každou částici zvlášť. Algoritmus rychle konverguje, nicméně snadno uvízne v lokálním minimu. Model A na obrázku 3-2 odpovídá globálnímu sousedství. [9].



Obr. 3-2: Typy topologií.

Druhým modelem je lokální sousedství, kterému odpovídá rovnice (3-5). Zde jsou mezi sebou propojeny pouze určité částice. Tento model se zkráceně označuje jako *lbest*. Každá částice má několik sousedů, od kterých čerpá informace a vybírá si z nich jednu nejlepší pozici. Sousednost ovšem není určována podle blízkosti částic ve stavovém prostoru, ale podle pořadového čísla částice

v rovnicích tedy podle proměnné i. Důvodem je menší výpočetní náročnost, protože není třeba provádět žádné shlukování částic. [10]. Z programátorského pohledu řečeno, jsou částice seřazeny v jednorozměrném poli. Pak proměnná l v rovnici (3-5) udává počet sousedů nalevo a napravo od částice. Pro celkovou velikost lokálního sousedství N_i (počet všech sousedů částice) platí $N_i = l * 2$. S rostoucí velikostí sousedství roste rychlost konvergence algoritmu a zároveň se zvyšuje riziko uváznutí v lokálních minimech. Globální sousedství lze považovat za speciální případ lokálního, kdy platí $N_i = N$, kde N je velikost celé populace. Schéma B na obrázku 3-2 je sousedství pro $N_i = 4$ v literatuře označované jako von Neumannovo. Sousedství C na obrázku je pro $N_i = 2$.

3.2.3 Parametry

Na rychlosti a na výsledku výpočtu se významnou měrou podílí parametry. Při špatně zvolených parametrech se algoritmus může stát nestabilním. Může dojít k explozi roje, uváznutí v lokálních minimech apod. Mezi základní parametry algoritmu patří velikost populace N, velikost sousedství l, maximální počet kroků K, kognitivní váha c_1 a sociální váha c_2 a konstanta setrvačnosti ω. Ideální hodnoty těchto parametrů se mohou pro různé typy úloh lišit. Praktická část práce se bude zabývat hledáním vhodných hodnot parametrů pro řešené úlohy a následným porovnáním získaných výsledků s referenčními hodnotami. Referenční hodnoty parametrů a jejich předpokládané vlivy na výsledek výpočtu budou diskutovány v následujících odstavcích.

Od velikosti populace se odvíjí doba výpočtu. S přibývajícími částicemi roste doba výpočtu, ale také se zvětšuje prohledávací schopnost algoritmu, proto je doporučeno pro problémy s velkým stavovým prostorem nebo pro problémy s velkým počtem lokálních minim počet částic zvýšit. Jako referenční hodnota pro průměrné problémy se uvádí $N = 20$ [10].

Konstanta setrvačnosti v původním návrhu PSO chyběla. Později se ukázalo, že je důležitým faktorem pro konvergenci algoritmu. Vhodně zvolená setrvačnost zlepšuje rychlost konvergence. Příliš vysoká setrvačnost vede k explozi roje a k související ztrátě prohledávací schopnosti. V literatuře se uvádí, že již hodnota $\omega > 1$ vede k časté nestabilitě algoritmu. Dále bylo zjištěno, že pro rychlejší konvergenci je výhodné mít na začátku běhu spíše vyšší hodnotu konstanty setrvačnosti a ke konci hodnotu nižší. Ze začátku algoritmus nemá žádné informace o přibližné pozici optima. Vyšší

hodnota setrvačnosti umožňuje prohledat větší prostor, což je pro začátek výhodné. Za předpokladu, že algoritmus postupem času konverguje k optimu, je vhodné prohledávaný prostor zmenšit. Sníží se tak riziko oscilace kolem hledaného optima. Zmenšením hodnoty setrvační konstanty se na výpočtu rychlosti více projeví vliv zbylých dvou částí rychlostní rovnice, důsledkem čehož se zúží prohledávaný prostor. Tyto poznatky byly využity v jedné z modifikací PSO, kdy se hodnota setrvační konstanty snižuje dynamicky za běhu. Jako optimální rozsah hodnot je uváděn interval $\omega \in < 0.4, 0.9 >$ [10].

Sociální a kognitivní váha jsou souhrnně nazývány akcelerační konstanty. Obě tyto konstanty se podílejí na pohybu částic a jejich hodnoty mají podobný efekt na průběh výpočtu jako konstanta setrvačnosti. Příliš vysoké hodnoty vedou k nestabilitě, příliš nízké k pomalé konvergenci. Aby se předešlo cyklickému opakování drah částic, byly zavedeny náhodné proměnné r_1 a r_2. Pro sociální a kognitivní váhu bylo experimentálně zjištěno, že má platit $c_1 + c_2 = 4$.

Maximální počet kroků nemá na samotný průběh algoritmu vliv. Ovlivňuje ale výsledek výpočtu. Protože tento parametr stanovuje, kdy se má PSO zastavit a vrátit svoje dosud nejlepší nalezené řešení jako výsledek optimalizace, jeho nízká hodnota může vést k vracení neoptimálních výsledků a jeho příliš vysoká hodnota vede ke zbytečnému prodlužování doby výpočtu, a to zejména v případech, kde již nastal stav konvergence. Proto je vhodné zavést další koncovou podmínku, která je splněna v momentě, kdy se již žádné částice nehýbou.

Vliv topologie byl probírán v kapitole 3.2.2. Podle [6] je pro většinu problémů nejvhodnější von Neumannovská topologie $l = 2$, kde je dosaženo dobrého poměru mezi rychlostí konvergence a odolností na uváznutí v lokálních optimech.

3.2.4 Omezení

Algoritmus popsaný v předchozích kapitolách žádným způsobem neomezuje pohyb částic. Ty se tak mohou dostat mimo stavový prostor. Metody, které se používají, jsou dvojího druhu. Lze použít určitý druh zdi nebo omezit maximální rychlost částice.

Při omezování maximální rychlosti částice se do výpočtu rychlosti přidává rovnice (3-6).

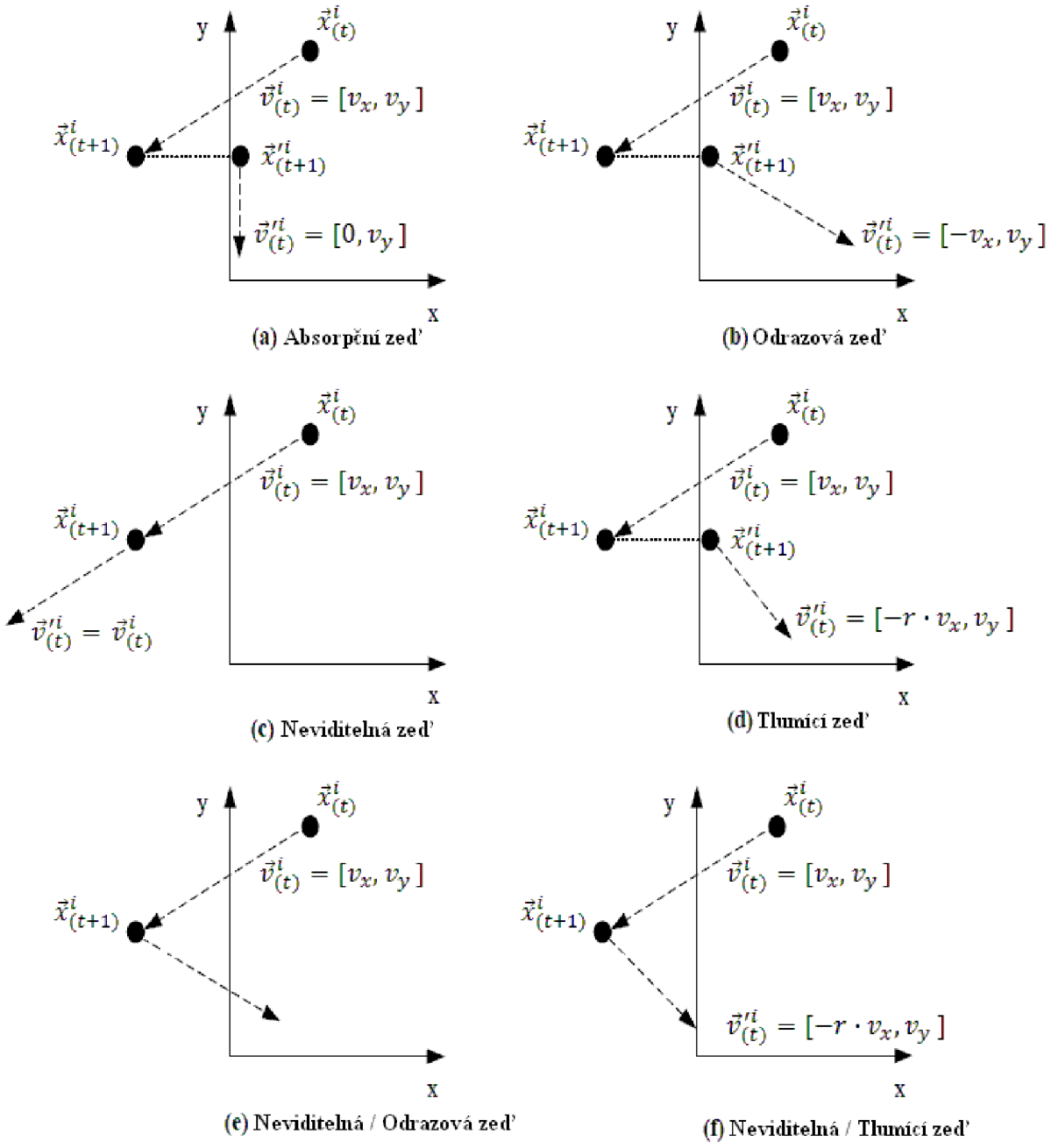
$$\vec{v}^{\,i}_{(t+1)} = min(\vec{v}^{\,i}_{(t+1)},\ \vec{V}_{max}) \qquad \textbf{(3-6)}$$

Rovnice (3-6) říká, že rychlost $\vec{v}_{(t+1)}^{i}$ spočítána podle (3-2) je nahrazena $\vec{V}_{max}$ platí-li $\vec{v}_{(t+1)}^{i} > \vec{V}_{max}$. Žádná částice se tak nemůže pohybovat rychleji než je maximální rychlost $\vec{V}_{max}$. Problém nastává při zvolení $\vec{V}_{max}$. Příliš nízká hodnota maximální rychlosti vede ke snížení prohledávací schopnosti a většímu riziku uváznutí v lokálních minimech. Příliš vysoká hodnota omezuje pohyb nedostatečně. Proto se maximální rychlost odvozuje od velikosti stavového prostoru, jak ukazuje rovnice (3-7), kde $\vec{x}_{max}, \vec{x}_{min}$ jsou hranice stavového prostoru a $\varepsilon \in (0,1>$ [13].

$$\vec{V}_{max} = \varepsilon(\vec{x}_{max} - \vec{x}_{min}) \quad \textbf{(3-7)}$$

Protože předchozí metoda nezaručuje, že se částice nedostanou mimo prostor, podle [8] bývá vhodnějším řešením použít určitý druh zdi zabraňující částicím opustit stavový prostor optimalizovaného problému. Částice, která opouští prostor, je vrácena zpět přepočítáním její polohy a rychlosti tak, aby ve stavovém prostoru zůstala. Obrázek 3-3 ukazuje různé typy zdí a způsob, jakým zacházejí s částicí opouštějící prostor.

Absorpční zeď zcela pohltí a vynuluje rychlost částice pro dimenzi, ve které částice opouští prostor. Rozdíl mezi odrazovou zdí a zdí tlumící je ten, že zatímco odrazová zeď pošle částici v opačném směru, tlumící zeď změní rychlost v dané dimenzi podle náhodné proměnné r z intervalu (0,1). Nejzajímavějším typem zdi je zeď neviditelná. Ta nechá částici vylétnout mimo prostor. Zde se nevyhodnocuje objektivní funkce a částice se pozvolna vrací zpět přitahována svoji individuální pamětí a sociálním vlivem za podmínky vhodně zvolené konstanty setrvačnosti. Výhodou tohoto přístupu je menší výpočetní náročnost, protože není nutno přepočítávat nové pozice a rychlosti pro částice, které vylétly mimo prostor. Poslední dva typy zdí na obrázku 3-3 jsou kombinace neviditelné zdi s odrazovou zdí a neviditelné zdi s tlumící zdí [13].



Obr. 3-3: Typy zdí.

3.3 Pseudokód

PSO algoritmus pro každou částici začíná náhodným výběrem startovní pozice a inicializací rychlosti na nulu nebo náhodnou hodnotu, která nesmí být příliš velká, aby se částice nedostaly mimo stavový prostor hned po první iteraci. V dalších cyklech se pro každou částici vyberou nejlepší pozice podle (3-3), (3-4) nebo (3-3), (3-5) a vyhodnocují rovnice (3-1) a (3-2) až do splnění koncové podmínky. Konec optimalizace nemusí být omezen pouze na maximální počet kroků, které má

algoritmus provést. Lze přidat i další koncové podmínky, které běh ukončí například po nalezení vyhovujícího řešení nebo v případě dosažení stavu konvergence. Posloupnost výše popsaných akcí provedených algoritmem ukazuje následující pseudokód s jedinou koncovou podmínkou ukončující algoritmus po dosažení určitého počtu iterací.

```
While K--                              // K počet kroků
     foreach i in N do                 // N počet částic
          if f(x_i) < f(p_i) then      // rovnice (3-3)
               p_i = x_i;
     end
     foreach i in N do
          lbest_update(i);             // rovnice (3-4) nebo (3-5)
          velocity_update(i);          // rovnice (3-2)
          position_update(i);          // rovnice (3-1)
     end
end
```

4 Úlohy

4.1 Shlukování

Shlukování nebo také shluková analýza „patří mezi metody učení bez učitele. Jejím cílem je v dané množině objektů nalézt její podmnožiny – shluky objektů – tak, aby si členové shluku byli navzájem podobní, ale nebyli si příliš podobní s objekty mimo tento shluk“ [citováno z 15, str. 1].

4.1.1 Algoritmus PSO pro shlukování

Shlukování je chápáno jako spojitý problém. Jako pro každý problém je zcela zásadní definovat stavový prostor, objektivní funkci a význam jednotlivých kandidátních řešení. Zde probíraný postup potřebuje ke svému běhu znát navíc informaci o počtu shluků $\mathcal{K}$. Jedno kandidátní řešení, tedy pozice částice, se skládá z vektorů $\vec{\mathcal{C}}$, jejichž počet je roven počtu shluků $\mathcal{K}$. Každý vektor $\vec{\mathcal{C}}$ reprezentuje jeden střed shluku. Obsahuje souřadnice středu shluku v $\mathcal{D}$-dimenzionálním prostoru, kde každá dimenze představuje určitou vlastnost shlukovaných objektů. Jinak řečeno pozice jedné částice reprezentuje jedno možné rozestavení středů shluků a její hodnota odpovídá souřadnicím v matici o $\mathcal{K}$ řádcích a $\mathcal{D}$ sloupcích. Definici objektivní funkce ukazuje rovnice (4-1).

$$f(\boldsymbol{X}_{(t)}^{i}) = w_1 \bar{d}_{max}(\boldsymbol{X}_{(t)}^{i}) + w_2(R_{max} - d_{min}(\boldsymbol{X}_{(t)}^{i})) + w_3 E(\boldsymbol{X}_{(t)}^{i}) \quad \textbf{(4-1)}$$

$$E(\boldsymbol{X}_{(t)}^{i}) = \frac{\sum_{k=1}^{\mathcal{K}} \sum_{\forall n_j \in \mathcal{C}_k} d(\vec{n}_j, \vec{\mathcal{C}}_k)/\mathcal{N}_k}{\mathcal{K}} \quad \textbf{(4-2)}$$

$$\bar{d}_{max}(\boldsymbol{X}_{(t)}^{i}) = max_{k \in 1,2,\ldots,\mathcal{K}} \{\sum_{\forall n_j \in \mathcal{C}_k} d(\vec{n}_j, \vec{\mathcal{C}}_k)/\mathcal{N}_k\} \quad \textbf{(4-3)}$$

$$d_{min}(\boldsymbol{X}_{(t)}^{i}) = min_{p,q \in 1,2,\ldots,\mathcal{K} \wedge p \neq q} \{d(\vec{\mathcal{C}}_p, \vec{\mathcal{C}}_q)\} \quad \textbf{(4-4)}$$

$$d(a,b) = \sqrt{(a_{x1} - b_{x1})^2 + (a_{x2} - b_{x2})^2 + \cdots + (a_{x\mathcal{D}} - b_{x\mathcal{D}})^2} \quad \textbf{(4-5)}$$

Z definice objektivní funkce vyplývá, že jde o vícekriteriální optimalizaci. Jednotlivá kritéria jsou chyba shlukování (4-2), průměrná vzdálenost objektů od středu shluku (4-3) a vzdálenost mezi středy shluků (4-4). Výsledná objektivní funkce je váženým součtem těchto tří kritérií, přičemž w_1, w_2, w_3 jsou jednotlivé váhy.

Pro všechny rovnice platí, že $\boldsymbol{X}_{(t)}^{i}$ je pozice částice i v čase t reprezentující matici se souřadnicemi všech středů shluků $\boldsymbol{X}_{(t)}^{i} = [\vec{c}_1, \vec{c}_2, \ldots, \vec{c}_k]$, $\vec{n}_j$ je objekt, $\mathcal{N}_k$ je počet objektů patřící ke shluku k, $d(a, b)$ je funkce k výpočtu euklidovské vzdálenosti mezi dvěma body v $\mathcal{D}$-dimenzionálním prostoru viz rovnice (4-5). Dále platí, že objekt patří ke shluku, k jehož středu má nejblíže a R_{max} je největší hodnota ze všech vlastností, kterou nabývá nějaký objekt neboli nejvzdálenější kladná souřadnice obsažená ve shlukované sadě objektů [17].

4.2 Problém obchodního cestujícího

Problém obchodního cestujícího (anglicky *Travelling Salesman Problem*), dále jen TSP, je úloha diskrétní kombinatorické optimalizace a patří do množiny NP-úplných problémů. Pro problémy této třídy není znám algoritmus, který by je dokázal řešit s polynomiální časovou složitostí. Proto je TSP řešen algoritmy, které vždy nedávají optimální řešení, ale snaží se k němu v rozumném čase přiblížit. Pro menší instance problému lze pomocí těchto algoritmů opravdové optimální řešení nalézt. Je však nutno dokázat, že žádné lepší řešení neexistuje.

Úloha je zadána následovně: obchodní cestující má za úkol navštívit právě jednou každé z měst na mapě a vrátit se do místa, odkud vycházel. Optimalizačním úkolem je, aby jeho cesta byla co nejkratší. Formálněji řečeno se hledá nejkratší hamiltonovská kružnice v ohodnoceném grafu [14]. TSP má několik variant. Zde se uvažuje varianta, kdy z každého města vede symetrická cesta do všech ostatních.

4.2.1 Algoritmus PSO pro TSP

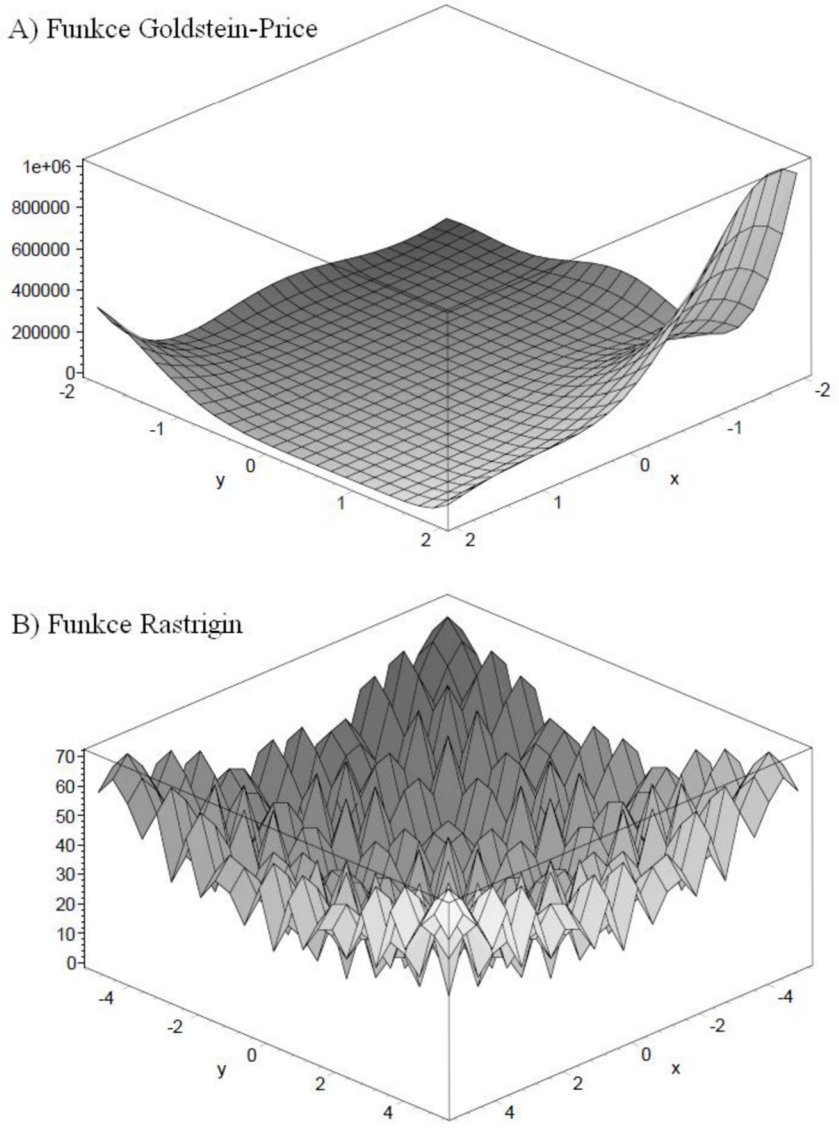
Kandidátní řešení (pozice částice) jsou posloupnosti měst seřazených podle pořadí, v jakém mají být navštívena. Například $\vec{x}_{(t)}^{i} = [1,2,3,4,5]$ znamená, že první bude navštíveno město číslo jedna, poté město číslo dva atd. Objektivní funkcí je pak součet vzdáleností mezi sousedními městy v dané posloupnosti. Protože jde o diskrétní problém a PSO bylo původně navrženo pro problémy spojité, operátory a rychlost v rovnicích (3-1) a (3-2) musí být přetíženy. Rychlost bude chápána jako seznam dvojic měst, která mají být navzájem prohozena a + v rovnici (3-1) znamená aplikace tohoto

prohození. Například je-li $\vec{x}_{(t)}^{i} = [1,2,3,4,5]$ a $\vec{v}_{(t)}^{i} = [(1,4),(2,5)]$, tak $\vec{x}_{(t)}^{i} + \vec{v}_{(t)}^{i} = [4,5,3,1,2]$. Výsledkem operátoru – v rovnici (3-2) je seznam dvojic měst, ve kterých se operandy odlišují. Po aplikaci + z rovnice (3-1) na výsledný seznam a na jeden z operandů, budou oba stejné; například pokud $[1,2,3,4,5] - [1,3,2,5,4] = [(3,2),(5,4)]$, tak $[1,3,2,4,5] + [(3,2),(5,4)] = [1,2,3,4,5]$. Poslední úpravou je operátor + z rovnice (3-2). Zde se jedná o konkatenaci dvou seznamů např. $[(1,2),(3,4)] + [(3,2),(5,4)] = [(1,2),(3,4),(3,2),(5,4)]$. Uvedený postup se nachází v [18] a [19]. Nového významu nabývají i konstanty z rovnice (3-2). Protože se implementační výstup práce v tomto bodě již neshoduje s výše zmíněnými zdroji a jde svoji vlastní cestou, bude o nich zmínka v kapitole 5.3 s popisem implementace.

4.3 Minimalizace funkcí

Jedná se o problém spojité optimalizace sloužící převážně k testování vlastností optimalizačních algoritmů. Úloha spočívá v hledání minima spojitých funkcí o *N*-proměnných. V tomto případě PSO není nutné nikterak upravovat. Funkční předpis minimalizované funkce je objektivní funkcí a pozice částice je bod z jejího definičního oboru. Funkce použité v praktické části jsou na obrázku 4-1 [16].

Testovací funkce lze podle [16] rozdělit do nekolika skupin podle obtížnosti jejich řešení optimalizačními algoritmy. V první skupině jsou funkce unimodálni (s jedním optimem), konvexní, obecně vícerozměrné. Mohou zde být funkce „pěkné“ i funkce s „divokým průběhem“ Do druhé skupiny patří funkce dvourozměrné, multimodální s malým počtem lokálních optim a jedním globálním optimem. Obě tyto skupiny se používají na testování běžných optimalizačních procedur. Z hlediska inteligentních algoritmů, kam spadá i PSO, jsou zajímavější poslední dvě skupiny, a to funkce dvourozměrné multimodální s velkým množstvím lokálních i globálních optim a funkce vícerozměrné multimodální s velkým množstvím lokálních i globálních optim. Právě tyto funkce jsou doporučeny pro testování inteligentních algoritmů. Funkce A z obrázku 4-1 patří do první skupiny a funkce B z obrázku patří do třetí skupiny.



Obr. 4-1: Grafy funkcí.

Funkce A – Goldstein-Price z obrázku 4-1 je definována:

$$f(x,y) = [1 + (x + y + 1)^2 \cdot (19 - 14x + 3x^2 - 14y + 6xy + 3y^2)] \cdot [30 + (2x - 3y)^2 \cdot (18 - 32x + 12x^2 + 48y - 36xy + 27y^2)]$$

Definice funkce B – Rastrigin:

$$f(x,y) = 10 \cdot 2 + [x^2 - 10\cos(2\pi x)] + [y^2 - 10\cos(2\pi y)]$$

5 Realizace

Tato kapitola popisuje jak návrh a implementaci samotného algoritmu PSO, tak i aplikaci pro jeho použití a vizualizaci při řešení zadaných úloh. Protože k implementaci uživatelského rozhraní bylo využito služeb Windows API, stává se aplikace platformě nepřenositelnou. Proto byly vytvořeny dvě varianty aplikace využívající stejné zdrojové kódy: varianta s grafickým uživatelským rozhraním schopná vizualizace běhu algoritmu a přenositelná konzolová varianta poskytující pouze numerické výsledky. Dalším důvodem vytvoření dvou variant byl fakt, že vizualizace shora omezuje dimenzionalitu problému. Přitom hlavní zásadou implementace bylo kladení důrazu na schopnost algoritmu řešit obecné instance zkoumaných problémů (například minimalizovat funkce o třech a více proměnných) a na schopnost algoritmu řešit i jiné problémy. Samotný PSO algoritmus byl vytvořen jako soubor knihovních funkcí, které obě varianty využívají. Knihovna je sama o sobě nezávislá na řešené optimalizační úloze, ale je k ní potřeba dodat modul s definicí úlohy a s objektivní funkcí. Navzdory tomu se povaha řešených úloh v této práci navzájem odlišuje natolik, že musely vzniknout tři různé alternativy algoritmu PSO, a to: diskrétní PSO pro TSP, PSO optimalizované na práci s maticemi pro shlukování a klasické spojité PSO pro minimalizaci funkcí. Jejich obecným popisem se zabývá následující kapitola.

Jako implementační jazyk praktických výstupů byl zvolen jazyk C/C++. Knihovna PSO a konzolová varianta aplikace jsou naprogramovány čistě v jazyku C a GUI grafické varianty aplikace je realizováno v C++.

5.1 Návrh implementace algoritmu PSO

Základní datovou strukturou algoritmu je struktura *Particle*, která reprezentuje jednu částici a uchovává v sobě všechny proměnné popsané v teoretické části: aktuální pozici, rychlost, nejlepší sousední pozici společně s její cenou a nejlepší individuální pozici také s cenou. Pole těchto struktur tvoří hejno. Částice v tomto poli jsou číslované od 0 do *N-1*, kde *N* je velikost populace. Knihovna používá několik privátních proměnných. Kromě samotného hejna *ps* jde především o konstanty

probírané v kapitole 3.2.3. Jmenovitě: kognitivní váha *c1*, sociální váha *c2*, konstanta setrvačnosti *w*, velikost populace *N* a parametr *K* udávající koncovou podmínku v podobě počtu kroků, po jejichž provedení algoritmus skončí. Dále je zde instance struktury *Function f*, která obsahuje položky pro definici objektivní funkce, a to: ukazatel na samotnou objektivní funkci, hranice definičního oboru a dimenzi definičního oboru.

Parametry z předchozího odstavce musí být nastaveny jednorázově pomocí funkce *pso_init()*, která inicializuje hejno. Důležitou součástí jsou privátní funkce realizující výpočty podle rovnic z kapitoly 3.2.1 (popř. 3.2.2). Funkce *position_update()* odpovídá rovnici (3-1), rovnici pro rychlost (3-2) řeší funkce *velocity_update()* a na výpočtu sociálního vlivu se podle rovnice (3-5) podílí funkce *lbest_update()*, která odpovídá topologickému modelu *lbest* s $l = 2$. Tyto privátní funkce jsou volány z funkce *pso_step()*, která zároveň počítá rovnici (3-3) pro nejlepší personální pozici. Funkce *pso_step()* tvoří výpočetní jádro celého algoritmu a odpovídá jí pseudokód z kapitoly 3.3.

Celý proces optimalizace začíná inicializací hejna funkcí *pso_init()*. V této fázi jsou částice náhodně rozmístěny po stavovém prostoru. Rychlost částic je nulová a jejich nejlepší osobní pozice a nejlepší sousedská pozice jsou rovny aktuální pozici. Po inicializaci je možno provést jeden krok algoritmu pomocí funkce *pso_step()* nebo více kroků funkcí *pso_run()*. Počet provedených kroků je určován parametrem *K*, který je zadán při inicializaci. Aktuální stav optimalizace lze v kterékoliv fázi získat zavoláním funkce *pso_getBest()*. Tato funkce vrací index částice s nejmenší cenou nejlepší osobní pozice. Nejlepší osobní pozice takto získané částice je jinými slovy nejoptimálnější řešení, které bylo algoritmem dosud nalezeno. Algoritmus implicitně nedisponuje žádnou ukončující podmínkou, avšak pomocí funkcí *pso_getBest()* a *pso_run()* či *pso_step()* lze snadno nějakou realizovat.

5.2 Implementace algoritmu pro shlukování

Zatímco klasická varianta PSO uvedená v teoretické části uvažuje pozice částic a rychlost za vektory obsahující souřadnice jednoho bodu v prostoru, varianta pro shlukování pracuje s několika body zároveň. Jejich souřadnice jsou uspořádány do matic, proto algoritmus pracuje s abstraktním

datovým typem matice - *Matrix*. Knihovna pro práci s maticemi realizuje operace potřebné pro výpočet rychlosti a pozice částice. Dále byl implementován typ tlumící zdi, kdy se nová pozice vypočítá náhodně i pro dimenze, ve kterých částice neopustila prostor.

Aby byla zaručena obecnost algoritmu, veškeré operace související s načítáním dat ze souboru, zařazováním dat do shluků, počítáním ceny apod. jsou implementovány v modulu C*luster*. Shlukované objekty jsou sdruženy v matici, jejíž každý řádek tvoří jeden objekt složený z vlastností, podle kterých se shlukuje. Každá vlastnost se nachází v jednom sloupci. Počet sloupců vyjadřuje dimenzionalitu problému. Podle těchto údajů jsou vymezeny hodnoty stavového prostoru problému. V každé dimenzi je stavový prostor omezen minimální a maximální hodnotou obsaženou v datech. Pro úplnou definici stavového prostoru je třeba zvolit počet shluků. Tím pádem se vymezí počet řádků v maticích kandidátních řešení. Objektivní funkce má jeden argument, a tím je právě jedno možné rozestavení středů shluků (kandidátní řešení). V této funkci jsou data přiřazena k dodaným shlukům a je spočítána cena podle rovnice (4-1). Váhy v rovnici jsou pevně stanovené na základě pokusů a to: $w_1 = 1$, $w_2 = 0.1$ a $w_3 = 2$. Objektivní funkce kromě ceny vrací matici přiřazení, kde se nachází informace o tom, který objekt je přiřazen k jakému shluku.

5.2.1 Formát dat pro shlukování

Algoritmus pro shlukování načítá shlukované objekty ze souboru. Objekty jsou definovány svými souřadnicemi pro každou dimenzi. Závazný formát těchto souborů je vidět na následujícím příkladu:

```
3, 2        // počet objektů, počet dimenzí
[
0, 1;       // první objekt
1, 2;       // druhý objekt
2, 1        // třetí objekt
]
```

5.3 Implementace algoritmu pro problém obchodního cestujícího

Oproti obecnému návrhu popsanému v kapitole 5.1 byly do implementace přidány nové funkce realizující potřebné operace z kapitoly 4.2.1. Jedná se o funkce *swap()* a *diff()*. První jmenovaná prohazuje dvě zadaná města v cestě. Druhá funkce vybírá rozdílná města ve dvou pozicích a postupně vytváří seznam měst, která mají být prohozena. Protože paměťový buffer pro výsledek má značně omezenou kapacitu a protože není předem známo, kolik výsledný seznam bude mít prvků, funkce *diff()* nevybírá všechny rozdíly. Některé zcela náhodně zamítne. Zamítnutím rozdílu se zároveň simuluje vliv konstant r_1 a r_2, které jinak ve výpočtu nefigurují.

Stěžejním bodem implementace PSO pro TSP je výpočet rychlosti neboli seznamu měst k prohození. Seznam rychlosti je složen ze tří menších podseznamů. Prvním z nich je seznam vytvořený z náhodně vybraných prvků z předešlé iterace. Jde tedy o reprezentaci setrvačnosti. Druhý a třetí seznam tvoří nejlepší osobní pozici a nejlepší sociální pozici. Oba jsou vytvořeny funkcí *diff()*. Výsledný seznam rychlost má konstantní velikost a platí, že součet podseznamů je roven velikosti výsledného seznamu rychlosti. O tom, jak velká část z kapacity rychlostního seznamu bude přidělena jednotlivým podseznamům, rozhodují konstanty ω, c_1 a c_2 podle následujících předpisů:

$$\omega\_size = (domLength / (\omega + c_1 + c_2)) * \omega * VEL\_SIZE\_RATE \quad \textbf{(5-1)}$$

$$c_1\_size = (domLength / (\omega + c_1 + c_2)) * c_1 * VEL\_SIZE\_RATE \quad \textbf{(5-2)}$$

$$c_2\_size = \big(domLength /(\omega + c_1 + c_2)\big) * c_2 * VEL\_SIZE\_RATE \quad \textbf{(5-3)}$$

Ve výše uvedených rovnicích odpovídají ω_size, c_1_size a c_2_size velikostem příslušných podseznamu, $domLength$ je počet měst obsažených v řešené instanci problému a VEL_SIZE_RATE udává, o kolik má být rychlostní seznam větší než počet měst. VEL_SIZE_RATE je na základě pokusů pevně nastavena na hodnotu 2. Z výše uvedeného dále vyplývá, že obě akcelerační konstanty i konstanta setrvačnosti by měly být celočíselné.

Protože každá nová pozice částice vzniká prohazováním měst, neexistuje možnost vylétnutí částic mimo stavový prostor. Z tohoto důvodu nebylo nutné implementovat žádnou zeď nebo jiná omezení. Nicméně musela být implementována jiná úprava. Algoritmus byl velmi náchylný na předčasnou konvergenci, kdy se všechny částice zcela zastavily, proto byl zaveden mechanismus na

detekci a zabránění zpomalování částic. Tento mechanismus, zjistí-li, že nějaká částice zpomaluje, tedy že přestává využívat kapacitu seznamu rychlosti, injektuje do volných míst seznamu náhodně vybraná města k prohození. Částice se tak nikdy nezastaví, ani v případě nalezení opravdového optima.

Jako v předchozím případě i zde jsou funkce pro práci s daty a objektivní funkce umístěny mimo modul s vlastním algoritmem. Teoreticky tak lze řešit i jiný kombinatorický problém. Modul *tsp* obsahuje objektivní funkci počítající délku cesty a parser pro čtení dat ve formátu TSPLIB.

5.3.1 Formát dat pro TSP

Data jsou programu předkládány ve formátu knihovny TSPLIB [20]. TSPLIB je volně dostupná datová knihovna obsahující mnoho instancí TSP. Tato data specifikují typ použitého souřadného systému, souřadnice měst, možné cesty mezi městy, ceny těchto cest apod. Specifikací v TSPLIB existuje celá řada. Aplikace pracuje pouze s těmi druhy instancí, které obsahují souřadnice v euklidovském dvourozměrném prostoru a obsahují pouze souřadnice měst, cesty jsou symetrické, přičemž vedou implicitně mezi všemi městy a jejich cena je euklidovskou vzdáleností mezi dvěma body.

5.4 Implementace algoritmu pro minimalizaci funkcí

Algoritmus pro minimalizaci funkcí využívá neviditelnou zeď. Jinak je shodný s návrhem i teorií. Modul s minimalizovanými funkcemi *objectives* implementuje jejich funkční předpisy a definuje definiční obory. Přestože tomu tak být nemusí, jedná se o dvourozměrné funkce s definičním oborem v obou rozměrech omezeným na interval <-5,12; 5,12>, jak bylo doporučeno v [16]. Dvourozměrné funkce byly vybrány z důvodu přehlednější vizualizace.

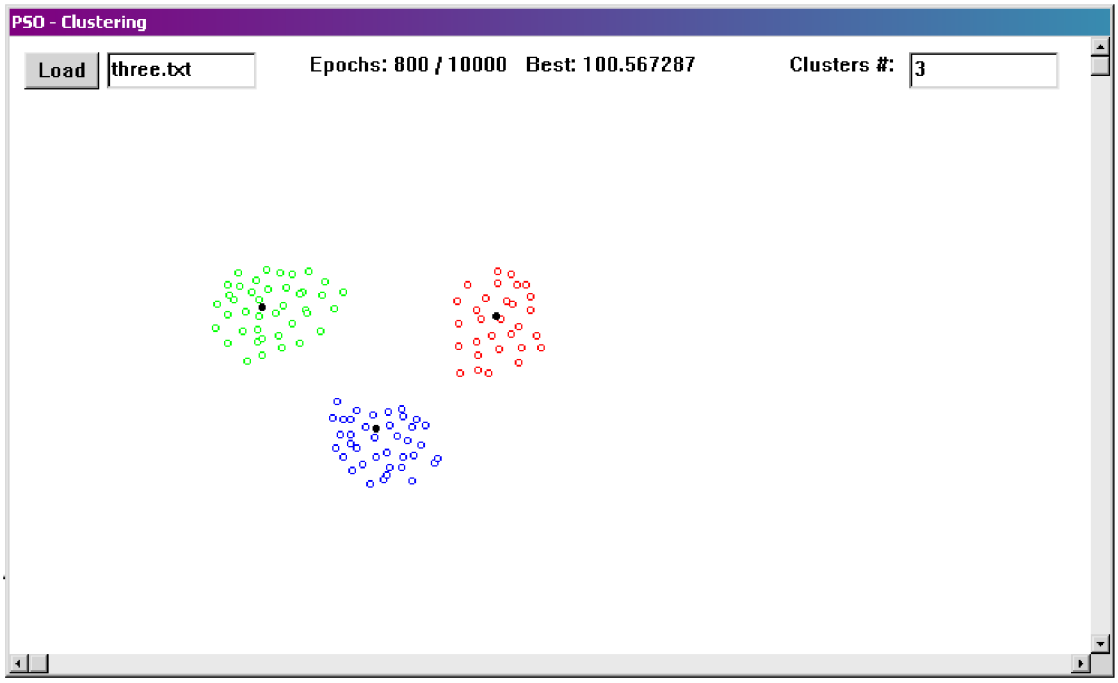
5.5 Popis aplikace

V této kapitole budou popsány vizualizační mechanismy a další stěžejní prvky související především s grafickou variantou aplikace. Podrobný popis ovládacích prvků – stejně tak jako seznam parametrů příkazové řádky pro konzolovou variantu – je dispozici v příloze.

Kromě vizualizace řešených problémů poskytuje aplikace možnost vytvářet nové instance TSP a shlukování pomocí grafického editoru. Protože se v obou případech definují body ve dvourozměrném prostoru, liší se tvorba instancí pro obě úlohy pouze způsobem uložení. Jsou-li souřadnice bodů uloženy jako instance TSP, editor k nim automaticky přidá potřebné metainformace tak, aby bylo vyhověno požadavkům formátu TSPLIB. V případě, že je definice bodů uložena jako instance shlukování, editor vytvoří soubor objektů podle formátu z kapitoly 5.2.1.

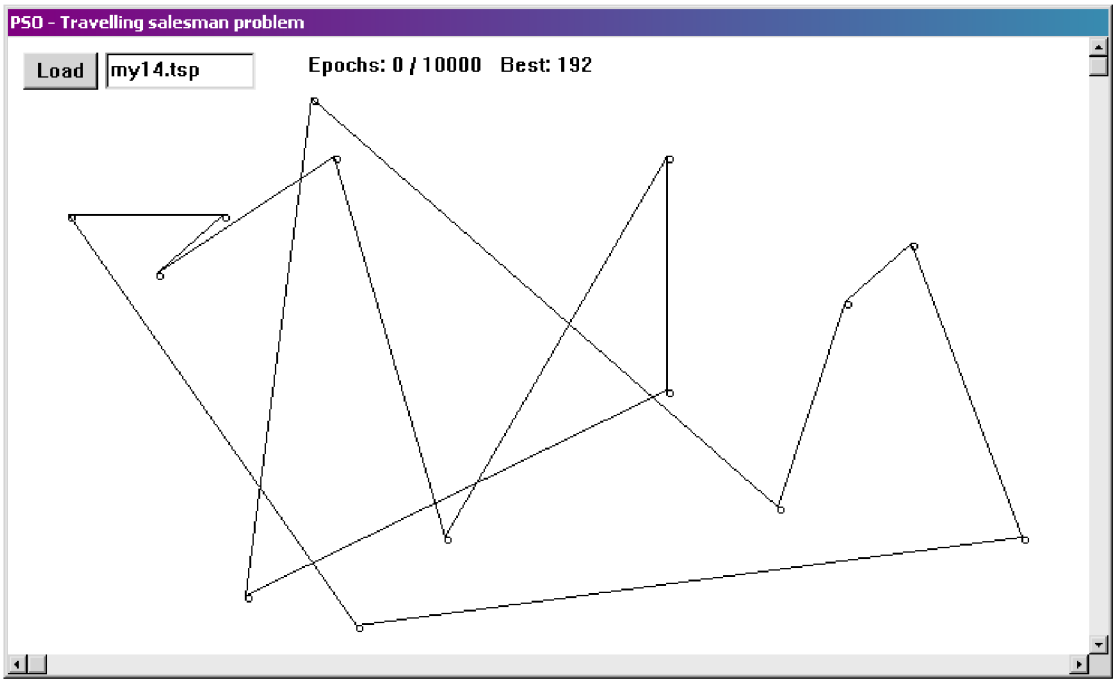
5.5.1 Vizualizace

Na obrázku 5-1 je okno s vizualizací shlukování. Černé body znázorňují středy shluků, barevné body objekty přiřazené k těmto shlukům. Důležité je, že vždy je vidět pouze jedna, a to nejlepší částice. Jinými slovy řečeno, vždy je vidět pouze nejlepší možné rozestavění středů shluků a k nim přiřazené objekty.

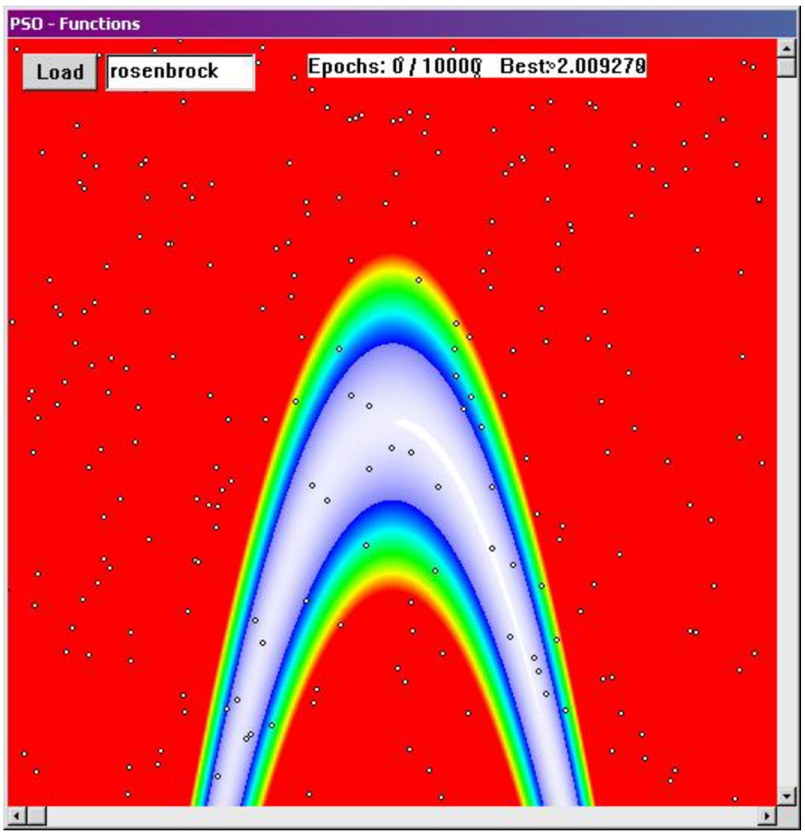


Obr. 5-1: Vizualizace shlukování.

Na obrázku 5-2 je zobrazen problém obchodního cestujícího. Body zde zastupují města a úsečky mezi nimi tvoří možnou cesta. Stejně jako v předchozím případě, vždy je vidět pouze ta nejlepší.



Obr. 5-2: Vizualizace problému obchodního cestujícího.



Obr. 5-3: Vizualizace hledání minima funkcí.

Obrázek 5-3 zobrazuje minimalizaci funkce (konkrétně Rosenbrockovi funkce). Na rozdíl od předchozích dvou případů, zde jsou zároveň vidět všechny částice – černě ohraničené bíle body. Na funkci je pohlíženo z vrchu. Funkční hodnoty jsou podle velikosti zařazeny do barevných skupin. Největší hodnoty jsou červené, nejmenší bíle.

5.5.2 Struktura

Jak již bylo řečeno v úvodu páté kapitoly, obě varianty aplikace využívají stejné moduly. Jádro tvoří moduly s implementací PSO, a to: *pso* – spojité PSO, *dpso* – diskrétní PSO a *xpso* – maticové PSO. Další nedílnou součástí jsou pak moduly s definicí problémů: *objectives* - minimalizace funkcí, *cluster* - shlukování a *tsp* – problém obchodního cestujícího. Modul *objectives* mimo jiné obsahuje i definici struktur *Function*, *DFunction* a *XFunction,* které umožňují mapovat problémy, jejich objektivní funkce a stavové prostory na požadovaný formát PSO knihoven. Aplikace dále obsahuje pomocné moduly *random* pro generování náhodných čísel, *matrix* pro práci s maticemi a *utilities*, kde jsou implementovány další pomocné funkce. Grafická varianta navíc využívá moduly implementující jednotlivá okna grafického uživatelského rozhraní.

6 Experimenty

Tato kapitola se zabývá experimenty s PSO a vhodným nastavením jeho parametrů pro jednotlivé úlohy. Testy probíhaly v konzolové verzi aplikace.

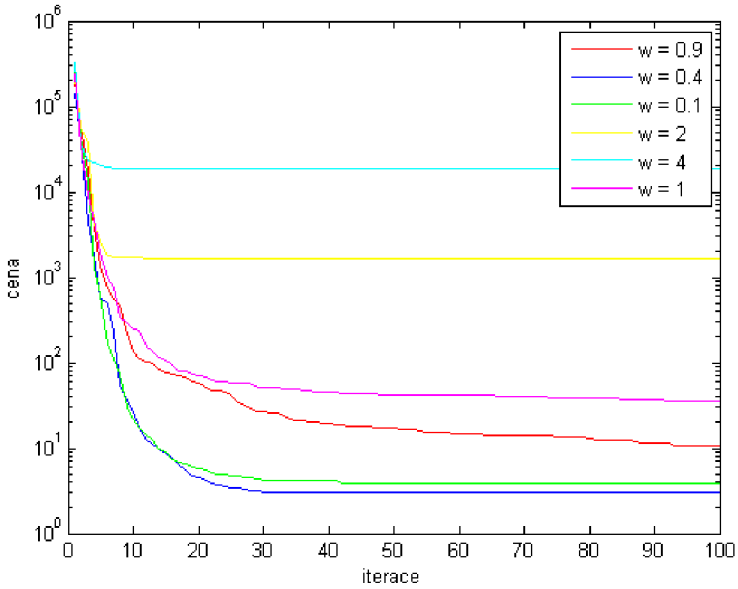
Testy vycházejí ze základního nastavení parametrů $c_1 = 2$, $c_2 = 2$, $\omega = 0.4$ (popř. $\omega = 1$ u TSP), $N = 20$ a $l = 2$. Není-li uvedeno jinak, mění se pouze hodnota zkoumaného parametru. Tato hodnota pak jednoznačně identifikuje test. Zbylé parametry zůstávají neměnné. Protože se PSO řadí mezi stochastické algoritmy, výsledky pouze jednoho spuštění testu nelze považovat za směrodatné, proto jsou získané hodnoty ve většině případů průměrem z vícenásobného běhu testu. Z důvodu vyšší časové složitosti pro některé parametry a úlohy se počet opakování test od testu různí.

6.1 Testovací funkce

Pro demonstraci problému byly zvoleny funkce goldstein-price (obrázek 4-1 A) a funkce rastrigin (obrázek 4-1 B). Na první z nich byly testovány vlivy parametrů c_1, c_2 a ω. Druhá funkce se vyznačuje výskytem velkého množství lokálních optim, proto je vhodná pro ukázku vlivu parametru N.

Výsledky testů jsou ukázány na grafech konvergence, vyjadřující nejlepší nalezenou hodnotu optimalizace v závislosti na čase (počet iterací od začátku výpočtu), a v tabulkách úspěšnosti. Ty procentuálně vyjadřují, kolik spuštění PSO v rámci jednoho testu našlo optimum funkce před koncem výpočtu. Konec výpočtu byl stanoven na $k = 100$ a jeden test byl spuštěn stokrát.

6.1.1 Experiment č. 1 – setrvačnost

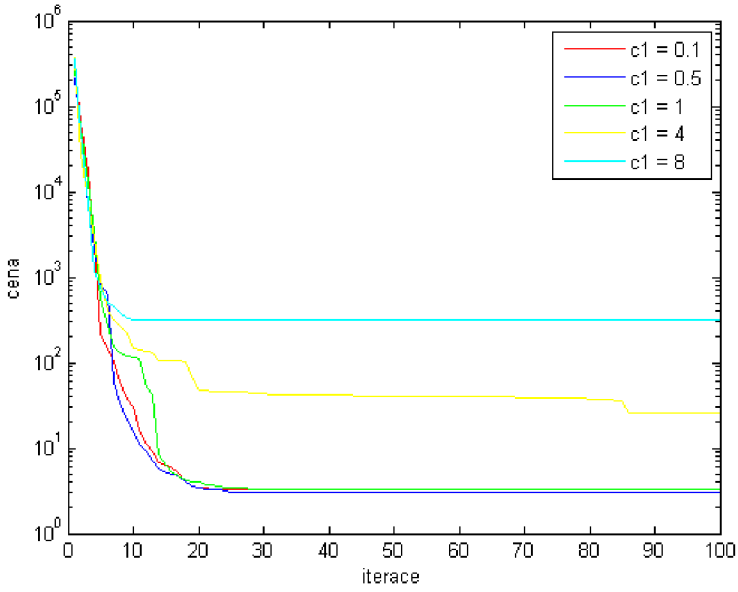


Graf 6-1: Konvergence - test setrvačnosti.

ω	Úspěšnost [%]
0.9	28
0.4	100
0.1	97
2	0
4	0
1	4

Tabulka 6-1: Úspěšnost - test setrvačnosti.

6.1.2 Experiment č. 2 – kognitivní váha

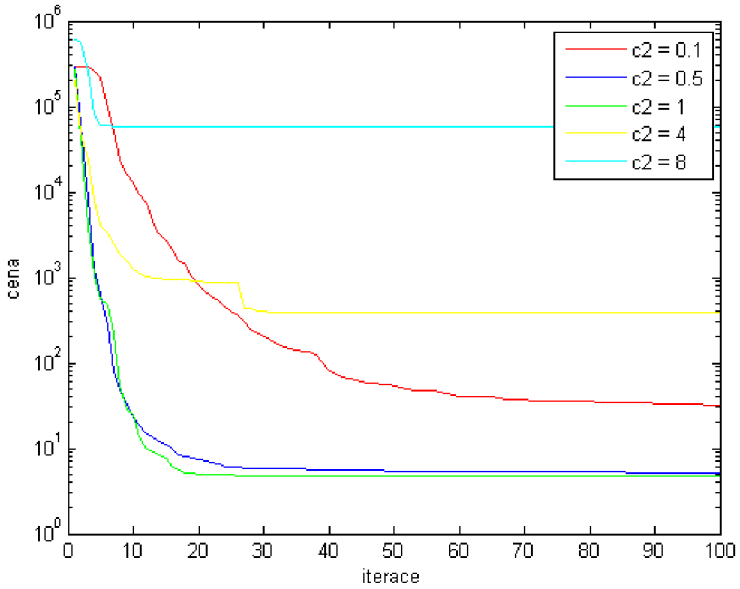


Graf 6-2: Konvergence - test kognitivní váhy.

c_1	Úspěšnost [%]
0.1	99
0.55	10
1	99
4	14
8	1

Tabulka 6-2: Úspěšnost - test kognitivní váhy.

6.1.3 Experiment č. 3 – sociální váha

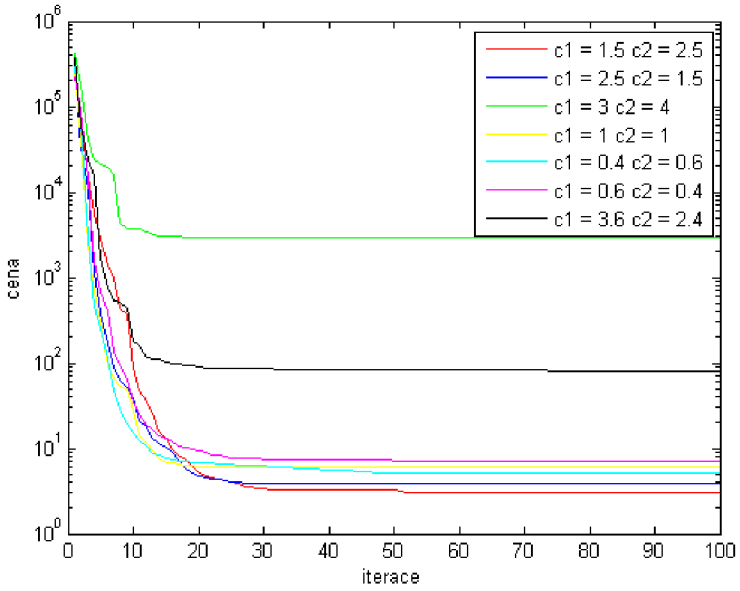


Graf 6-3: Konvergence - test sociální váhy.

c_2	Úspěšnost [%]
0.1	24
0.5	93
1	96
4	2
8	1

Tabulka 6-3: Úspěšnost - test sociální váhy.

6.1.4 Experiment č. 4 – kognitivní váha a sociální váha

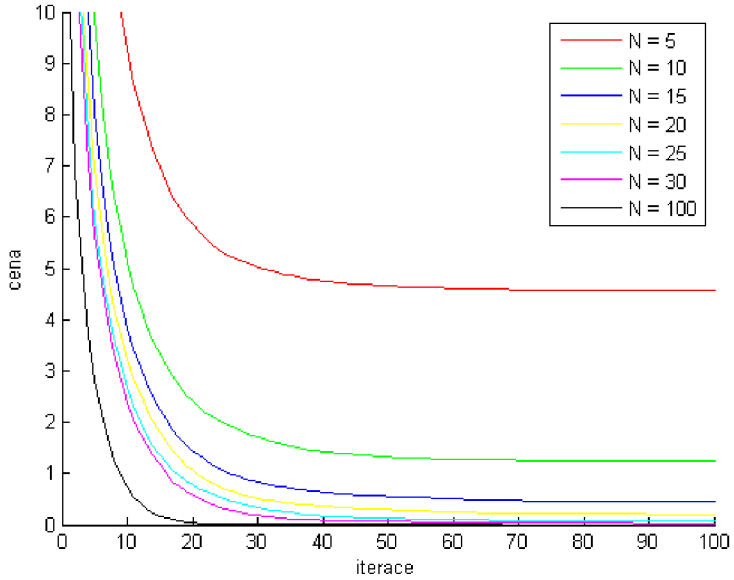


Graf 6-4: Konvergence - test kognitivní a sociální váhy.

c_1	c_2	Úspěšnost [%]
1.5	2.5	100
2.5	1.5	99
3	4	1
1	1	93
0.4	0.6	93
0.6	0.4	79
3.6	2.4	16

Tabulka 6-4: Úspěšnost – test kognitivní a sociální váhy.

6.1.5 Experiment č. 5 – velikost populace



Graf 6-5: Konvergence - test velikosti populace.

N	Úspěšnost [%]
5	31
10	69
15	88
20	94
25	98
30	99
100	100

Tabulka 6-5: Úspěšnost - test velikosti populace.

6.1.6 Zhodnocení

Z experimentů č. 1, č. 2, č. 3 a č. 4 lze vyvodit, že nejlepších výsledků algoritmus dosahuje při kombinaci parametrů, kdy platí, že $c_1 + c_2 = 2$ a $\omega \leq 1$. Tato kombinace odpovídá doporučené kombinaci parametrů uvedené v kapitole 3.2.3 v teoretické části práce. Dále se potvrdil vliv velikosti populace (parametr N) při řešení problémů s velkou koncentrací lokálních optim. U takových problémů se s rostoucí populací snižuje riziko uváznutí, jak ukazuje experiment č. 5.

6.2 Problém obchodního cestujícího

Při experimentování s PSO pro TSP byl nejdříve zkoumán vliv různých kombinací parametrů na rychlost, s jakou algoritmus nalezne optimální cestu pro vyřešenou instanci problému se 14 městy. Druhý experiment hledá souvislosti mezi rychlostí konvergence k optimu, počtem měst a velikostí populace. Každý test byl zopakován desetkrát. Optimum všech použitých instancí bylo předem známé a po jeho nalezení se algoritmus se zastavil. Výsledky jsou prezentovány v tabulkách. Ve sloupci iterace jsou zaneseny počty kroků, které algoritmus provedl, než dospěl k optimální cestě.

6.2.1 Experiment č. 1 – parametry

ω	c_1	c_2	N	iterace
1	2	2	100	546
4	2	2	100	770
1	4	2	100	585
1	2	4	100	634
1	2	2	200	446
1	2	2	300	338
1	2	2	400	312
2	3	4	100	776
4	2	3	100	791
3	4	2	100	1103

Tabulka 6-7: Kombinace parametrů.

6.2.2 Experiment č. 2 – počet měst

Počet měst	N	iterace
5	50	0
8	50	9
11	50	100
14	50	865
17	50	8794
20	50	15419
23	50	66405
5	300	0
8	300	4
11	300	74
14	300	338
17	300	2162
20	300	7132
23	300	40760

Tabulka 6-8: Doba výpočtu v závislosti na počtu měst.

6.2.3 Zhodnocení

Ačkoliv je význam parametrů u PSO pro TSP odlišný od spojitých implementací, experiment č. 1 ukázal, že mírně lepších výsledků dosahovala kombinace parametrů, kdy opět platilo, že $c_1 + c_2 = 2$ a $\omega \leq 1$. Zde je nutno podotknout, že pro $\omega < 1$ vykazuje algoritmus nedefinované chování. Je proto třeba volit celočíselné hodnoty parametrů. Nejlepší hodnota konstanty setrvačnosti je tedy nejmenší možná a to $\omega = 1$. Nicméně parametry c_1, c_2 a ω jsou zastíněné parametrem N. Při nalézání

optima hraje podle tabulky 6-8 největší roli právě velikost populace. Experiment č. 2 ukázal, že tato implementace PSO problém obchodního cestujícího v polynomiálním čase nevyřeší. Jak plyne z tabulky 6-8 doba výpočtu roste s počtem měst exponenciálně.

6.3 Shlukování

Protože skutečné nejlepší řešení u shlukovaných instancí není známo, cílem testů je najít takovou kombinaci parametrů, se kterou nalezne algoritmus lepší řešení než s ostatními kombinacemi. Testy byly prováděny nad dvěma instancemi. První obsahovala 121 objektů v 5 shlucích a druhá obsahovala 397 objektů po 8 shlucích. Výsledky pro první instanci jsou v tabulce 6-9 a pro druhou v tabulce 6-10. Jeden test proběhl patnáctkrát a počet provedených kroků byl stanoven na $k = 10000$.

6.3.1 Experiment č. 1

ω	c_1	c_2	N	Nejlepší nalezené řešení
0.4	2	2	20	81.356895
0.7	2	2	20	81.641831
1	2	2	20	81.999987
4	2	2	20	82.169119
0.4	1.5	2.5	20	81.384355
0.4	3.5	0.5	20	81.723251
1.7	4	2	50	81.020469
2	2	4	50	81.000288
0.4	2	2	100	80.995140
0.4	2	2	200	80.994960
0.4	2	2	300	80.991747

Tabulka 6-9: Vliv kombinace parametrů na řešení instance shlukování se 121 objekty.

ω	c_1	c_2	N	Nejlepší nalezené řešení
0.4	2	2	20	144.779865
0.7	2	2	20	142.974064
1	2	2	20	148.047512
4	2	2	20	148.033059
0.4	1.5	2.5	20	146.041553
0.4	3.5	0.5	20	150.272848
1.7	4	2	50	126.694826
0.4	2	2	50	120.172486
0.4	2	2	100	111.675522
0.4	2	2	200	111.551332
0.4	2	2	300	111.478732

Tabulka 6-10: Vliv kombinace parametrů na řešení instance shlukování s 397 objekty.

6.3.2 Zhodnocení

Z experimentů vyplývá, že důležitost vhodného nastavení parametrů roste s počtem shlukovaných objektů. Dále lze vyvodit závěr, že i shlukování patří mezi problémy, kde PSO může snadno uváznout v lokálních optimech. Důkazem jsou lepší výsledky pro testy s větším počtem částic, i přes nevhodnou kombinaci zbylých parametrů, u které neplatí doporučení z kapitoly 3.2.3. Tento jev je patrný v tabulce 6-10. Velikost populace výrazně ovlivňuje výpočetní čas, proto je třeba udržovat

jeho hodnotu v rozumné míře. Z tohoto hlediska se testy pro $N = 200$ a pro $N = 300$ jeví jako nadbytečné, protože nalezené řešení není příliš rozdílné od testu s $N = 100$.

7 Závěr

V první kapitole teoretické části byl vybudován pojmový aparát, který měl za cíl uvést čtenáře do oblasti optimalizace a poskytnout teoretický základ k pochopení činnosti optimalizačních algoritmů. Tyto pojmy jsou pak využity v kapitole č. 3, kde jsou s jejich pomocí vysvětleny principy algoritmu PSO a význam jeho parametrů.

Na základě teoretických kapitol č. 3 a č. 4, kde jsou definovány vybrané úlohy, byl ve dvou variantách implementován program pro demonstraci činnosti PSO a nezávislá knihovna poskytující funkci algoritmu. Tato knihovna je výpočetním jádrem aplikací a na rozdíl od nich je napsána obecně a s její pomocí lze řešit libovolný problém, je-li implementován.

V poslední kapitole je pak zhodnocena kvalita implementace porovnáním chování algoritmu pro různé kombinace parametrů s referenčním chováním nastudovaným v literatuře. Zejména pro problém hledání minima vícerozměrných funkcí se algoritmus chová tak, jak je v literatuře předpokládáno. Na druhou stranu se algoritmus příliš nehodí pro řešení problému obchodního cestujícího. U instancí TSP s 26 a více městy se z hledání nejkratší cesty stává takřka náhodný proces, a to z toho důvodu, že částice mají tendenci zpomalovat a v pozdějších fázích běhu se na výpočtu podílí pouze mechanismus na zabránění zpomalení, který vytváří pozice částic čistě náhodně. Doba výpočtu pak roste nade všechny meze.

O poznání lépe si algoritmus vedl při řešení shlukování. Protože numerické výsledky nemají v tomto případě dostatečně vypovídající hodnotu, v příloze jsou k dispozici vizuální výsledky shlukování. Z těch plyne, že algoritmus nejlépe pracuje s instancemi, ve kterých jsou všechny shluky kruhovitého tvaru a přibližně stejně velké, nebo dostatečně daleko od sebe. Je-li tomu jinak, výsledek se výrazně liší od intuitivního rozdělení shluků. Tento jev není ovšem způsoben algoritmem jako takovým, nýbrž použitou objektivní funkcí, která je založena jen na vzdálenostech bodů v prostoru.

Nejpozitivněji z výstupů této práce hodnotím c/c++ knihovnu implementující klasické spojité PSO. Přínosem je hlavně její obecnost. Při realizaci bylo záměrem, aby algoritmus dokázal řešit libovolné problémy. Ačkoliv nakonec vznikly tři varianty algoritmu pro tři různé úlohy, tento cíl byl

splněn alespoň pro spojité problémy, a to v případech, kde je dodržen formát kandidátních řešení v podobě vektorů. Je nutné si uvědomit, že PSO pro shlukování se různí od klasického především použitým formátem pro kandidátní řešení. Kandidátní řešení jsou zde matice a algoritmus pro práci s nimi používá jiný datový typ[2]. Nicméně jde spíše o záležitost definice úlohy. Jiná definice shlukování může převést matice do vektoru a splnit tak požadovaný formát kandidátních řešení pro PSO klasické. U TSP je situace odlišná. Zde bylo nutno algoritmus zcela změnit a vytvořit tak jeho modifikaci pro diskrétní problémy.

Pro další vývoj knihovny by bylo vhodné knihovnu do budoucna rozšířit o několik dalších možností. Jedná se zejména o možnost měnit topologický model hejna, o možnost výběru jedné ze zdí a o možnost práce s více hejny zároveň. Další možné vylepšení by mohlo spočívat v přidání různých modifikací PSO, které sice nebyly předmětem této práce, ale při studiu problematiky se jim nelze vyhnout. Co se týče experimentů, zajímavé by bylo provést porovnání činnosti PSO s dalšími algoritmy z třídy hromadné inteligence a s algoritmy zaměřené na konkrétní problém, konkrétně u shlukování např. s k-means apod.

[2] Knihovna je implementována v jazyce C a postrádá tak výhody objektově orientovaného programovaní.

8 Literatura

[1] BLONDIN, James. *Particle Swarm Optimization*: *A Tutorial*. Armstrong Savannah, 2009. Dostupné z: http://cs.armstrong.edu/saad/csci8100/pso_tutorial.pdf

[2] WEISE, Thomas. *Global Optimization Algorithms*: *Theory and Application* [online]. 2009, 820s. [cit. 2012-01-26]. Dostupné z: http://www.it-weise.de/projects/book.pdf

[3] BLUM, Christian a Daniel MERKLE. *Swarm Intelligence*: *Introduction and Applications*. Berlin: Springer, 2008, 296 s. ISBN 978-3-540-74088-9.

[4] SIVANANDAM a DEEPA. *Introduction to Genetic Algorithms*. New York: Springer, 2007, 443 s. ISBN 978-3-540-73189-4.

[5] SCHUTTE, Jaco F. *The Particle Swarm Optimization Algorithm*. Gainesville, 2005, 33 s.

[6] LI, Xiaodong. *Particle Swarm Optimization*: *An introduction and its recent developments*. Melbourne, 2006. Dostupné z: http://www.particleswarm.info/Li_tutorial_pso_seal_06.pdf

[7] TALUKDER. *Mathematical Modelling and Applications of Particle Swarm Optimization*. Karlskrona, 2011. 65 s. Dostupné z: http://www.bth.se/fou/cuppsats.nsf/all/bfe14dc68f2ed7eec125783b004629a1/$file/MS%20Thesis%20PSO%20algorithm.pdf. Master's Thesis. Blekinge Institute of Technology.

[8] ČAPEK a HAZDRA. PSO OPTIMALIZACE V MATLABU. In: *Mezinárodní konference Technical Computing Prague 2008* [online]. 2008 [cit. 2012-01-26]. Dostupné z: http://phobos.vscht.cz/konference_matlab/MATLAB08/prispevky/018_capek.pdf

[9] DORIGO, Marco. Particle swarm optimization. *Scholarpedia* [online]. 2008(3) [cit. 2012-01-26]. Dostupné z: http://code.ulb.ac.be/dbfiles/DorMonEng2008sch-pso.pdf

[10] VAN DEN BERGH, F. *An analysis of particle swarm optimizers*. Pretoria: University of Pretoria, 2001. 283 s. disertační práce.

[11] ROHAN, Eduard. *Moderní metody v mechanice.* (přednáška) Plzeň: katedra mechaniky ZČU, 13. 12. 2006.

[12] POLI, Riccardo. An Analysis of Publications on Particle Swarm Optimisation Applications. In: *University of Essex*: *Technical Report* [online]. 2007 [cit. 2012-04-17]. ISSN 1744-8050. Dostupné z: http://cswww.essex.ac.uk/technical-reports/2007/tr-csm469.pdf

[13] TALUKDER, Satyobroto. *Mathematical Modelling and Applications of Particle Swarm Optimization*. Karlskrona, 2011. Master's Thesis. School of Engineering at Blekinge Institute of Technology. Vedoucí práce Prof. Elisabeth Rakus-Andersson.

[14] Problémy a algoritmy: Problém obchodního cestujícího. TUHÁČEK. *Jiří Tuháček* [online]. 2001 [cit. 2012-04-17]. Dostupné z: http://www.volny.cz/jtuhacek/school/paa_tsp/index.htm

[15] LHOTSKÁ, Lenka. *Úvod do biomedicínské informatiky.* (přednáška) Praha: katedra kybernetiky ČVUT, 2006.

[16] MOLGA, Marcin a Czeslaw SMUTNICKI. *Test functions for optimization needs*. Polsko, 2005.

[17] ABRAHAM, Ajith, DAS, Swagatam, ROY, Sandip. *Swarm Intelligence Algorithms for Data Clustering*.

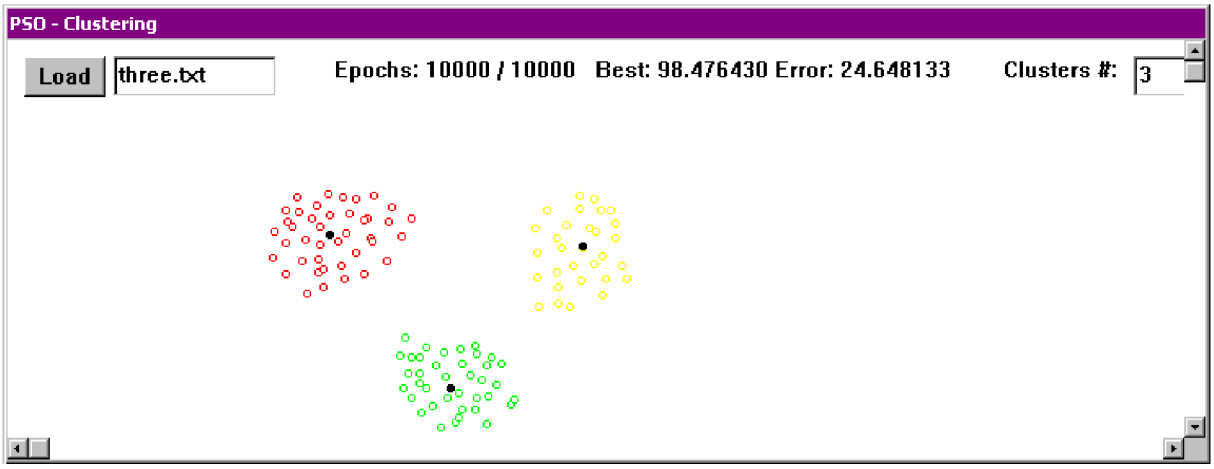
[18] FAN, Huilian. Discrete Particle Swarm Optimization for TSP based on Neighborhood. *Journal of Computational Information Systems* [online]. 2010, č. 6 [cit. 2012-04-20]. ISSN 1553-9105. Dostupné z: http://www.jofcis.com/

[19] GOLDBARG, Elizabeth F. G., GOLDBARG, Marco C., SOUZA, Givanaldo R. de. *Particle Swarm Optimization Algorithm for the Traveling Salesman Problem.*

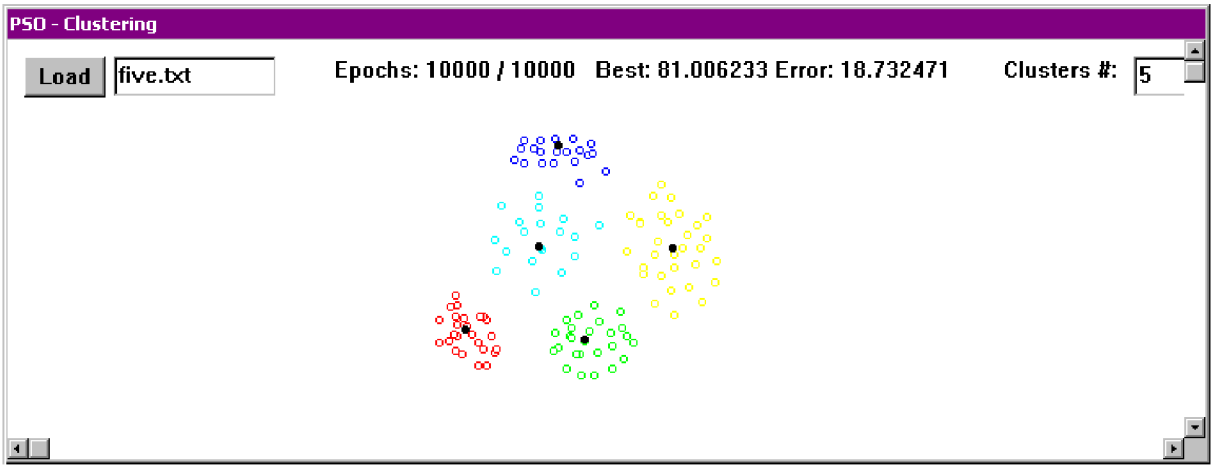
[20] *TSPLIB* [online]. 2008 [cit. 2012-05-10]. Dostupné z: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

9 Seznam příloh

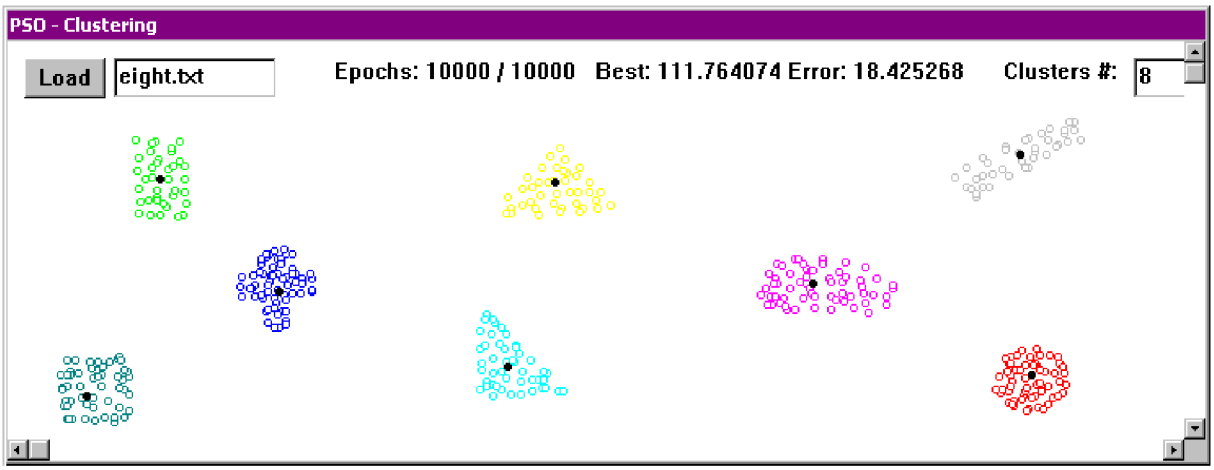
A. Grafické výsledky shlukování



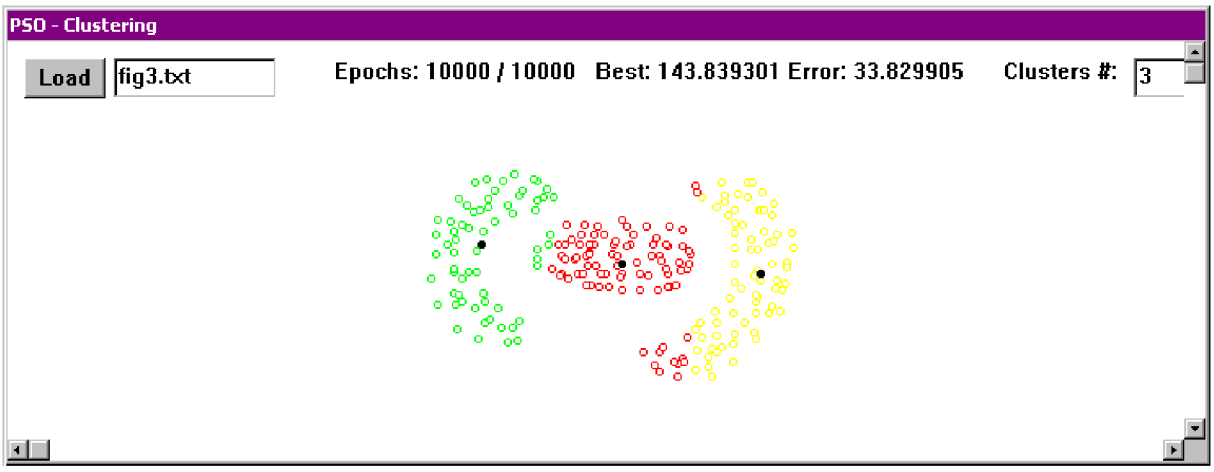
Obr. A-1: Tři shluky - rovnoměrné.



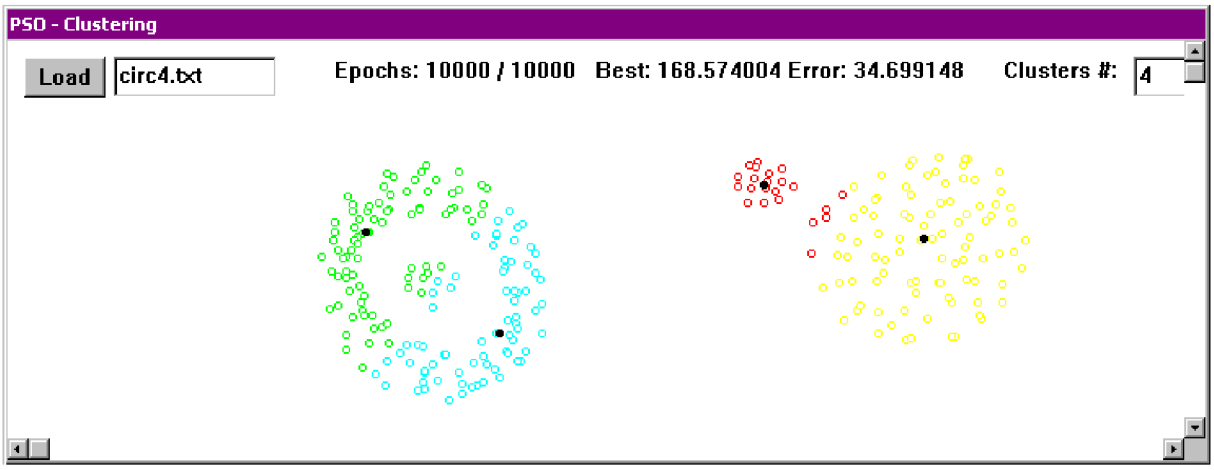
Obr. A-2: Pět shluků - rovnoměrné.



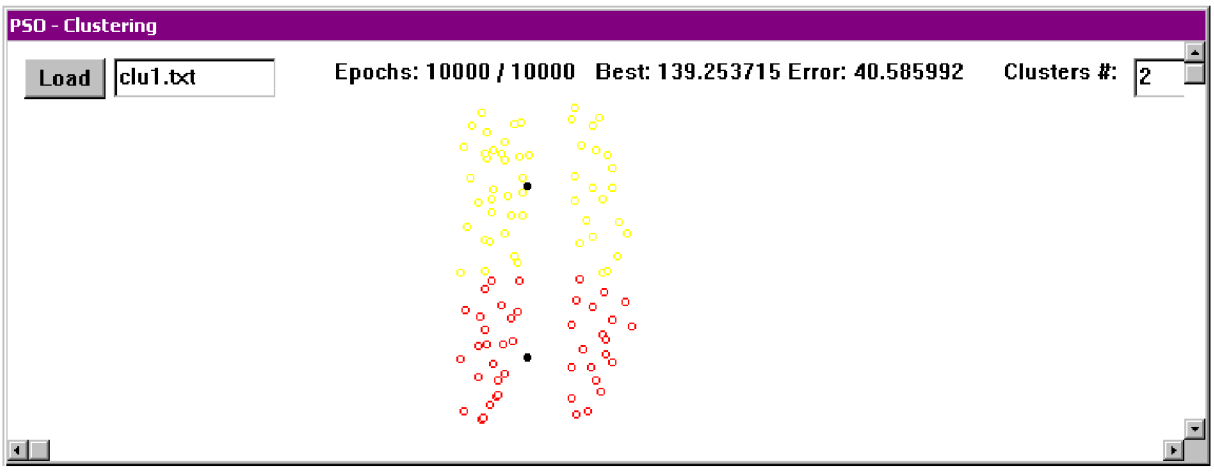
Obr. A-3: Osm shluků - daleko od sebe.



Obr. A-4: Tři shluky - nepravidelné .



Obr. A-5: Čtyři shluky - nepravidelné.



Obr. A-6: Dva shluky – nekulaté.

B. Uživatelská příručka

Návod k instalaci

Pro použití ve Windows jsou k dispozici binární tvary jak grafické tak konzolové varianty aplikace přeložené na 32 bitové architektuře pod Windows XP.

Pro použití v UNIXově orientovaných systém je přiložen příslušný Makefile, který po zadání příkazu *make* zkompiluje aplikaci. V tomto případě je k dispozici pouze varianta konzolová.

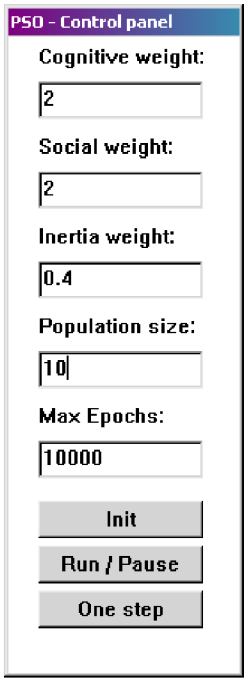
Všechny instance shlukování musí být uloženy ve složce *cluster_data* a všechny instance TSP ve složce *tsp_data*. Obě tyto složky se musí nacházet ve stejném adresáři jako spustitelný soubor aplikace. Společně s nimi zde také musí být příslušný konfigurační soubor *conf.txt*. Obě složky a konfigurační soubor jsou součásti balíku.

Parametry příkazové řádky

Mezi parametry příkazové řádky patří všechny parametry algoritmu jako takového jmenovitě: *-c1* – kognitivní váha, *-c2* – sociální váha, *-w* – váha setrvačnosti, *-n* – velikost populace, *-k* – počet cyklů. Další parametry specifikují typ úlohy a konkrétní instance těchto úloh. Jedná se o parametr *-prob*, který může nabývat hodnot *tsp*, *cluster a fun* podle typu úlohy, a o parametr *-src* specifikující jméno souboru s instancí TSP či s objekty pro shlukování. V případě minimalizace funkcí parametr *-src* obsahuje název funkce, která však musí být implementována. Známé názvy funkcí jsou: *rastrigin*, *goldstein*, *rosenbrock, branin, himmelblau, sixhump*. Pro shlukování je dále nutno zadat počet shluků parametrem *-K*. Všechny tyto uvedené parametry jsou povinné. Jediným volitelným parametrem je parametr *-steps*, který stanovuje, po kolika krocích má být vypisován průběžný výsledek optimalizace. Implicitně je jeho hodnota 1. Je-li nastaven na 0, je po provedení všech kroků vypsán až konečný výsledek.

Grafické uživatelské rozhraní

K nastavení parametrů algoritmu a k jeho ovládání slouží kontrolní panel, který je pro všechny tři typy úloh stejný. Kontrolní panel je na obrázku B-1.



Obr. B-1: Kontrolní panel.

Aby se projevila změna v nastavení parametrů, je nutné použít tlačítko *Init*. Tímto se hejno restartuje s novými parametry. Ke spuštění a zastavení algoritmu slouží tlačítko *Run/Pause*. Aplikace také dovoluje krokovat běh algoritmu pomocí tlačítka *One step*. Kontrolní panel není jediným ovládacím prvek. V oknech s problémy se nachází tlačítko *Load*, které okamžitě nahraje novou instanci problému podle zadaného jména souboru a znovu inicializuje hejno. V případě minimalizace funkcí je potřeba napsat jméno požadované funkce. Známé názvy funkcí jsou: *rastrigin*, *goldstein*, *rosenbrock, branin, himmelblau, sixhump*. Je-li zadán neznámý soubor či funkce, aplikace zachová stávající instanci problémů. V okně se shlukováním je navíc pole *Clusters #* pro zadání počtu shluků. Zde se změna projeví po nové inicializaci hejna tlačítkem *Init*.

Aplikace také disponuje editorem, který umožňuje vytvářet instance problému TSP a shlukování. Pomocí myši lze vytvářet body v prostoru. Poté je nutno zadat název cílového souboru a kliknout na tlačítko *Save matrix* pro vytvoření instance shlukování nebo použít tlačítko *Save tsp* pro vytvoření instance TSP.

V sekci *Settings* lze měnit rychlost práce algoritmu.