VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

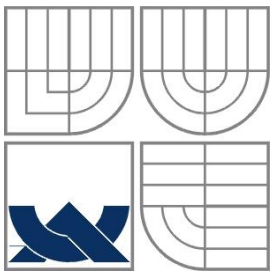
OPTIMALIZAČNÍ ÚLOHY NA BÁZI ČÁSTICOVÝCH HEJN (PSO)

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

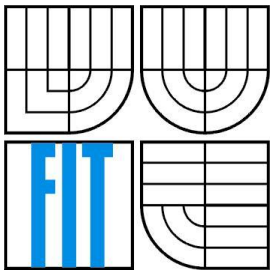
AUTOR PRÁCE
AUTHOR

BC. PATRIK NĚMEČEK

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

OPTIMALIZAČNÍ ÚLOHY NA BÁZI ČÁSTICOVÝCH HEJN (PSO)

PSO – PARTICLE SWARM OPTIMIZATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. PATRIK NĚMEČEK

VEDOUCÍ PRÁCE
SUPERVISOR

DOC. ING. JOSEF SCHWARZ, CSC.

BRNO 2014

Abstrakt

Práce se zabývá optimalizací na bázi částicových hejn. V teoretické části je nejprve stručně popsána problematika optimalizace. Poté se značná část věnuje celkovému popisu optimalizačního algoritmu na bázi částicových hejn (PSO). Jsou popsány jeho princip, chování, parametry, struktura a modifikace. Následuje rešerše variant PSO, včetně hybridizací PSO. V praktické části práce jsou nejprve blíže rozebrány dynamické problémy. Poté je popsán nově navržený algoritmus pro dynamické problémy AHPSO (z čeho vychází, čím byl inspirován a jaké prvky používá a proč). Algoritmus je spuštěn na sadě úloh (Moving peaks benchmark) a porovnán s dosud nejlepšími veřejně dostupnými algoritmy variant PSO na dynamické problémy.

Abstract

This work deals with particle swarm optimization. The theoretic part briefly describes the problem of optimization. The considerable part focuses on the overall description of particle swarm optimization (PSO). The principle, behavior, parameters, structure and modifications of PSO are described. The next part of the work is a recherché of variants of PSO, including hybridizations of PSO. In practical part the dynamic problems are analyzed and new designed algorithm for dynamic problems AHPSO is described (what it is based on, what was inspired, what elements are used and why). Algorithm is executed on the set of tasks (Moving peaks benchmark) and compared with the best publicly available variants of algorithm PSO on dynamic problems so far.

Klíčová slova

Optimalizace na bázi částicových hejn, varianty a hybridizace PSO, dynamické problémy, Moving peaks benchmark, AHPSO.

Keywords

Particle Swarm Optimization, PSO variants and hybridizations, dynamic problems, Moving peaks benchmark, AHPSO.

Citace

Němeček Patrik: Optimalizační úlohy na bázi částicových hejn (PSO), diplomová práce, Brno, FIT VUT v Brně, 2014

Optimalizační úlohy na bázi částicových hejn (PSO)

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Ing. Josefa Schwarze, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Patrik Němeček
28. 5. 2014

Poděkování

Děkuji doc. Ing. Josefu Schwarzovi, CSc. za poskytnuté studijní materiály a přínosné rady při zpracovávání této problematiky.

Obsah

1 Úvod

Cílem této práce bylo popsat optimalizaci založenou na bázi částicových hejn – Particle Swarm Optimization (PSO) – a její varianty, poté navrhnout a implementovat vlastní variantu algoritmu PSO a nakonec jeho výkon porovnat s dosud nejlepší veřejně dostupnou variantou PSO. Navržená varianta je vhodná na řešení dynamických problémů, proto je praktická část práce zaměřena právě na tyto problémy. K porovnání kvality algoritmu byly vybrány ty varianty, které dosahovaly skvělých výsledků v dynamickém prostředí. Nový algoritmus byl pojmenován AHPSO (adaptivní hybridní PSO).

Optimalizace je termín pro proces nalezení optimálního řešení daného problému. Pokud je problém reprezentován funkcí, jejíž parametry odpovídají hledanému řešení tohoto problému, pak optimalizace má za cíl nalézt extrémy této funkce (minimum nebo maximum).

Existuje velké množství různých algoritmů pro optimalizaci. Jednou velkou skupinou jsou algoritmy nacházející inspiraci v přírodě. Do této skupiny patří právě PSO, který je analogií chování zvířecích druhů ve skupině v přírodě (hejna ptáků, ryb, …). Ne moc inteligentní jedinci vykazují ve skupině inteligentní chování, které je způsobeno lokálními interakcemi mezi těmito jedinci ve skupině. Jedinci se tedy více či méně ovlivňují. Díky získávání informací ze společenství a schopnosti učení dokážou jedinci spolu řešit poměrně složité problémy.

Mezi hlavní výhody algoritmů založených na inteligenci hejna patří jejich robustnost, flexibilita a jednoduchá implementovatelnost. Za nevýhodu lze považovat fakt, že nemusí být vždy nalezen optimální výsledek, protože algoritmus pracuje s prvkem náhody. To lze ovšem jednoduše eliminovat vícenásobným spuštěním algoritmu.

V kapitole 2 je podrobněji popsána podstata optimalizace a ukázáno možné rozdělení optimalizačních algoritmů. Dále je zmíněn No Free Lunch teorém, který souvisí s výběrem nejlepšího optimalizačního algoritmu. Nakonec je v kapitole ukázáno, podle čelo lze klasifikovat funkce a do jakých kategorií.

V kapitole 3 je popsán samotný PSO. Je uvedeno, na čem je PSO založeno, jeho principy a samotná matematická definice. Je přiblížena základní verze PSO i s jeho drobnými modifikacemi, které se staly později standardem. Rovněž jsou popsány parametry a typy sousedství a topologií a jejich souvislosti na chování celého algoritmu. Jsou vybrány úspěšné varianty PSO, které mnohdy základní verzi PSO překonávají ve výkonu a rychlosti, a více či méně rozebrány jejich modifikace. Některé varianty se liší svojí pouze drobnou úpravou stěžejní rovnice, jiné zavádějí poměrně drastické změny do celého algoritmu, nicméně stále vycházejí ze základního principu. Výsledné varianty PSO se liší i co se týče vhodnosti na konkrétní typy funkcí. Určité varianty jsou účinné např. na funkce s více optimy, jiné varianty dosahují dobrých výsledků u dynamických problémů a ještě

jiné u více-kriteriálních problémů. Nakonec jsou představeny některé hybridizace PSO, tedy spojení PSO s jiným algoritmem za účelem kombinace výhod obou přístupů s cílem zlepšit výkon.

Čtvrtá kapitola je zaměřena na popis navrženého algoritmu AHPSO na dynamické problémy. Podrobněji jsou popsány dynamické problémy i přístupy, které se k jejich řešení v algoritmech PSO používají. V prostřední části je navržen nový algoritmus. Nejprve je naznačeno, z čeho navržený algoritmus vychází a proč, a poté jsou rozebrány jednotlivé prvky samotného algoritmu i motivace, proč byly tyto prvky aplikovány. Jsou zmíněny i přístupy a algoritmy, kterými byly části navrženého algoritmu inspirovány. V poslední části je zobrazen vývojový diagram nově vytvořeného algoritmu a zapsán jeho pseudokód.

Kapitola 5 obsahuje informace týkající se implementace algoritmu AHPSO.

Kapitola 6 se věnuje experimentům a jejich výsledkům. Nejprve je vysvětleno, jaká sada úloh byla použita a jaké kritérium bylo zkoumáno. Poté byl zkoumán a popsán vliv jednotlivých nových parametrů na chování algoritmu. Na základě těchto experimentů byly parametry nastaveny tak, aby algoritmus AHPSO dosahoval nejlepších výsledků.

Předposlední sedmá kapitola porovnává výsledky dosažené algoritmem AHPSO s výsledky nejlepších veřejně dostupných algoritmů na dynamické problémy na několika testovacích úlohách. Nakonec jsou dosažené výsledky zhodnoceny.

Ze semestrálního projektu byly převzaty první tři teoretické kapitoly.

2 Optimalizace

Podle [1] je optimalizace termín, který je používán v odvětví počítačových věd zabývající se nalezení „nejlepšího“ řešení. „Nejlepší“ znamená dostatečně dobré řešení, které může být absolutně nejlepší řešení nebo jedno z několika kandidátních řešení. Charakteristiky a požadavky problému udávají, zda úplně nejlepší řešení může být nalezeno.

Každý optimalizační problém se skládá z několika složek. Každá optimalizace má účelovou (fitness) funkci, která musí být optimalizována tak, že její výsledek je buď maximální, nebo minimální. Podle toho, jestli se hledá maximum, nebo minimum. Jakoukoli funkci, která hledá maximum (minimum), lze změnit na hledání minima (maxima) pouhým vynásobením funkce hodnotou -1.

Dále optimalizace obsahuje množinu neznámých proměnných, které ovlivňují hodnotu účelové funkce. Pokud *x* reprezentuje vektor proměnných, pak *f(x)* udává hodnotu účelové funkce *f* kandidátního řešení *x*.

Poslední složkou optimalizace je množina omezení, která ohraničuje prohledávací prostor problému. Jedná se o omezení hodnoty, kterou může nabýt neznámá proměnná. Tato omezení definují množinu přípustných hodnot proměnné. Omezení mohou být i více komplexní, kdy například celé kandidátní řešení může být na základě těchto omezení penalizováno.

Účelovou funkci je nutné nejprve vytvořit. Je snaha, aby co nejvíce odpovídala řešenému problému. Hledají se takové hodnoty parametrů, aby výstupy optimalizačního modelu odpovídaly výstupům reálného problému při stejných vstupech.

Optimalizace je stále aktivní oblast výzkumu. Mnoho reálných optimalizačních problémů se stává více a více složitými, proto je vždy potřeba získat lepší optimalizační algoritmus. Optimalizační problém bez omezení může být formulován jako *D*-dimenzionální minimalizační problém *f* následovně:

$$\min f(x),\ x = [x_1, x_2, \dots, x_D]\,, \tag{2.1}$$

kde *D* je počet parametrů k optimalizaci [2]. Tedy pokud je *X* množina všech kandidátních řešení funkce *f*, pak $y \in X$ je optimální řešení, pokud $\forall x \in X$: $f(y) \leq f(x)$.

2.1 Optimalizační (a heuristické) algoritmy

Většina problémů inženýrské praxe může být definována jako optimalizační úlohy, kde řešený problém lze převést na matematickou úlohu danou vhodným funkčním předpisem. Obvykle se tyto algoritmy používají tam, kde je řešení daného problému analytickou cestou nereálné nebo nevhodné, tzn. řešení je možné, ale značně komplikované a zdlouhavé [3].

Optimalizační algoritmy jsou prohledávací metody, které mají za cíl najít řešení optimalizovaného problému. Toto řešení musí být dostatečně dobré a musí se nacházet v dané množině omezení. Optimalizační algoritmy musí čelit mnoha problémům. Řešení může obsahovat kombinaci různých datových typů (celočíselné, reálné, komplexní,…), nelineární omezení prohledávajícího prostoru, může existovat mnoho kandidátních řešení, charakteristiky problému se můžou měnit s postupem času nebo může existovat několik protichůdných požadavků na řešení [1].

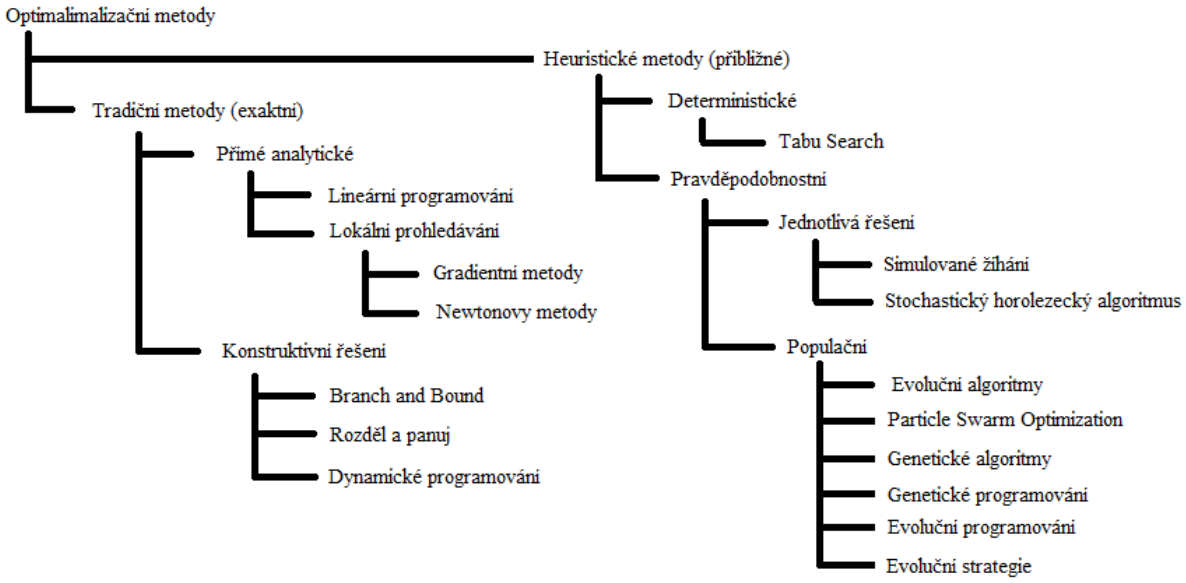
Tedy jednoduše řečeno, optimalizační algoritmy slouží k nalezení minima dané účelové funkce tak, že hledají optimální numerickou kombinaci jejich argumentů.

2.1.1 Rozdělení optimalizačních algoritmů

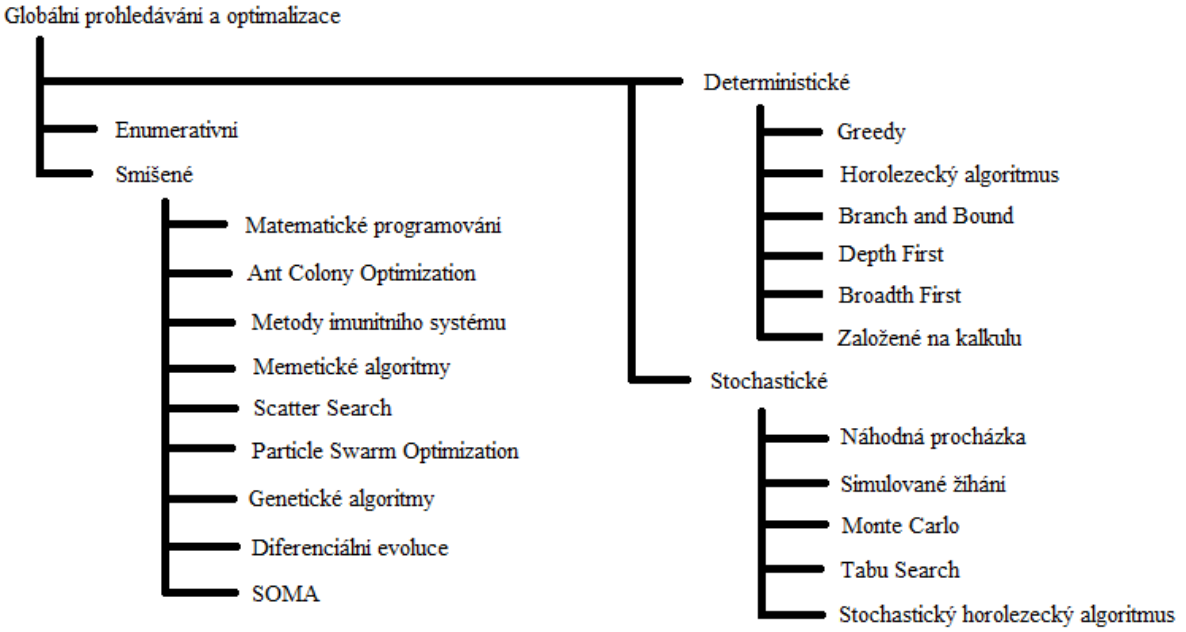
Optimalizační algoritmy lze rozdělit podle principů jejich činnosti, podle složitosti algoritmu, podle tříd problému, pro které jsou „předurčeny“, atp. Názory na jejich klasifikaci se mírně liší. Různé pohledy na klasifikaci optimalizačních algoritmů demonstrují obrázky 2.1 a 2.2, které i přes společnou linii mají některé odlišnosti. Jednotlivé třídy algoritmů představují obecně způsoby řešení daného problému metodami s různým stupněm efektivity a složitosti. Podle jejich vlastností se dělí algoritmy do těchto kategorií [3]:

- **Enumerativní.** Algoritmy provádí výpočet všech možných kombinací daného problému (vhodné pouze pro problémy s diskrétními parametry malého rozsahu).
- **Deterministické.** Algoritmy jsou postaveny pouze na rigorózních metodách klasické matematiky (vhodné pro lineární, konvexní unimodální funkce s malým a souvislým počtem možných řešení).
- **Stochastické.** Algoritmy jsou založeny na využití náhody, kde se náhodně hledají hodnoty argumentů účelové funkce a výsledkem je celkově nejlepší dosud nalezené řešení (vhodné pro hrubý odhad).
- **Smíšené.** Algoritmy spojují metody deterministické a stochastické, jejichž spoluprácí je dosaženo dobrých výsledků, jsou robustní, efektivní a výkonné. Hlavní přednosti jsou: nezávislost na počátečních podmínkách, nalezení kvalitního řešení během malého počtu ohodnocení účelové funkce, téměř žádná potřeba informací o řešeném problému, není potřeba analytický popis problému.

Příklady algoritmů patřících do jednotlivých kategorií jsou uvedeny na obrázku 2.2.



Obrázek 2.1 Dělení optimalizačních metod [4]



Obrázek 2.2 Dělení optimalizačních metod [3]

2.2 No Free Lunch Teorém

Byly snahy o vytvoření univerzálního řešitele (algoritmu) různé škály problémů. Existuje ale jistá statisticky významná skutečnost, která se nazývá No Free Lunch Teorém, který tvrdí, že žádný algoritmus nemůže být lepší než jiný v řešení úplně všech možných funkcí [5], neboli existuje podmnožina problémů, pro které je algoritmus A lepší než algoritmus B a naopak.

Na základě tohoto teorému nelze tedy očekávat, že lze použít jakýkoliv algoritmus na řešení libovolného problému; spíše různé algoritmy se hodí k řešení různých problémů. Nelze na základě výsledků experimentů nad velkým množstvím testovacích funkcí označit za „matematickou pravdu“ např. to, že jeden algoritmus je obecně zcela nepoužitelný, jelikož zákonitě musí existovat naopak třída takových problémů, které bude tento algoritmus řešit velice úspěšně [3].

Různé optimalizační techniky a algoritmy jsou vhodnější na různé typy problémů (unimodální/multimodální, lineární/nelineární,…), proto je při optimalizaci účelové funkce velmi důležité vybrat správnou optimalizační metodu. Tento výběr často závisí na zkušenostech řešitele a mnohdy ani nelze předem odhadnout vhodný algoritmus. Nicméně mnohé smíšené algoritmy jsou natolik robustní, že dokážou nalézt dostatečně dobré řešení za relativně krátkou dobu.

2.3 Klasifikace problémů (funkcí)

Každý problém lze definovat matematickou funkcí. Jsou snahy, aby vytvořená účelová funkce co nejvíce odpovídala řešenému problému. Dosáhnout toho ovšem není vždy jednoduchý proces. Čím „nekvalitnější“ účelová funkce je, tím jsou výsledky optimalizace méně relevantní, protože se řeší (optimalizuje) trochu jiný problém. Účelovou funkci vytváří často odborník z praxe, který řešenému problému dobře rozumí.

Problémy a jejich odpovídající účelové funkce lze rozdělit podle úhlu pohledu do několika tříd:

- Podle počtu dimenzí.
  - **Jednodimenzionální.** V účelové funkci je jen jedna proměnná, jejichž hodnotu optimalizace hledá.
  - **Vícedimenzinální.** V účelové funkci je více proměnných. Optimalizační algoritmy pracují právě s obecně *n* proměnnými.
- Podle počtu extrémů.
  - **Unimodální.** Funkce má na daném intervalu jen jeden extrém.
  - **Multimodální.** Funkce má více extrémů. Hodnoty těchto extrémů se buď rovnají – funkce má několik globálních extrémů, nebo jsou různé – funkce má jeden (případně více) globální extrém a několik lokálních extrémů.
- Podle stability účelové funkce.
  - **Statické.** Hodnoty účelové funkce se v čase nemění.
  - **Dynamické**. Účelová funkce se mění v čase, extrém se tedy může posunovat.
- Podle počtu kritérií (účelových funkcí).
  - **Jedno-kriteriální.** Problém je charakterizován jednou účelovou funkcí.
  - **Více-kriteriální.** Problém je charakterizován více účelovými funkcemi, které jdou často proti sobě, tzn., že hodnoty těchto funkcí pro stejné kandidátní

řešení jsou hodně odlišné (pro jednu funkci je řešení dost dobré, pro druhou dost špatné).

Řeší se to např. skalarizací, kdy se všechny kritéria kombinují do jedné skalární hodnoty. Používá se především agregační funkce, kde se pomocí nastavení vah určují preference jednotlivých účelových funkcí. Jelikož mohou být kritéria sémanticky odlišná, je potřeba provést normalizaci.

Lepší (ale složitější) způsob řešení více-kriteriální optimalizace je nalezení kompromisu, který hledá taková řešení, která jsou Pareto optimální. Pareto optimalita je založena na Pareto dominanci. Pareto dominantních řešení může být více, tato řešení se nachází na tzv. Pareto frontě a jsou všechna stejně dobrá (jsou vůči sobě indiferentní). Řešení je Pareto optimální, pokud neexistuje žádné řešení, které by ho Pareto dominovalo. Řešení a Pareto dominuje řešení b, pokud pro $\forall i \in 1,\ldots, n_f$: $f_i(a) \geq f_i(b)$ a zároveň $\exists i \in 1,\ldots, n_f$: $f_i(a) > f_i(b)$, kde n_f je počet účelových funkcí k optimalizaci [6].

3 Particle Swarm Optimization

Particle Swarm Optimization neboli Optimalizace na bázi částicových hejn (zkráceně PSO) je smíšený optimalizační algoritmus s prvky determinismu a náhody. Původní PSO vyvinuli již v roce 1995 sociální psycholog Dr. James Kennedy a elektrický inženýr Dr. Russel C. Eberhart a je zaměřen na výpočetní inteligenci využívající analogii sociální interakce, zejména analogii ptačích hejn hledajících kukuřici. Díky své robustnosti a flexibilitě je dodnes PSO považován za silnou optimalizační metodu [7].

3.1 Inspirace přírodou

Skupinové formace byly vypozorovány u mnoha druhů zvířat. Některé zvířecí druhy nebo skupiny jsou ovládány vůdcem, například u lvů, paviánů nebo jelenů. V některých společenstvích je chování jedinců silně ovlivněno sociální hierarchií.

Zajímavá jsou i samo-organizační chování druhů žijících ve skupinách, kde není žádný vůdce, např. ptáci, ryby nebo ovce. Uvnitř těchto skupin jedinci nemají žádnou znalost globálního chování celé skupiny ani globální informaci o prostředí. Naproti tomu mají schopnost se shromažďovat a přemísťovat společně na základě lokální interakce mezi jedinci. Pomocí těchto lokálních interakcí mezi jedinci skupina vykazuje komplexní kolektivní chování. Kolektivní chování bylo vypozorováno např. u ptáků, ryb, velryb, opic nebo žraloků [8].

3.2 Obecný princip

PSO je založeno na sociologicko-psychologickém modelu společenského vlivu a učení. Jedinci v hejnu částic následují velmi jednoduché chování: berou do úvahy úspěch sousedních jedinců. Kolektivní chování způsobí prohledání optimální oblasti ve vysoce dimenzionálním prohledávacím prostoru.

PSO má některé společné vlastnosti s jinými evolučními výpočetními technikami, jako jsou např. genetickými algoritmy, které jsou hodně rozšířené. Systém je inicializován skupinou náhodně vygenerovaných řešení (jedinců). S přibývajícím počtem generací každý jedinec hledá lepší řešení, než je jeho současné. Kvalita řešení se zjišťuje dosazením parametrů, které jedinec reprezentuje, do účelové funkce. Čím vyšší/nižší hodnota účelové funkce, tím lépe. Na rozdíl od genetických algoritmů PSO neobsahuje operátory křížení ani mutace. Jedinci následují jedince s dosud optimálním řešením [9].

PSO algoritmus simuluje chování ptačího hejna. Skupina ptáků prohledává oblast a hledá nejvyšší vrchol. Hejno neví, kde se vrchol nachází, ale po každé iteraci ví, kdo našel zatím nejvyšší

místo. Tedy každý jedinec reprezentující řešení „prolétává“ skrz mnoho-dimenzionální prohledávací prostor, kde každá jeho další pozice je upravena podle jeho vlastní zkušenosti o jeho sousedech. Učí se tedy ze svých předchozích pozic a od svých lépe postavených sousedů.

Každý jedinec je charakterizován svou pozicí a rychlostí. Nová pozice je dána novou rychlostí. Právě rychlost se v čase mění na základě předchozích pozic jedince a výměny informace ze sousedství. Rychlost udává i směr následujícího pohybu jedince.

Pozice je dána vektorem, jehož velikost odpovídá počtu dimenzí řešeného problému. PSO tedy pracuje s vektorem reálných čísel, a tedy odpadá problém zakódování jedinců. PSO je tedy vhodné pro řešení problémů, které pracují s reálnou reprezentací proměnných.

3.2.1 Explorace a exploitace

Pro celý PSO je esenciální udržet rovnováhu mezi explorací a exploitací. Existují různé problémy, kde u některých je výhodnější se zaměřit na exploraci (multimodální funkce) a u jiných na exploitaci (unimodální funkce).

Explorace (exploration) je v kontextu PSO výraz pro prohledání či prozkoumání co nejširší oblasti prohledávacího prostoru. Tento prostor sice není prohledán nijak do hloubky, ale hejno má zato povědomí o velkém prostoru. Částice se musí vyhnout uváznutí v lokálních extrémech, aby byla udržena diverzita (rozmanitost), a proto je snaha o exploraci, tedy o překonání lokálních oblastí a prohledání jiných, mnohdy hodně vzdálených.

Exploitace (exploitation) je naopak výraz pro podrobné prozkoumání určité konkrétní oblasti. Jde sice o malý prostor, ale zato pořádně prozkoumaný, a tedy sledovaná „stopa“ je maximálně využita a potenciální optimální řešení nalezeno. Částice se nemůže chaoticky či náhodně pohybovat po prohledávacím prostoru, musí postupně konvergovat k jednomu řešení, a proto je snaha o exploitaci.

Udržení diverzity snižuje míru konvergence, která snižuje pravděpodobnost stagnace ve špatném lokálním optimu, a to může vést k nalezení lepšího konečného řešení. Na druhou stranu nižší konvergence vede ke zvýšení času potřebného prohledávacím procesem k nalezení konečného výsledku.

3.3 Princip původního PSO

V základním PSO jsou jedinci (částice) umístěny do prohledávajícího prostoru nějakého problému nebo funkce. Každá částice vypočítá účelovou (fitness) funkci v tomto místě. Poté určí svůj pohyb na základě historie své vlastní pozice a dosud nejlepší pozice jednoho nebo více členů hejna. Další iterace se provede, až je přesun všech částic dokončen. Nakonec se hejno jako celek pravděpodobně přesune blízko k optimu fitness funkce.

Každá částice uchovává v paměti tři *D*-dimenzionální vektory: současná pozice *x* (souřadnice v prostoru), dosud nejlepší částicí nalezená pozice *pBest* a rychlost *v*. V každé iteraci je vypočítáno nové *x* (řešení problému) a nové *v* (lze považovat za velikost kroku). Pokud je toto nové *x* dosud pro částici nejlepší, souřadnice se uloží do *pBest*. Cílem je neustále zlepšovat *pBest*.

Komunikací částic se zvyšuje úspěšnost PSO – od interakce se odvíjí individuální chování částic. Populace je organizována podle nějakého druhu komunikační struktury (topologie, sociální síť), tvoří se sousedství (obousměrné). Každá částice komunikuje s jinými částicemi a je ovlivněna dosud nejlepším nalezeným řešením *gBest* (jakýmkoli členem svého sousedství). Druhy sociálních sítí mohou být různé, v praxi se ale objevují jen určité typy. Protože je v každém cyklu měněna rychlost každé částice, částice osciluje kolem bodů *pBest* a *gBest* [7].

Celý princip je snadno implementovatelný, jak ukazuje následující kapitola.

3.3.1 Algoritmus původního PSO

Jak už bylo popsáno výše, každý jedinec je charakterizován svou pozicí a svou rychlostí, která udává směr, kterým se bude jedinec v další iteraci ubírat. Nová pozice je dána rovnicí

$$x_i(t+1) = x_i(t) + v_i(t+1)\,, \tag{3.1}$$

kde x_i je vektor udávající pozici jedince *i*, v_i je vektor rychlosti jedince *i* a *t* je aktuální generace; $x_i(0)$ je náhodné číslo z intervalu x_{min} a x_{max}, kde x_{min} a x_{max} jsou hranice prohledávajícího prostoru.

Nová rychlost jedince je dána vztahem

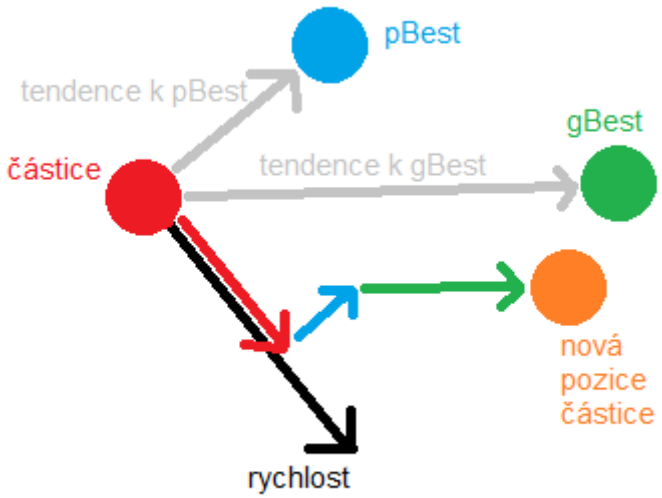
$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)\left[pBest_{ij}(t) - x_{ij}(t)\right] + c_2 r_{2j}(t)\left[gBest_j(t) - x_{ij}(t)\right], \tag{3.2}$$

kde proměnné mají následující význam:

- $v_{ij}(t+1)$ - rychlost jedince *i* v dimenzi *j* v následující iteraci *t+1*
- $v_{ij}(t)$ - rychlost jedince *i* v dimenzi *j* v aktuální iteraci *t*
- $x_{ij}(t)$ - pozice jedince *i* v dimenzi *j* v aktuální iteraci *t*
- $pBest_{ij}(t)$ - dosud nejlepší nalezená pozice jedince *i* v dimenzi *j* v aktuální iteraci *t*
- $gBest_j(t)$ - dosud nejlepší nalezená pozice v celé populaci (nebo $lBest_{ij}(t)$ v sousedství jedince *i*, viz kapitola 3.4) v dimenzi *j* v aktuální iteraci *t*
- c_1, c_2 - učící faktory (akcelerační koeficienty)
- r_{1j}, r_{2j} - náhodná čísla z intervalu 0 až 1 v dimenzi *j*

Rovnice 3.2 naznačuje, že každý jedinec má tendenci pokračovat v prozkoumávání prohledávajícího prostoru třemi směry. Jedinec jde svým vlastním individuálním směrem, zároveň bere do úvahy návrat ke své dosud nejlepší pozici a taktéž následuje jedince s dosud nejlepší pozicí. Tento princip je znázorněn na obrázku 3.1. Poté, co jedinec nalezne novou pozici na základě rovnic 3.2 a 3.1, se na ni přesune. Spočte se hodnota účelové funkce a celý cyklus se opět opakuje [8].

Protože rovnice (3.2) obsahuje prvek náhody, není zaručeno, že algoritmus pokaždé nalezne správný výsledek. Tento problém lze eliminovat vícenásobným spuštěním algoritmu.



Obrázek 3.1 Znázornění skutečného pohybu částice

3.3.2 Parametry PSO

Pro správné fungování algoritmu PSO je potřeba správně nastavit některé parametry. Žádný zaručený postup, na jakou hodnotu tyto parametry nastavit, neexistuje. Vše je závislé na řešeném problému. Nastavení parametrů může ovlivnit průběh a výsledek řešení. Některé parametry jsou dány přímo podstatou problému, některé musí uživatel nastavit na základě testování [8].

3.3.2.1 Akcelerační koeficienty

Akcelerační koeficienty c_1 a c_2 určují, jak velký vliv budou mít pozice $pBest_i$ a $gBest$ na změnu rychlosti (kterým směrem se jedinec bude vydávat). Učící faktor c_1 představuje paměť částice a dává přednost k návratu na dosud svou nejlepší pozici. Učící faktor c_2 představuje kooperaci mezi částicemi a preferuje posun k dosud nejlepšímu řešení. Vliv těchto parametrů není ale úplný, protože vše závisí i na náhodně vygenerované hodnotě, která je násobena s učícím faktorem. Priority jedince se tak mohou v jakékoli iteraci i obrátit (dochází k vytvoření nerovné cyklické trajektorie). Tím je zajištěna dostatečná náhodnost [7].

Správně definované akcelerační koeficienty c_1, c_2 mohou zabránit explozi nebo mohou přimět částice konvergovat do lokálního optima. Aplikování omezujících koeficientů dovoluje řízení nad dynamickými charakteristikami hejna částic, včetně explorace vs. exploitace [10].

Tedy v globálu se spolu se změnami hodnot těchto učících faktorů radikálně mění chování PSO. Změna těchto parametrů může učinit PSO nestabilní (nekontrolovatelné zvyšování rychlosti): malá hodnota omezuje částice, velká hodnota může způsobit divergenci. Obecně se nastavuje $c_1 = c_2 = 2.0$ (průměrná hodnota obou stochastických faktorů je tedy 1.0, tzn., že částice přeletí bod, ke

kterému směřuje, v polovině případů prohledávání). Se zvyšující se $c = c_1 + c_2$ stoupá četnost oscilací kolem optimálního bodu. Pro menší c se vzor trajektorie podobá sinusoidálnímu průběhu. Pro hodnotu $c > 4.0$ jde trajektorie do nekonečna [8].

3.3.2.2 Stanovení maximální rychlosti

Všudypřítomně přijaté nastavení akceleračních koeficientů na $c_1 = c_2 = 2.0$ vnáší zmíněnou nestabilitu, proto bylo zavedeno omezení rychlosti do rozsahu $(-V_{max}, V_{max})$. Pokud částice v nějakém směru překročí rychlost V_{max}, je jí vygenerována rychlost nová, nebo je rychlost v daném směru nastavena na hodnotu $\pm V_{max}$.

Příliš nízká hodnota V_{max} vzhledem k rozsahu prohledávané oblasti vede k tomu, že jedinec se moc nepohne od své počáteční pozice. Prohledaná oblast tímto jedincem je příliš malá, ale zato důkladně prozkoumaná, nicméně globální extrém nemusí být nalezen. Příliš vysoká hodnota V_{max} způsobí nekontrolovatelnou (nevyzpytatelnou) trajektorii a častý výskyt jedince mimo prohledávací oblast, protože částice mají tendenci svou rychlost neustále zvyšovat a brzy se mohou dostat na pozici mimo prohledávací prostor. Pokud se tak stane, je částici vygenerována nová pozice. Pokud se nová pozice generuje příliš často, stává se z algoritmu náhodné prohledávání prostoru řešení [8].

Velikost V_{max} má vliv na rovnováhu mezi explorací a exploitací. Pokud se V_{max} v průběhu provádění algoritmu postupně zmenšuje, pak dochází k postupnému přechodu od explorativního k exploitativnímu chování hejna; nicméně pořád je kvůli V_{max} tlumena dynamika [7]. Takové dynamické změny mohou být prováděny podle rovnice $V_{max} = (X_{max} - X_{min})/N_i$, kde N_i je počet intervalů v k-té dimenzi vybraný uživatelem a X_{min}, X_{max} jsou krajní hodnoty dosud nalezené částicemi [11].

3.3.2.3 Ukončující podmínka

Ukončující podmínkou může být například minimální rozdíl fitness hodnot nejlepší a nejhorší částice, nebo neměnící se rozdíl nejlepších částic v po sobě jdoucích iteracích. Někdy to může trvat hodně dlouhou dobu, tak se také udává maximální počet iterací nebo maximální počet funkčních ohodnocení, po nichž se algoritmus zastaví [8].

3.3.2.4 Dimenze

Je dána problémem, který je optimalizován. Většinou jde o počet parametrů účelové funkce. Čím je dimenze vyšší, tím se prohledávací prostor exponenciálně zvětšuje. Vektor rychlosti a vektor pozice má tolik argumentů, kolik je číslo dimenze prohledávaného problému [8].

3.3.2.5 Rozsah

Rozsah udává hranice prohledávacího prostoru v jednotlivých dimenzích. Každá dimenze má obecně jiný rozsah. Rozsah je dán samotným problémem [8].

3.3.2.6 Počet částic

Tento parametr udává počet jedinců, kteří budou řešení hledat. Počet je v průběhu celé optimalizace konstantní, nijak se nemění. Čím více jedinců, tím hustěji bude prohledávací prostor prohledán, ale časová náročnost bude vyšší. Typicky stačí kolem 20 – 50 jedinců. Některé obtížné problémy ale vyžadují i mnohem více (200 a více), tedy obecně pro více dimenzionální problémy je lepší větší populace [8].

3.3.2.7 Pseudokód algoritmu

Algoritmus PSO je následující [8]:

```
for každá částice do
     inicializace částice – vygenerování náhodné pozice a náhodné
     rychlosti;
end
repeat
     for každá částice do
          vypočítej fitness současné pozice;
          if fitness současné pozice je lepší než fitness dosud
             nejlepší pozice pBest then
               současnou pozici nastav jako pBest;
     end
     vyber částice s nejlepší fitness a tu nastav jako gBest;
     for každá částice do
          vypočítej rychlost částice pomocí rovnice 3.2;
          vypočítej pozici částice pomocí rovnice 3.1;
     end
until ukončovací podmínka je splněna;
```

3.3.3 PSO s vahou setrvačnosti

Kvůli neustále zvyšující se rychlosti jedince, díky čemuž může částice divergovat (blížit se nekonečnu), byla zavedena setrvačnost, která lépe kontroluje rozsah hledání a snižuje důležitost V_{max}. Jde o mírnou úpravu rovnice rychlosti. Tento parametr se značí *w*, jeho hodnota je z intervalu 0 až 1 a je jím vážena předcházející rychlost jedince. Tedy rychlost jedince se mění pomaleji, je řízena tendence částice pokračovat ve stejném směru. Malá hodnota *w* podporuje hledání v lokálních oblastech, velká hodnota *w* podporuje prohledávání na větším prostoru. Rovnice 3.2 je pak upravena na

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_{1j}(t)\left[pBest_{ij}(t) - x_{ij}(t)\right] + c_2 r_{2j}(t)\left[gBest_j(t) - x_{ij}(t)\right]. \quad (3.3)$$

Bylo objeveno, že nejlepší výkony PSO vykazuje, když je hodnota *w* lineárně snižována (TVIW). Nejprve je prostor prohledáván po velkých skocích a postupně se kroky zmenšují. Váha *w* je nastavena na hodnotu w_{max} a snižuje se k hodnotě w_{min}, většinou $w_{max} = 0.9$ a $w_{min} = 0.4$. Nevýhodou ovšem je, že pokud jednou *w* klesá, hejno ztrácí schopnost hledat nové oblasti. Pro $w_{max} > 1$ je PSO nestabilní, postupně se dostává do stabilní oblasti (závislost i na c_1 a c_2) [12]. V každé iteraci má *w* jinou hodnotu, ta je dána vztahem

$$w = w_{max} - \frac{(w_{max} - w_{min}) act\_iter}{num\_iter}, \quad (3.4)$$

kde *act_iter* je aktuální iterace a *num_iter* je celkový počet iterací [8].

Velmi dobré výsledky vykazuje i PSO s náhodným *w* (RANDIW), které je dáno

$$w = 0.5 + \frac{r}{2}, \quad (3.5)$$

kde $r \in \langle 0, 1 \rangle$. Průměrná hodnota *w* je tedy 0.75. Tato modifikace byla inspirována omezujícím faktorem χ, jehož optimální hodnota je 0.7289 (viz následující kapitola), což zhruba odpovídá průměrné hodnotě *w*. Proto je RANDIW používán s $c_1 = c_2 = 1.494$ (viz následující kapitola) [13].

Vhodné *w*, c_1 a c_2 mohou učinit PSO mnohem více stabilní (parametr V_{max} být přítomný nemusí, nebo jeho hodnota může být vysoká). Právě stabilita dynamiky částic byla zkoumána (i se všemi stochastickými parametry) pomocí použití metody analýzy Lyapunovy stability a konceptu pasivních systémů, aby byly určeny dostatečné podmínky pro asymptotickou stabilitu a konvergenci k rovnovážnému bodu. Aby byla dynamika systému stabilní, je potřeba, aby byla splněna podmínka

$$c < \frac{2(1 - 2w + w^2)}{1 + w}, \quad (3.6)$$

kde $c = c_1 + c_2$, $w < 1$ a $w \neq 0$. Nestabilita (hejno uteče z dané oblasti) roste s *c*. Při zvýšení *w* je pro zajištění stejné úrovně stability potřeba snížit *c*. Menší počet nestabilit se vyskytuje u spíše menšího *w* a většího *c*. Pokud má být udržena stejná úroveň explorace a konvergence, mělo by se s klesajícím *w* ($0 < w < 1$) zvyšovat *c*. Pro lokální průzkum je nežádoucí, pokud $-1 < w < 0$ vykazuje stabilitu a trajektorie částic mají střídavá znaménka, která vedou k velkým skokům v pohybu částic (rychlost díky zápornému *w* mění každou iteraci znaménko) [14].

3.3.4 PSO s omezujícím faktorem

Později pokračovaly experimenty s omezením rychlosti, protože nějaký druh tlumení dynamiky částic je nezbytný (jelikož tradiční verze algoritmu má nežádoucí dynamické vlastnosti). Neomezením by se rychlost dostala do neakceptovatelných úrovní během pár iterací. Především byly zkoumány

akcelerační koeficienty a výsledkem bylo, že pokud $c_1 + c_2 \in \langle 0.0,\ 4.0 \rangle$, pak trajektorie nestochastických 1-dimenzionálních částic obsahovaly zajímavé pravidelnosti. Byla navržena strategie pro umístění omezujícího faktoru, který jednotně ovlivňuje předchozí rychlost a akcelerační koeficienty. Tím je pak řízena konvergence částic, brání se explozi rychlosti a eliminuje se V_{max} [7].

Existuje mnoho způsobů, jak implementovat omezující faktor χ. Nejjednodušším způsobem je vynechání váhy setrvačnosti w, tedy nová rovnice rychlosti je

$$v_{ij}(t+1) = \chi\big(v_{ij}(t) + c_1 r_{1j}(t)\big[pBest_{ij}(t) - x_{ij}(t)\big] + c_2 r_{2j}(t)\big[gBest_j(t) - x_{ij}(t)\big]\big), \quad (3.7)$$

kde

$$\chi = \frac{2}{c - 2 + \sqrt{c^2 - 4c}}, \quad (3.8)$$

kde $c = c_1 + c_2 > 4$. Nejlepší výsledky byly dosaženy při $c_1 = c_2 = 2.05$ a $\chi = 0.7298$. PSO bude konvergovat, ale lepší přístup je pojistit se i nastavením limitu pro V_{max} na X_{max} (rozsah každé proměnné v každé dimenzi). Výsledkem je PSO s žádnými problémově závislými parametry. PSO s omezujícími koeficienty je algebraicky ekvivalentní s PSO s váhou setrvačnosti, kde jsou pak parametry následující: $w = 0.7298$, $c_1 = c_2 = 1.49618$.

Obecně χ zlepšuje konvergenci ztlumením oscilací, pokud se částice zaměří na bod v optimální oblasti. Nevýhodou může být, že pokud je *pBest* daleko od *gBest* (jsou v různých oblastech), tak částice nemusí konvergovat (opisují širší cykly).

PSO jen s V_{max} je schopné najít optimální oblast, ale už nemá prostředky k tomu, aby byl schopen konvergovat do optima. Techniky s omezeními přinutí hejno ke konvergenci. Varianty s V_{max} zaostávaly za variantami s omezujícími podmínkami. Tyto modifikace garantují konvergenci, kterou tradiční algoritmus s V_{max} negarantuje [10].

3.4 Sousedství a topologie v PSO

První PSO (kapitola 3.3) bylo vyvinuto ze simulace hejn ptáků, které je popsáno v [15]. V tomto modelu je trajektorie každého letu ptáka změněna aplikací několika pravidel, především vlivem ptáků z fyzického okolí. To, o jaké konkrétní částice (ptáky) v PSO půjde, závisí na volbě topologie a sousedství. Topologie určuje, s kým každá jednotlivá částice bude komunikovat. Tyto komunikující částice pak tvoří sousedství, ve kterém se navzájem ovlivňují. Každá částice má své sousedství, z tohoto svého sousedství vybere nejlepší částici – *gBest*. Každá částice má obecně jiné sousedství. V závislosti na tom, od kterého jedince informaci převezme, může algoritmus vykazovat mírně odlišné chování v rychlosti a kvalitě nalezeného řešení.

Topologie ovlivňuje prohledávání na nižší úrovni. Částice ze stejného sousedství tíhnou k prohledávání stejné oblasti. Definicí vztahů mezi lokálními sousedstvími lze ovlivnit hledání na

vyšší úrovni. Sociální sítě můžou být charakterizovány svými údaji, jako jsou informace o struktuře a rychlost komunikačního toku. Kdykoli částice objeví dobrou oblast prostoru, přímo ovlivní jen přímé sousedy (spojeni jednou hranou). Sousedé spojení přes dvě hrany budou ovlivněny až poté, co se vliv projeví u přímých sousedů. Tedy informace proudí grafem se zpožděním [5].

Původní PSO topologie byly založeny na blízkosti v prohledávaném prostoru (geografické sousedství). V tomto sousedství vyvstává problém s definicí hranic vzdáleností, kdy jedinec patří do sousedství a kdy už ne. Bohužel tento druh komunikační struktury měl (kromě výpočetní náročnosti) nežádoucí konvergenční vlastnosti, a proto tato euklidovská sousedství byla brzy opuštěna (až na výjimky).

Jiné (v současnosti používané) topologie jsou založeny na sociálním sousedství. Jedinci jsou sousedy bez ohledu na to, kde se v prohledávacím prostoru nacházejí. O jejich příslušnosti do sousedství rozhoduje pouze „pořadové číslo“ jedince. Toto sousedství se používá nejčastěji. Je taky nutno definovat počet jedinců v sousedství.

3.4.1 Statická sousedství

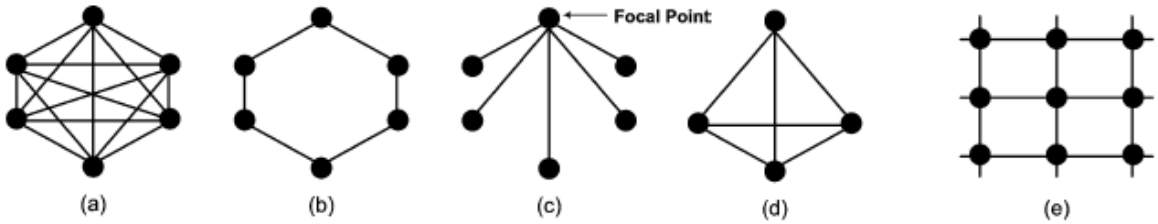
Používají se především dva druhy statických sousedství. Prvním je *globální sousedství* (Global Best topologie), jež je reprezentováno plně propojeným grafem, ve kterém každou částici ovlivňuje nejlepší soused z úplně celé populace. Jedinec se moc nedívá kolem sebe, ale vždy má tendenci se přesunout k doposud nalezenému globálnímu extrému. U této varianty hrozí, že se algoritmus zasekne v lokálním extrému, protože při nalezení tohoto lokálního extrému se jedinci kolem něj shromáždí a existuje zvýšená šance, že už se od něj nehnou. Výhodou je, že u funkcí s jedním výrazným globálním extrémem konverguje hodně rychle.

Druhým používaným sousedstvím a pro sociální sousedství relevantnější je *lokální sousedství* (Local Best topologie). V tomto sousedství se většinou používá jako proměnná pro nejlepší nalezené řešení v sousedství v rovnici (3.2) *lBest* místo *gBest*. Nejčastější topologie je kruhová mřížka, kde každý jedinec je připojen k oběma sousedním členům populační řady (kruhu, $K = 2$). Obecně $K > 2$, kde K je počet sousedů. Výměna informace uvnitř sousedství jedinců odráží lokální znalost prostředí. Jednotlivá sousedství nejsou oddělenými jednotkami, ale jsou navzájem propojená a mohou se navzájem ovlivňovat. Výhoda této topologie je v možnosti paralelního prohledávání, protože jednotlivé populace mohou konvergovat v odlišných oblastech prohledávacího prostoru. Je tedy stabilní pro více stejně dobrých optim. Ačkoli toto sousedství konverguje pomaleji než globální sousedství, je méně zranitelné na uváznutí v lokálním optimu [7][8].

Byly zkoumány různé komunikační struktury (kruh, hvězda, pyramida, vonNeumann, náhodné přiřazení hran) na různé problémy, ale žádná nebyla obecně lepší než jiná. Příklady topologií lze vidět na obrázku 3.2. Lehce lepší byla von Neumannova topologie (sousedství na mřížce, $K = 4$).

Výkony se lišily s testovanou funkcí. Vyjmutí samotné částice ze svého sousedství nemělo téměř žádný vliv na chování hejna.

Obecně pro unimodální problémy jsou vhodné vysoce propojené topologie, pro multimodální řídce propojené topologie [7].



Obrázek 3.2 Příklady topologií hejna [16]. (a) Globální. (b) Kruh. (c) Hvězda. (d) Pyramida. (e) Von Neumann.

3.4.2 Dynamická sousedství

Dynamické (adaptivní) topologie, které vycházejí ze statických, mohou být někdy výhodné. Jak bylo popsáno výše, lokální sousedství lépe prohledává prostor, zatímco globální sousedství konverguje rychleji. Proto byla navržena technika, kde hejno částic nejprve prohledává prostor s lokální kruhovou mřížkou s malým počtem sousedů, v průběhu se pomalu zvyšuje velikost sousedství, až na konci běhu je populace plně propojená a sousedství maximální.

Další možností je topologie, kde jsou sousedé definováni svou blízkostí v prostoru a počet sousedů se dynamicky zvyšuje během běhu. Později byla použita k nalezení souseda vážená euklidovská vzdálenost. Každé částici je přiřazen jako soused jedinec s nejvyšší hodnotou poměru fitness aktuální a sousedské částice k jejich vzdálenostem v dané dimenzi. Algoritmus používá globální sousedství. Tím je zajištěno, že vybraný soused je dobrý představitel a zároveň neleží ve vzdálené oblasti prostoru (pro všechny dimenze nemusel být vybrán stejný soused).

Další technika pracuje s tím, že se nejprve vytvoří více náhodných populací velikosti n a příležitostně se náhodně mění všechny spoje. Byly získány dobré výsledky na multimodálních problémech s $n = 3$ a se změnou struktury topologie po každých 5 iteracích.

Dobrý výkon byl získán i uspořádáním částic do dynamické hierarchie, každá částice byla ovlivněna svým dosud nejlepším výsledkem a částicí přímo nad sebou. Částice s lepší fitness jsou v hierarchii posunuty nahoru (mají větší vliv na horší částice).

Kvalitní výsledky dosahuje i bezparametrický Particle Swarm systém nazvaný TRIBES, ve kterém se detaily topologie a velikost populace vyvíjejí v čase v závislosti na výkonu. Populace je rozdělena na více menších populací, kde každá obsahuje vlastní pořadí a strukturu. Pro výkonné kmeny (tribes) může být výhodné odstranit jejich nejslabší člen (už mají dobré řešení problému a mohou si dovolit zredukovat populaci), pro nevýkonné kmeny je lepší přibrat nového člena (zvýší

možnost zlepšení). Nové částice jsou náhodně generovány. Modifikace populační struktury se provádí v každé L/2-té iteraci (L je počet spojení v populaci) [7].

3.5 Varianty PSO

Základní algoritmus podstoupil mnoho úprav, které vyústily k lepšímu výkonu (obecnému nebo jen u konkrétních druhů problémů). Varianty 3.5.1 a 3.5.2 pracují s jinou než reálnou reprezentací problému. Ve variantách 3.5.3 a 3.5.4 částice interagují s mnoha částicemi najednou. Varianty 3.5.4 - 3.5.9 obsahují změny pro podporu diverzity hejna. Varianty 3.5.9 - 3.5.12 se zaměřují spíše na konvergenci, aby se hejno v průběhu vykonávání nezastavilo. Ve variantách 3.5.13 a 3.5.14 je využita operace křížení známá z genetických algoritmů. Varianty 3.5.14 - 3.5.18 byly vyvinuty pro řešení multimodálních problémů. Varianta 3.5.18 je typickým zástupcem pro řešení dynamických problémů. Varianty 3.5.19 - 3.5.21 se zaměřují na řešení multi-kriteriálních problémů. (Další varianty na multi-kriteriální problémy lze nalézt např. v [17].) Mnoho variant (3.5.2, 3.5.16, 3.5.18, 3.5.21 - 3.5.24) využívá více-rojový přístup, kde jednotlivá hejna spolu kooperují.

3.5.1 Diskrétní PSO

Pro diskrétní problémy byla vyvinuta varianta diskrétního (celočíselného) PSO. Jednou z možností je použití klasického PSO se zaokrouhlením reálných hodnot. U tohoto přístupu se předpokládá souvislost dvou po sobě jdoucích čísel [18].

Nejběžnější přístup rozšíření PSO na kombinatorické problémy je založen na principu permutací. V těchto algoritmech je pozice částice konkrétní permutace a rychlost se stává operátor výměny, který mění jednu permutaci v druhou. Příklad konkrétního algoritmu na kombinatorické problémy je popsán v kapitole 3.5.2.

Speciálním případem diskrétních problémů jsou binární problémy, jejichž řešení obsahují jen číslice 0 a 1 (řetězce bitů). Takovéto problémy řeší binární PSO.

3.5.1.1 Původní binární PSO

PSO pro problémy s binární reprezentací zachovává koncept sociálního a kognitivního učení, ale mění způsob upravování vlastností částic. Složky vektoru polohy mohou nabývat pouze hodnot 1 nebo 0. Rovnice rychlosti zůstává nezměněná kromě toho, že x_{ij}, $pBest_{ij}$ a $gBest_{ij}$ mohou nabývat hodnot 0 nebo 1.

Rychlost je použita jako práh pravděpodobnosti pro stanovení nové pozice. Toho je dosaženo pomocí sigmoidální funkce

$$sig(v_{ij}) = \frac{1}{1+e^{-v_{ij}}}. \tag{3.9}$$

Rovnice pro výpočet nové pozice je následující ($rand \in \langle 0, 1\rangle$): Pokud $rand \leq sig(v_{ij})$, pak $x_{ij} = 1$, jinak $x_{ij} = 0$. Aby se vyhnulo tomu, že práh (*sig*) bude příliš blízko 0.0 nebo 1.0, je V_{max} typicky nastaven na 4.0; to zaručí, že nejmenší pravděpodobnost bude $sig(V_{max}) = 0.018$ [18].

Algoritmus dosahoval na náhodných binárních problémech lepších výsledků než různé druhy genetických algoritmů.

3.5.1.2 Binární PSO s novými operátory

Další technika binární PSO (BPSO) představila nové operátory pro aktualizaci rovnic rychlosti a pozice. V této technice je každé potenciální řešení reprezentováno svou pozicí a rychlostí, což jsou *d*-bitové binární řetězce, a jsou aktualizovány podle rovnic:

$$v_i(t+1) = wv_i(t)\ or\ \alpha\big(pbest_i(t)\ xor\ p_i(t)\big)\ or\ \beta(gbest_i(t)\ xor\ p_i(t)) \qquad (3.10)$$

a

$$p_i(t+1) = p_i(t)\ xor\ v_i(t+1)\ , \qquad (3.11)$$

kde *w* je váha setrvačnosti (obecně menší než 1) a α, β jsou parametry řídící konvergenční rychlost algoritmu (autory nastavené na $w = 0.729$, $\alpha = \beta = 1.94$). V rovnicích jsou použity pro práci s binárními řetězci operátory *or* a *xor*.

Nové rovnice jsou analogií k původním rovnicím standardní PSO pro spojité problémy; operátor *or* nahradil operátor plus a operátor *xor* operátor mínus. Průběh algoritmu odpovídá průběhu u standardního PSO. Algoritmus prokazoval na „knapsack“ problému lepší výsledky než genetický algoritmus [19].

3.5.2 Kooperativní PSO pro kombinatorické problémy

V kooperativním PSO pro kombinatorické problémy (CCPSO) se každé řešení jako obvykle skládá z několika částí (= dimenzí), každá část je reprezentována jinou částicí a tyto částice tvoří hejno [20]. Tyto hejna pak tvoří jedno velké hejno. Každá část může nabývat určitý počet konkrétních diskrétních hodnot daného kombinatorického problému. Tyto hodnoty jsou reprezentovány statickými částicemi (atraktory), které nejsou součástí žádného hejna. Tyto atraktory, které ovlivňují pohyb částic v prostoru, jsou na sebe ortogonální a tvoří multidimenzionální prostor. Pozice každé částice se mění pohybem v tomto prostoru, vektor pozice má tedy tolik položek, kolik je atraktorů. Podle úspěšnosti atraktorů, která je dána účelovou funkcí kandidátního řešení, které daný atraktor obsahuje, je upravována jejich přitažlivost pro konkrétní částice. Tato přitažlivost je pak promítnuta do rovnice rychlosti a tedy i do rovnice pozice. V rovnici rychlosti je *pBest* nahrazena nejlepším řešením nalezeném dosud v hejnu a *gBest* nejlepším řešením nalezeným v množině hejn, které jsou sousedy daného hejna (záleží na topologii). Kandidátní řešení je pak sestrojeno z odpovídajících částic, které

reprezentují hodnoty nejlepších (nejpřitažlivějších) atraktorů (je kooperativně dekódována pozice každé částice pro tu danou část řešení).

Na tento algoritmus lze nahlížet jako na hejna (= analogie částic) v hejně. Každé hejno poskytuje po každé iteraci jedno kandidátní řešení (částice v hejně reprezentuje jednu dimenzi), s těmito řešeními se pracuje z pohledu *pBest* a *gBest* stejně jako s řešeními ve standardním PSO. Počet hejn většinou odpovídá počtu částic ve standardním PSO.

Toto rozdělení řešení na části, kde každá částice prohledává jednu část vektoru, dovoluje důkladné nastavení každé části, protože šablona kandidátního řešení se nemění v jedné iteraci. Nestává se to, že je prozkoumáno řešení, až když se změní všechny části (dimenze) v prostoru jako u klasické PSO. Algoritmus měl lepší výsledky než zaokrouhlování výsledků spojitých hodnot nebo diskrétní PSO na problému Side-chain Packing proteinů.

3.5.3 Plně informovaný PSO

V plně informovaném PSO (FIPS) byl upraven způsob, jakým částice interagují se svými sousedy. Zatímco v tradičním algoritmu je každá částice ovlivněna sebou a nejlepším dosud nalezeným ve svém sousedství, ve FIPS je částice ovlivněna všemi svými sousedy (někdy bez vlivu svého předchozího úspěchu *pBest*) [5].

Ve standardním PSO není zaručeno, že nejlepší soused našel lepší oblast než druhý nebo třetí nejlepší soused. Důležitá informace o prostoru je tedy zanedbána zaměřením se pouze na jednoho souseda. Jak bylo naznačeno v kapitole 3.3.4, omezující koeficient χ je odvozen z hodnot akceleračních koeficientů c_1 a c_2, resp. jejich součet c určuje χ. To zestručnilo rovnici rychlosti na

$$v_{i+1} = \chi(v_i + c(p_{im} - x_i)) \ , \tag{3.12}$$

kde

$$p_{im} = \frac{c_1 \cdot r_1 \cdot pBest_i + c_2 \cdot r_2 \cdot gBest}{c_1 \cdot r_1 + c_2 \cdot r_2} \tag{3.13}$$

a kde je akcelerační váha $c = c_1 + c_2$ je rozdělena mezi $pBest_i$ a *gBest*. Rovnice 3.12 je ekvivalentní k rovnici 3.7. Prohledávání částice *i* konverguje do bodu p_{im} v prostoru.

Variabilita je zavedena několika způsoby. Jednak je bod vážen náhodným číslem; to ovšem nezabrání, aby se rychlost limitně neblížila nule (např. pokud $p_{im} - x_i = 0$, rychlost se bude blížit nule). Tedy dokud se bude pozice částice lišit od předchozí nejlepší pozice, částice se bude stále pohybovat. Dokud budou nalezeny lepší body v prostoru a měnit se p_{im} díky *gBest*, rychlost částice neustane. Pro konvergenci je důležité, aby p_{im} (*gBest*) zůstalo fixní. Dále je pro variabilitu důležitý rozdíl mezi $pBest_i$ a *gBest*. Náhodné vážení drží hledání mezi a za těmito body. Limity pro tyto dvě rovnoměrné rozdělení (pro $pBest_i$ a pro *gBest*) jsou obvykle stejné, proto je celková váha *c* rozdělena do dvou rovnajících se částí – c_1 a c_2. Je nezbytné, aby součet těchto hodnot odpovídal omezující váze χ.

Algoritmus bude probíhat konvergenčně a rozpínavě správně, ať už bude *c* rozděleno do dvou, tří, nebo *N* částí. A tak byl odvozen alternativní výpočet p_{im}, který po zobecnění rovnice 3.13 je

$$p_{im} = \frac{\sum_{k \in N} W(k) c_{ik} \cdot pBest_k}{\sum_{k \in N} W(k) c_{ik}}, \tag{3.14}$$

kde c_{ik} je akcelerační koeficient, který udává, jak moc částice *i* následuje částici *k*, a má tvar

$$c_{ik} = rand\left(0, \frac{c}{N}\right) \quad \forall k \in N \tag{3.15}$$

a kde *N* je množina sousedů částice *k* a $pBest_k$ je nejlepší pozice nalezená částicí *k*. *W* je funkce, popisující nějaký aspekt částice, který je považován za důležitý (fitness nejlepší nalezené pozice částice *k*, vzdálenost dané a aktuální částice, konstanta). Protože všichni sousedé přispívají ke změně rychlosti, částice je plně informovaná. FIPS není radikální odklon od předchozích verzí. Standardní PSO s dvěma nejlepšími pozicemi (své a globální) je chápáno jako speciální případ FIPS.

V obyčejném PSO platí, že čím početnější sousedství, tím lepší částice je pravděpodobně vybrána. U FIPS to neplatí. Velikost sousedství určuje, jak rozmanité bude prohledávání. Větší diverzita přináší rozředěné hledání (místo zlepšeného).

Nejlepší výsledky dosahovala FIPS s kruhovou nebo von Neumannovou topologií s 3 – 5 sousedy, kde funkce *W* reprezentovala fitness $pBest_k$. FIPS ve všech měřených ohledech překonal standardní PSO.

3.5.4 PSO s úplným učením

PSO s úplným učením (CLPSO) používá novou strategii učení, kde *pBest* všech ostatních částic je využita k aktualizaci rychlosti částice [2]. Tato strategie umožňuje zvýšit diverzitu hejna, aby se předešlo předčasné konvergenci. Kromě předčasné konvergence to může řešit i další problém: Jelikož fitness hodnota částice je určena hodnotami všech *D* parametrů (*D* je počet parametrů problému), částice, která objevila oblast odpovídající globálnímu optimu v nějakých dimenzích, může mít malou fitness kvůli špatnému řešení v jiných dimenzích.

V této strategii se rychlost aktualizuje následující rovnicí:

$$v_i^d \leftarrow w \cdot v_i^d + c \cdot rand_i^d \cdot (pbest_{fi(d)}^d - x_i^d), \tag{3.16}$$

kde $f_i = (f_i(1), f_i(2), \ldots, f_i(D))$ definuje, které *pBest* ze všech částic by měla částice *i* následovat v jednotlivých dimenzích. V závislosti na pravděpodobnosti učení *Pc* je $pBest_{fi(d)}{}^d$ buď odpovídající dimenze *pBest* jakékoli jiné částice, nebo odpovídající dimenze vlastního *pBest*. Pro každou dimenzi částice *i* se tedy generuje náhodná hodnota. Pokud je tato hodnota větší než Pc_i, odpovídající dimenze se bude učit z vlastního *pBest*, jinak se bude učit z *pBest* jiné částice.

Byla použita procedura turnajové selekce, kdy se dimenze částice učí z *pBest* jiné částice následovně: Náhodně jsou vybrány dvě částice z populace (sebe nebere v potaz). Porovnají se fitness

hodnoty *pBest* těchto částic a vybere se lepší z nich. Vítězovo *pBest* je použito jako exemplář, od kterého se učit pro danou dimenzi. (Pokud je pro všechny dimenze vybrána vlastní *pBest*, pak se v náhodné dimenzi vybere jiná částice.)

Všechny tyto $pBest_{fi}$ můžou generovat nové pozice v prostoru použitím informace odvozené od historicky nejlepších pozic různých částic. K ujištění, že částice se bude učit od dobrého exempláře, a k minimalizaci zbytečného času, který se stráví špatným směřováním, je dovoleno částici se učit od jednoho exempláře jen určitou dobu. Pokud se částice nezlepšuje po určitý počet iterací, který se nazývá obnovovací propast *m*, pak se částici znovu přiřadí jiné f_i. Tedy místo učení se v každé iteraci ze dvou různých exemplářů (*pBest* a *gBest*) stejných pro všechny dimenze (klasické PSO) se každá dimenze částice učí od právě jednoho exempláře po několik iterací (každá dimenze se učí od jiného *pBest*).

Pokud se ve standardním PSO dostane *pBest* ke *gBest*, které je jen lokálním optimem, už nikdy se nemusí částice dostat z oblasti lokálního optima. Avšak v CLPSO má částice díky učení se od jiné částice schopnost vyskočit z lokálního optima přes kooperativní chování celého hejna. Strategie CLPSO dává větší potenciální prohledávací prostor než původní PSO. Zvětšením potenciálního prohledávacího prostoru každé částice se zvyšuje diverzita. Protože *pBest* každé částice je potenciálně dobrá oblast, prohledávání CLPSO není ani slepé ani náhodné. CLPSO prohledá více slibných oblastí, aby nalezl globální optimum.

Různé *Pc* pro každou částici dává podobné výsledky u unimodálních i multimodálních problémů. Proto je vhodné, aby každá částice měla jiné *Pc*. To má pak každá částice jinou úroveň exploračních a exploitačních schopností a jsou schopny řešit rozmanité problémy. Empiricky byl vyvinut výraz pro výpočet Pc_i pro každou částici *i*, např. čím vyšší pořadové číslo částice, tím vyšší hodnota Pc_i (např. Pc_i pro i z $\{1, \ldots, 30\}$ nabývá postupně hodnot z $\langle 0.05, 0.5 \rangle$). Pokud je $m = 0$, je dosaženo rychlejší konvergence rychlosti a lepších výsledků na koulovitých funkcích. Na jiných testovaných funkcích se získalo lepších výsledků, pokud $m = 7$.

Díky více exemplářů, od kterých se částice učí, má částice větší potenciální prostor kam letět. Učící strategie dovoluje využít informaci v hejnu efektivněji a generovat kvalitnější řešení častěji. Výsledky demonstrují dobrý výkon nad multimodálními problémy v porovnání s jinými současnými variantami PSO (PSO s vahou setrvačnosti (kapitola 3.3.3), PSO s omezujícím faktorem (kapitola 3.3.4), unifikovaný PSO, CPSO-H (kapitola 3.5.24), FIPS (kapitola 3.5.3)). Naopak je CLPSO trochu horší pro unimodální problémy.

3.5.5 PSO se soustředným prostorovým rozšířením

Další variantou je PSO založené na soustředném prostorovém rozšíření (CSE-PSO). V tomto schématu se kombinuje prostorové rozšíření a soustředné řazení potomstva [21]. Soustředné řazení potomstva je známé např. z mravenčích kolonií a jde o chování, jež vede k jednomu shluku

potomstva, který tvoří soustředné kruhy jednotlivých potomků, kde nejmladší jsou uprostřed a nejstarší na kraji shluku. Čím je pozice okrajovější, tím bude mravenec pravděpodobně dříve nakrmen.

V PSO s prostorovým rozšířením (SE-PSO) má každá částice prostorový objem, který je definován poloměrem *r*. V základním PSO se se zvyšujícím se počtem iterací částice shlukují příliš těsně kolem optima, které může být lokální. To může způsobit stagnaci hejna, které nemá schopnost utéct z tohoto lokálního optima. Aby nedocházelo k tomuto shlukování a zvýšila se diverzita hejna, každá částice obsahuje toto konstantní prostorové rozšíření, díky kterému se částice odrážejí jedna od druhé. Dvě částice *i* a *j* kolidují, pokud $\|x_i - x_j\| \leq (r_i + r_j)$. Pokud došlo ke kolizi (částice se překrývají a pravděpodobně prohledávají stejné oblasti), obě částice se odrazí na druhou stranu. Nová pozice a rychlost odrazené částice se vypočítá podle

$$x'_{id}(t+1) = x_{id}(t+1) - 2 \cdot v_{id}(t+1) \tag{3.17}$$

a

$$v'_{id}(t+1) = -v_{id}(t+1)\,. \tag{3.18}$$

V PSO se soustředným prostorovým rozšířením (CSE-PSO) se oproti SE-PSO využívá i schématu soustředného řazení potomstva v mravenčích koloniích (popsaného výše) pro adaptivní změnu poloměru, protože výkon algoritmu kriticky závisí na výběru poloměru každé částice. Poloměr se přizpůsobuje na základě vzdálenosti od středu shluku. Za střed shluku se považuje *gBest* pozice. Tedy čím dále je částice od *gBest*, tím více neznámý prostor je prohledán. Naopak čím blíže ke *gBest*, tím více prostoru kolem známého *gBest* je důkladně prohledán. Proto částice dále od *gBest* mají větší poloměr a částice blíže *gBest* menší poloměr. Poloměr částice *i* může být spočítán

$$r_i(t+1) = pL\frac{\|x_i(t+1) - gbest(t)\|}{max_i\{\|x_i(t+1) - gbest(t)\|\}}\,, \tag{3.19}$$

kde *L* je nejdelší úhlopříčka možných oblastí a $p \in \langle 0, 1\rangle$. CSE-PSO umožní některým částicím prohledat jiné oblasti prostoru, zatímco ostatní zůstanou u optima, aby nalezli ještě lepší řešení (kompromis mezi exploraci a exploitací).

CSE-PSO vykazuje lepší výsledky než SE-PSO s $r = 0.01L$. Maximální poloměr každé částice v PSO je nastaven také na $r = 0.01L$, tedy $p = 0.01$. CSE-PSO má větší konvergenční rychlost než SE-PSO. CSE-PSO má také přesnější optimální řešení na všech testovacích funkcích.

3.5.6 PSO s vyhýbáním se lokálnímu optimu

PSO s vyhýbáním se lokálnímu optimu (LOAPSO) značně překonává standardní PSO tím, že se umí vyhnout zachycení v lokálním optimu a má lepší konvergenční výkon [22]. PSO často trpí předčasnou konvergencí na problémech s mnoha lokálními optimy. Kvůli tomu může hejno uvíznout v lokálním

minimu a už neprohledá slibné oblasti v úplně jiných částech prostoru. Je tedy potřeba najít rovnováhu v prohledání současného lokálního minima důkladně skrz rychlou konvergenci a v prohledání jiných oblastí.

Aby se překonal problém uvíznutí v lokálním minimu, byla navržena strategie založená na vyhýbání se lokálnímu optimu a na zvyšování diverzity. Populace je rozdělena na dvě části pomocí nové proměnné γ, která se nazývá míra vyhýbání. Pokud $t < 0.75t_{max}$, pak $\gamma(t)$ nabývá náhodné hodnoty z intervalu $\langle 0, 1\rangle$, jinak $\gamma = 1$ (t_{max} je maximální počet iterací). Nechť N je počet částic, pak γN je počet částic, které upravují rychlost pomocí standardní rovnice (3.3), a $(1-\gamma)N$ je počet částic upravující rychlost podle

$$v_i(t+1) = \omega v_i(t) - L(t)[c_1 r_1\big(pBest_i(t) - x_{iL}(t)\big) + c_2 r_2\big(gBest(t) - x_i(t)\big)]\,, \quad (3.20)$$

kde $L(t)$ je koeficient vyhýbání, který je vybrán iterativně podle $L(t) = 2(1-t/t_{max})$. První skupina směřuje směrem k nejlepšímu řešení, druhá směřuje pryč od nejlepší částice první skupiny – vyhýbání se lokálnímu minimu a zvýšení diverzity.

Testy ukazují, že LOAPSO dosahuje lepších výsledků než standardní PSO a CPSO (kapitola 3.5.24). V porovnání s CLPSO (kapitola 3.5.4) bylo LOAPSO lepší jen na polovině testovacích funkcí.

3.5.7 Dissipativní PSO

Algoritmus zavádí do modelu negativní entropii, aby stimuloval PSO. Tím se vytváří disipativní struktura, která brání před předčasnou stagnací. Negativní entropie zavádí chaos do rychlostí a pozic částic následovně: Pokud $rand < c_v$, pak $v_{id} = rand \cdot V_{max,d}$, a pokud $rand < c_x$, pak $x_{id} = Rand(l_d, u_d)$, kde $rand$, c_v, c_x jsou náhodné čísla z intervalu $\langle 0, 1\rangle$ a $Rand(l_d, u_d)$ je náhodné číslo z intervalu $\langle l_d, u_d\rangle$. Chaos zabraňuje systému, aby se dostal do rovnovážného stavu. Samo organizace disipativních struktur a nelineární interakce v hejně vedou ke kolísání částic [16].

3.5.8 PSO s pasivním shromažďováním

Pasivní shromažďování je mechanismus, který dovoluje živočichům se shlukovat do skupin. Tento algoritmus chrání PSO před uváznutím v lokálním optimu a zlepšuje jeho přesnost a konvergenční rychlost. K rovnici rychlosti přičítá ještě $c_3 \cdot r_3 \cdot (X - \boldsymbol{x}_i(t-1))$, kde c_3 je koeficient pasivního shromažďování, r_3 je náhodné číslo z intervalu $\langle 0, 1\rangle$ a X je náhodně vybraná částice z hejna. Není specifikována hodnota c_3 ani její vliv na výkon [16].

3.5.9 PSO s měnícími se akceleračními koeficienty v čase

V PSO s měnícími se akceleračními koeficienty (PSO-TVAC) byly navrženy strategie automatizace parametrů. Hlavním cílem je zlepšit výkon po předdefinovaném počtu iterací. Kromě měnící se

w v čase (TVIW) podle rovnice (3.4) se v čase mění i koeficienty c_1 a c_2. Obecně v optimalizačních metodách založených na populaci je žádoucí podporovat, aby se částice v brzkých fázích optimalizace pohybovaly skrz celý prohledávací prostor bez shlukování se kolem lokálního optima a aby se v pozdějších fázích zvýšila konvergence hejna ke globálnímu optimu. Proto byl zaveden TVAC. Cílem je nejprve zvýšit globální prohledávání a po čase zvýšit konvergenci směrem ke globálnímu optimu [13].

Změnou c_1 a c_2 se redukuje kognitivní a zvyšuje sociální složka. Na počátku je sociální složka malá a kognitivní složka velká, částice se pohybují po celém prostoru. Ke konci je sociální složka velká a kognitivní složka malá, částice konverguje ke globálnímu optimu. Tato metoda běží s TVIW. Tedy c_1 a c_2 se v čase mění podle

$$c_1 = c_{1i} + \frac{(c_{1f} - c_{1i}) act\_iter}{num\_iter} \tag{3.21}$$

a

$$c_2 = c_{2i} + \frac{(c_{2f} - c_{2i}) act\_iter}{num\_iter}, \tag{3.22}$$

kde *act_iter* je aktuální iterace, *num_iter* je celkový počet iterací a c_{1i}, c_{1f}, c_{2i} a c_{2f} jsou konstanty. Nejlepší výkon byl při snižování c_1 od 2.5 do 0.5 a při zvyšování c_2 od 0.5 do 2.5. Na většině funkcí byl nejlepší HPSO-TVAC (kapitola 3.5.9.2), na pár funcích MPSO-TVAC (kapitola 3.5.9.1) nebo PSO-RANDIW (kapitola 3.3.3 s rovnicí (3.5)) Porovnávání proběhlo ještě s PSO-TVIW (kapitola 3.3.3), PSO-TVAC, MPSO-FAC (fixní akcelerační koeficienty) a HPSO-FAC.

3.5.9.1 MPSO-TVAC

První variantou PSO-TVAC je PSO-TVAC s mutací (MPSO-TVAC). V pozdějších fázích optimalizace mnohdy chybí diverzita (předčasná konvergence k lokálnímu optimu). Ke zvýšení možnosti globálního prohledávání je poskytnuta dodatečná diverzita pomocí operátoru mutace. Bylo vypozorováno, že PSO rychle najde dobré lokální řešení, ale někdy zůstane v lokálním optimu značný počet iterací bez zlepšení. Tedy použitím operátoru mutace lze toto řídit, tím se zvýší globální prohledávání. (To je koncepčně blízké operátoru mutace v GA.)

Tedy pokud se globální optimální řešení nezlepšuje během zvyšujícího se počtu iterací, je náhodně vybrána částice a její náhodně zvolené složce (dimenzi) vektoru rychlosti je s předdefinovanou pravděpodobností (pravděpodobnost mutace) přidána náhodná perturbace - zmatek (velikost kroku mutace). Velikost kroku mutace je nastavena proporcionálně k maximální povolené rychlosti. Lze tak krok mutace měnit v čase.

3.5.9.2 HPSO-TVAC

Druhou variantou PSO-TVAC je samo-organizační hierarchické PSO-TVAC (HPSO-TVAC). Pro komplexní multimodální funkce řízení diverzity populace s lineární TVIW může vést částice k předčasné konvergenci do lokálního optima. Proto v této strategii se v rovnici pro výpočet rychlosti bere do úvahy pouze kognitivní (*pBest*) část a sociální (*gBest*) část a úplně chybí část s předchozí rychlostí, resp. předchozí rychlost je nastavena na nulu.

V takových případech se ovšem částice rychle přesunou k lokálnímu optimu a stagnují kvůli absenci hybnosti, tedy optimální řešení silně závisí na inicializaci populace. S touto stagnací se algoritmus vypořádal tak, že pokud k ní dojde a rychlost částice v určité dimenzi je nula, pak se rychlost v této dimenzi znovu inicializuje na náhodnou rychlost (reinicializační rychlost je vynásobena číslem z $\langle 0, 1\rangle$). Velikost reinicializační rychlosti je nastavena proporcionálně k maximální povolené rychlosti. Lze taky velikost reinicializační rychlosti měnit v čase.

3.5.10 Gaussovo PSO

Klasický PSO hledá uprostřed *gBest* a *pBest*. Hledání a konvergence hejna v optimální oblasti tedy závisí na tom, jak budou parametry (w, c_1, c_2) nastaveny. K odstranění tohoto problému je použita Gaussova funkce, která řídí pohyb částic. V nově navržené rovnici se místo konstant objevuje náhodné číslo s Gaussovým rozdělením. Díky těmto změnám je převážně prohledávána oblast mezi *gBest* a *pBest*. Jakmile se *gBest* a *pBest* k sobě blíží, tak klesá odchylka a prohledávací oblast konverguje [16].

Byla navržena další varianta Gaussova PSO, který má rovnici pro změnu rychlosti následující

$$v_i(t) = rand_1 \cdot \big(pBest_i - x_i(t-1)\big) + rand_2 \cdot (gBest - x_i(t-1)) \,, \tag{3.23}$$

kde $rand_1$ a $rand_2$ jsou kladné náhodné hodnoty z Gaussova rozdělení $abs(N(0, 1))$. Není potřeba specifikovat žádný další parametr. Navíc u Gaussova PSO není potřeba ani V_{max}.

3.5.11 PSO s garantovanou konvergencí

Pokud $x_i = pBest_i = gBest$, pak rychlost závisí pouze na wv_i. Pokud je předchozí rychlost částic blízká nule, pak se částice přestanou pohybovat, až doženou *gBest*, což může vést k předčasné konvergenci algoritmu. Algoritmus dokonce nemusel najít ani lokální optimum, pouze nejlepší dosud nalezené řešení. Proto byla upravena rovnice rychlosti pro částici, která reprezentuje nejlepší dosud nalezené řešení, na

$$v_g(t+1) = -x_g(t) + pBest_g(t) + wv_g(t) + \rho(t)(1 - 2r(t)) \,, \tag{3.24}$$

kde ρ je škálovací faktor a $r \in \langle 0, 1\rangle$. Výraz $-x_g(t)$ resetuje aktuální pozici částice (v rovnici pro výpočet pozice) na pozici $pBest_g(t)$. K této pozici je přidán vektor reprezentující současný směr

prohledávání ($wv_g(t)$). Poslední část generuje náhodnou vzdálenost (vzorek) až do velikosti $\rho(t)$ na obě strany, to způsobí náhodné prohledávání kolem $pBest_g$.

ρ se mění v každé iteraci následovně: Pokud *#successes* > s_c, pak $\rho(t+1) = 2\ \rho(t)$; pokud *#failures* > f_c, pak $\rho(t+1) = 0.5\rho(t)$; jinak $\rho(t+1) = \rho(t)$. *#successes* je počet po sobě jdoucích úspěchů, kdy $f(pBest_g(t)) \neq f(pBest_g(t-1))$. *#failures* je počet po sobě jdoucích neúspěchů, kdy $f(pBest_g(t)) = f(pBest_g(t-1))$. $\rho(0) = 1$. Pokud *#successes*$(t+1)$ > *#successes*(t), pak *#failures*$(t+1) = 0$. Pokud *#failures* $(t+1)$ > *#failures* (t), pak *#successes* $(t+1) = 0$. s_c a f_c jsou prahy, jejich optimální hodnota závisí na účelové funkci. Např. při $f_c = 5$ a $s_c = 15$ algoritmus rychleji trestá špatné nastavení ρ, než oceňuje úspěšnou hodnotu ρ.

Alternativně optimální hodnoty f_c a s_c mohou být nastavovány dynamicky. Např. s_c může být zvýšena pokaždé, když *#failure* > f_c, tedy stane se mnohem obtížnější dosáhnout úspěšného stavu, pokud se nezdary objevují často. To chrání před rychlým kmitáním ρ. Pokud určité ρ opakované ústí v úspěch, větší objem vzorku je vybrán ke zvýšení maximální vzdálenosti dosažitelné v jednom kroku. Naopak pokud ρ produkuje f_c po sobě jdoucích neúspěchů, objem vzorku je příliš rozsáhlý a musí být zmenšen [23].

V naprosté většině případů GCPSO dosáhl lepších výsledků než standardní PSO.

3.5.12 Vyvážené PSO

Nechť lokální oblasti kolem dobré pozice (= LEA) jsou D-dimenzionální hyperkostky. Tyto dobré pozice jsou lokální nejlepší pozice nalezené hejnem. Míra prohledávání LEA v iteraci t je dána: $r(t) = N_E(t)/N$, kde N_E je počet částic, které jsou uvnitř LEA E, a N je počet všech částic. Standardní PSO ($N = 10+2D^{1/2}$) má $r(t)$ nízké. Se zvyšujícím se počtem iterací se $r(t)$ zvyšuje jen velmi lehce, když hejno konverguje. Průměrná hodnota $r(t)$ rychle klesá se zvyšujícím se počtem dimenzí. To ústí v nevyvážené PSO, které skutečně netěží danou oblast (nevyužívá ji maximálně k nalezení optima). Je proto navržena varianta pro zlepšení takového PSO.

Vyvážené PSO (BPSO) je upravené PSO tak, aby se zvýšila míra prohledávání oblasti kolem dobré pozice. Základní myšlenkou je přinutit nějaké částice k pohybu směrem k danému LEA. Lze to chápat jako jistý druh lokálního prohledávání. Přístup obsahuje nové parametry.

Pravděpodobnost α, která udává, s jakou pravděpodobností bude částice vybrána k přemístění do některé LEA. Druhý parametr určuje, do které části LEA bude vybraná částice přesunuta (do středu, na náhodnou pozici s rovnoměrným rozdělením, na náhodnou pozici s Gaussovým rozdělením). Třetí parametr určuje, do které LEA bude částice přesunuta. Poslední parametr ρ udává, jak velká každá LEA bude. Pokud $\rho = 0.5$, pak hranice jedné LEA se dotýká s hranicí vedlejší LEA. Pokud $\rho = 1$, pak hranice jedné LEA protíná střed vedlejší LEA. Hodnota ρ může být konstantní, adaptivní (pokud se $r(t)$ zvýší, ρ se sníží), alternativní $\{1/N, \rho_0\}$, náhodné s rovnoměrným rozdělením $\langle 0, \rho_0 \rangle$ nebo náhodné s Gaussovým rozdělením $(1/N, \rho_0)$ [24].

Volba α = 0.5, vždy přesun do LEA s nejlepší částicí (*gBest*) na náhodně (rovnoměrné rozdělení) vybranou pozici uvnitř této LEA a $\rho = \{1/N, \rho_0\}$ (každá hodnota vybrána s pravděpodobností 0.5; $\rho_0 = 0.5$) vždy ukázala lepší výkon než standardní PSO.

3.5.13 PSO s Laplacovým křížením pro sdílení informace

V této variantě byla navržena metodologie ke sdílení informace mezi dvěma částicemi použitím Laplacova operátoru navrženém z Laplacovy funkce hustoty pravděpodobnosti. Dvě částice sdílejí svoji informaci o pozici v prohledávacím prostoru a jejich křížením je vytvořena nová částice. Tato částice nazvaná Laplacova částice nahradí nejhorší částici v hejně. Jsou navrženy dva algoritmy – PSO s Laplacovým křížením a s váhou setrvačnosti (LXPSO-TVIW) a PSO s Laplacovým křížením s omezujícím faktorem (LXPSO-C) [25].

Algoritmus zahrnuje operátor křížení, který rozvíjí model sdílení informace mezi dvěma náhodně vybranými částicemi. Toto PSO s křížením používá Laplacovo rozložení. Tento rodičovský centrální operátor se nazývá Laplacův operátor křížení (LX). Funkce hustoty pravděpodobnosti pro Laplacovo rozložení je

$$f(x|a,b) = \frac{1}{2b}\exp\left(-\frac{|x-a|}{b}\right), -\infty < x < \infty, \tag{3.25}$$

kde a je parametr umístění a $b > 0$ je parametr škály. Oba vzniklí potomci jsou umístěni symetricky s ohledem na pozici rodičů. Pro menší b jsou potomci blíže rodičům a naopak. Pro neměnné parametry a a b LX rozděluje potomky proporcionálně s rozpětím rodičů: Pokud jsou rodiče blízko sebe, očekává se, že potomci budou taky blízko sebe, a pokud jsou rodiče daleko od sebe, pak potomci budou taky pravděpodobně daleko od sebe. Vytvoření potomků y_1 a y_2 se provádí podle $y_1=x_1+\beta_i|x_1-x_2|$ a $y_2=x_2+\beta_i|x_1-x_2|$, kde β_i udává, jak daleko bude potomek od rodiče, a získá se z rovnice hustoty pravděpodobnosti (3.25) a z náhodného čísla $u_i \in \langle 0, 1\rangle$ po úpravách následovně

$$\beta_i = \begin{cases} a - b \cdot \log_e(1 - 2u_i), & u_i \le \frac{1}{2} \\ a - b \cdot log_e(2u_i - 1), & u_i > \frac{1}{2} \end{cases}. \tag{3.26}$$

V algoritmu se nejprve provedou rovnice pro úpravu rychlosti a pozice. Poté po vytvoření dvou potomků pomocí Laplaceova operátoru z náhodně vybraných rodičů je ten lepší potomek vybrán a je nazván Laplaceova částice. Tato částice nahradí buď jednoho z rodičů, nebo nejhorší částici v hejně. Protože je graf Laplaceovy funkce hustoty pravděpodobnosti podobný Gaussově funkci, β_i bude ve více případech spíše menší než větší, a tedy potomek bude generován spíše blíže rodiči.

Testování ukázalo, že LXPSO-C dává nejlepší výsledky. Porovnávalo se s LXPSO-TVIW, PSO-C (kapitola 3.3.4) a PSO-TVIW (kapitola 3.3.3).

3.5.14 PSO s *pBest* křížením

V této variantě je PSO chápán jako systém se dvěma populacemi: Populace tvořená současnými pozicemi a populace tvořená *pBest* pozicemi. Nově navržený algoritmus využívá křížení k vytvoření nových řešení. Stejně jako u GA jsou vybrány z množiny nejlepších řešení (populace *pBest*) rodiče k procesu křížení [26].

Do PSO se přidá pouze jeden nový parametr, který značí četnost křížení. Oproti dřívějším přístupům, kde se křížily současné pozice částic, dochází ke křížení dosud nejlepších nalezených pozic. Tento přístup vychází z algoritmu, kde se v určitých intervalech resetovala pozice částice a ta se vrátila na své *pBest*. Od této techniky už je to jen kousek ke křížení *pBest* dvou jedinců, jelikož jde taky zčásti o reset (kříží se *pBest* a ne současné pozice). V porovnání se standardním PSO je nový přístup více exploitativní (díky resetům) i více explorativní (díky křížení).

Na základě parametru četnosti křížení je v určitých iteracích proveden krok křížení. V tomto kroku jsou tradiční rovnice pro změnu rychlosti a pozice nahrazeny jedinou operací – křížením. Tato operace nahradí současnou pozici každé částice novou pozicí použitím vztahu $x_i = (pBest_i + pBest_{rand}) / 2$, kde $pBest_{rand}$ je *pBest* náhodně vybrané částice. Parametr četnosti křížení ovlivňuje to, jestli se algoritmus bude chovat spíše jako standardní PSO, nebo jako extrémně nenasytný GA, kde algoritmus nestihne konvergovat k optimu.

Aby mohlo hejno plně konvergovat k lokálnímu optimu, je potřeba omezit diverzitu. Protože křížením se částice dostane do úplně jiné vzdálené pozice, a tedy je zvýšena rychlost částice, bylo potřeba upravit omezující faktor χ. Experimenty ukázaly, že nejlepší výsledky algoritmus dosahuje při $\chi = 0.95 \cdot 0.792 = 0.752$ a křížení se provádí jednou za 100 iterací. Oproti standardnímu PSO je dosaženo zlepšení na většině multimodálních prohledávacích prostorů, kde nová řešení dosažená po křížení mohou pomoci algoritmu uniknout z lokálního optima. Zlepšení oproti PSO s křížením aktuálních pozic (x) je konzistentní na mulitmodálních problémech, dokonce výrazné na unimodálních vysoce dimenzionálních problémech. Celkový výkon (na více funkcích) je lepší u PSO s *pBest* křížením než u locust swarm PSO (viz kapitola 3.5.16).

3.5.15 PSO s vychýlením, rozpínáním a odpory

Navržená technika má za cíl pomocí PSO (jakožto jedna z metod inteligentních hejn) efektivně spočítat všechna globální optima účelové funkce. Přístup zahrnuje použití vychýlení, rozpínání a odporů v místech detekovaného optima [27]. Tyto techniky v kontextu PSO ústí v efektivní algoritmus se schopností vyhnout se předchozímu detekovanému optimu, a tedy nalézt všechny globální optima funkce (např. vypočítání Nashova ekvilibria). Vychýlení a rozpínání jsou techniky pro překonání lokálních minim, které se spoléhají na koncept transformace účelové funkce tak, že v její nové podobě je zařazena znalost předchozích nalezených optim. Odpory jsou techniky pro

odražení částic pryč od již lokalizovaného minima. Mnohem lepší výsledky dosahovala tato varianta PSO s omezujícím faktorem než s vahou setrvačnosti.

3.5.15.1 Technika vychýlení

Nechť x_i^*, $i = 1, \ldots, m$, kde m je počet nalezených minim. Potom technika vychýlení je definována

$$F(x) = T_1(x; x_1^*, \lambda_1)^{-1} \cdot \ldots \cdot T_m(x; x_2^*, \lambda_m)^{-1} \cdot f(x) \ , \tag{3.27}$$

kde λ_1, $i = 1, \ldots, m$ jsou volitelné parametry a $T_1, \ldots, T_m$ jsou odpovídající funkce takové, že výsledná funkce F má úplně stejné minima jako f, kromě bodů $x_1^*, \ldots, x_m^*$. Jakékoli sekvenci bodů $\{x_k\}_{k=0}^{\infty}$ konvergující k nějakému minimu x_i^* se již ve funkci F bod $x = x_i^*$ nejeví jako minimum. Tuto vlastnost splňuje např. funkce

$$T_i(x; x_i^*, \lambda_i) = tanh\ (\lambda_i \| x - x_i^* \|) \ . \tag{3.28}$$

Nastavení parametru λ ovlivňuje tvar transformované funkce. Pro větší hodnotu λ je efekt vychýlení na účelovou funkci relativně mírný (zvýší funkční hodnotu jen malému počtu sousedních bodů). Pokud $\lambda < 1$, výsledkem je funkce F s velkými funkčními hodnotami v sousedství vychýleného minima (zvýší funkční hodnotu velkému počtu sousedních bodů - může dokonce působit na hodnoty sousedního minima). Tato transformace zavádí nová lokální minima po obou stranách vychýleného minima (tvoří kolem něj „mexický klobouk“).

Tato technika by neměla být použita samotná na funkci f, jejíž globální minimum je 0 (F by měla taky nulovou funkční hodnotu). To lze řešit pomocí $f = f+c$, kde $c > 0$ je konstanta. Lze to řešit taky pomocí použití odporů.

3.5.15.2 Technika rozpínání

Tato technika se skládá ze dvou fází transformace účelové funkce. První fáze je rozpínaní účelové funkce nahoru, tím se eliminují všechny minima s hodnotami většími než hodnota získaného minima. V druhé fázi transformace je detekované minimum přetočeno na maximum. Všechny minima s nižší hodnotou účelové funkce zůstávají nezměněné. Nechť x^* je získané minimum účelové funkce f, pak rozpínání je definováno novými účelovými funkcemi

$$G(x) = f(x) + \gamma_1 \| x - x^* \| (sign\big(f(x) - f(x^*)\big) + 1) \tag{3.29}$$

a

$$H(x) = G(x) + \gamma_2 \frac{sign\big(f(x) - f(x^*)\big) + 1}{\tanh\left(\mu\big(G(x) - G(x^*)\big)\right)} \ , \tag{3.30}$$

kde γ_1, γ_2 a μ jsou volitelné parametry a $sign()$ je signoidální funkce. Parametr γ_1 řídí rozpínání směrem nahoru. Vyšší hodnoty γ_1 jsou použity ve vícedimenzionálních problémech k ujištění, že lokální minima s funkční hodnotou vyšší než detekované minimum budou eliminovány. Parametry

γ_2 a μ určují rozsah a sílu vyvýšení. Čím vyšší γ_2, tím větší oblast kolem lokálního minima je ovlivněna. Čím nižší μ, tím extrémnější (vyšší) jsou funkční hodnoty.

Tato technika ovlivňuje všechna minima, která mají vyšší nebo stejnou funkční hodnotu než získané minimum. To funguje správně na lokální minima, ale pokud je to aplikováno na globální minimum, všechny ostatní globální minima (s funkční hodnotou stejnou jako získané minimum) jsou zvýšena. I zde se objevuje efekt „mexického klobouku“. To lze řešit buď správným nastavením parametrů (to je mnohdy obtížné), nebo pomocí použití odporů. Odpory brání hejnu konvergovat k jednomu z lokálních minim, která byla uměle vytvořena efektem „mexického klobouku“.

3.5.15.3 Technika odporů

Transformace použité na účelovou funkci *f* zavádí nová lokální minima v „mexickém klobouku“ – vytvarovaná oblast výsledné funkce. Není tedy jisté, že hejno nebude konvergovat k sousedství jednoho z už detekovaných minim. Použití odporů řeší tento problém. Po detekci minima a po aplikaci vychýlení nebo rozpínání je aktivován algoritmus použití odporů. Ten zajistí, že pokud se částice přemístí směrem k již detekovanému minimu, bude odpuzena pryč.

Nejdřív se vypočítají nové pozice všech částic. Pokud už bylo detekováno nějaké minimum, tak se zkontroluje, zda nějaká částice leží uvnitř oblasti kolem nějakého minima. Oblast je dána poloměrem *r*, které je spíše malé (při velkém *r* může byt minimum uvnitř tohoto poloměru). Pokud částice leží uvnitř této oblasti, je odpuzena pryč. To je dáno pomocí

$$x_i = \frac{x_i + p(x_i - x_j^*)}{\left\| x_i - x_j^* \right\|}, \tag{3.31}$$

kde *p* je konstanta určující sílu odporu a x_j^* je *j*-té detekované minimum. Konstanta *p* by měla být dostatečně velká, aby odrazila částici dostatečně daleko od minima (např. $r = 0.5$ a $p = 0.8$.).

3.5.16 Locust Swarm PSO

Hejna kobylek (Locust Swarms) je vícehejnový systém na bázi PSO, který byl navržen na řešení multimodálních problémů [28]. Algoritmus má dvě fáze: prozkoumávající proces, který má za úkol nalézt dobré počáteční pozice pro částice, a proces vykonávání jednotlivých hejn, kdy je prohledávána podrobně konkrétní oblast. První hejno a jeho prohledávání je provedeno nad celým prohledávacím prostorem, následující hejna začínají někde kolem optima nalezeného předchozím hejnem. Každá fáze zabere přibližně stejné množství funkčních ohodnocení.

V kobylkovém hejnu je prohledávání řízeno „sežráním a přestěhováním“. V této strategii poté, co hejno „sežere“ relativně malou oblast prostoru, aby nalezlo lokální optimum, jsou vypuštěni průzkumníci, aby nalezli novou slibnou oblast k „přestěhování“. K přestěhování dojde, až se hejno dostane k bodu, kdy je nepravděpodobné další zlepšování. Proces prozkoumávání se odehrává v dostatečné vzdálenosti od optima, aby došlo k exploraci. Průzkumníci jsou generováni náhodně nad

prostorem, nejlepší řešení jsou použity jako počáteční hodnoty pro další hejno. Nové počáteční rychlosti míří pryč od optima, to vede k rozmanitosti.

Tato strategie odděluje prohledávací proces na fázi intenzivního prohledávání (exploitace) a fázi vysokého prozkoumávání (explorace). Kompromis je tedy nahrazen střídáním dvou nezávislých mechanismů.

3.5.17 PSO s redukcí shlukování

Nejlepší dosud nalezené pozice všech částic mohou rychle konvergovat. To se děje, pokud *pBest* první částice je *gBest* pro druhou částici a tato druhá částice přepíše své *pBest* hodnotou, která se nachází blízko své *gBest* (*pBest* první částice) To je žádoucí pro unimodální problémy, ne pro multimodální. Proto byla navržena varianta, kde je tato konvergence redukována tak, že jsou aktualizovány ty *pBest*, ke kterým je částice přitahována (*gBest*), a ne vlastní *pBest* částice [29].

Pokud dva nebo více jedinců jsou blízko sebe, dochází ke shlukování (snížení diverzity a schopnosti explorace). Lze se tomu vyhnout např. tak, že nové kandidátní řešení nahradí jedno z dvou podobných řešení; tím se nevytvoří shluk (redukce shlukování).

Při situaci popsané výše, kde jedno *pBest* ovlivňuje druhé *pBest* tak, že jsou nakonec tato *pBest* hodně podobná, se formují shluky. K redukci shlukování se nejdříve zkontroluje, zda potenciálně nové *pBest* ($f(x_i) > f(pBest_i)$) je blízko *gBest*. Pokud je vzdálenost nové *pBest* a *gBest* menší než práh, jsou porovnány a potenciálně je toto *gBest* upraveno. Potenciální úprava spočívá v přepsání *gBest* novou pozicí, pokud je nová pozice lepší než toto *gBest*, a původní *pBest* zůstane nezměněné. (Pokud je nová pozice blízko *pBest* i *gBest*, je přepsána ta bližší.) Je tedy použita prahová funkce k řízení minimální potřebné diverzity. Hejno jako celek si stále pamatuje nejlepší známou pozici (*gBest*), ale více udržuje diverzitu v populaci složené ze všech *pBest*. Velikost prahu se snižuje v průběhu vykonávání algoritmu.

Jiné strategie pro udržení diverzity založené na nikách (rozdělení celé populace do několika menších populací, kde každá prohledává oblast kolem odlišného lokálního optima) a shlukování jsou buď výpočetně náročné, nebo náchylné na „chyby nahrazení“ (porovnává se jen určitá podmnožina jedinců). Tento návrh zlepšuje exploraci udržením diverzity a je přitom výpočetně nenáročný. To vede k značnému zlepšení výkonu v multimodálních problémech. Klíčem je snadná identifikace existujícího jedince, s kterým nové řešení může vytvořit shluk.

Na unimodálních funkcích se tato varianta nechovala konzistentně, na multimodálních dosahovala výrazného zlepšení oproti standardnímu PSO.

3.5.18 PSO na dynamické problémy

Hlavní myšlenkou je rozdělit populaci částic na interagující hejna. Tato hejna komunikují lokálně pomocí parametru vyloučení a globálně skrz nový anti-konvergenční operátor. Každé hejno udržuje

diverzitu použitím nabitých nebo kvantových částic. Tento více-rojový algoritmus se používá pro řešení různých problémů s několika dynamickými (pohybujícími) optimy [30].

Pokud se dynamicky změní problém (funkce), informace uložená v paměti (nejlepší řešení a jeho fitness) nemusí být pravdivá a může tedy být zavádějící – zastaralá paměť. Protože se konvergující hejno smršťuje kolem optima (ztráta diverzity), výrazný posun tohoto optima daleko od hejna způsobí to, že hejno bude kmitat kolem špatných atraktorů (hejno nemá rychlost uniknout).

Celé hejno je rozděleno do několika menších hejn s cílem umístit každé takové hejno na odlišný slibný vrchol krajiny. Avšak jednoduché rozbití sousedství a rozdělení globálního hejna do několika nezávislých hejn pravděpodobně nebude efektivní, protože hejna by neinteragovali. Jsou aplikovány dvě formy interakce: vyloučení a anti-konvergence.

Vyloučení: Pokud je hejno rozděleno do několika malých hejn, může se stát, že se částice z různých hejn shluknou kolem jednoho vrcholu. To je nežádoucí, protože motivace je mít různé hejna na různých vrcholech. Aby se toho dosáhlo, je používána jednoduchá soutěž mezi hejny, které jsou blízko sebe. Vítěz je hejno s nejlepší funkční hodnotou ve svém atraktoru hejna (*gBest*). Poražený je vyloučen a znovu inicializován v prostoru (vítěz zůstává). Hejna jsou považována, že jsou blízko sebe (překrývají se), pokud jsou jejich hejnové atraktory uvnitř vylučovacího poloměru r_{excl}. Vyloučení tedy představuje místní interakci mezi kolidujícími hejny.

Anti-konvergence: Hejno konverguje, pokud je prostorový rozsah (největší vzdálenost složek mezi jakýmikoli dvěma částicemi) neutrálního hejna menší než poloměr konvergence r_{conv}. Pokud je počet hejn menší než počet vrcholů a všechny hejna konvergují k nějakému (odlišnému – vyloučení) vrcholu, systém ztratí schopnost detekovat vrchol. Proto kdykoliv všechny hejna konvergují, anti-konvegence vyloučí nejhorší hejno z jeho vrcholu a znovu ho inicializuje v prostoru. Výsledkem je nejmíň jedno hejno vyhlížející nové vrcholy. Anti-konvergence je globální operátor, protože se předpokládá, že každé hejno si je vědomo konvergenčního statusu ostatních hejn – informace je sdílená mezi hejny. Naopak aktualizace atraktoru hejna (*gBest*) je informace sdílená mezi částicemi ve stejném hejně.

Nabité nebo kvantové částice: Aby byly hejna schopné sledovat pohybující se vrchol, je nutné, aby byla diverzita mezi částicemi uvnitř hejna. To řeší nabité PSO (CPSO) obsahující kromě neutrálních i nabité částice. Diverzita částic je udržována odpory mezi nabitými částicemi. Míra této diverzity je rovna prostorové velikosti nabitého hejna. Avšak kvůli chaotické povaze oběžných drah velikost nabitého hejna kolísá. To může mít nevýhody v multi-CPSO, kde je žádoucí najít hejna kolem lokalizovaného optima. Navíc CPSO trpí kvadratickou složitostí, protože jsou počítány odpory mezi všemi páry částic v nabitém hejně. Proto je spíše používáno kvantové PSO (QPSO) obsahující kvantové hejno (místo nabitého), které je založeno na jednoduché randomizaci nabitých částic v oblasti kolem neutrálního hejna. Tato oblast je tvaru koule s poloměrem r_{cloud} s centrem v atraktoru hejna. Toto jednoduché pravidlo odpovídá rovnoměrnému pravděpodobnostnímu rozložení. Rychlost nabité částice je irelevantní a nabité částice nejsou vyloučeny od jiných nabitých částic nebo

přitahovány k žádným atraktorům. Avšak pokud díky svým náhodným pozicím kolem atraktoru naleznou dobrou pozici, může to být užitečná informace pro neutrální hejno. Multi-QPSO je založeno na tomto principu – jediná interakce, která se objeví mezi těmito hejny, je vyloučení, když hejna kolidují.

Po inicializaci algoritmus iteruje přes hlavní smyčku s pěti fázemi: test konvergence, test pro vyloučení, test pro změnu stavu, aktualizace částic a aktualizace atraktorů. Nejdříve testuje, zda všechny hejna konvergují, pokud ano, tak vyber nejhorší hejno pro randomizaci. Vyloučení testuje všechny páry hejn, zda jsou blízko sebe, a pokud ano, vyber horší pro randomizaci. Pokud je detekována změna prostředí, všechny *pBest* částic se znovu vyhodnotí, randomizace zruší a částice jsou znovu inicializovány nebo pouze aktualizovány. Každá částice obsahuje index *n*, který odkazuje na hejno, kam patří (částice n_i je částice *i* hejna *n*). Aktualizační pravidlo je definováno v závislosti na tom, zda je částice neutrální, nabitá nebo kvantová. Pozice neutrální je aktualizována jako standardní PSO. Pozice kvantové částice je dána náhodným číslem z rovnoměrného rozdělení z množiny bodů uvnitř *D*-dimenzionální koule s centrem v $gBest_n$. Pozice nabité částice je dána jako standardní PSO plus d_{ni}, které se získá z Coulombova zákona o působení nabitých částic na sebe.

Hejna interagují jak lokálně (vyloučení), tak globálně (anti-konvergence). Nové parametry jsou r_{excl}, r_{conv} a Q (nabité) nebo r_{cloud} (kvantové). Celé hejno je složeno buď z neutrálních (N) a nabitých (N^+) částic nebo z neutrálních (N) a kvantových (N^q) částic. Vhodná reprezentace konfigurace celého multi-hejna je $M(N+N^+)$ nebo $M(N+N^q)$, kde M je počet hejn v celém multi-hejně.

Tento přístup je velmi vhodný pro dynamické problémy, je velmi robustní a překonává předchozí přístupy. Slabý výkon se objevuje, jen když je počet vrcholů mnohem menší než počet hejn (řeší se to samo-adaptací počtu hejn). Celkově je lepší QPSO než CPSO. Na statických problémech QPSO a CPSO překonávají standardní PSO. Na dynamických problémech je QPSO výrazně lepší než standardní PSO s inicializací části populace po posunu vrcholů (nejlepší výsledky při znovu inicializaci 50 – 90 % populace).

3.5.19 Multi-kriteriální PSO

V této variantě je do PSO zahrnuta Pareto dominance, aby tato heuristika pomohla řešit problémy s několika účelovými funkcemi. Algoritmus používá vedlejší hejno částic (archiv částic), které pomáhá částicím z hlavního hejna určovat jejich vlastní let. Je zahrnut také speciální mutační operátor, který určuje rozpínavé možnosti algoritmu. Algoritmus je založený na uložení dosud nejlepších nedominovaných řešení do archivu, tato řešení leží na Pareto frontě (více o Pareto dominanci a Pareto frontě viz kapitola 2.3). Použitím mechanismu globální přitažlivosti, kterou poskytuje samotné PSO, a archivu předchozích nalezených nedominovaných vektorů se podnítí konvergence směrem ke globálně nedominovaným řešením [31].

Při inicializaci se pozice všech částic nastaví náhodně po prohledávacím prostoru a jejich rychlosti se nastaví na nulu. Každá částice se ohodnotí a pozice částic, které reprezentují nedominované vektory, se uloží do archivu. Vygeneruje se mřížka, která je složena z hyperkostek. Dimenze této mřížky odpovídá počtu účelových funkcí. Souřadnice této mřížky odpovídají fitness hodnotám jednotlivých částic v daných účelových funkcích. Paměť (*pBest*) všech částic je nastavena na aktuální pozici.

Během každé iterace se provádí následující kroky: Vypočítá se rychlost částic podle

$$v_i = w \cdot v_i + rand_1 \cdot (pbest_i - x_i) + rand_2 \cdot (arch_h - x_i)\,, \qquad (3.32)$$

kde w je nastavena na hodnotu 0.4. $rand_1$ a $rand_2$ jsou náhodná čísla z intervalu $\langle 0, 1 \rangle$. $pBest_i$ je nejlepší pozice částice i. $arch_h$ je hodnota z archivu z hyperkostky h. Hyperkostka h je vybrána následujícím způsobem: Těm hyperkostkám, které obsahují více než jednu částici, je přiřazena fitness, která je rovna výsledku dělení jakéhokoli čísla $x > 1$ (je použito $x = 10$) počtem částic, které tato hyperkostka obsahuje. Tím se sníží fitness té kostky, která obsahuje více částic (lze vidět druh sdílení fitness). Pak se aplikuje výběr pomocí ruletového kola používající tyto fitness hodnoty k výběru hyperkostky, ze které se vezme odpovídající částice. Když je hyperkostka vybrána, náhodně se vybere částice uvnitř ní.

Nová pozice se vypočítá stejně jako u klasického PSO. Pokud je nová pozice částice mimo danou ohraničenou oblast, je daná složka pozice nastavena na hraniční hodnotu a rychlost je vynásobena -1, tím částice v tom směru získá opačnou rychlost. Poté se vyhodnotí všechny částice. Pokud je nová pozice částice lepší než pozice v paměti (*pBest*), nová pozice se uloží do paměti místo té původní.

Poté se aktualizuje obsah archivu spolu s geografickou reprezentací částic uvnitř hyperkostek. Archiv se skládá ze dvou částí: kontrolér a mřížka. Kontrolér rozhoduje, zda by určité řešení mělo být přidáno do archivu nebo ne. Nedominované vektory nalezené v každé iteraci v hlavní populaci jsou porovnány každý s každým s ohledem na obsah archivu, který je na počátku prázdný. Pokud je archiv prázdný, aktuální řešení je akceptováno (vloženo do archivu). Pokud je nové řešení dominované částicí uvnitř archivu, je takové řešení automaticky vyřazeno. Jinak pokud žádná částice z archivu nedominuje vstupující částici, je taková částice uložena do archivu. Pokud jsou nějaká řešení v archivu, která jsou dominovaná novou vstupující částicí, je tato řešení z archivu odstraněna. Pokud archiv dosáhne své maximální kapacity, je vyvolána procedura adaptivní mřížky.

K produkci dobře distribuovaných Pareto front je použita variace adaptivní mřížky. Prostor účelových funkcí je rozdělen do oblastí (hyperkostek). Pokud leží jedinec, který je vkládaný do archivu, mimo současné hranice mřížky, pak mřížka musí být přepočítána a každá částice uvnitř musí být přemístěna. Hyperkostka má tolik složek, kolik je účelových funkcí. Každá hyperkostka může být interpretována jako geografická oblast, která obsahuje určitý počet jedinců. Hlavní výhodou adaptivní mřížky je její výpočetní náročnost (je nižší než u nik). Pokud by byla mřížka přepočítána každou

iteraci, je její časová složitost $O(N^2)$ (stejná jako u nik). Adaptivní mřížka je použita k rovnoměrnému rozdělení co největšího možného množství hyperkostek; k tomu je nezbytné poskytnout a získat určitou problémově závislou informaci.

Vysoká konvergenční rychlost PSO může být škodlivá v kontextu multikriteriální optimalizace, protože algoritmus může konvergovat k falešné Pareto frontě (ekvivalence k lokálnímu optimu v globální optimalizaci). Mutační operátor je aplikován na všechny částice na začátku prohledávání, poté se rapidně snižuje počet částic, které jsou tímto operátorem ovlivněny. Mutační operátor je aplikován nejen na částice hejna, ale také na rozsah v prostoru, do kterého jsou částice po zmutování umístěny. To ústí v pokrytí celého rozsahu každé proměnné na začátku prohledávání, pak se zužuje rozsah pokrytí v čase (použitím nelineární funkce). Tedy ze začátku je chování v algoritmu vysoce explorativní. Při stoupající iteraci vliv mutace klesá – klesá počet zasažených částic i rozsah, do kterého jsou tyto zmutované částice umístěny.

3.5.20 PSO s dynamickým sousedstvím

PSO s dynamickým sousedstvím byl vyvinut pro řešení více-kriteriálních problémů. PSO je upraven, aby lokalizoval Pareto frontu.

Více kritérií je rozděleno do dvou skupin – F_1 a F_2. F_1 je definována jako kritérium sousedství, F_2 jako optimalizační kritérium. Výběry F_1 a F_2 jsou libovolné. V každé iteraci každá částice definuje své sousedství výpočtem vzdáleností ke všem ostatním částicím a výběrem *M* nejbližších sousedů. V tomto případě je vzdálenost chápána jako rozdíl mezi fitness hodnotami pro první skupinu fitness funkcí. Jakmile bylo určeno sousedství, je mezi sousedy nalezena nejlepší lokální hodnota jako fitness hodnota druhé skupiny funkcí. Aktualizace globální nejlepší pozice je provedena, pouze když řešení dominuje současné *pBest* hodnotě. PSO parametry se nastaví na $c_1 = c_2 = 1.494$, $w \in \langle 0.5; 1.0 \rangle$ [32].

Rozšířená paměť pro uložení všech Pareto optimálních řešení v současné generaci byla navržena ke snížení výpočetního času.

3.5.21 Vektorově ohodnocené PSO

Vektorově ohodnocené PSO je založen na konceptu vektorově ohodnocených GA. K prohledávání prostoru jsou použity dvě nebo více hejn. Každé hejno je vyhodnocováno podle jedné účelové funkce a informace si mezi sebou předávají. Výsledky těchto informací z jiných hejn jsou použity k vedení trajektorie částic směrem k Pareto optimálním bodům. Rovnice pro úpravu rychlosti pro *M*-kriteriální problém je

$$v_i^{[j]}(t+1) = \chi^{[j]}\left(w^{[j]}v_i(t) + c_1 r_1(t)\left[pBest_i^{[j]}(t) - x_i(t)\right] + c_2 r_2(t)\left[gBest^{[s]}(t) - x_i(t)\right]\right), \quad (3.33)$$

kde j = 1, 2, …, M definuje číslo hejna, i = 1, 2, …, N odpovídá číslu částice, $pBest_i^{[j]}$ je nejlepší pozice nalezená částicí i hejna j a $gBest^{[s]}$ je nejlepší dosud nalezená pozice hejnem s. Pokud je použita kruhová topologie, pak s je nastaveno následovně: Pokud j = 1, pak $s = M$. Jinak pokud j = 2, 3, …, M, pak $s = j - 1$. Existuje i paralelní verze tohoto algoritmu [33].

3.5.22 Agentní paralelní PSO

Agentní paralelní PSO (APPSO) je založen na dvou typech agentů: koordinační agent (CA) a několik rojových agentů (SA). Hejno je rozděleno na více menších hejn, jedno menší hejno pro každého agenta. Rojoví agenti provádějí výpočty, koordinační agent řídí ostatní věci. Menší hejna buď pracují spolu na hledání jednoho globálního optima, nebo se rozdělí po prostoru pro nalezení různých globálních/lokálních optim (multimodální funkce) [34].

SA buď komunikují s CA, nebo spustí PSO svého hejna. CA řídí rozdělení částic mezi SA. SA nekomunikují sami se sebou, pouze s CA. CA řídí výměnnou strategii podle toho, který SA poskytuje a žádá své částice.

CA rozlišuje mezi třemi stavy: počáteční stav, nečinný stav a vyměňovací kolo. V počátečním stavu CA inicializuje systém SA, rozešle fitness funkci a nastavení parametrů. Poté se CA přepne do nečinného stavu. Uvnitř vyměňovacího kola CA koordinuje nabídky SA.

Poté, co CA rozešle detaily optimalizace (parametry, počet částic, oblast inicializace), každé SA spustí své PSO. Jakmile první SA dokončí výpočet po předem daném počtu iterací, CA zahájí vyměňovací kolo. Na základě vyměňovacích parametrů a počtu ohodnocení, které byly provedeny v poslední počítací fázi, CA určí pro každé SA, kolik částic musí být koupeno a prodáno. Poté, co SA spočítají seznam částic na prodej, CA pošle seznam všech dostupných částic všem SA a čeká na nabídky. Na základě nabídek CA rozdělí a rozešle dostupné částice.

To, jaké částice se koupí a prodají, ovlivní chování jednotlivých hejn. Pokud si hejna vymění své nejlepší částice, chovají se tato hejna jako jeden celek. Pokud je koupená částice lepší než nejlepší v hejně, dojde ke konvergenci směrem k této nové částici. Pokud se SA rozhodne zaměřit na určitou niku (výklenek v prostoru, shluk), prodá částici s nejdelší vzdáleností od středu tohoto shluku a koupí částici, která je blíže. Různé strategie výměny částic SA mohou řídit chování odpovídajících hejn.

Bylo dosaženo lepších výsledků v porovnání se standardním PSO.

3.5.23 Hierarchický duální PSO

Jde o variantu, kde se automaticky ladí parametry pro dosažení zvýšení výkonu. Je používán hierarchický duální PSO systém s vysoko úrovňovým PSO, který ladí parametry jiného PSO, jenž hledá optima funkce. Mezi tyto laděné parametry patří omezující faktor χ, akcelerační koeficienty c_1 a c_2 a velikost hejna [35].

Zkoumá se počet iterací, který algoritmu potřebuje k dosáhnutí určité chyby od optima, nejlepší dosažená hodnota funkce po určitém daném počtu iterací a celkový počet funkčních ohodnoceních. Protože se hledá optimum 4 parametrů, je vysokoúrovňové PSO (super PSO, sPSO) čtyř-dimenzionální. Každá částice super hejna definuje parametry PSO pro optimalizaci funkce (funkční PSO, fPSO). Rozsah prostorů jednotlivých parametrů jsou: Velikost hejna $N_f \in [20, 80]$, $\chi \in [0.4, 0.9]$, $c_1, c_2 \in [0.5, 2.5]$. sPSO používá standardní parametry: $N_s = 20$, $\chi = 0.729$, $c_1 = c_2 = 2.05$. Protože počáteční pozice částic ovlivňují výkon PSO, je tento vliv minimalizován tím, že se jednotlivé fPSO běhy nastavily na stejné počáteční pozice. Fitness hodnota částice v super hejně je vypočítána použitím průměrného výkonu fPSO. Tento výkon se vypočítá jako průměr několika běhů, kde v každém běhu je vynásobena hodnota fitness s velikostí hejna fPSO a s počtem iterací vykonaných k dosažení určité fitness (práh chyby). To všechno bere do úvahy výkon – kvalita řešení a výpočetní cena.

Negativem je vyšší náročnost algoritmu, protože je provedeno mnohonásobně více ohodnocení. Pozitivem je možnost pro určitou třídu fitness funkcí podobné povahy použít jednu množinu optimalizovaných PSO parametrů.

3.5.24 Kooperativní PSO

Algoritmus využívá několik hejn k optimalizaci různých komponent (dimenzí) vektoru řešení. Kooperativní chování těchto hejn značně zlepšuje výkon. Prohledávací prostor je explicitně rozdělený rozdělením vektorů řešení na menší vektory. Takto jsou navrženy dva nové algoritmy CPSO-S a CPSO-H [36].

V algoritmu CPSO-S je hejno *n*-dimenzionálních částic rozděleno na *n* hejn jedno-dimenzionálních částic, kde každé hejno optimalizuje jednu komponentu vektoru řešení. Vypočtení fitness funkce každé částice je provedeno tak, že každé hejno dosadí do funkce jen ten parametr, který zastupuje, za ostatní parametry jsou dosazeny konstanty. Tyto konstanty jsou *gBest* z odpovídajících hejn. To podporuje kooperaci mezi jednotlivými hejny, protože ony všechny přispívají k sestrojení vektoru použitého do fitness funkce. V čase je měněna jen jedna komponenta, a tedy mnoho kombinací různých částic z různých hejn je vytvořeno. To ústí v jemné prohledávání a ke zvýšení diverzity řešení.

Problém může nastat s korelovanými proměnnými. To se řeší hejny s více dimenzemi, které reprezentují právě korelované proměnné. Takový algoritmus se nazývá CPSO-S_k, kde vektor pozice je rozdělen na *k* částí (místo *n*).

Algoritmus CPSO-H_k kombinuje PSO a CPSO-S_k. Tyto dva algoritmy si po každém svém běhu vymění informace svých nejlepších řešení (kooperace). Funguje to tak, že jednu iteraci CPSO-S_k následuje jedna iterace PSO. Po jedné iteraci CPSO-S_k části (první polovina algoritmu CPSO-H_k) se pozicí nejlepšího řešení (skládá se z *gBest* jednotlivých hejn) přepíše náhodně vybraná částice v PSO

části (druhá polovina algoritmu CPSO-H_k). Následuje jedna iterace PSO části, jejímž výsledkem bude její nový *gBest* vektor, který bude rozdělen do několika menších vektorů (podle dimenzí odpovídajících hejn), které přepíšou pozice náhodně vybraných částic v odpovídajících hejnech. Všechny tyto náhodně vybrané částice nemohou být ty nejlepší v daných algoritmech. Pokud je mechanismus výměny informací příliš častý, je zpomalen pokrok algoritmu (všechny částice by měly pozici globálně nejlepšího – snížení diverzity). To lze řešit např. pomocí omezení počtu částic, které se můžou zúčastnit výměny informace (např. polovina částic populace).

3.5.25 Adaptivní PSO

Byl navržen adaptivní PSO (APSO), jenž je založen na strategii, která provádí změnu velikosti hejna, váhy *w* a velikosti sousedství za běhu [37]. Algoritmus je změněn přidáním indexu zlepšení pro částice, který se vypočítá následovně:

$$\delta(x_i) = \frac{f(x_i(t_0)) - f(x_i(t))}{f(x_i(t_0))} \tag{3.34}$$

kde $f(x_i(t))$ je fitness hodnota částice i v iteraci t. $\delta(x_i)$ se porovná s prahem, aby se určilo zlepšení částice i. Práh je na začátku definován jako $1 - f_{best}/f_{worst}$, poté je v průběhu upravován (pokud se generují a odebírají částice nebo pokud se mění velikost sousedství).

Změna velikosti hejna: Pokud se částice dost zlepšila a zároveň je to nejhorší částice ve svém sousedství, pak se tato částice odstraní. Naopak pokud se částice dost nezlepšila, ale je nelepší ve svém sousedství, generuje se nová částice.

Změna váhy *w*: Čím více se částice zlepšila, tím menší oblast tato částice musí prohledat. Pokud se částice nelepší, pak musí zvětšit svůj prohledávací prostor. Velikost prohledaného prostoru určuje právě váha *w*.

Změnu velikosti sousedství: Pokud je částice nejlepší ve svém sousedství, ale dost se nezlepšila, pak částice potřebuje více informací a velikost sousedství se musí zvětšit. Pokud se částice zlepšila uspokojivě, pak nepotřebuje moc sousedů a velikost sousedství se může zmenšit.

APSO poskytlo spolehlivější optimalizaci než standardní PSO.

3.6 Hybridizace PSO

Existuje několik hybridizací PSO. Hybridizace je označení pro proces kombinace několika podobných (či různých) algoritmů za účelem zlepšení výkonu celého algoritmu. Tohoto zlepšení je dosaženo využitím výhod jednotlivých přístupů. Hybridní PSO je PSO, které využívá technik známých z jiných algoritmů, především evolučních.

3.6.1 PSO s genetickým algoritmem

Tento algoritmus kombinuje výhody inteligence hejna z PSO a mechanismu přírodního výběru z genetických algoritmů (GA), aby se zvýšil počet vysoce ohodnocených jedinců a zároveň snížil počet nízce ohodnocených jedinců v každém iteračním kroku. Tedy nejen že je možné úspěšně měnit současnou prohledávací oblast pomocí *pBest* a *gBest*, ale také je možné skákat z jedné oblasti do druhé selekčním mechanismem, což ústí ve zvýšenou rychlost konvergence celého algoritmu.

Algoritmus zahrnuje aspekt klasického GA přístupu – reprodukční systém, který modifikuje pozici i rychlost náhodné vybraných částic, aby se zlepšil potenciál PSO dosáhnout optimum, podle rovnic

$$child_1(x) = p \cdot parent_1(x) + (1 - p) \cdot parent_2(x) \tag{3.35}$$

a

$$child_1(v) = (parent_1(v) + parent_2(v)) \frac{parent_1(v)}{parent_1(v) + parent_2(v)}, \tag{3.36}$$

rovnice pro $child_2$ jsou analogické, kde p je náhodné číslo z intervalu $\langle 0, 1 \rangle$, $parent_{1,2}(x)$ reprezentuje vektory pozic náhodně vybraných částic, $parent_{1,2}(v)$ jsou odpovídající vektory rychlosti každého rodiče a $child_{1,2}(x)$ a $child_{1,2}(v)$ jsou potomci rozmnožovacího procesu [38].

Může se použít také nahrazení jedinců s nízkými fitness jedinci s vysokými fitness podle hodnoty selekce S_T. Tím se udrží závislost na vysoce ohodnocených jedincích.

Další možností hybridizace PSO a GA je, že se v průběhu provádění po několika stovkách iterací tyto algoritmy vymění. Ukázalo se, že nejlepší výsledky má přechod z PSO na GA, protože PSO efektivněji prohledává celý prostor pro nalezení nejlepší oblasti a GA je efektivní v hledání optima, až když populace konvergovala do určité jedné oblasti.

3.6.2 PSO s evoluční strategií (s evolučním programováním)

Do PSO, jehož parametry jsou upravovány pomocí evoluční strategie (ES), je začleněna metoda turnajového výběru, který se používá v evolučním programování (EP) [39]. V tomto přístupu zůstávají aktualizační rovnice stejné, pouze se mění výběr částic. Fitness hodnota každé částice je porovnána s k jinými částicemi a zaznamená se bod pro každou částici s horším fitness. Populace je pak seřazena na základě těchto bodů. Současné pozice a rychlosti nejhorší poloviny hejna jsou nahrazeny pozicemi a rychlostmi nejlepší poloviny. *pBest* každé částice hejna (nejlepší i nejhorší poloviny) zůstávají nezměněné. Tedy v každé iteraci je polovina jedinců přesunuta na pozice v prostoru, které jsou blíže k optimálnímu řešení než předchozí pozice (přesto si ponechají své *pBest*). To klade důraz na mechanismus exploitativního hledání oproti klasickému PSO. Toto by mělo pomoci nalézt optimum více důsledně.

Navíc k selekčnímu mechanismu jsou přidány samo-adaptační možnosti hejna změnou množiny strategických parametrů (vah). Obecné schéma zahrnuje replikaci (každá částice je replikována *r*-krát), mutaci (každá částice zmutuje své váhy), reprodukci (každá zmutovaná částice generuje potomky na základě pravidla pohybu částice), vyhodnocení (každý potomek má fitness hodnotu) a selekci (stochastický turnaj pro výběr nejlepší částice, která přežije do další generace). Rovnice rychlosti je tedy upravena na

$$v_i(t) = w_{i1}^* \cdot v_i(t-1) + w_{i2}^* \cdot rand_1 \cdot \left(pBest_i - x_i(t-1)\right) + w_{i3}^* \cdot rand_2 \cdot (gBest^* - x_i(t-1)) , \quad (3.37)$$

kde $w_{ik}^* = w_{ik} + \tau \cdot rand$ a *rand* je Gaussovo rozložení ($N(0, 1)$). *gBest* je taky zmutovaná rovnicí $gBest^* = gBest + \tau' \cdot rand$, kde τ a τ' jsou učící parametry, které mohou být buď neměnné, nebo se mohou dynamicky měnit. Z hlediska výkonu měl tento algoritmus (EPSO) převahu nad klasickým PSO.

3.6.3 PSO s diferenční evolucí

V této hybridizaci (DEPSO) jsou použity prvky z diferenční evoluce (DE). Rovnice pro výpočet rychlosti a pozice z původní PSO jsou vykonány v liché iteraci; v sudé iteraci je vykonán operátor mutace algoritmu DE na *pBest* částice, kde je bod $T_i = pBest_i$ pro *d*-tou dimenzi odvozen následovně:

Pokud $rand < CR$ nebo $d = k$, pak $T_{id} = gBest_{gd} + \delta_{2d}$, kde *k* je náhodná celočíselná hodnota z intervalu $\langle 1, n \rangle$, což zajistí, že mutace bude alespoň v jedné dimenzi, $CR \leq 1$ je konstanta křížení a $\delta_2 = (\Sigma_1^N \Delta)/2$, kde Δ je rozdíl *pBest* dvou náhodně vybraných částic. Pokud je T_i lepší než $pBest_i$, pak ho T_i nahradí. Poté, co je DE operátor aplikován na všechny částice, vybere se *gBest* standardně jako nejlepší ze všech *pBest*. To poskytne možnost sociálního učení, která může urychlit konvergenci [40].

V jiných případech je DE použita pro řešení problému nastavení vah (w, c_1, c_2). Nejprve jsou váhy vygenerovány náhodně, poté se v průběhu algoritmu střídá fáze PSO a fáze DE, která aplikuje své operátory na váhy. Hodnoty vah s nejlepšími výsledky jsou uchovávány [16].

Ve většině případů byla DEPSO lepší než samotné PSO nebo samotná DE.

3.6.4 PSO s lokálním Pattern Search

Tradiční PSO je vylepšen přidáním Local Pattern Search (PS) – PSPSO – za účelem urychlení konvergence. PS má schopnost efektivního lokálního prohledávání v hranicemi ohraničených optimalizačních problémech. PS je jednoduchá přímá metoda. Klasický PS algoritmus pohybuje kandidátním řešením podél os (nebo jiných uživatelsky definovaných množin) s určitým krokem, aby zlepšil existující řešení „greedy“ způsobem. Velikost kroku Δ je zmenšována (typicky na polovinu), pokud není v žádném směru nalezeno zlepšení. PSO je použito jako globální prohledávací metoda, PS jako lokální prohledávací metoda [41].

PS je provedeno po aktualizaci pozice a rychlosti, aby vylepšil pozici částice. Krok je inicializován na Δ_0. Dokud není splněna ukončovací podmínka PS (max. počet iterací), vypočte se pro každou dimenzi funkční hodnota řešení $x_{ij} + \Delta$ a $x_{ij} - \Delta$. Pokud je toto nové řešení lepší, nahradí původní řešení x_{ij}. Pokud není dosaženo zlepšení pro žádnou dimenzi, pak $\Delta_{k+1} = \Delta_k / 2$ (zkrácení kroku na polovinu), a pokud $\Delta_{k+1} = \Delta_{tolerance}$, pak $\Delta_{k+1} = \Delta_0$ (resetování kroku).

Experimenty ukázaly, že PS zlepšilo výkon PSO algoritmu. Až na pár výjimek bylo PSPSO lepší než samotné PSO nebo PS.

3.6.5 Binární PSO s imunitním klonovým algoritmem

Tento přístup zavádí novou strategii pro aktualizaci vektoru pozice v binárním PSO (BPSO), který se dále kombinuje s klonováním z imunitních systémů pro zlepšení schopnosti optimalizace [42].

Umělý imunitní systém je inspirován přírodním imunitním systémem, díky kterému jsou lidé chráněni (použitím protilátek) před vniknutím škodlivých látek (antigenů). Klonový selekční přistup je typ adaptivního imunitního systému, který je směřován proti specifickému antigenu a který obsahuje dva hlavní typy lymfocytů – B-buňky (bílé krvinky odpovědné za produkci protilátek) a T-buňky (bílé krvinky odpovědné za detekci antigenů). Tyto dva hlavní typy jsou zahrnuty v procesu identifikace a odstranění antigenu. Základní myšlenka tohoto přístupu spočívá v klonování aktivovaných B-buněk, které jsou vhodné na určitý antigen (dokážou jej rozpoznat). Klony jsou zmutovány za účelem zlepšení vhodnosti na antigen, nejlepší buňky jsou udržovány v paměti a buňky, které nejsou schopny rozeznat antigen, jsou odstraněny. Pro udržení diverzity jsou generovány náhodní jedinci.

Binární PSO s imunitním klonovým algoritmem (NPSOCLA) kombinuje upravený BPSO, klonový selekční algoritmus a podmnožinu náhodné populace s cílem dosažení rovnováhy mezi explorací a exploitací. Pozice je aktualizována bez použití rychlosti; krok směrem k *pBest* a *gBest* je řízen pomocí logických operátorů. Pomocí operátoru *xor* se vypočítá vzdálenost mezi aktuální pozicí a *pBest*, invertuje se volitelný (parametrem daný) počet těch bitů aktuální pozice, ve kterých se aktuální pozice a *pBest* liší (výsledek *xor* je 1) (pokud se invertují všechny takové bity, částice se dostane na pozici *pBest*), a s určitou pravděpodobností se zmutuje jeden bit; tím se získá jeden mezivýsledek. Druhý mezivýsledek se získá podle stejného postupu, jen je *pBest* nahrazen *gBest*. Nová pozice je dána: první mezivýsledek *xor* druhý mezivýsledek.

Pokud se *gBest* nezmění v určitém počtu iterací, provede se klonový selekční algoritmus. Vyberou se z populace jedinci s největší *pBest*. Každé takové *pBest* se několikrát naklonuje, počet klonů je přímo úměrný fitness *pBest*. Každý klon je zmutován nepřímo úměrně k jeho fitness. Nakonec se tyto výsledné klony ohodnotí, uspořádají a vyberou se ty nejlepší do další generace (bez redundance). Úplně nová populace se tedy skládá z nejlepších jedinců těchto tří populací: ze staré PSO populace, nové náhodně vygenerované populace a klonové selekční populace.

Na všech testovaných funkcích byl NPSOCLA lepší než PSO.

3.6.6 PSO s diferenční evolucí a pravděpodobnostním algoritmem EDA

Tato hybridizace v sobě zahrnuje více-rojový systém kobylkových hejn (kapitola 3.5.16) a prvky diferenční evoluce (DE) a pravděpodobnostního algoritmu EDA (UMDA), ve kterém jsou jedinci generováni podle odhadnutého pravděpodobnostního rozložení (UMDA-PSO) [43]. Více-rojový systém využívá různá hejna k rovnováze (kompromisu) mezi explorací a exploitací. Každé hejno se zaměří na jednu oblast, která je pořádně prohledána (exploitace), a tyto hejna jsou umístěna po celém prohledávacím prostoru. To, jak jsou hejna přemísťována po prostoru (explorace), závisí na specifických metodách pro rozmanitost. Ve více-rojových systémech je dosaženo lepších výsledků, pokud počáteční pozice a počáteční rychlosti nejsou náhodné.

V kobylkovém hejnu je prohledávání řízeno „sežráním a přestěhováním“. V této strategii poté, co hejno „sežere“ relativně malou oblast prostoru, aby nalezlo lokální optimum, jsou vypuštěni průzkumníci, aby nalezli novou slibnou oblast k „přestěhování“. Právě proces prozkoumávání (náhodné prohledávání) a výběr počáteční rychlosti (směřování pryč od předchozího optima) jsou klíčové části, které hybridní systém zlepšuje. Náhodné prohledávání je nahrazeno technikou UMDA, která provádí optimalizaci během prohledávání. UMDA, která obsahuje jen navzájem nezávislé proměnné, je nejjednodušší varianta algoritmu EDA. K lepšímu využití gradientní informace získané pomocí UMDA jsou použity diferenční vektory z DE k určení počátečních rychlostí.

Počáteční vektor rychlosti je vypočítán vztahem $v_0 = F(x_a - x_b)$, , kde x_a, x_b jsou náhodně vybraná řešení z množiny zvolených UMDA řešení (x_a má lepší fitness než x_b) a F je škálovací faktor z DE z intervalu $(0, 1\rangle$. UMDA je standardní implementace používající Gaussovy funkce hustoty. V každé iteraci je odhadnuta střední hodnota a standardní odchylka pro každou proměnnou. Tato střední hodnota a odchylka jsou použity ke generovaní inverzní Gausovy funkce, která je pak vzorkována, aby úplně nahradila předchozí populaci.

Celý algoritmus se skládá z několika cyklů. Každý cyklus je rozdělen do dvou fází: zesílení a obměna. Každá z těchto fází obsahuje UMDA k nalezení počátečních pozice, DE k nastavení počátečních rychlostí a PSO prohledávání s těmito nastaveními. Každá fáze začíná kolem optima nalezeného předchozí fází. Rozdíl mezi fázemi zesílení a obměny je rozsah dolní a horní hranice oblasti, na které je UMDA spuštěna. Tyto hranice určují interval kolem předchozího optima, který bude prohledán. Hranice omezuje pouze generování UMDA řešení, částice z PSO se mohou normálně za tyto hranice dostat. Fáze zesílení má tuto hranici užší, fáze obměny širší. Tyto hranice jsou na začátku celého algoritmu rovny prohledávacímu prostoru, postupem cyklů jsou hranice čím dál menší, tedy prohledávací oblasti v pozdějších cyklech jsou více omezené – to vede k postupnému jemnějšímu hledání.

UMDA-PSO v naprosté většině případů dosahoval lepší výsledky než samotný locust swarm PSO, klasické PSO nebo UMDA.

3.6.7 PSO s umělou včelí kolonií

Pro řešení dynamických problémů byla navržena hybridizace PSABC, která spojuje dva přístupy: PSO a umělou včelí kolonii (ABC) [44]. ABC je jeden z včelích algoritmů, který je založen na populaci, je tak velice blízký PSO. Kolonie obsahuje tři druhy včel: dělnice, vyčkávací včely a průzkumníci. Algoritmus vychází z chování včel. Řešení je reprezentováno pozicí zdroje potravy a množství nektaru ve zdroji potravy odpovídá hodnotě fitness funkce. Počet dělnic je roven počtu řešení. Každá včela je charakterizována svou pozicí v prohledávacím prostoru řešení a své nejlepší řešení si ukládá. Dělnice se pohybují v prostoru. Poté, co se dělnice dostanou na nové pozice, jsou na tyto pozice přemístěny vyčkávací včely (více vyčkávacích včel jde přímo úměrně na lepší řešení). Vyčkávací včely prohledávají okolí kolem řešení dělnice, a pokud se objeví lepší řešení, je na něj dělnice přemístěna (dělnice se dívá jen na řešení svých vyčkávacích včel). Průzkumníci náhodně prohledávají prostor. Pokud dělnice po určitou dobu nezlepší své řešení, stává se z ní průzkumník [45].

V PSABC ABC i PSO pracují se stejnou populací. V každém cyklu po vypočítání fitness všech včel jsou označeny dělnice, které zlepší své řešení pomocí PSO. Kolem dělnic jsou umístěny vyčkávací včely. Pokud jsou nalezeny lepší pozice, jsou na ně dělnice přemístěny. Opět se na tyto pozice dělnic aplikuje PSO. Průzkumníci neustále hledají nová náhodná řešení. Ukládá se nejlepší řešení. Cyklus se opakuje. Tím se zvyšuje schopnost prohledávání.

Na dynamických funkcích v naprosté většině případů dosahoval PSABC nejlepších výsledků. Porovnáván byl s algoritmy adaptivní mQPSO, adaptivní CPSO, mQPSO (kapitola 3.5.18) a RPSO (s odpory).

4 Nový algoritmus pro dynamické problémy AHPSO

V této kapitole je podrobně popsán nově navržený algoritmus AHPSO. Nejprve je naznačeno, z čeho algoritmus vychází, poté jsou rozebrány jednotlivé prvky samotného algoritmu i motivace, proč byly tyto prvky aplikovány. Jsou zmíněny i přístupy a algoritmy, kterými byly části navrženého algoritmu inspirovány. Algoritmus byl vytvořen pro dynamické problémy, proto je popsáno i chování dynamického prostředí.

4.1 Dynamické problémy

Jak bylo naznačeno v kapitole 2.3, problémy a jejich odpovídající účelové funkce lze rozdělit mimo jiné i podle jejich stability. Z tohoto úhlu pohledu lze problémy dělit na statické a dynamické. Jak z názvů těchto tříd problémů vyplývá, statické problémy jsou ty, které nemění hodnoty účelové funkce v čase, a dynamické jsou logicky naopak ty, jejichž extrémy se v čase pohybují.

Lze říci, že dynamické problémy jsou typicky charakterizovány měnící se účelovou funkcí. Obecně se účelová funkce mění libovolně v čase, tzn., že ve dvou různých časových okamžicích je funkční hodnota totožných proměnných různá, tedy pokud $t_1 \neq t_2$, pak neplatí vztah $f_{t1}(x_1, x_2, \ldots, x_n) = f_{t2}(x_1, x_2, \ldots, x_n)$, protože f_{t1}, f_{t2} mohou představovat jiné funkce.

Často se stává, že účelová funkce se nemění libovolně náhodně, ale v pravidelných intervalech, kdy je po určitou neznámou dobu účelová funkce neměnná. Pokud je změna účelové funkce extrémně častá, je téměř zbytečné používat optimalizační algoritmy. V takovém případě se optimalizace pak rovná náhodnému prohledávání, které by bylo samotné vhodnější. Při přiměřené frekvenci změny funkce má algoritmus relativně čas opět najít globální extrém.

Optimalizační algoritmy pro dynamické problémy musí být pružné a přizpůsobivé. Především musí být schopné detekovat jinou účelovou funkci a reagovat v co nejkratším čase na tuto změnu prostředí. Detekce změny se většinou provádí pomocí znovu ohodnocení již známého bodu (případně více bodů). Pokud tento bod nabývá jiné funkční hodnoty, dá se úspěšně předpokládat změna účelové funkce. Reakce na změnu bývá různá v závislosti na použitém algoritmu. Obecně musí mít algoritmus povědomí o prohledávaném prostředí. Pokud by algoritmus tento alespoň minimální přehled neměl, rovnala by se optimalizace obyčejnému restartu algoritmu po každé změně funkce.

4.2 PSO v dynamických problémech

V dynamických problémech má PSO dva zásadní problémy. Tím prvním je ten, který je společný pro většinu optimalizačních algoritmů a byl již zmíněn v předchozí kapitole: Pokud se změní problém, informace uložená v paměti v podobě *gBest* nemusí být pravdivá, a může tedy být zavádějící. Jedná se o zastaralou paměť částic. Druhým problémem je ten, že pokud hejno konverguje, pak se hejno smršťuje v míře určené omezujícím faktorem, vahou setrvačnosti nebo akceleračními koeficienty a lokálním prostředím kolem optima. Tedy v PSO se s vyšším počtem iterací vytrácí diverzita.

Např. pokud se optimum posune v prostoru jen o kousek v rámci sousedství, je optimalizace efektivní. Naopak pokud je optimum posunuto významně daleko od hejna, ztráta rychlosti částic znemožní jeho sledování a hejno bude kmitat kolem špatných atraktorů.

Aby se PSO dokázal adaptovat na optimální výsledky v dynamických problémech, je potřeba, aby algoritmus znal dobu změny v prostředí. Tu většinou nezná, a tak se změna detekuje – opět se vypočítá účelová funkce bodu, který je uložen v paměti. Problém zastaralé paměti je obvykle řešen buď nastavením *pbest* na současnou pozici (smazání paměti), nebo znovu vyhodnocením *pbest* a nechat v *pbest* lepší hodnotu (buď *pbest*, nebo současnou pozici).

Ukázalo se, že PSO může sledovat pomalu se měnící optimum beze změny. Ovšem není úplně jasné, o jakou frekvenci změn se jedná.

Dalším možným řešením je změna váhy ω na náhodné číslo z intervalu $\langle 0.5, 1.0 \rangle$. To vychází z úvahy, že pokud se sleduje jedno dynamické optimum, nelze předpovědět, zda bude v daném čase vhodnější explorace nebo exploitace.

V následujících kapitolách jsou naznačeny techniky v řešení dynamických problému PSO. Tyto techniky jsou shrnuty v [7] a [30].

4.2.1 Strategie pro zvýšení diverzity

Dále lze dynamické problémy řešit zvýšením diverzity, např. randomizací celé nebo části populace. Buď jako odpověď na změnu funkce, nebo v pravidelných intervalech. Problémem je, že randomizace implikuje ztrátu dosud sesbíraných informací a že je těžké odhadnout správnou míru randomizace – moc velká připomíná restart, moc malá neřeší konvergenci.

Diverzitu lze udržovat za běhu zavedením odporů. Např. jejich využití v prostorově rozšířených částicích a následné řešení jejich kolizí, aby se vyhnulo konvergenci. Nebo použití odporů, aby se udržely částice pryč od už detekovaného optima [27].

Jiný přístup zavádí nabité částice do hejna. Tyto částice se navzájem odpuzují a krouží kolem jádra neutrálních částic. Neutrální částice lze považovat za klasické konvergenční hejno. Oběžné dráhy nabitých částic jsou spíše chaotické než eliptické. Právě hejno nabitých částic udržuje populační diverzitu, přinejmenším uvnitř prostorového rozsahu oběžných drah nabitých částic. Díky

tomu je změna funkce ihned automaticky zaznamenána a hejno se dokáže adaptovat. Hejno normálních částic mezitím pokračuje v detailním prohledávání sousedství optima. To dobře funguje v unimodálních problémech [46].

Diverzitu lze zvýšit i pomocí použití mřížky jako lokálního sousedství, což redukuje tlak pohybovat se směrem ke globálnímu optimu [47], nebo použití hierarchické struktury sousedství (naznačená v kapitole 3.4.2). Hierarchická struktura sousedství je vhodná na unimodální dynamické problémy. V závislosti na dosud nejlepších nalezených pozicích se částice pohybuje v hierarchii nahoru a dolů. Dobré částice vysoko v hierarchii mají větší vliv na hejno [48].

Diverzita se udržuje i pomocí zavedení úprav do rovnic rychlosti a pozice [16][49].

4.2.2 Více-rojové metody

Výše zmíněné strategie (restart, zvýšení diverzity) jsou méně úspěšné v situacích, kdy se mění nejen pozice vrcholu, ale i jeho maximum. Změnou maxima se může stát nejvyšším vrcholem ten, který leží v úplně jiné oblasti prohledávajícího prostoru. Následkem tedy je velká změna v pozici globálního optima. K tomu může docházet v dynamických multimodálních prostorech. Lokální extrém se může změnit na globální a naopak. V těchto situacích se výhodně jeví více-populační přístupy. Pokud se hejna ve více-rojovém modelu umístí na odlišné vrcholy, pak bude se změnou sub-optimálního vrcholu na optimální moct hejno ihned začít se znovu optimalizací.

Na více-rojový přístup lze nahlížet i z pohledu sousedství (viz kapitola 3.4). Každé hejno představuje lokální sousedství, kde částice uvnitř tohoto hejna komunikují pouze se svými sousedy.

Více-rojovým modelem je například varianta, která je podrobněji popsána v kapitole 3.5.18, využívající vylučovací princip za účelem zabránění soupeření více hejn o jeden stejný vrchol a anti-konvergenci k získání diverzity multi-hejna jako celku.

Jiný takový model dynamicky mění počet a velikost rojů uspořádáním částic do tzv. druhů (původně navržené pro hledání optima v multidimenzionálních problémech). Hejno je rozděleno na několik menších hejn na základě jejich podobnosti. Každý druh se shromáždí kolem dominantní částice [50].

Více-rojové modely se používají i na multimodální funkce – např. mechanismus nik v PSO. Z jednoho velkého hejna se vytvoří více menších hejn pomocí sledování fitness hodnot jednotlivých částic. Algoritmus využívá mimo jiné i PSO s garantovanou konvergencí (viz kapitola 3.5.11). Mechanismus nik v PSO je úspěšný na statické multimodální problémy, ale neumí se moc přizpůsobit dynamickým problémům [51]

V jiném přístupu se zavádí pojmy rodič a potomek. V tomto více-rojovém přístupu je jedno hejno rodič a zbylá hejna jsou potomci. Rodič slouží k prozkoumání prohledávacího prostoru (explorace) a potomci k podrobnému prozkoumání slibných oblastí (exploitace), které rodič nalezl. K udržení diverzity uvnitř potomka je použito náhodné lokální prohledávání [52].

4.2.3 Hybridizace

Kromě více-rojových modelů byly na dynamické problémy použity různé hybridizace PSO s jiným algoritmem, např. s umělou včelí kolonií (kapitola 3.6.7) nebo s fuzzy shlukováním [53].

4.2.4 Noisy fitness funkce

Speciální podkategorií dynamických problémů jsou tzv. noisy fitness funkce obsahující šum. Na tyto funkce se naráží v problémech reálného světa. V těchto problémech (na rozdíl od výše popsaných) zůstává fitness funkce stejná, ale pokud PSO objeví stejnou pozici více než jednou, fitness hodnota se může lišit. Bylo zjištěno, že PSO zůstává efektivní i v přítomnosti šumu, dokonce je PSO schopné se vyhnout zachycení v lokálním optimu.

Byla také navržena šumu odolná varianta PSO, která spočívá v tom, že vícenásobné vyhodnocení stejného řešení je agregováno (sloučeno) do lepšího vyhodnocení fitness tohoto řešení [7].

4.3 Výchozí algoritmus mQPSO

Nově navržený algoritmus pro řešení dynamických problémů využívá více-rojového přístupu, který byl poprvé využit v algoritmech mQPSO nebo mCPSO popsaných v kapitole 3.5.18. Více-rojový model byl zvolen z důvodu řešení obecných dynamických problémů, které mohou obsahovat několik lokálních extrémů. Výhoda tohoto přístupu tkví ve zmapování a obsazení více extrémů současně, proto trvá méně času opět nalézt globální extrém po změně účelové funkce.

V dynamických funkcích obsahujících více vrcholů (lokálních nebo globálních) je dynamika vyjádřena malou změnou pozice, šířky a výšky vrcholů. Pokud dojde k situaci, kdy se změní výšky několika vrcholů natolik, že se extrém objeví v úplně jiné části prostředí, než byl předtím, jedno hejno má jen malou možnost se k tomuto novému optimu rychle dostat. Více rojů má větší šanci, že rychleji najde nové optimum, protože ty lokální extrémy, které mají nejvyšší pravděpodobnost, že budou nové globální extrémy, jsou obsazeny právě těmito jinými roji.

Více-rojový přístup je tedy založen na tom, že hejna náhodně prolétávají prohledávacím prostorem; pokud hejno nalezne volný vrchol, podrobně prohledá jeho okolí. Pokud je vrchol již obsazen jiným hejnem, ostatní hejna putují po prohledávacím prostoru dále a hledají další neobsazené vrcholy. Toho je dosaženo mechanismem exkluze (vyloučení). Pokud jsou dvě hejna v prohledávacím prostoru moc blízko sebe, dá se předpokládat, že prohledávají stejný vrchol, a to horší hejno s horším atraktorem je opět náhodně rozmístěno do prostoru (znovu inicializováno). Tedy nezáleží na tom, které hejno bylo u vrcholu dříve a které později.

Může dojít k situaci, že v prostoru je více vrcholů než hejn. V takovém případě není vyloučeno, že všechna hejna se shluknou kolem různých lokálních extrémů a globální extrém zůstane

nepokrytý. To je nežádoucí chování, a proto byla zavedena anti-konvergence. Ta funguje tak, že pokud je více vrcholů v prohledávaném prostředí než hejn a pokud všechna hejna nalezla svůj vlastní vrchol, který prohledávají, a konvergují, tak je hejnu, které prohledává nejhorší vrchol (oblast), přikázáno, aby se znovu inicializovalo v celém prohledávacím prostoru. Tím se otvírá možnost obsadit neobsazený globální extrém.

Pokud se při změně funkce extrém neobjeví na úplně jiném místě v prostoru, ale jen se posune o kousek od své předchozí pozice, je nutné, aby bylo hejno prohledávající tento vrchol schopné reagovat na tento posun. Pokud by byly částice tohoto hejna příliš blízko sebe, hejno by se hůře vypořádalo s jakoukoli dynamickou změnou. Proto byla v tomto přístupu zavedena technika pro udržení diverzity mezi částicemi uvnitř hejna. Tato technika spočívá v tom, že kromě obyčejných částic se v hejně vyskytují i nabité nebo (nověji) kvantové částice. Tyto částice se náhodně vyskytují v určité vzdálenosti od atraktoru, nemají rychlost a nejsou atraktorem přitahovány. V každé iteraci se tyto částice náhodně přemístí, ovšem stále v určité vzdálenosti od atraktoru, aby byly schopné detekovat posunutý extrém [30][54].

Stěžejní prvky více-rojového algoritmu mQPSO jsou vyloučení, anti-konvergence a kvantové částice. Nově navržený algoritmus pro dynamické problémy vychází právě z tohoto algoritmu mQPSO z výše popsaných důvodů.

4.4 Navržený algoritmus AHPSO

Nově navržený algoritmus pro dynamické problémy je nazván Adaptivní hybridní PSO (AHPSO) a vychází z algoritmu mQPSO. Využívá základ tohoto algoritmu, ale v mnohých věcech se liší. mQPSO je vysoce neefektivní v tom ohledu, že někdy zbytečně ohodnocuje neperspektivní body v prostoru. Tím se zmenšuje příležitost ohodnotit relevantní body blízko extrému, a tedy není čas přesně nalézt globální extrém kvůli (protože jde o dynamický problém).

Ze čtyř hlavních pilířů mQPSO navržený algoritmus využívá plně jen více hejn a techniku vyloučení; anti-konvergence je využita v upravené verzi a kvantové částice jsou vynechány úplně – pro udržení diverzity jsou použity jiné prvky.

V následujících kapitolách jsou popsány prvky, které algoritmus AHPSO obsahuje, a důvody jejich zavedení. Jedná se o prvky, na které lze pohlížet jako vnitřní a vnější interakce aktivních členů algoritmu. Vnitřní probíhají na úrovni částic, kde částice spolu komunikují uvnitř hejna (krok algoritmu (kapitola 4.4.9), přesunutí nezlepšujících se částic (4.4.6), přesunutí částic po změně funkce (4.4.10)); vnější interakce probíhají na úrovni hejn, kde si hejna mezi sebou předávají informace (vyloučení (4.4.1), anti-konvergence (4.4.2), přidávání a ubírání hejn (4.4.2 a 4.4.4), uspání hejn (4.4.7), určení fáze hejn – PSO nebo SOMA (4.4.8)).

Byla snaha nastavit rovnováhu mezi explorací a exploitací, mezi diverzitou a konvergencí (viz kapitola 3.2.1). Diverzitu je potřeba udržovat mezi částicemi (přesun částic po změně funkce

(4.4.10)) i mezi hejny (exkluze (4.4.1), anti-konvergence (4.4.2)). Konvergence je zajištěna samotným tradičním PSO s omezujícím faktorem a algoritmem SOMA. Tím, že je v případě potřeby lépe pokrytý prostor (zvyšování hejn (4.4.2), prohledávací částice (4.4.3)), je splněn požadavek na exploraci. Přítomnost více hejn a jejich schopnost lokálního prohledávání, především u hejna kolem globálního extrému (změna pozice nelepšících se částic blíže k nejlepšímu (4.4.6 a 4.4.8)), napomáhají exploitaci.

4.4.1 Vyloučení (exkluze)

Jak již bylo několikrát zmíněno, vyloučení se používá k tomu, aby se více hejn neshluklo kolem stejného vrcholu (extrému, optima), ale aby každé hejno prohledávalo jinou oblast. V mQPSO pokud se dvě hejna dostanou moc blízko sebe, pak je hejno s horším *gBest* znovu inicializováno v prostoru. Dvě hejna jsou blízko sebe, pokud je vzdálenost jejich *gBest* menší než poloměr vyloučení r_{excl}.

V mQPSO je poloměr vyloučení odvozen vztahem

$$r_{excl} = \frac{1}{2}\frac{(maxCoord - minCoord)}{p^{\frac{1}{D}}}, \tag{4.1}$$

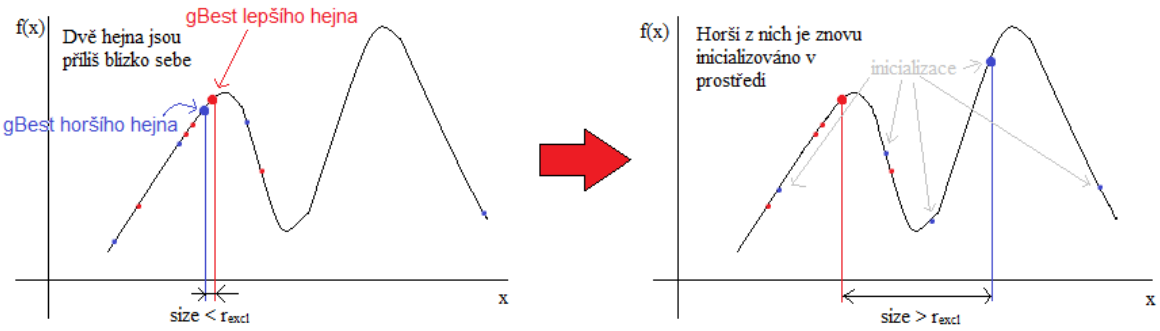
kde *minCoord a maxCoord* jsou hranice prohledávacího prostoru, *p* je počet vrcholů v prostoru a *D* je počet dimenzí problému. Protože v rovnici (4.1) je použitý parametr počtu vrcholů, potřebuje algoritmus mQPSO v této podobě externí informaci o zkoumaném problému.

Vzdálenost dvou hejn se počítá podle vztahu

$$dist_{kl} = \sqrt{\sum_{d=1}^{D}(x_{kd} - x_{ld})^2} \tag{4.2}$$

kde *k, l* jsou hejna a *D* je počet dimenzí problému.

Aplikace vyloučení na 2 hejna v 1D prostoru je zobrazena na obrázku 4.1.



Obrázek 4.1 Zobrazení použití vyloučení v 1D prostoru

V AHPSO je použita stejná rovnice pro výpočet poloměru exkluze, jen parametr počtu vrcholů je nahrazen parametrem pravděpodobného počtu vrcholů (*probNumOfPeaks*). Ten se

v průběhu optimalizace mění většinou podle počtu hejn (viz kapitoly 4.4.2, 4.4.4 a 4.4.11). Tímto zobecněním není dopředu potřeba znát počet vrcholů a je vhodnější pro reálné problémy, kde se častokrát neví, kolik je tam lokálních nebo globálních extrémů.

4.4.2 Anti-konvergence a zvyšování počtu hejn

Anti-konvergence je prospěšná, pokud má funkce více vrcholů než je počet hejn. Pokud je počet hejn menší než počet vrcholů a všechna hejna konvergují k nějakému (z důvodu vyloučení různému) extrému, je potřeba, aby jedno hejno stále procházelo prostor. Tím je možné zachytit vrchol, který se po změně stane maximem a který zároveň není obsazen žádným hejnem.

V mQPSO hejno konverguje, pokud je jeho velikost menší než r_{conv}. Přitom platí, že $r_{conv} \leq r_{excl}$, protože v opačném případě by hejno nikdy nekonvergovalo (nestihlo by to). Velikost hejna je dána největší vzdáleností mezi jakýmikoli dvěma částicemi tohoto hejna a vypočítá se podle vztahu

$$size = max_{ij}\sqrt{\sum_{d=1}^{D}\left(x_{id} - x_{jd}\right)^2}\,, \tag{4.3}$$

kde *i, j* jsou částice hejna a *D* je počet dimenzí problému. Pokud všechna hejna konvergují, vybere se to nejhorší a to se znovu inicializuje v prostoru. Nejhorší hejno je takové, jehož *gBest* je nejhorší.

Přístup v mQPSO není dostatečný, protože při větším počtu vrcholů je prohledávání celého prostoru jedním hejnem pomalé. Znovu inicializované hejno může při velkém počtu vrcholů lehce konvergovat k neglobálnímu extrému. V takovém případě se musí čekat, než hejno zkonverguje a to nejhorší je znovu inicializované. Tato situace se může několikrát opakovat.

Z toho důvodu je v AHPSO anti-konvergence trochu upravena. Pokud všechna hejna konvergují a pokud je počet hejn menší než maximální počet hejn (*maxNumOfSwarms*), přidá se několik (*numAddSwarms*) hejn navíc. Pokud je v prohledávacím prostoru už maximální počet hejn (*maxNumOfSwarms*), provede se stejná věc jako v mQPSO – znovu se inicializuje nejhorší hejno.

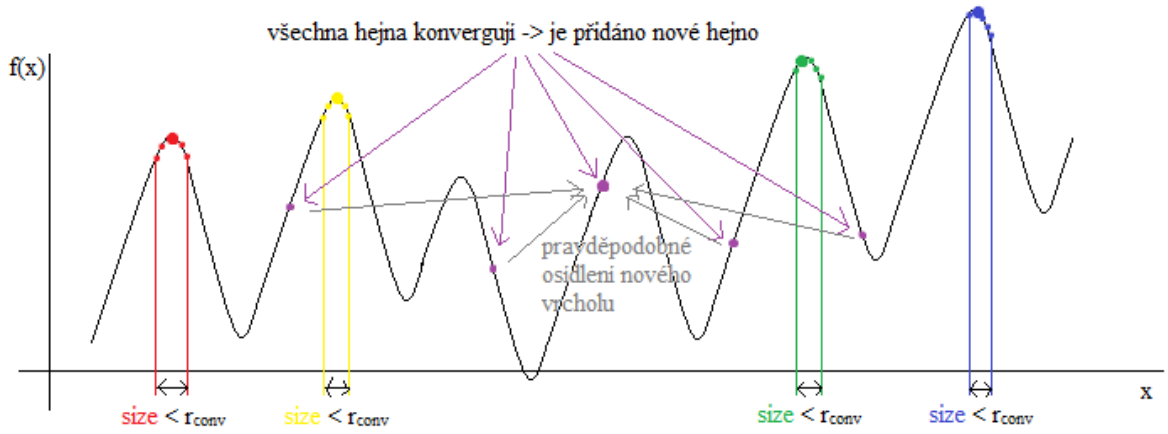
Od této změny se očekává lepší pokrytí prostoru. Pokud všechna hejna konvergují, je pravděpodobné, že je v prostoru vrcholů více nebo rovno počtu hejn; proto je počet hejn navýšen. Po změně funkce je zachyceno více vrcholů než v algoritmu mQPSO.

Maximální počet hejn je omezen, protože nelze zvyšovat počet hejn donekonečna. Více hejn znamená více částic a tedy i více ohodnocení. Čím více ohodnocení za jednu iteraci, tím méně iterací proběhne, než dojde ke změně funkce. Proto se v určitém okamžiku nevyplatí neustále zvyšovat počet hejn. I z toho důvodu AHPSO při dosažení maximálního počtu hejn sníží počet částic v každém hejně o určitý počet (*numOfDelParticles*).

Pokud je počet vrcholů menší nebo roven počtu hejn, lze předpokládat, že každý vrchol je již hejnem obsazen a není potřeba posílat nejhorší hejno prohledávat prostor. V takovém případě je anti-konvergence vypnuta.

Když při maximálním povoleném počtu hejn dojde ke konvergenci všech hejn, existuje pravděpodobnost, že je v prostředí další nepokrytý vrchol. Proto je inkrementován parametr reprezentující pravděpodobný počet vrcholů (*probNumOfPeaks*), který se využívá ve vztahu (4.1) pro určení poloměru vyloučení.

Případ zvýšení počtu hejn v 1D prostoru je vizuálně zobrazen na obrázku 4.2.



Obrázek 4.2 Zobrazení situace, kdy dojde ke zvýšení počtu hejn, v 1D prostoru

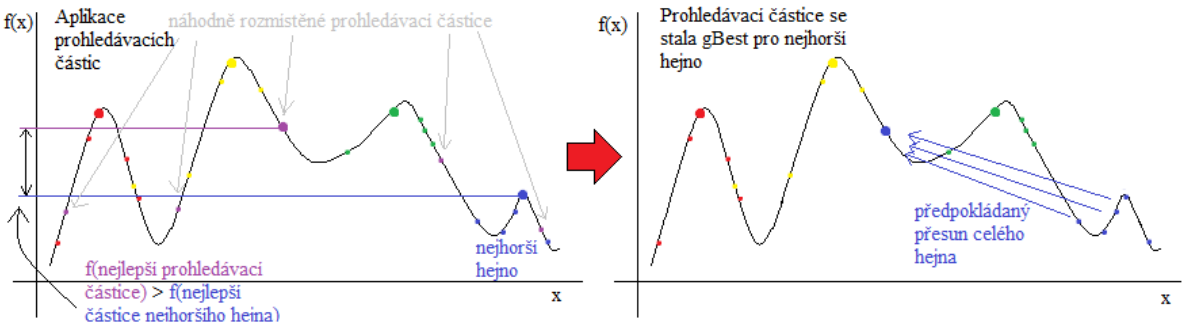
4.4.3 Prohledávací částice

Kromě pokrytí multimodálního prostoru s více vrcholy více hejny se navíc v každé iteraci pošle několik částic (*numOfSearchParticles*) náhodně do celého prostoru. Pokud je funkční hodnota nějaké z těchto částic větší než *gBest* nejhoršího hejna, stává se právě tato částice *gBest* tohoto nejhoršího hejna. Tzn., že hejno bude přitahováno touto lepší pozicí – tímto atraktorem.

Důsledkem je, že algoritmus nečeká, až budou všechna hejna konvergovat, ale při objevené lepší oblasti se nejhorší hejno ihned přemísťuje. Počet těchto prohledávacích částic nesmí být příliš velký, aby se neplýtvalo ohodnoceními, která by mohla být využita částicemi hejn kolem extrémů. Pokud se jedná o problém unimodální nebo pokud je počet vrcholů menší nebo roven počtu hejn, dá se předpokládat, že každý vrchol je obsazen jedním hejnem, a proto nejsou v takovém případě prohledávací částice nutné – nejsou vůbec vypuštěny (nejsou tedy povoleny).

Tyto prohledávací částice se dají považovat za ekvivalenty skautů v algoritmu umělé včelí kolonie (ABC), kteří také náhodně prohledávají prostor, aby našli lepší řešení, než dosud nalezly dělnice. Těmito skauty byly prohledávácí částice inspirovány.

Využití prohledávajících částic je zobrazeno na obrázku 4.3.



Obrázek 4.3 Zobrazení využití prohledávacích částic v 1D prostoru

4.4.4 Snižování počtu hejn

V multimodálním prostředí, kde se vyskytuje velký počet vrcholů, se využívá technik zvyšování hejn, anti-konvergence a vyslání prohledávacích částic. Naopak pokud je problém unimodální, případně je počet vrcholů menší nebo roven počtu hejn, je zbytečné, aby v prostředí bylo hodně hejn. V takovém případě je vhodné snižovat jejich počet.

V každé iteraci se porovnávají funkční hodnoty nejlepších částic (*gBest*) jednotlivých hejn s funkční hodnotou nejlepší částice nejlepšího hejna; tedy porovnávají se atraktory těchto hejn. Pokud je po určitou dobu (po určitý počet iterací) (*maxNumIterForDeleteSwarm*) rozdíl těchto funkčních hodnot dostatečně velký (větší než práh) (*distSwarmFromBestThreshold*), dá se předpokládat, že to hejno, které se porovnává s nejlepším, nemůže nalézt vrchol, který by důkladně prohledal, a jen se potuluje ve špatných místech nebo hodně nízkých vrcholech. Je tedy vysoce pravděpodobné, že v prostoru už žádný neobsazený vrchol není. Takové toulavé hejno je zbytečné, protože jen ubírá ohodnocení částicím těch hejn, které prohledávají skutečné vrcholy. V takovém případě je hejno odstraněno.

Počet iterací (*maxNumIterForDeleteSwarm*), po který je rozdíl funkčních hodnot atraktorů dvou hejn dostatečně velký, musí být vhodně zvolen. Pokud je tento počet malý, může dojít k nesprávnému odstranění příliš mnoha hejn. Pokud je tento počet příliš velký, hejna mohou být odstraňována příliš pomalu a přínos snižování počtu hejn je eliminován.

Dostatečně velký rozdíl funkčních hodnot dvou atraktorů (*distSwarmFromBestThreshold*) (práh) musí být taky vhodně nastaven. Příliš velký zapříčiní, že se počet hejn snižovat nebude; příliš malý zase odstraní hejna i z nějakého vrcholu. Vhodná je alespoň okrajová znalost prostředí, jaké jsou zhruba nejnižší a nejvyšší hodnoty. Pokud znalost prostředí není, je vhodnější konzervativnější přístup, a to nastavit počet iterací a práh spíše vyšší, tím se hejna odstraňují opatrněji – lepší je mít více hejn než moc málo. Výchozím nastavením prahu by měla být průměrná hodnota nejvyšší a nejnižší hodnoty fitness funkce nebo vyšší

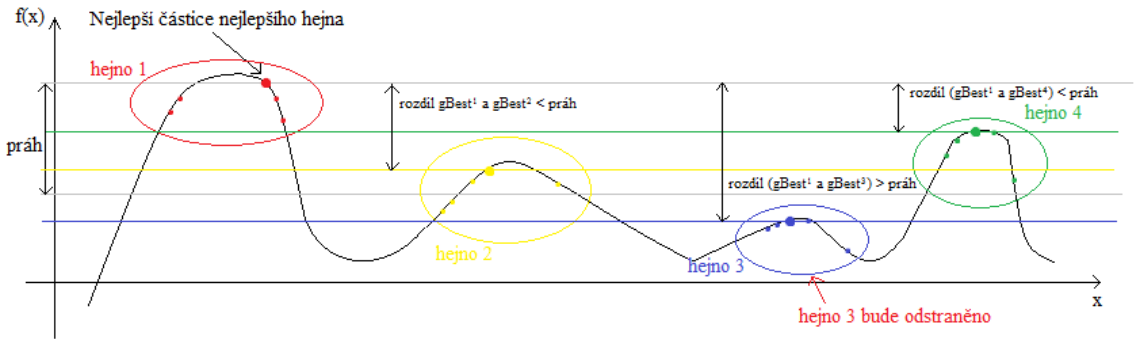
$$distSwarmGromBestThreshold \geq \frac{f(best)-f(worst)}{2}, \tag{4.4}$$

kde *best* a *worst* jsou nejlepší a nejhorší řešení problému.

Pokud je odstraněno i jen jedno hejno, je vypnuta anti-konvergence a nejsou do prostoru posílány prohledávací částice, protože se nepředpokládá, že je v prostoru ještě nějaký neobsazený vrchol. Když by anti-konvergence nebyla vypnuta, nejhorší hejno by bylo znovu inicializováno a mohlo by dojít k tomu, že by toto hejno nebylo schopné dostatečně rychle najít svůj původní vrchol a bylo by po určité době nesprávně odstraněno.

S odstraněním hejna se sníží o jedna i parametr udávající pravděpodobný počet vrcholů v rovnici pro vypočtení poloměru vyloučení (*probNumOfPeaks*).

Zobrazení případu, kdy dojde ke snížení hejn, je na obrázku 4.4.



Obrázek 4.4 Zobrazení snížení počtu hejn v 1D prostoru

4.4.5 Prohledávání kolem nejlepšího hejna

K efektivnímu prohledání nejperspektivnější oblasti – oblasti kolem nejlepší částice v nejlepším hejnu – bylo navrženo další prohledávání. Prohledávání kolem nejlepší částice celého multi-hejna není úplně náhodné, ale vychází z pozic ostatních částic v nejlepším hejně.

Tento přístup vychází z algoritmu Umělé včelí kolonie (ABC), kde vyčkávací včely zkoumají oblasti kolem dělnic, zda nenajdou lepší pozici. Vyčkávací včely si vyberou dělnice proporcionálně k jejich fitness funkcím – ruletový výběr. Dělnice s lepší fitness funkcí mají větší pravděpodobnost, že budou vybrány. V AHPSO v této části tyto „vyčkávací včely“ prohledávají oblast jen kolem té úplně nejlepší částice, takže žádný ruletový výběr není potřeba. V jiné části AHPSO se prohledávají oblasti všech částic nejlepšího hejna, tam už se ruletový výběr využívá (viz kapitola 4.4.8).

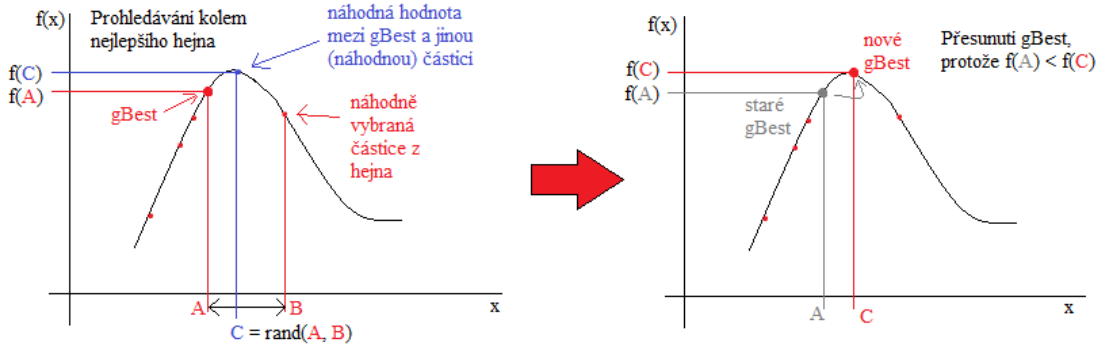
Nejlepší částice nejlepšího hejna se několikrát (*numOfHelpParticlesAroundBest*) v této iteraci podívá na nějakou náhodnou pozici, která se nachází mezi současnou nejlepší pozicí a pozicí nějaké jiné náhodné částice v hejně. Pokud je nová pozice $pBestNew_i$ lepší než dosud nejlepší $pBest_i$, pak se na ni nejlepší částice přesune. Výpočet nové zkoumané pozice kolem nejlepší částice je dán vztahem

$$pBestNew_{id} = pBest_{id} + rand_d(-1, 1) \cdot (pBest_{id} - x_{kd}), \quad (4.5)$$

kde *i* je nejlepší částice, *pBest* je nejlepší dosud nalezená pozice nejlepší částice, tedy *gBest* celého hejna ($pBest_i$ a *gBest* jsou totožné), a $k \neq i$ je náhodná částice z tohoto (nejlepšího) hejna.

Na tuto část lze nahlížet jako na poslání několika (*numOfHelpParticlesAroundBest*) pomocných částic do oblasti té úplně nejlepší částice.

Vizuální ukázka prohledávání kolem nejlepšího hejna je zobrazena na obrázku 4.5.



Obrázek 4.5 Vizuální ukázka prohledávání kolem nejlepšího hejna v 1D prostoru

4.4.6 Přesun částic, které se nezlepšují

Je pravděpodobné, že částice, která se přemísťuje určitým směrem a po několik iterací nemůže najít lepší pozici, letí špatným směrem nebo má malou rychlost. Takové částici dlouho trvá, než se dostane do oblasti kolem *gBest* nebo *pBest*, která je perspektivní a vhodná pro detailní prohledání.

Pro každou částici každého hejna platí, že pokud se určitý počet iterací (*maxNumOfSamePBest*) nezlepší její pozice, je přemístěna do blízkosti své *pBest*. V každé dimenzi je náhodně zvolena pozice kolem *pBest* pomocí vztahu

$$x_{id} = rand_d(pBest_{id} - distFromPBest,\ pBest_{id} + distFromPBest), \quad (4.6)$$

kde *i* je částice, jejíž *pBest* se několik iterací nemění, *d* je dimenze a *distFromPBest* je maximální vzdálenost od *pBest*, na kterou se částice *i* přesune. Částice se na nové pozici neohodnotí, tzn., že se přesune, i když se její pozice zhorší.

Tento přístup byl inspirován klonováním s hypermutací v umělých imunitních systémech. Částice je nahrazena klonem nejlepší částice, na který je použita mutace – posun v jednotlivých dimenzích prostoru.

Po přesunu se také vynuluje rychlost.

4.4.7 Uspání hejn na neglobálních (lokálních) extrémech

Každé hejno se po určitém čase shlukne kolem nějakého vrcholu. Pokud ne, pak je odstraněno (viz kapitola 4.4.4). Pokud je hejno dlouhou dobu jen na lokálním extrému, tzn., že nejde o nejlepší hejno v multi-hejně, pak v něm není až tak potřeba dělat krok algoritmu úplně každou iteraci, protože při změně funkce se vrchol stejně posune, a tedy není potřeba mít přesně nalezený tento lokální extrém.

Z tohoto důvodu hejno, které se shluklo pouze kolem lokálního extrému a tento extrém je lokální po delší dobu (*maxNumOfConstLocalPeaks*), jednou za čas vynechá svůj krok. Dá se říci, že hejno se na jednu iteraci uspí. Tím se ušetří několik evaluací, které se využijí např. pro prohledávací částice (kapitola 4.4.3) nebo pro prohledání oblasti kolem nejlepší částice nejlepšího hejna (kapitola 4.4.5).

4.4.8 Použité techniky, pokud se nejlepší hejno nezlepšuje

Ideální situace je, pokud se nejlepší hejno – hejno shromážděné kolem globálního extrému – neustále zlepšuje, tedy že efektivně využívá čas, který je mu dán na nalezení řešení dynamického problému. Může se lehce stát, že hejno se delší dobu nezlepšuje, že jsou jeho částice zaseknuté v určité oblasti a nemohou z ní vybřednout. Pokud tato situace trvá delší dobu, je vhodné hejnu i nějak pomoci. Z tohoto důvodu byly zavedeny prvky, které by měly hejnu pomoci najít lepší pozici.

Pokud se nejlepší částice nejlepšího hejna nezlepší po určitý počet iterací, použijí se techniky, které jsou trochu jiné než obyčejné PSO a které by měly pomoci hejnu vylepšit svoji pozici.

První technikou, která se použije, pokud se nejlepší hejno několik iterací (*maxNumOfSameGBestToSendHelp*) nezlepší, je ta, která je popsána v kapitole 4.4.5. Jde o prozkoumání určité oblasti jiným způsobem – technikou známou z algoritmu ABC. Rozdíl od prohledávání oblasti kolem nejlepšího hejna v každé iteraci (kapitola 4.4.5) tkví v tom, že teď se nezkoumá oblast pomocnými částicemi (*numOfHelpParticlesAroundAll*) kolem nejlepší částice v hejnu, ale proporcionálně se prozkoumají oblasti kolem všech částic v hejnu – více se prozkoumá oblast kolem částice s lepší fitness funkcí. Je to naprosto stejný způsob, který využívají vyčkávací včely algoritmu ABC. Zkoumaná pozice je vypočtena podle vztahu (4.5), kde tentokrát *i* není nejlepší částice, ale částice vybraná náhodně (proporcionálně) na základě fitness funkce – ruletový výběr. Pravděpodobnost pr_i, že bude vybrána částice *i* pro prohledání její oblasti, je

$$pr_i = \frac{f(x_i)}{\sum_{j=1}^{N} f(x_j)}, \tag{4.7}$$

kde *f* je fitness funkce problému a *N* je počet částic.

Tím se dosáhne prohledání prostoru, který je mezi částicemi nejlepšího hejna, do kterého se částice obyčejným PSO krokem nedostane, protože ty směřují jen k *pBest* a *gBest*.

Další technikou, která by měla řešit nezlepšující se nejlepší hejno, je použití algoritmu SOMA. Pokud se nejlepší hejno po několik iterací nezlepší (*maxNumOfSameGBestToSOMA*), je v následující iteraci jednou použit algoritmus SOMA. V tomto algoritmu každá částice jde s určitou pravděpodobností po pevně daných krocích směrem k nejlepší částici a v každém tomto kroku zkoumá, zda je tato nová pozice lepší než dosavadní pozice této částice. Pokud ano, pozici si částice uloží jako svou novou aktuální pozici a zkoumá, zda je lepší než dosud nejlepší pozice částice. Pokud je i tohle pravda, pak si novou pozici uloží i do své *pBest*. Částice jde nejen k nejlepší částici, ale jde i za ni (viz kapitola 4.4.9).

Rozdíl od předchozího přístupu (prohledání okolí podle ABC) spočívá v tom, že oba přístupy prohledávají trochu jiné oblasti a SOMA nemusí jít k nejlepší částici přímo, ale v nějaké dimenzi může zůstat stát na svém původním místě.

4.4.9 Krok algoritmu (PSO nebo SOMA)

V každé iteraci částice provede jeden krok. V závislosti na stavu daného hejna je to buď krok PSO, nebo krok SOMA. Částice krok neprovede jen v případě popsaném v kapitole 4.4.7.

Pokud má částice vykonat fázi PSO, jsou použity klasické rovnice pro novou rychlost (3.7) a pro novou pozici (3.1). Jde tedy o PSO s omezujícím faktorem (viz kapitola 3.3.4), kde $\chi = 0.7298$ a $c_1 = c_2 = 2.05$. V AHPSO PSO používá pro změnu rychlosti globální sousedství mezi částicemi. Tedy topologie je plně propojený graf, ve kterém každou částici ovlivňuje nejlepší soused z úplně celé populace (viz kapitola o statických sousedstvích 3.4.1). Maximální rychlost V_{max} je nastavena na čtvrtinu rozsahu prohledávajícího prostoru. Pokud tedy rychlost $v_{i,d} > V_{max,d}$, pak $v_{i,d} = V_{max,d}$. Pokud se částice dostane v určité dimenzi mimo prohledávací prostor, je v této dimenzi částice znovu inicializována. Tradičně, pokud je nově nalezená pozice aktivní částice lepší než jeho *pBest*, tato nová pozice se stává novým *pBest*.

Pokud částice vykonává fázi SOMA, je pro určení nové pozice použit vztah

$$xNew_{ij} = pBest_{ij} + \left(gBest_j - pBest_{ij}\right) \cdot r \cdot PRTVector_j, \tag{4.8}$$

kde $r = step, 2 \cdot step, 3 \cdot step, \ldots, pathLength$. *Step* je pevný krok, po kterém částice jde k nejlepší částici, určuje „zrnitost“, s jakou bude mapována cesta aktivní částice. *PathLength* je celková délka cesty, po které částice jde, určuje, jak daleko se aktivní částice zastaví za vedoucí částicí, obvykle $pathLenght \geq 1$. *PRTVector* se generuje každou iteraci a jde o vektor jedniček a nul takový, že

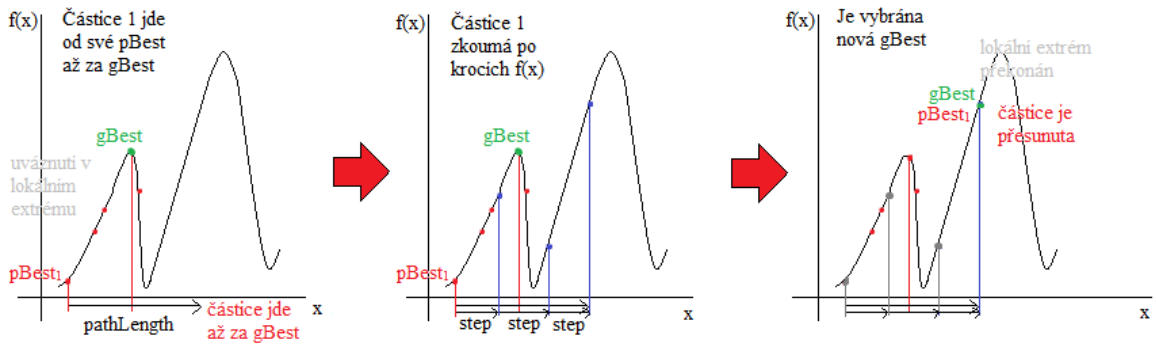
$$PRTVector_j = \begin{cases} 1, & rand_j\langle 0,1\rangle < PRT \\ 0, & jinak \end{cases}, \tag{4.9}$$

kde $j = 1, \ldots, D$, kde D je počet dimenzí, a *PRT* znamená perturbace a udává, zda se aktivní jedinec bude pohybovat přímo k vedoucímu jedinci či ne. Pokud je $PRT = 1$, pak se jedinci pohybují přímo k nejlepší částici [3]. Čím je *step* menší, tím je podrobněji prozkoumána oblast, ale tím je taky

optimalizace pomalejší, protože je spotřebováno mnoho evaluací v jedné iteraci. Pokud je *pathLength* < 1, pak částice až ke *gBest* nedojde; pokud je *pathLength* > 1, částice jde za *gBest*. Pokud např. *pathLenght* = 2, pak se částice zastaví za *gBest* v takové vzdálenosti, v jaké stojí před ní.

V AHPSO ve fázi SOMA částice nezkoumá prostředí směrem k nejlepší částici od své aktuální pozice, ale od své dosud nejlepší pozice, jak lze vidět v rovnici (4.8). Pokud je v nějakém z těchto kroků (*step*) nalezena lepší pozice ($xNew_i$), než je aktuální pozice aktivní částice (x_i), částice se na tuto pozici přesune. Pokud je zároveň tato nová pozice lepší než $pBest_i$, je na tuto novou pozici $xNew_i$ změněno i $pBest_i$.

Jeden krok algoritmu SOMA je zobrazena na obrázku 4.6.



Obrázek 4.6 Zobrazení jednoho kroku algoritmu SOMA v 1D prostoru

Algoritmus SOMA se musel trochu přizpůsobit PSO, protože v originálním SOMA nejsou žádné *pBest* ani *gBest*. V původní SOMA *r* nabývá i hodnotu 0. To je zbytečné, protože tím se testuje pozice, na které částice už je, proto *r* nabývá až hodnoty *step* a vyšší. Tím se ušetří evaluace.

Po jedné iteraci fáze SOMA se fáze automaticky přepíná opět na PSO. Po změně z fáze SOMA do fáze PSO je vynulována rychlost.

Ať je provedena fáze PSO nebo fáze SOMA, vždy se na konci iterace algoritmu určí nejlepší částice *gBest* ze všech *pBest* částic v hejně.

4.4.10 Změna funkce (prostředí)

AHPSO si uchovává pozici a fitness funkci nejlepší částice. Pokud je v nějaké iteraci pozice nejlepší částice stejná, ale hodnota fitness funkce jiná, došlo ke změně funkce. Jakmile je změna funkce detekována, jsou znovu ohodnoceny nejlepší pozice (*pBest*) všech částic.

Protože jak algoritmus postupuje, hejna konvergují, nachází se po určitém počtu iterací částice realitně blízko sebe. To je v době změny funkce nežádoucí, protože jak se posune hejnem okupovaný vrchol, tak příliš shluklé částice nemusí být schopné adekvátně zareagovat. Z toho důvodu je potřeba zavést dodatečnou diverzitu do hejna.

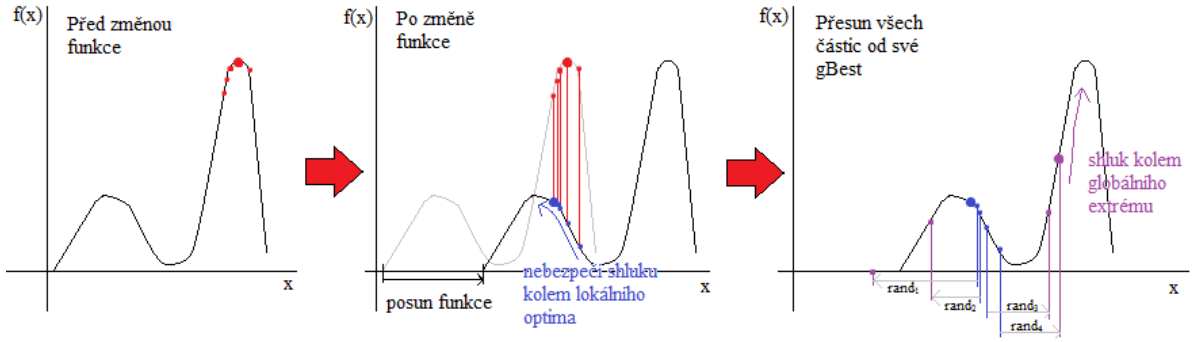
V AHPSO jakmile dojde ke změně prostředí (funkce), všechny částice všech hejn se přesunou náhodně do určité vzdálenosti (*distFromGBest*) od svého atraktoru (nejlepší částice daného hejna) – vzdálí se od něj. Nová pozice je tedy vypočítána podle vztahu

$$x_{id} = rand_d(gBest_d - distFromGBest,\ gBest_d + distFromGBest), \qquad (4.10)$$

kde *i* je číslo částice, *d* je dimenze, *gBest* je nelepší částice hejna, ve kterém se nachází částice *i*, a *distFromGBest* je maximální vzdálenost od *GBest*, na kterou je částice *i* přesunuta. Tyto nové pozice se neohodnocují, tudíž se částice mohou dostat do horších pozic, než byly předtím. To nevadí, protože již nově ohodnocené *pBest* všech částic zůstávají stejné.

Rychlosti všech částic se vynulují.

Přesun částic po změně funkce a z toho plynoucí výhoda zvýšení diverzity v hejně je zobrazen na obrázku 4.7.



Obrázek 4.7 Zobrazení přesunu částic po změně funkce v 1D prostoru

4.4.11 Adaptace počtu hejn na počet vrcholů

Jak bylo popsáno v kapitolách 4.4.2 a 4.4.4, algoritmus AHPSO na dynamické problémy je schopný po svém spuštění měnit počet hejn na základě předpokládaného počtu vrcholů. Algoritmus počet vrcholů nemusí znát, přesto ho dokáže odhadnout. Z toho plyne, že je AHPSO obecně použitelný na unimodální i multimodální dynamické problémy.

Nepotřebou znát předem počet vrcholů se algoritmus AHPSO liší od algoritmu mQPSO, ze kterého vychází. Algoritmus mQPSO využívá znalost počtu vrcholů v rovnici (4.1) pro výpočet poloměru vyloučení a k tomu, zda použít anti-konvergenci. AHPSO místo něj používá parametr předpokládaného počtu vrcholů (*probNumOfPeaks*), který je nejprve roven počtu hejn a poté, co dosáhne maximálního počtu hejn (*maxNumOfSwarms*), se zvyšuje o jedna při každé situaci, kdy všechny hejna konvergují.

Tedy tento parametr vypovídá o pravděpodobném počtu vrcholů jen do doby, než je v prostoru maximální počet hejn. Jakmile je v prostoru více vrcholů, než je maximální možný počet

hejn, pak všechna hejna budou vždy konvergovat a parametr pravděpodobného počtu vrcholů se bude vždy zvyšovat. Už to tedy moc nevypovídá o počtu vrcholů. To ale nemá žádný velký vliv, protože tento parametr je použit pouze do rovnice poloměru vyloučení, na kterou nemá při velkém počtu vrcholů radikální vliv.

Parametry pro snížení hejn (*maxNumIterForDeleteSwarm* a *distSwarmFromBestThreshold*) se musí nastavit hodně citlivě, protože pokud je jednou odstraněno nějaké hejno, už nikdy jejich počet nebude navýšen, protože se vypíná anti-konvergence, tzn., že i když budou všechna hejna konvergovat, nic se nestane. Naopak pokud se v jedné iteraci přidá i více hejn najednou, lze je v dalších iteracích postupně odstraňovat až na požadovaný počet. Navíc se po odstranění i jen jednoho hejna vypíná náhodné prohledávání prostoru. Jak je popsáno v kapitole 4.4.4, konzervativnějším přístupem je nastavení těchto parametrů spíše vyšší než nižší a práh nastavit podle rovnice (4.4).

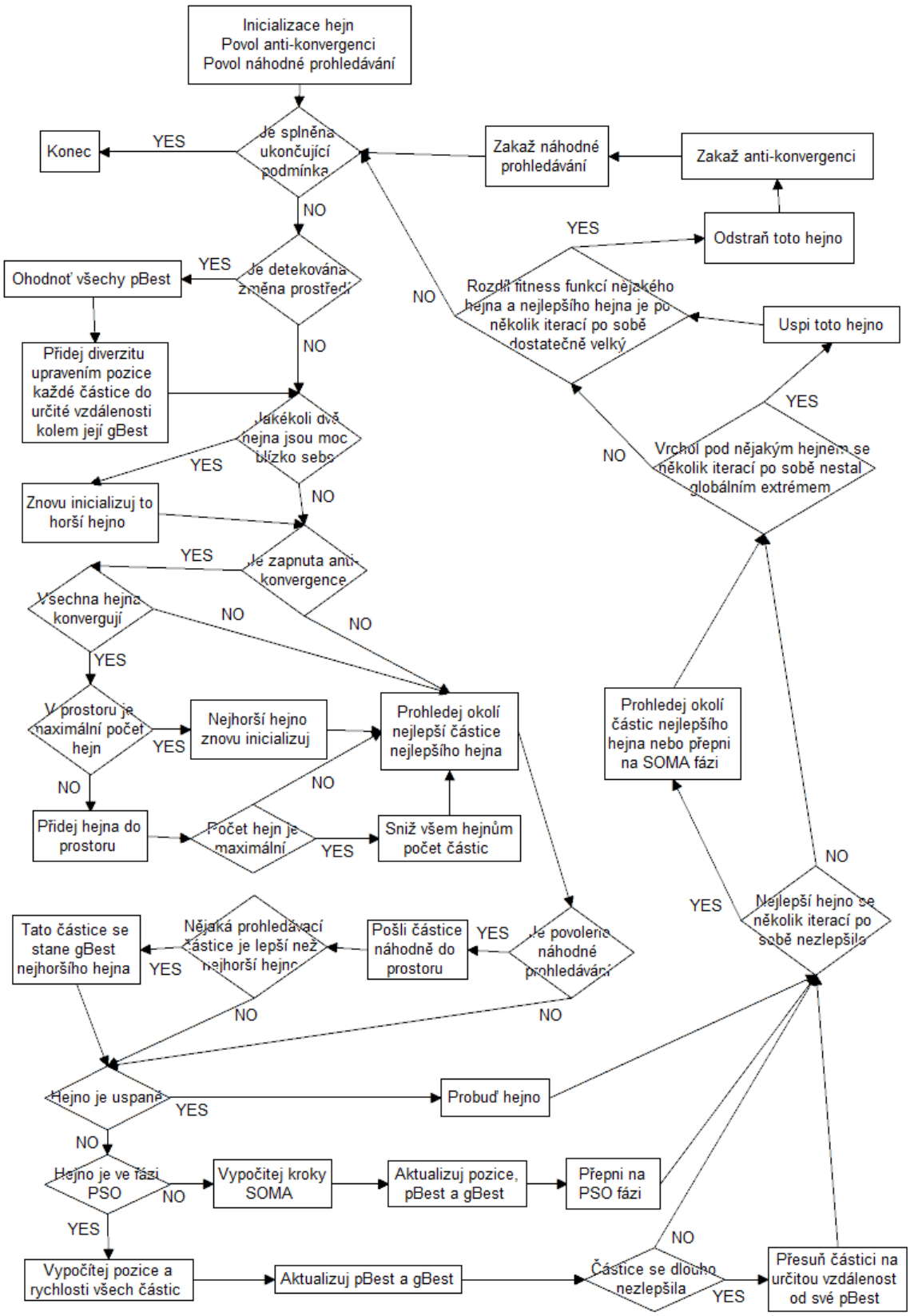
Nejvíce jsou tedy parametry pro snížení počtu hejn ohroženy problémy obsahující méně než 10 vrcholů, protože výchozí počet hejn algoritmu AHPSO je 10 (viz dále). Multimodálních problémů s počtem vrcholů větším než 10 se pravděpodobně mechanismus snižování počtu hejn týkat nebude.

Schopnost algoritmu AHPSO se adaptovat na předem neznámý počet vrcholů jej dělá mnohem efektivnějším, než je tomu u algoritmu mQPSO s fixním počtem hejn.

4.5 Konkrétní tvar algoritmu AHPSO

4.5.1 Vývojový diagram algoritmu AHPSO

Vývojový diagram algoritmu AHPSO se nachází na obrázku 4.1.



Obrázek 4.8 Vývojový diagram algoritmu AHPSO

4.5.2 Pseudokód algoritmu AHPSO

Pseudokód algoritmu AHPSO je následující:

```
Inicializace;
repeat
      if gBest_best(t) == gBest_best(t-1) and
         f(gBest_best(t)) ≠ f(gBest_best(t-1)) then
            Detekována změna prostředí;
      Test exkluze;
      if anti-konvergence je povolena then
            Test anti-konvergence;
      Prohledání okolí nejlepšího hejna;
      if náhodné prohledávání je povoleno then
            Náhodné prohledávání;
      Krok algoritmu;
      Test stavu nejlepšího hejna;
      Test stavu všech ostatních horších hejn;
      Test počtu hejn;
      t = t + 1;
until je dosaženo určitého počtu evaluací;
```

Jednotlivé metody pseudokódu AHPSO jsou rozvedeny v následujících kapitolách.

4.5.2.1 Inicializace:

```
povol anti-konvergenci;
povol náhodné prohledávání;
numOfParticles = initNumOfParticles;
vytvoř numOfSwarms hejn;
for každé hejno s do
      vytvoř numOfParticles částic o D dimenzích;
      for každá částice i do
            x_i^s = rand(minCoord, maxCoord);
            v_i^s = 0;
            samePBest_i^s = 0;
```

4.5.2.2 Detekována změna prostředí:

```
for každé hejno s do
    for každá částice i do
        vypočítej f(pBest_i);
    if f(pBest_i^s) > f(gBest^s) then
        gBest^s = pBest_i^s;
    for každá částice i do
        x_i^s = rand(gBest^s-distFromGBest, gBest^s+distFromGBest);
        v_i^s = 0;
```

4.5.2.3 Test exkluze:

```
vypočítej r_excl podle rovnice (4.1);
for každé dvě hejna s, t do
    vypočítej vzdálenost dist^st hejn s, t pomocí rovnice (4.2);
    if dist^st < r_excl then
        if f(gBest^s) > f(gBest^t) then
            znovu inicializuj hejno t;
        else
            znovu inicializuj hejno s;
```

4.5.2.4 Test anti-konvergence:

```
for každé hejno s do
    vypočítej velikost hejna size^s podle rovnice (4.3);
if ∀s ∈ {s_1, s_2, …, s_numOfSwarms}: size^s < r_conv then
    if numOfSwarms < maxNumOfSwarms then
        numOfSwarms = numOfSwarms + numAddSwarms;
        probNumOfPeaks = numOfSwarms;
    else
        for každé hejno s do
            if f(gBest^s) < f(gBest^worst) then
                worst = s;
        znovu inicializuj hejno worst;
        if numOfParticles == initNumOfParticles then
            for každé hejno do
                numOfParticles = numOfParticles –
                numOfDelParticles;
```

```
        if probNumOfPeaks < 200 then
            probNumOfPeaks = probNumOfPeaks + 1;
```

4.5.2.5 Prohledání okolí nejlepšího hejna:

```
for každé hejno s do
    if f(gBest^s) > f(gBest^best) then
        best = s;
for numOfHelpParticlesAroundBest do
    vyber nejlepší částici i z hejna best;
    náhodně vyber částici k ≠ i z hejna best;
    vypočítej novou pozici gBestNew^best podle rovnice (4.5);
    if f(gBestNew^best) > f(x_i^best) then
        x_i^best = gBestNew^best;
        if f(gBestNew^best) > f(gBest^best) then
            gBest^best = gBestNew^best;
        samePBest_i^s = 0;
    else
        samePBest_i^s = samePBest_i^s + 1;
```

4.5.2.6 Náhodné prohledávání:

```
for každé hejno s do
    if f(gBest^s) < f(gBest^worst) then
        worst = s;
for numOfSearchParticles do
    prohledávací částice i se náhodně umístí do prostoru;
    if f(x_i) > f(gBest^worst) then
        gBest^worst = x_i;
```

4.5.2.7 Krok algoritmu:

```
for každé hejno s do
    if hejno s je uspané then
        hejno s probuď;
    else
        if hejno s je ve fázi PSO then
            for každá částice i do
                vypočítej rychlost v_i^s podle rovnice (3.7);
                if v_i^s > V_max then
                    v_i^s = V_max;
```

```
                    vypočítej pozici x_i^s podle rovnice (3.1);
                    if x_i^s ∉ (minCoord, maxCoord) then
                        znovu inicializuj částici i v prostoru;
                    if f(x_i^s) > f(pBest_i^s) then
                        pBest_i^s = x_i^s;
                        samePBest_i^s = 0;
                    else
                        samePBest_i^s = samePBest_i^s + 1;
                    if samePBest_i^s ≥ maxNumOfSamePBest then
                        samePBest_i^s = 0;
                        v_i^s = 0;
                        vypočítej pozici x_i^s podle rovnice (4.6);
            if hejno s je ve fázi SOMA then
                for každá částice i do
                    for ∀r ∈ {step, 2·step, …, pathLength} do
                        vypočítej PRTVector podle rovnice (4.9);
                        vypočítej novou pozici xNew_i^s podle
                        rovnice (4.8);
                        if f(xNew_i^s) > f(x_i^s) then
                            x_i^s = xNew_i^s;
                    if f(x_i^s) > f(pBest_i^s) then
                        pBest_i^s = x_i^s;
                        samePBest_i^s = 0;
                    else
                        samePBest_i^s = samePBest_i^s + 1;
                změn fázi hejna s na PSO;
            for každá částice i do
                if f(pBest_i^s) > f(gBest^s) then
                    gBest^s = pBest_i^s;
```

4.5.2.8 Test stavu nejlepšího hejna:

```
for každé hejno s do
    if f(gBest^s) > f(gBest^best) then
        best = s;
```

```
if f(gBest^best(t)) ≤ f(gBest^best(t-1)) then
      sameGBest^best = sameGBest^best + 1;
else
      sameGBest^best = 0;
if sameGBest^best == maxNumOfSameGBestToSendHelp then
      for numOfHelpParticlesAroundAll do
            for každá částice i z hejna best do
                  vypočítej pravděpodobnost pr_i^best výběru částice i
                  použitím rovnice (4.7);
            ruletovým výběrem nad hodnotami pr_i^best vyber náhodně
            částici i z hejna best;
            náhodně vyber částici k ≠ i z hejna best;
            vypočítej novou pozici gBestNew^best podle rovnice (4.5);
            if f(gBestNew^best) > f(x_i^best) then
                  x_i^best = gBestNew^best;
                  if f(gBestNew^best) > f(gBest^best) then
                        gBest^best = gBestNew^best;
                  samePBest_i^s = 0;
            else
                  samePBest_i^s = samePBest_i^s + 1;
if sameGBest^best == maxNumOfSameGBestToSOMA then
      nastav hejno best do fáze SOMA;
```

4.5.2.9 Test stavu všech ostatních horších hejn:

```
for každé hejno s do
      if f(gBest^s) > f(gBest^best) then
            best = s;
constLocalPeak^best = 0;
for každé hejno s do
      if s ≠ best then
            constLocalPeak^s = constLocalPeak^s + 1;
            if constLocalPeak^s == maxNumOfConstLocalPeak then
                  constLocalPeak^s = 0;
                  hejno s uspi;
```

4.5.2.10 Test počtu hejn:

```
for každé hejno s do
      if f(gBest^s) > f(gBest^best) then
            best = s;
for každé hejno s do
      if |f(gBest^s) − f(gBest^best)| > distSwarmFromBestThreshold then
            numIterForDeleteSwarm^s = numIterForDeleteSwarm^s + 1;
      else
            numIterForDeleteSwarm^s = 0;
if ∃s ∈ {s_1, s_2, …, s_numOfSwarms}:
   numIterForDeleteSwarm^s ≥ maxNumIterForDeleteSwarm then
      odstraň hejno s;
      probNumOfPeaks = probNumOfPeaks − 1;
      zakaž anti-konvergenci;
      zakaž náhodné prohledávání;
```

5 Implementace

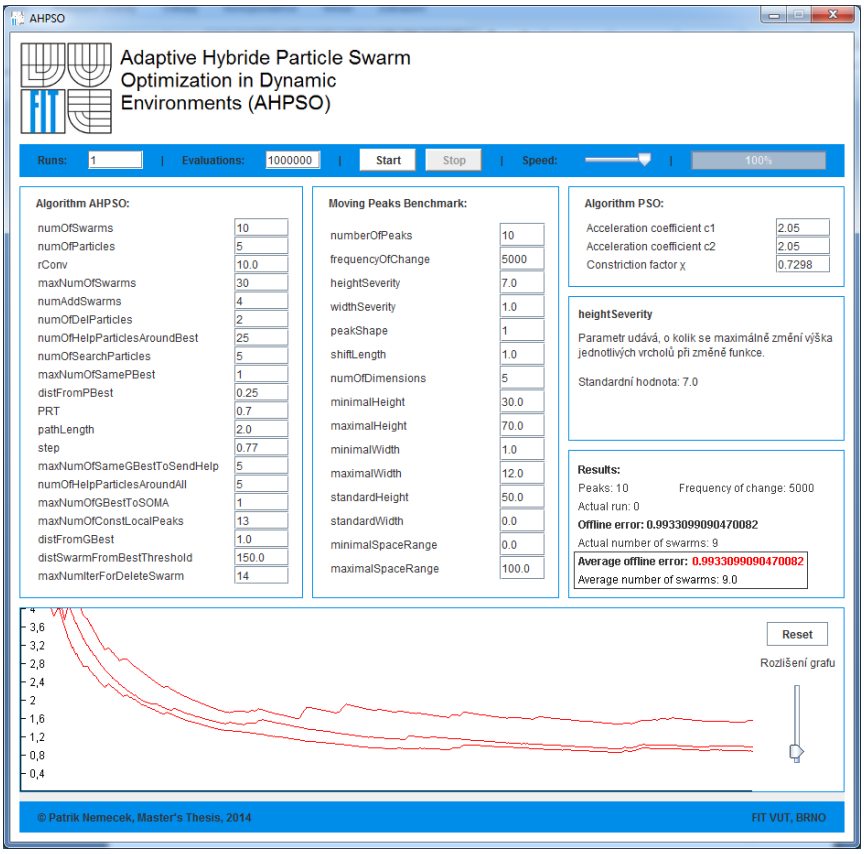
Algoritmus AHPSO byl implementován v programovacím jazyce Java. Při implementaci bylo využito vývojové prostředí NetBeans. Výsledný program byl nazván AHPSO_APP. Jazyk Java byl vybrán díky jeho univerzálnosti, program lze bez problémů spustit na každém stroji, kde je nainstalován Java RE (Java Runtime Environment).

Aplikace se skládá z 6 zdrojových souborů (tříd):

- `AHPSO_APP.java` – okno obsahující prvky GUI;
- `ParametersPanel.java` – panel s nastavitelnými parametry;
- `GraphPanel.java` – panel se zobrazujícím se grafem;
- `AHPSO.java` – samotný algoritmus spouštějící se v novém vlákně;
- `Swarm.java` – jedno hejno;
- `Particle.java` – jedna částice.

Zdrojové soubory obsahují celkem asi 2400 řádků kódu. Velikost výsledného programu je 130 kB.

Po nastavení parametrů algoritmu AHPSO a úlohy s pohybujícími se vrcholy se po spuštění aplikace zobrazí výsledky a grafy offline chyby (viz kapitola 6.1.1) a/nebo průměrné hodnoty offline chyby v několika bězích. Na obrázku 5.1 je vidět uživatelské rozhraní programu.



Obrázek 5.1 Uživatelské rozhraní aplikace algoritmu AHPSO

6 Experimenty

Bylo provedeno několik experimentů s cílem nalézt optimální hodnoty nastavitelných parametrů algoritmu AHPSO. Experimenty probíhaly na úloze pohybujících se vrcholů (Moving Peaks Benchmark). Nejprve bylo třeba určit, jaké kritérium (metrika) se bude na dynamických pohybujících se vrcholech zkoumat (měřit).

6.1 Metriky pro dynamické problémy

Protože dynamický problém je charakteristický svou častou změnou účelové funkce po určitém počtu evaluací, nemá smysl sledovat fitness hodnotu nejlepšího jedince v populaci. Výsledná hodnota fitness funkce nejlepšího jedince celého algoritmu hodně závisí na tom, kdy je algoritmus ukončen. Pokud je algoritmus ukončen ihned po změně funkce, pak nejlepší jedinec nemá dobrou fitness hodnotu, protože pozice nejlepší částice ještě odráží řešení předchozí funkce.

Navíc údaj o fitness hodnotě na konci provádění algoritmu neodráží schopnost algoritmu se rychle přesunout na nejlepší pozici po změně funkce. Je potřeba nějak sledovat nejlepší hodnotu v průběhu celého provádění algoritmu, který teoreticky běží nekonečně dlouho.

Tento problém řeší offline chyba (viz kapitola 6.1.1), což je globální parametr kvality optimalizačního procesu.

6.1.1 Offline chyba

Nejdůležitější metrikou pro určení kvality a efektivity algoritmu pro dynamické problémy je tzv. offline chyba (offline error). Offline chyba je definována

$$e_{off} = \frac{1}{T}\sum_{t=1}^{T}(f(global(t)) - f(gBest^{best}(t))), \tag{5.1}$$

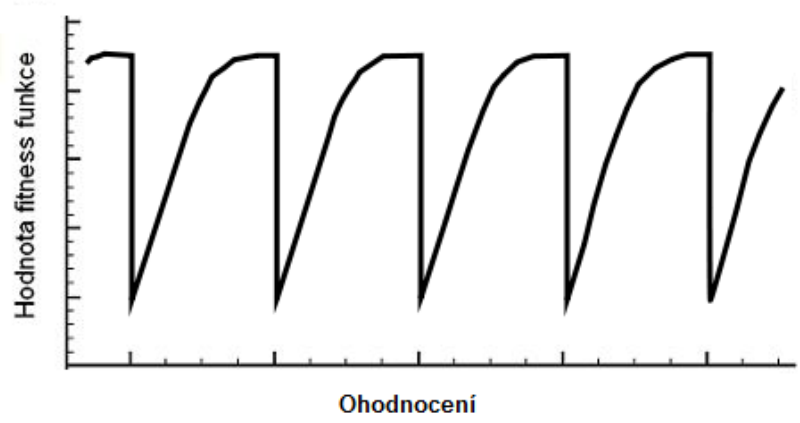
kde T je celkový počet evaluací, f je účelová funkce v evaluaci t, $global$ je skutečné optimum funkce f a $gBest^{best}$ je nejlepší dosud nalezené řešení nejlepším hejnem $best$.

Offline chyba je tedy definována jako průměr všech chyb (odchylek od optimální hodnoty) dosažených při konečném počtu evaluací. Počet evaluací by měl být teoreticky nekonečný, aby se eliminoval vliv prvotní náhodné inicializace jedinců na začátku algoritmu, kdy je offline chyba největší. Po určitém počtu evaluací by se měla offline chyba ustálit kolem konkrétní hodnoty.

6.1.2 Časový průběh účelové funkce

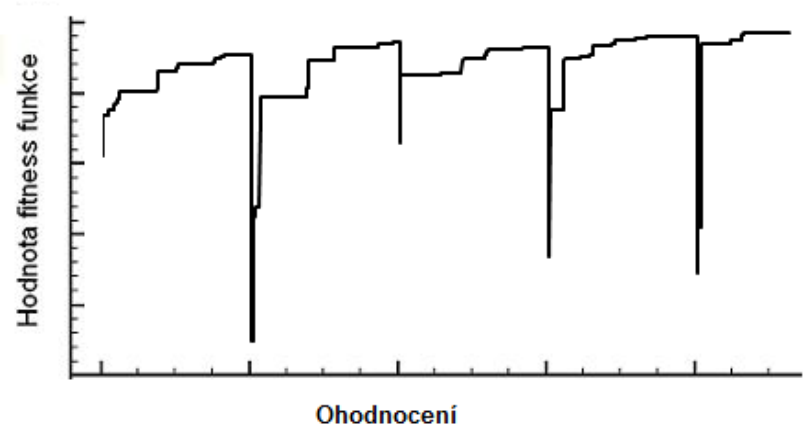
Samotná hodnota účelové funkce po skončení algoritmu je nic neříkající, ale její časový průběh (nebo průběh chyby) podává detailní grafický náhled o dynamice evoluce. Většinou se pro každou

generaci vynáší nejlepší dosažená fitness hodnota (nebo nejmenší dosažená chyba). Teoretický časový průběh fitness hodnoty nejlepší částice je zobrazen na obrázku 6.1. Na něm lze vidět pravidelné náhlé zhoršení fitness hodnoty v takovém ohodnocení, ve kterém dojde ke změně funkce. Po tomto drastickém propadu následuje postupné zlepšování, jak pracuje optimalizace, až do optimální fitness hodnoty.



Obrázek 6.1 Teoretický časový průběh fitness hodnoty nejlepší částice

V reálu není graf časového průběhu tak pravidelný, protože v algoritmu vystupuje prvek náhody, který ovlivňuje pozice částic. Jak vypadá reálný časový průběh fitness hodnoty nejlepší částice, je zobrazeno na obrázku 6.2.

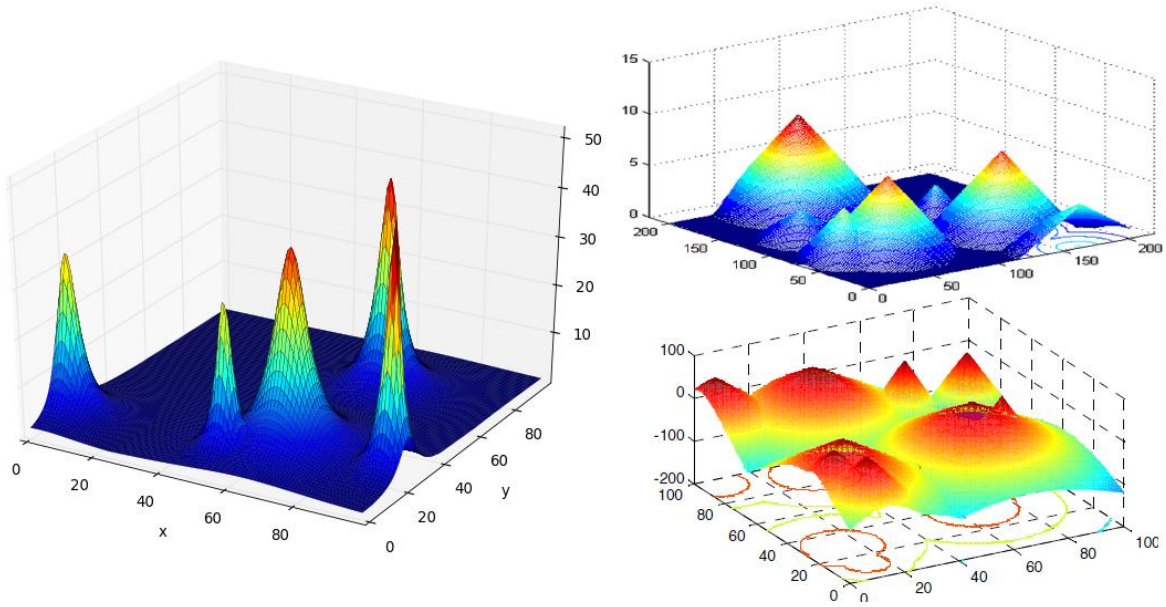


Obrázek 6.2 Reálný časový průběh fitness hodnoty nejlepší částice

6.2 Moving Peaks Benchmark

Naprostá většina veřejných volně dostupných algoritmů na dynamické problémy používá k testování sadu úloh s pohybujícími se vrcholy (Moving Peaks Benchmark). A aby bylo možné porovnat výsledky jednotlivých algoritmů a algoritmu AHPSO, byla pro účely testování výkonu algoritmu AHPSO použita právě tato sada. Implementace Moving Peaks Benchmark v jazyce Java je volně dostupná například na webové stránce [55].

V úloze pohybujících se vrcholů lze nastavit počet vrcholů v prostředí a počet evaluací, po kterém se účelová funkce pokaždé změní. Změna spočívá v posunutí, změně šířky a výšky všech vrcholů. Příklady 2-dimenzionálního prostředí pohybujících se vrcholů lze vidět na obrázku 6.3.



Obrázek 6.3 Příklady úloh pohybujících se vrcholů [52][53][56]

Úloha pohybujících se vrcholů má velké množství nastavitelných parametrů. Standardní nastavení parametrů použité při testování a jejich význam je uvedeno v tabulce 6.1.

Parametr	**Hodnota**	**Význam**
number of peaks	10	počet vrcholů funkce
frequency of change	každých 5000 ohodnocení	počet ohodnocení funkce, po kterých dojde k její změně
height severity	7.0	o kolik se maximálně změní výška jednotlivých vrcholů při změně funkce
width severity	1.0	o kolik se maximálně změní šířka jednotlivých vrcholů při změně funkce

peak shape	kužel	typ funkce (standardní funkce, kužel, koule)
shift length	1.0	o kolik se maximálně posunou jednotlivé vrcholy při změně funkce
number of dimensions	5	počet dimenzí funkce
minimal height	30.0	minimální výška jednotlivých vrcholů
maximal height	70.0	maximální výška jednotlivých vrcholů
minimal width	1	minimální šířka jednotlivých vrcholů
maximal width	12	maximální šířka jednotlivých vrcholů
standard height	50	standardní výška vrcholů (při 0 je výška náhodná hodnota mezi minimální a maximální výškou)
standard width	0	standardní šířka vrcholů (při 0 je šířka náhodná hodnota mezi minimální a maximální šířkou)
search space range	0 - 100	rozsah prohledávajícího prostoru ve všech dimenzích

Tabulka 6.1 Výchozí hodnoty parametrů úlohy pohybujících se vrcholů a jejich význam

6.3 Experimenty s nastavitelnými parametry

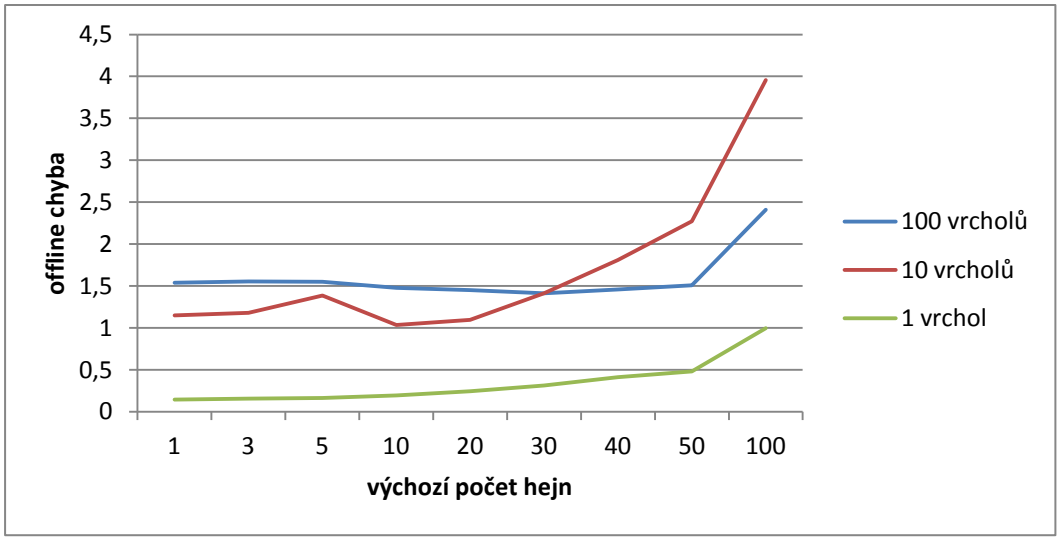
Protože algoritmus AHPSO obsahuje relativně velké množství nastavitelných parametrů, byla provedena sada experimentů, které měly za cíl tyto parametry optimalizovat. K ladění parametrů byly použity právě úlohy s pohybujícími se vrcholy. Jejich parametry byly nastaveny tak, jak je uvedeno v tabulce 6.1, jen počet vrcholů se měnil.

Počet vrcholů byl v jednotlivých úlohách pro testování nastaven na hodnoty 1, 10 a 100. Byly zvoleny právě tyto hodnoty, aby byl vidět vliv parametrů v různých situacích – unimodální problém, multimodální probémy se stejným počtem hejn jako vrcholů a s mnohem větším počtem vrcholů.

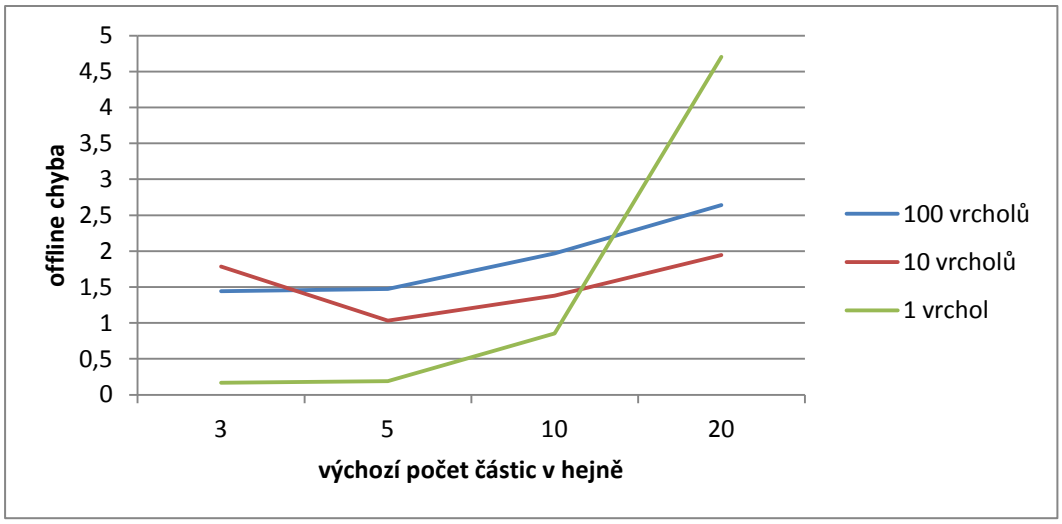
Při testování různých hodnot jakéhokoli parametru AHPSO zůstaly ostatní parametry konstantní. V testech nešlo o absolutní hodnoty offline chyb, ale o porovnání výsledků (při různých hodnotách zkoumaného parametru) mezi sebou, o vliv parametru na algoritmus. Proto byly pro zobrazení výsledků upřednostněny grafy před tabulkami s konkrétními hodnotami. Jako dostatečná ukončovací podmínka bylo nastaveno milion evaluací. Všechny výsledky testů jsou průměrné hodnoty z 50 běhů.

6.3.1 Počet hejn a počet částic

Nejprve je potřeba vhodně nastavit výchozí počet hejn a částic, s kterými se algoritmus pouští do optimalizace. Vliv počtu hejn na výkon je na grafu 6.1 a vliv počtu částic na grafu 6.2.



Graf 6.1 Závislost offline chyby na výchozím počtu hejn



Graf 6.2 Závislost offline chyby na výchozím počtu částic v každém hejně

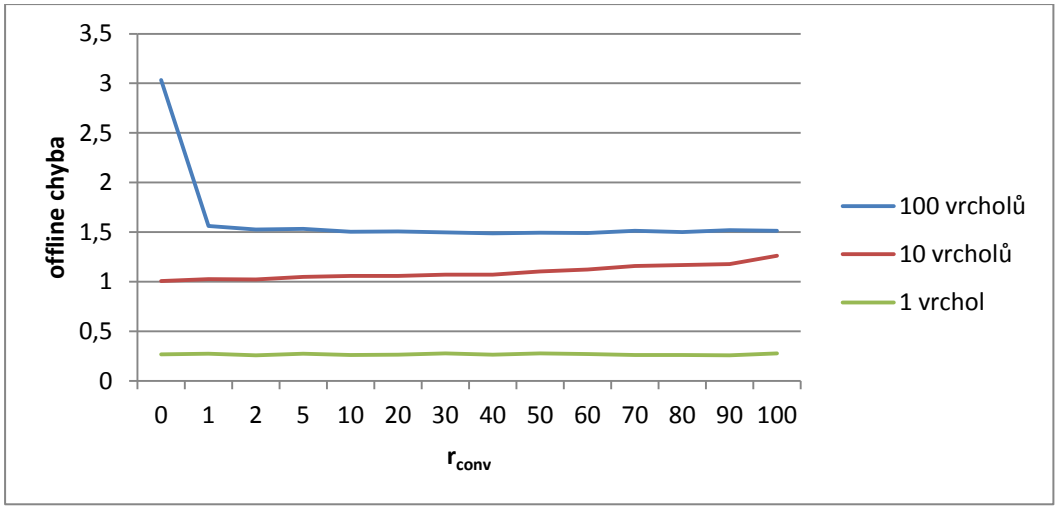
V úloze se 100 vrcholy je optimální počet hejn 30. S vyšším počtem hejn dochází překvapivě ke zhoršení, i když by např. při 100 hejnech byl teoreticky pokrytý každý vrchol. Naopak v úlohách s 1 a 10 vrcholy je nejvhodnější výchozí počet hejn nepřekvapivě rovný počtu vrcholů. Při velkém počtu hejn je výkon drasticky horší.

V úlohách s 1 a 100 vrcholy má algoritmus téměř shodné výsledky, pokud je počet částic v hejně 3 nebo 5. V úloze s 10 vrcholy má algoritmus daleko nejlepší hodnotu offline chyby s 5 výchozími částicemi v hejně. S vyšším výchozím počtem částic v hejně se zhoršuje výkon algoritmu na všech úlohách.

Byl zvolen kompromis – výchozí počet hejn 10 a výchozí počet částic v každém hejně 5.

6.3.2 r_{conv}

Parametr poloměr konvergence r_{conv} slouží k určení, zda hejno konverguje. Pokud je velikost nějakého hejna menší než r_{conv}, pak toto hejno konverguje (viz kapitola 4.4.2). Vliv tohoto parametru na offline chybu je vidět v grafu 6.3.



Graf 6.3 Závislost offline chyby na parametru r_{conv}

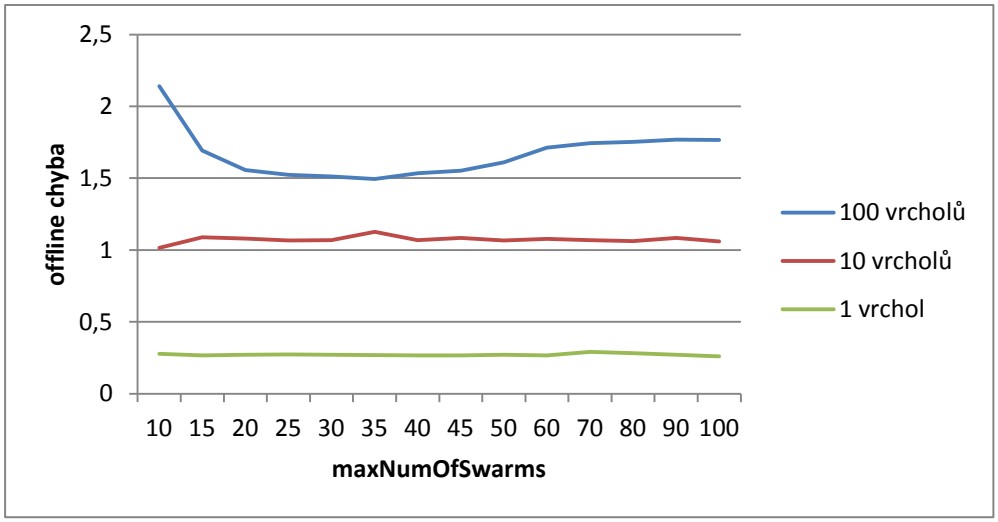
Z něj lze vyčíst, že bez anti-konvergence je algoritmus slabý na úloze se 100 vrcholy. To se dalo předpokládat, protože při $r_{conv} = 0$ nikdy nedojde na zvyšování počtu hejn, a tedy zůstane většina vrcholů nepokryta. Naopak v úloze s 10 vrcholy, protože je počet hejn na začátku nastaven na 10, tak není potřeba anti-konvergence – zvyšovat počet hejn nebo neustále nejhorší hejno inicializovat.

Protože je zlepšení při 100 vrcholech při zapnuté anti-konvergenci evidentní a v ostatních dvou případech jen nepatrné, bylo zvoleno $r_{conv} = 10$.

6.3.3 *maxNumOfSwarms*

Parametr *maxNumOfSwarms* udává maximální možný počet hejn, který se může v algoritmu objevit. Pokud je zapnuta anti-konvergence a všechna hejna konvergují, může dojít ke zvýšení počtu hejn. To, zda se počet navýší, nebo ne, určuje právě tento parametr (viz kapitola 4.4.2). Vliv tohoto parametru na výkon algoritmu lze vidět v grafu 6.4.

Jde vidět, že pokud je maximální počet hejn 10 jako na začátku algoritmu, tedy počet se nezvyšuje, algoritmus má mírně lepší výsledky při 10 vrcholech a výrazně horší při 100 vrcholech. Na 1 vrchol nemá parametr vliv, protože v takovém případě se o zvyšování počtu hejn vůbec neuvažuje (vůbec k němu nedojde). Od určité hodnoty (30 až 40) je zvyšující se počet hejn při 100 vrcholech neefektivní. Zřejmě pozitivní vliv velkého pokrytí prostoru nevynahradí negativní vliv vysokého počtu evaluací za jednu iteraci.

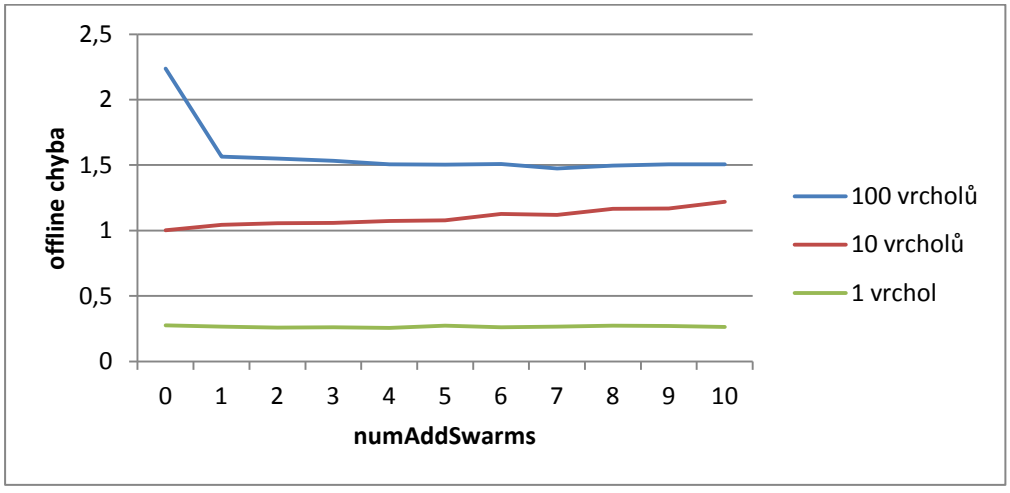


Graf 6.4 Závislost offline chyby na parametru *maxNumOfSwarms*

Jako optimální hodnota bylo zvoleno *maxNumOfSwarms* = 30.

6.3.4 numAddSwarms

Parametr *numAddSwarms* určuje, kolik hejn najednou v jedné iteraci se přidá, pokud má dojít ke zvýšení počtu hejn (všechna hejna konvergují, anti-konvergence je povolena a počet hejn není maximální) (viz kapitola 4.4.2). Vliv parametru na celkový algoritmus je vidět na grafu 6.5.



Graf 6.5 Závislost offline chyby na parametru *numAddSwarms*

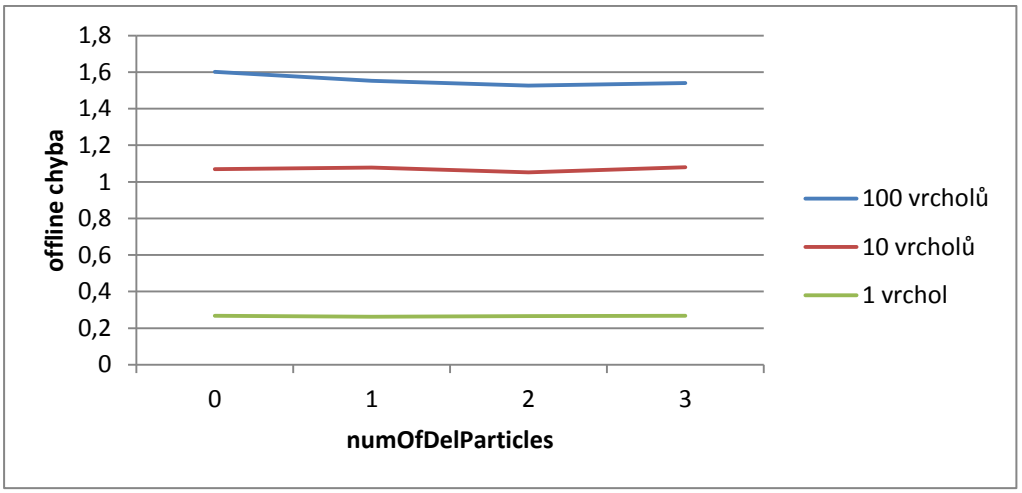
Vliv nepřidání žádného hejna na jednotlivé úlohy je stejný jako v předchozích případech (grafech), protože se dosáhne (nastavením různých parametrů) stejného (podobného) efektu – hejna se nepřidávají. Úloh s 1 vrcholem se tento parametr opět netýká. V úloze s 10 vrcholy je výkon algoritmu horší se zvyšujícím se *numAddSwarms*. Důvodem je to, že čím více se v jedné iteraci přidá nových hejn, tím více zabere iterací je snížit opět na 10. Zbytečné ohodnocování neperspektivních

oblastí zapříčiní zpomalení, a tedy zhoršení algoritmu. V úloze se 100 hejny je vliv parametru *numAddSwarms* od určité hodnoty (asi 4) zhruba stejný.

Kompromisem mezi jednotlivými úlohami je nastavení *numAddSwarms* = 4.

6.3.5 numOfDelParticles

Parametr *numOfDelParticles* určuje, o kolik se v každém hejně sníží počet částic ve chvíli, kdy je dosaženo maximálního počtu hejn (viz kapitola 4.4.2). Vliv parametru *numOfDelParticles* lze vidět v grafu 6.6.



Graf 6.6 Závislost offline chyby na parametru *numOfDelParticles*

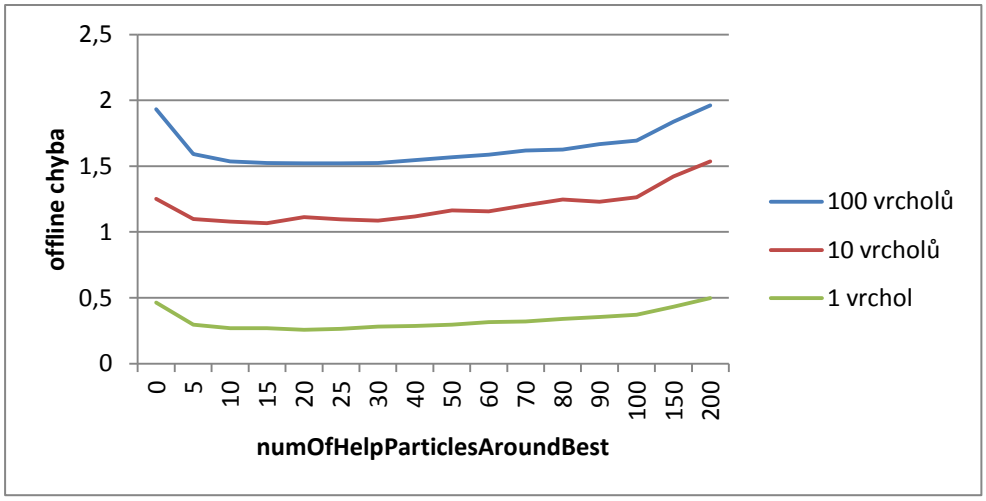
Na úlohu s 1 a 10 vrcholy parametr nemá vliv, protože k jeho použití vůbec nedojde (v prostředí nikdy nebude maximální počet hejn). V úloze se 100 vrcholy je algoritmus mírně lepší, pokud se sníží počet částic v hejně, než když se jejich počet nechá na výchozí hodnotě (5). Pokud je tedy prostředí pokryto hodně hejny, není potřeba tolik částic v každém hejně. Příliš nízký počet částic v hejně (2) již zhoršuje offline chybu.

Kvůli mírně lepším výsledkům byl nastaven parametr *numOfDelParticles* = 2.

6.3.6 numOfHelpParticlesAroundBest

Parametr *numOfHelpParticlesAroundBest* určuje, kolik částic se v každé iteraci pošle k nejlepší částici nejlepšího hejna, aby prohledaly její okolí (viz kapitola 4.4.5). Vliv parametru na výkon algoritmu lze vidět na grafu 6.7.

Pokud se nebudou posílat žádné pomocné částice kolem nejlepší částice, algoritmus se viditelné zhorší ve všech úlohách. I příliš velký počet takových částic (30 a více) má negativní vliv na výkon (nejspíše z důvodu příliš mnoha zbytečných evaluací).

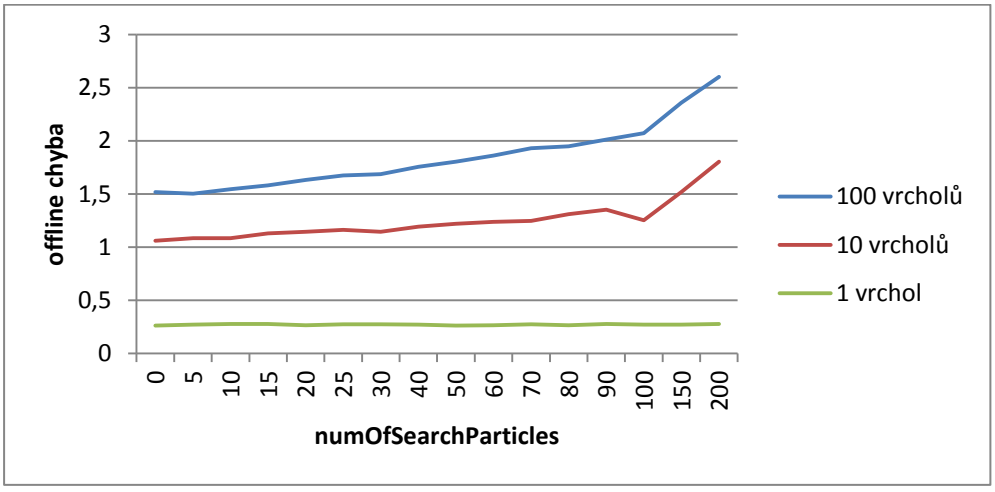


Graf 6.7 Závislost offline chyby na parametru *numOfHelpParticlesAroundBest*

Nakonec byl zvolen počet pomocných částic *numOfHelpParticlesAroundBest* = 25.

6.3.7 numOfSearchParticles

Parametr *numOfSearchParticles* určuje, kolik částic se každou iteraci pošle náhodně do prostoru, aby nalezly potenciálně dobrou oblast pro prohledání. Na tyto dobré oblasti se přesune nejhorší hejno (viz kapitola 4.4.3). Vliv tohoto parametru na celý algoritmus je vidět na grafu 6.8.



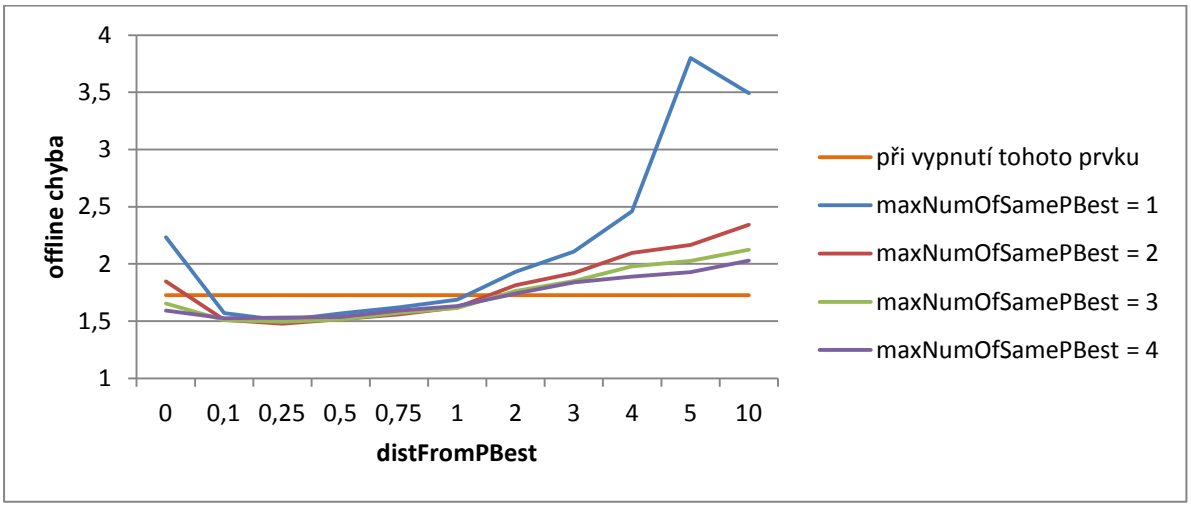
Graf 6.8 Závislost offline chyby na parametru *numOfSearchParticles*

Na úlohu s 1 vrcholem tento parametr nemá žádný vliv, protože se prohledávací částice vůbec nevyužijí. V úloze se 100 vrcholy je s navyšujícím se počtem prohledávacích částic výkonnost algoritmu nižší. Nejspíš jsou ty nejlepší oblasti prostoru již dobře obsazeny hejny, a proto není potřeba nějak moc prohledávat prostor jinak. Jde tedy vidět, že prohledávací částice nejsou nijak zvlášť přínosné vylepšení.

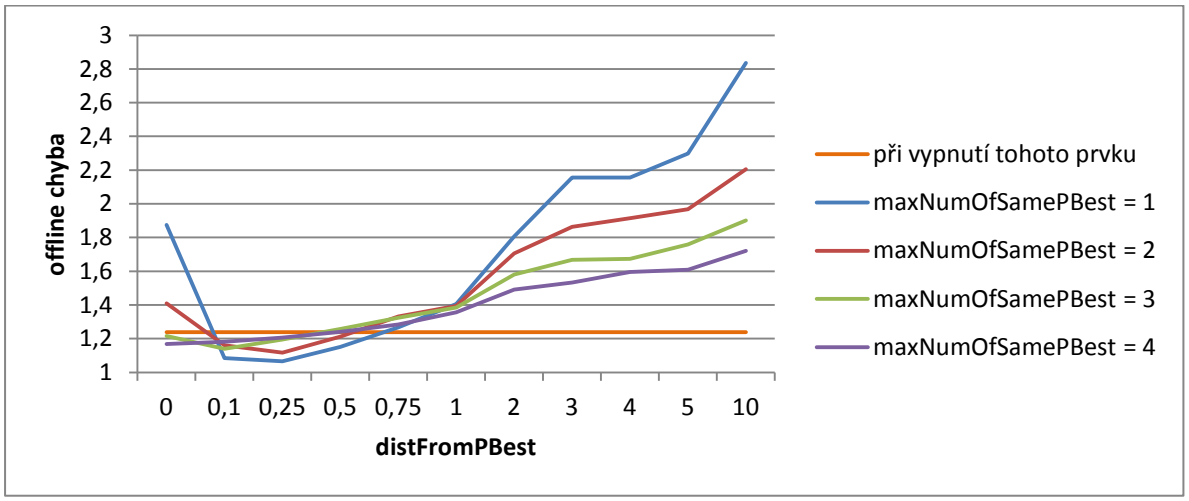
Z důvodu mírného zlepšení algoritmu v úloze se 100 vrcholy bylo zvoleno *numOfSearchParticles* = 5.

6.3.8 *maxNumOfSamePBest* a *distFromPBest*

Parametry *maxNumOfSamePBest* a *distFromPBest* spolu souvisí. Používají se pro přesun nezlepšujících se částic. Pokud se částice *maxNumOfSamePBest* iterací po sobě nezlepší, je přesunuta na náhodnou vzdálenost od své *pBest*, maximálně však do vzdálenosti *distFromPBest* (viz kapitola 4.4.6). Vliv těchto parametrů na algoritmus je vidět na grafech 6.9 (100 vrcholů), 6.10 (10 vrcholů) a 6.11 (1 vrchol).



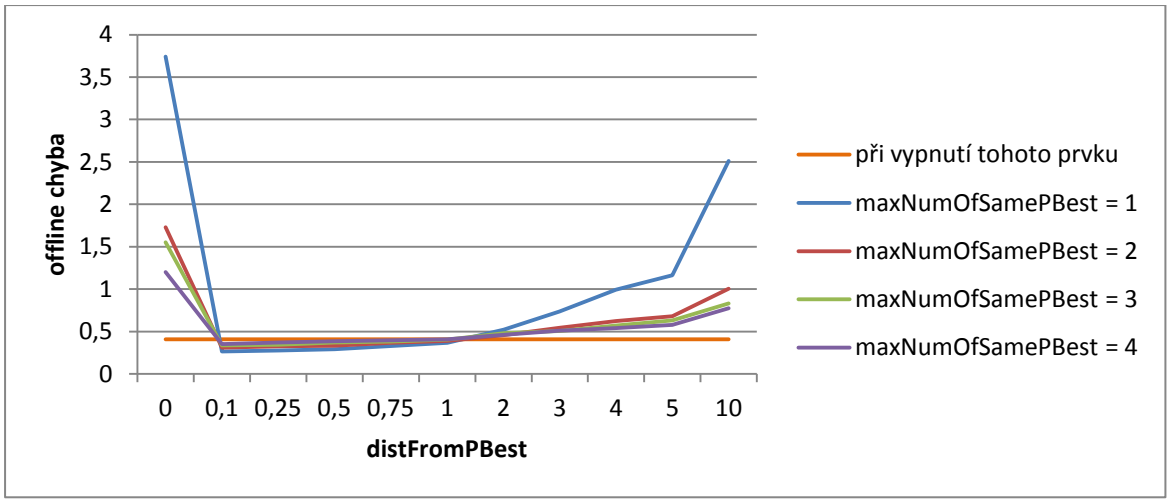
Graf 6.9 Závislost offline chyby na parametrech *maxNumOfSameBest* a *distFromPBest* v úloze se 100 vrcholy



Graf 6.10 Závislost offline chyby na parametrech *maxNumOfSameBest* a *distFromPBest* v úloze s 10 vrcholy

Ve všech úlohách lze vidět, že umístění částice na své *pBest* nebo příliš daleko od něj zhoršuje optimalizaci algoritmu. Nejlepší výsledky jsou dosaženy, pokud se s posunem částice nečeká

příliš mnoho iterací bez zlepšení. V úlohách s 1 a 10 vrcholy je nejlepší přesunout částici, pokud se i jen jedenkrát nezlepší; v úloze se 100 vrcholy jsou výsledky téměř srovnatelné, pokud se částice jedenkrát či dvakrát nezlepší. Vynechání tohoto prvku (nepřesunovat částice vůbec) má za následek horší výsledky – jde tedy o vylepšení.

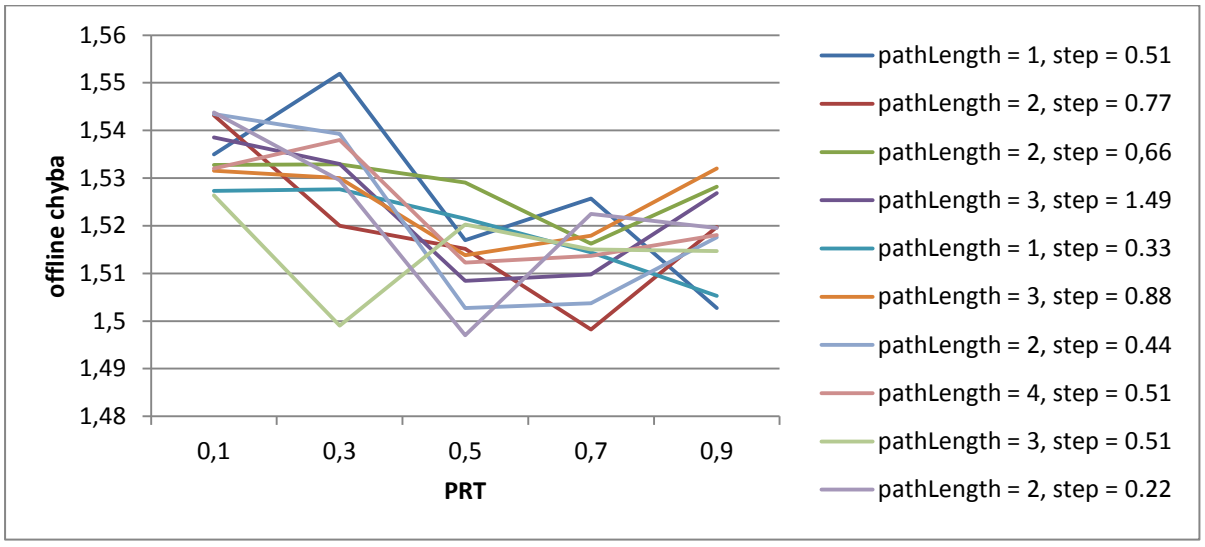


Graf 6.11 Závislost offline chyby na parametrech *maxNumOfSameBest* a *distFromPBest* v úloze s 1 vrcholem

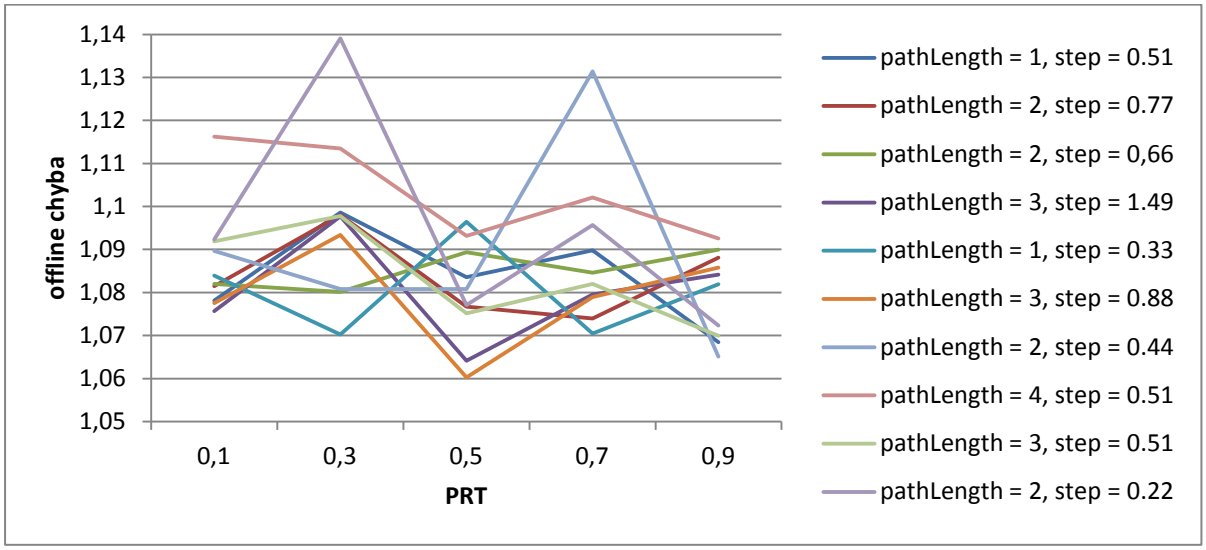
Nakonec byly parametry nastaveny na *maxNumOfSamePBest* = 1, tedy částice se musí v každé iteraci zlepšit, jinak je přesunuta, a *distFromPBest* = 0.25.

6.3.9 *PRT, pathLength* a *step*

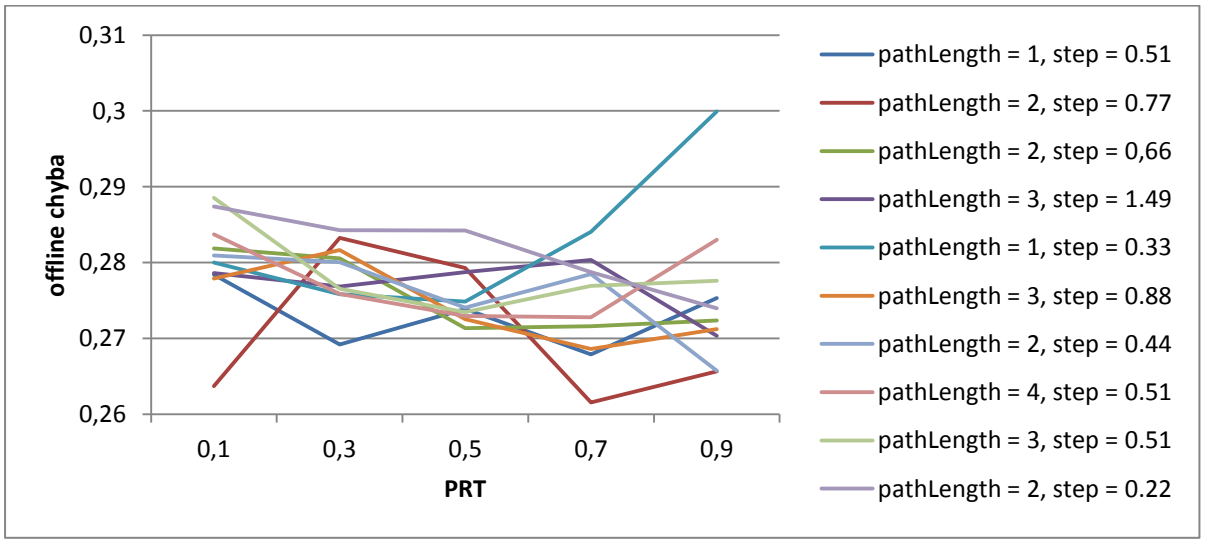
Parametry *PRT*, *pathLength* a *step* se používají v algoritmu SOMA, který je použit, pokud se nejlepší hejno určitou dobu nezlepšuje (viz kapitola 4.4.9). Vliv nastavení těchto parametrů na algoritmus AHPSO je vidět na grafech 6.12 (100 vrcholů), 6.13 (10 vrcholů) a 6.14 (1 vrchol).



Graf 6.12 Závislost offline chyby na parametrech *PRT*, *pathLenght* a *step* v úloze se 100 vrcholy



Graf 6.13 Závislost offline chyby na parametrech *PRT*, *pathLenght* a *step* v úloze s 10 vrcholy



Graf 6.14 Závislost offline chyby na parametrech *PRT*, *pathLenght* a *step* v úloze s 1 vrcholem

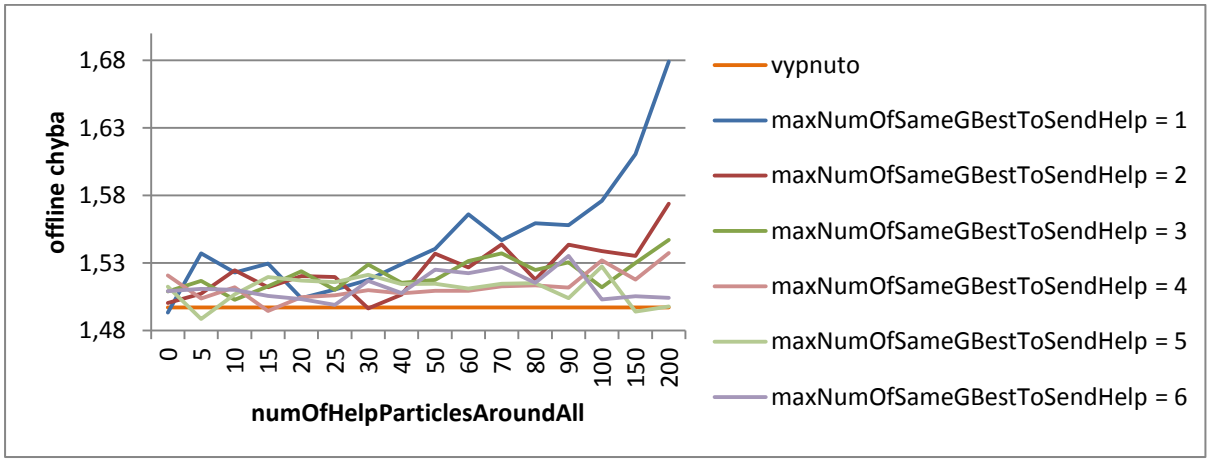
Hodnoty těchto parametrů udávají, kolik evaluací algoritmu SOMA se provede v jedné iteraci. Při dlouhé cestě a malém kroku se počet evaluací zvyšuje. Pro každou úlohu je nejlepší jiné nastavení. Nicméně většinou jsou výsledky jednotlivých nastavení ve všech úlohách hodně těsné.

Hodnoty parametrů byly zvoleny *PRT* = 0.7, *pathLength* = 2 a *step* = 0.77.

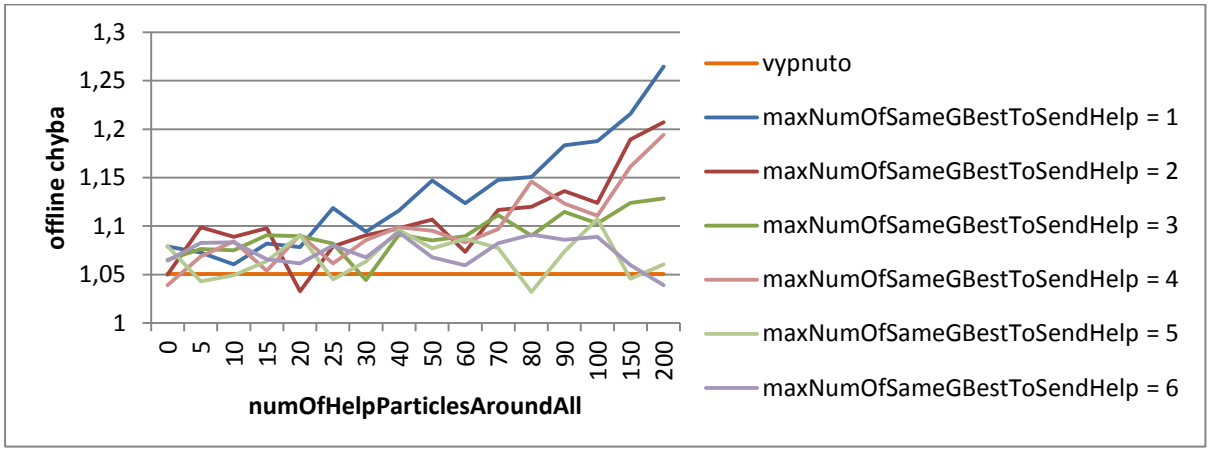
6.3.10 *maxNumOfSameGBestToSendHelp* a *numOfHelpParticlesArnoundAll*

Parametry *maxNumOfSameGBestToSendHelp* a *numOfHelpParticlesAroundAll* určují, zda a kolik částic bude vysláno ke všem částicím nejlepšího hejna, které se delší dobu nezlepšilo, aby mu

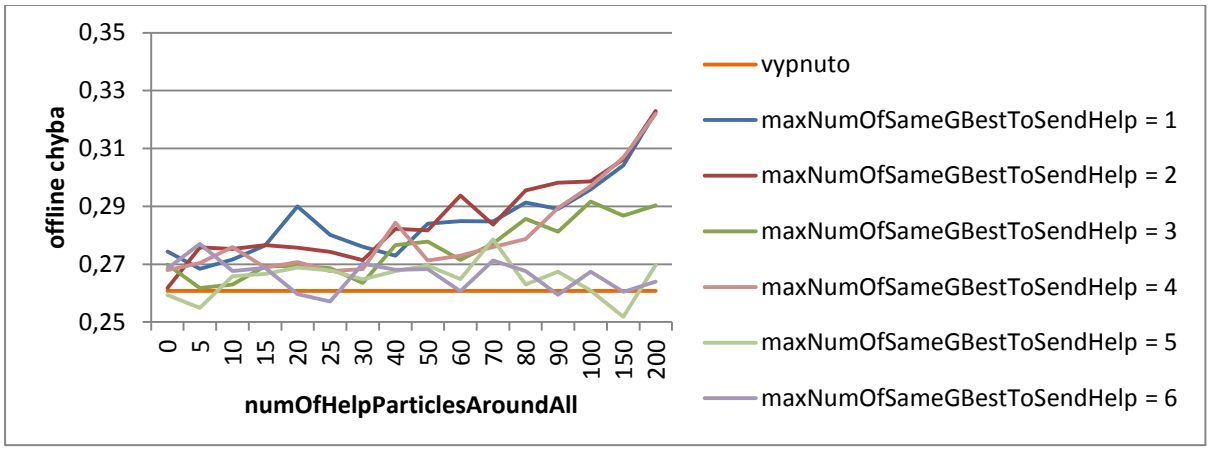
pomohly najít lepší pozici (viz kapitola 4.4.8). Vliv těchto parametrů na algoritmus je vidět na grafech 6.15 (100 vrcholů), 6.16 (10 vrcholů) a 6.17 (1 vrchol).



Graf 6.15 Závislost offline chyby na parametrech *maxNumOfSameGBestToSendHelp* a *numOfHelpParticlesAroundAll* v úloze se 100 vrcholy



Graf 6.16 Závislost offline chyby na parametrech *maxNumOfSameGBestToSendHelp* a *numOfHelpParticlesAroundAll* v úloze s 10 vrcholy



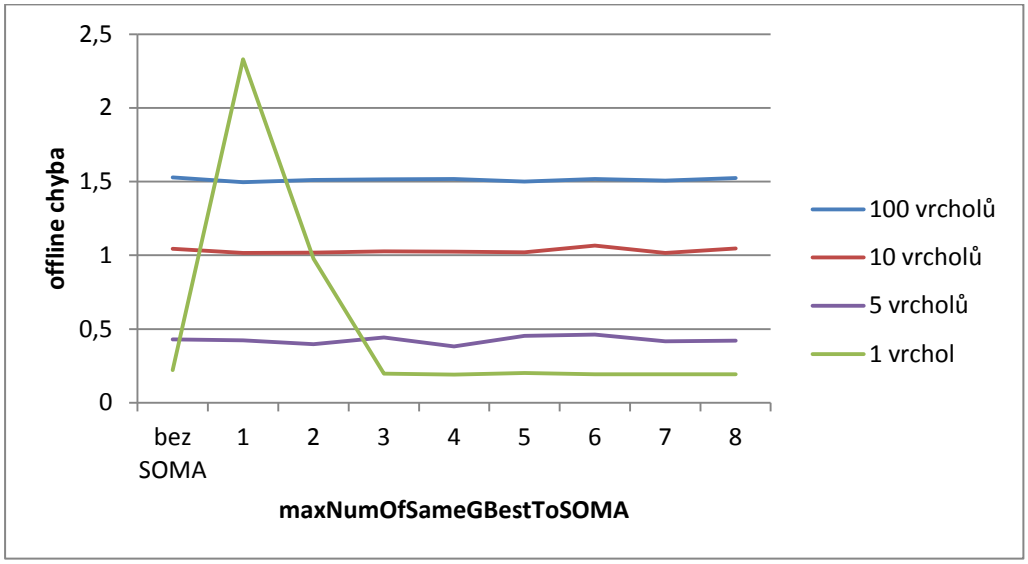
Graf 6.17 offline chyby na parametrech *maxNumOfSameGBestToSendHelp* a *numOfHelpParticlesAroundAll* v úloze s 1 vrcholem

Lze říci, že čím častěji se posílá velké množství pomocných částic, tím jsou výsledky horší. Množství poslaných částic nemá radikální vliv, pokud se posílají, až pokud se hejno delší dobu nezlepší. Je to zřejmě z toho důvodu, že k takovým situacím příliš často nedochází, tedy hejno se téměř pořád zlepšuje. Odstraněním tohoto prvku se algoritmus jen nepatrně zhorší, nejde tedy o nějaké zásadní vylepšení.

S ohledem na všechny 3 úlohy bylo nastaveno *maxNumOfSameGBestToSendHelp* = 5 a *numOfHelpParticlesAroundAll* = 5.

6.3.11 maxNumOfSameGBestToSOMA

Parametr *maxNumOfSameGBestToSOMA* určuje, kolik iterací se nejlepší hejno musí nezlepšit, aby došlo k přepnutí na SOMA fázi, tedy že v příští iteraci provede nejlepší hejno SOMA algoritmus (viz kapitola 4.4.8). Vliv tohoto parametru lze vidět v grafu 6.18, kde je přidána i úloha s 5 vrcholy.



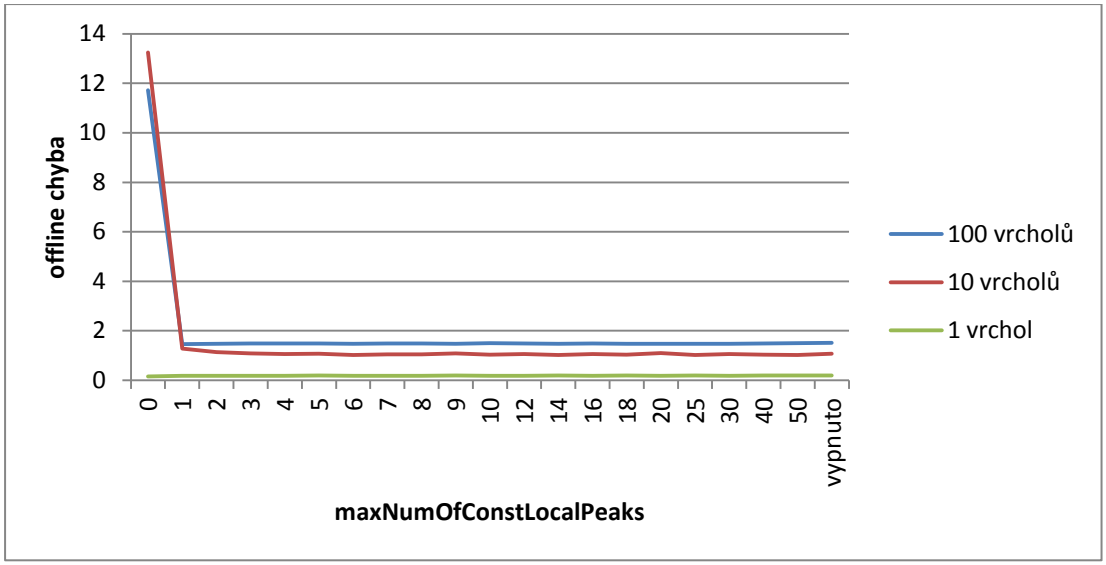
Graf 6.18 Závislost offline chyby na parametru *maxNumOfSameGBestToSOMA*

Drasticky negativní vliv má časté (každou iteraci, kdy nedojde ke zlepšení) provádění SOMA algoritmu na úlohu s 1 vrcholem. V úlohách s 10 a 100 vrcholy je mírně lepší, pokud se SOMA algoritmus provede pokaždé, kdy nedojde ke zlepšení. Naopak v úlohách s 1 a 5 vrcholy je lehce lepší nastavení parametru na 4.

Z výše popsaných skutečností je parametr nastaven na *maxNumOfSameGBestToSOMA* = 1, a pokud dojde k odstranění i jen jednoho hejna, je hodnota parametru změněna na *maxNumOfSameGBestToSOMA* = 4.

6.3.12 maxNumOfConstLocalPeaks

Parametr *maxNumOfConstLocalPeaks* určuje, kolik iterací za sebou se nesmí hejno kolem nějakého vrcholu stát nejlepším hejnem, aby bylo na jednu iteraci uspáno – krok algoritmu by se u tohoto hejna neprovedl (viz kapitola 4.4.7). Vliv tohoto parametru je vidět na grafu 6.19.



Graf 6.19 Závislost offline chyby na parametru *maxNumOfConstLocalPeaks*

V úlohách s 10 a 100 vrcholy má algoritmus při nastavení parametru na 0 drasticky horší výsledky. Je to z důvodu uspání všech hejn po celou dobu provádění algoritmu, tedy optimalizace se účastní jen jedno hejno (proto to nemá na úlohu s 1 vrcholem vliv). Vypnutí tohoto prvku má mírně horší výsledky. Výsledky jednotlivých nastavení se liší maximálně o desetinu.

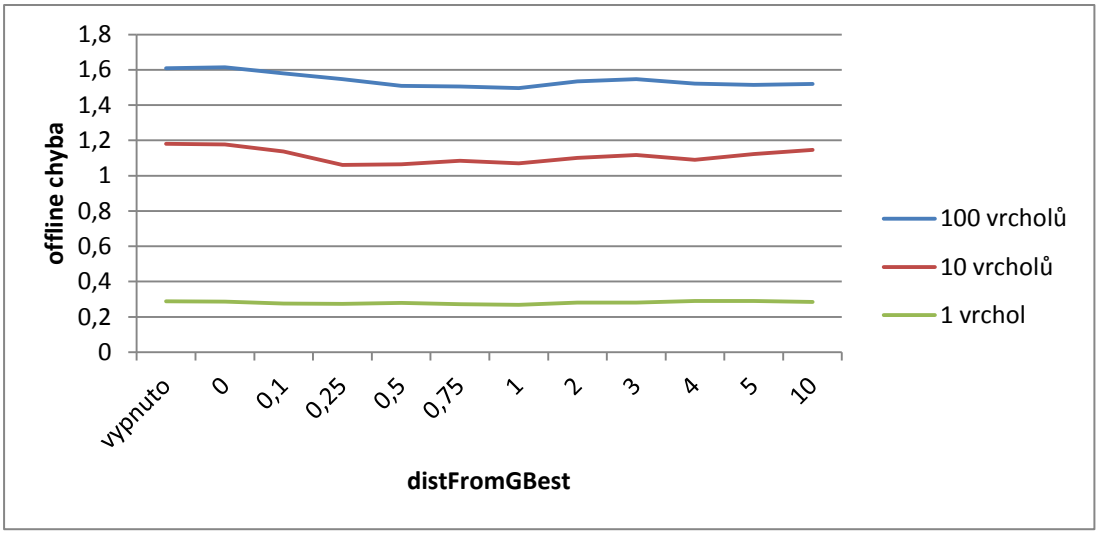
Nakonec bylo zvoleno *maxNumOfConstLocalPeaks* = 13.

6.3.13 distFromGBest

Parametr *distFromGBest* určuje, na jakou maximální vzdálenost od nejlepší částice ve svém hejně se náhodně přemístí každá částice po změně funkce (viz kapitola 4.4.10). Vliv tohoto parametru je vidět na grafu 6.20.

Mírně lepší výsledky algoritmus dosahuje, když je parametr nastaven na hodnotu v rozmezí 0.25 – 1. Při nepřesunování částic po každé změně funkce má algoritmus mírně horší výkon. I přesun na maximální vzdálenost 10 je lepší než nepřesunovat vůbec. To je z toho důvodu, že bez přesunutí nemá hejno dostatečnou diverzitu, aby mohlo adekvátně reagovat na změnu funkce.

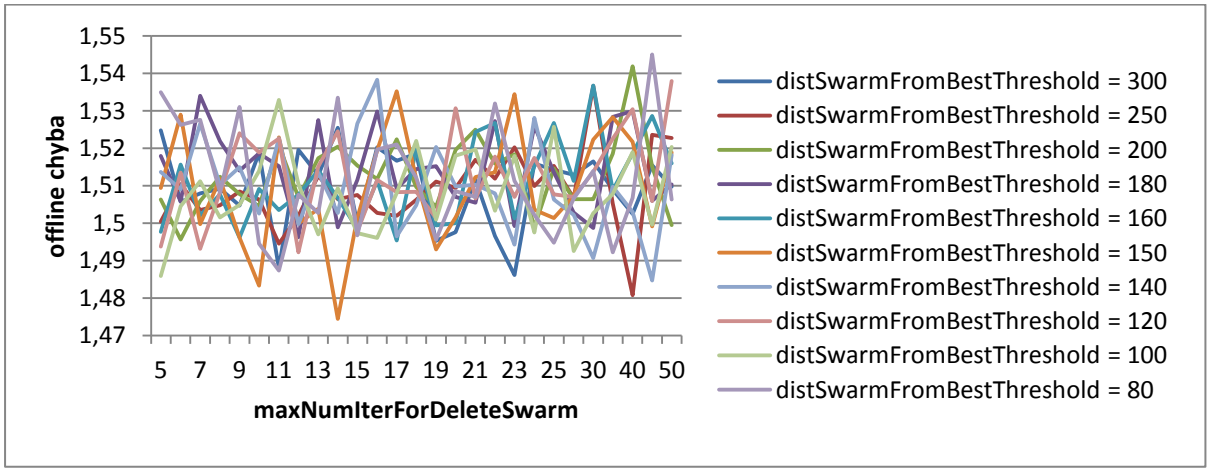
Parametr byl nastaven na *distFromGBest* = 1.



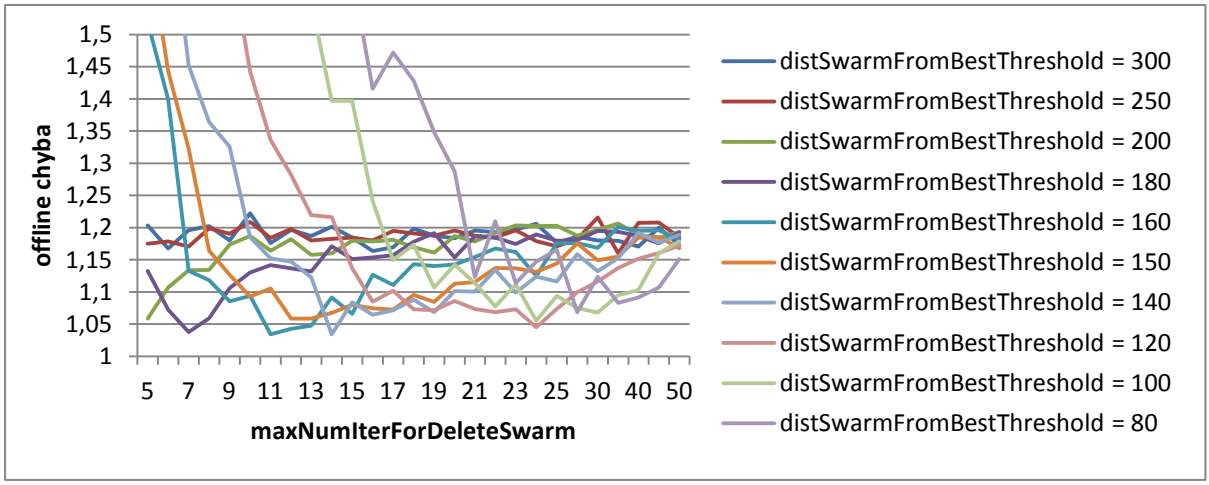
Graf 6.20 Závislost offline chyby na parametru *distFromGBest*

6.3.14 *distSwarmFromBestThreshold* a *maxNumIterForDeleteSwarm*

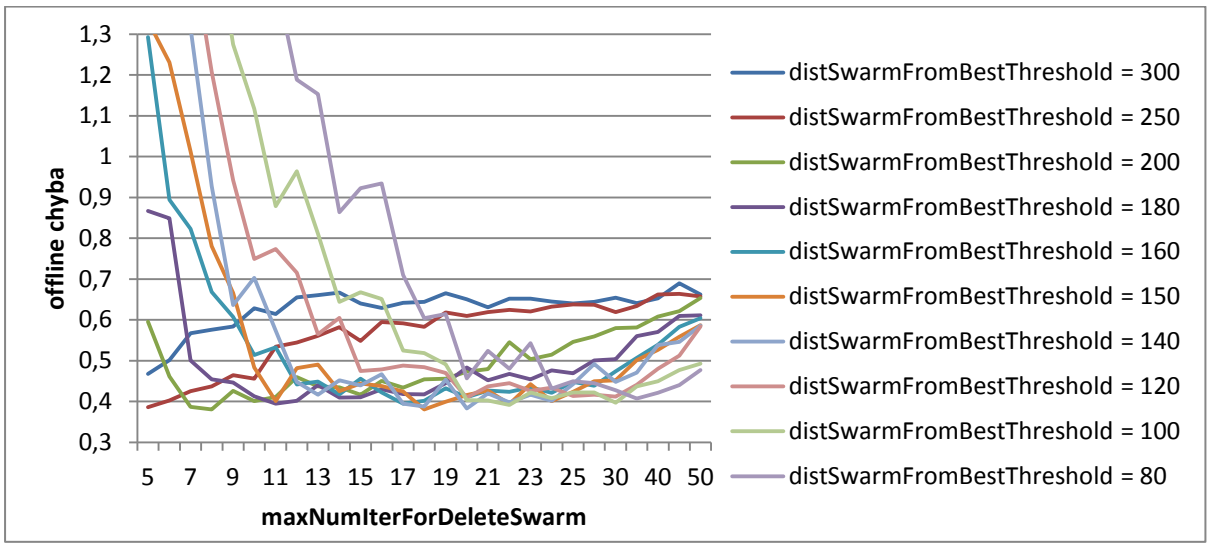
Parametry *distSwarmFromBestThreshold* a *maxNumIterForDeleteSwarm* určují, zda a kdy bude nějaké hejno odstraněno. Pokud bude rozdíl fitness hodnot zkoumaného hejna a nejlepšího hejna po dostatečně dlouhou dobu dostatečně velký, bude toto hejno odstraněno (viz kapitola 4.4.4). Vliv těchto parametrů je vidět na grafech 6.21 (100 vrcholů), 6.22 (10 vrcholů), 6.23 (5 vrcholů) a 6.24 (1 vrchol). Testováno bylo i 5 vrcholů, protože ony jsou přímo ovlivněny snižováním počtu hejn.



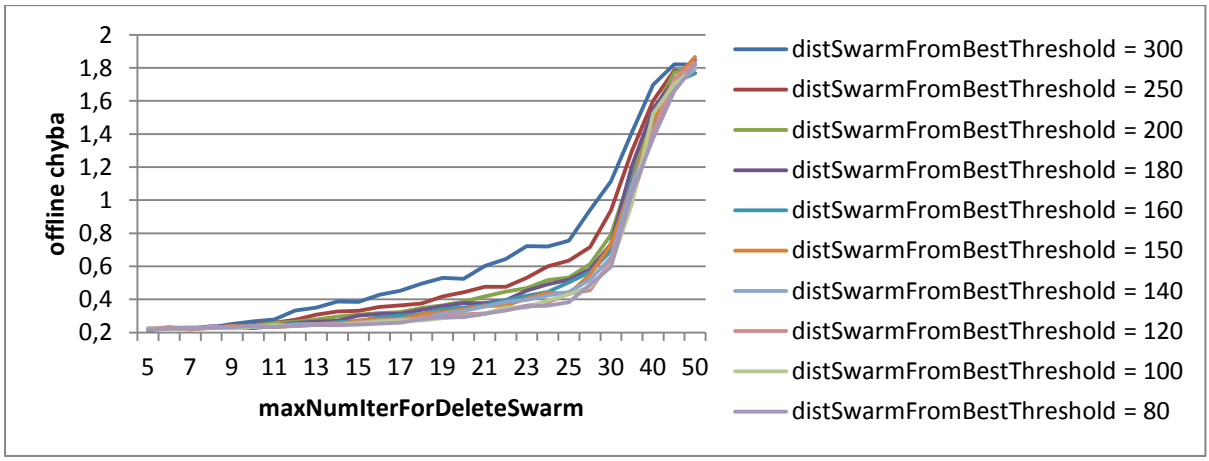
Graf 6.21 Závislost offline chyby na parametrech *distSwarmFromBestThreshold* a *maxNumIterForDeleteSwarm* v úloze se 100 vrcholy



Graf 6.22 Závislost offline chyby na parametrech *distSwarmFromBestThreshold* a *maxNumIterForDeleteSwarm* v úloze s 10 vrcholy



Graf 6.23 Závislost offline chyby na parametrech *distSwarmFromBestThreshold* a *maxNumIterForDeleteSwarm* v úloze s 5 vrcholy



Graf 6.24 Závislost offline chyby na parametrech *distSwarmFromBestThreshold* a *maxNumIterForDeleteSwarm* v úloze s 1 vrcholem

Výsledné hodnoty v úloze se 100 vrcholy nejsou nijak jednoznačné, protože by zde nemělo docházet ke snižování počtu hejn. V úloze s 5 a 10 vrcholy jde vidět, že při velkém prahu nedochází ke snižování počtu hejn a výsledná offline chyba je podobná.

Dále pro 10 vrcholů platí, že při nižším prahu je potřeba více iterací, aby byly výsledky srovnatelné s těmi, ve kterých je práh vysoký. Příliš nízký práh totiž způsobuje příliš rychlé snížení počtu hejn, tedy mnohem horší výsledky. Počet iterací nesmí být v této úloze moc velký, jinak nedojde po případném zvýšení počtu hejn k jeho snížení, a tedy k vypnutí anti-konvergence a prohledávacích částic.

Pro úlohy s 5 vrcholy platí, že při nízkém prahu a malém počtu iterací dojde k odstranění téměř všech hejn a algoritmus se výrazně zhorší. Čím větší práh, tím je možné nastavit méně iterací. S narůstajícím počtem iterací začíná docházet k tomu, že se žádná hejna neodstraní, a výsledky se tedy blíží k těm s vysokým prahem.

V úlohách s 1 vrcholem pro každý práh platí, že s rostoucím počtem iterací roste offline chyba. A čím menší práh, tím lepší výsledky. To je pro 1 vrchol pochopitelné, protože je snaha co nejdříve odstranit všechna nepotřebná hejna a toho se dosáhne co nejnižším prahem a počtem iterací.

Podle rovnice (4.4) by se práh nastavil na hodnotu asi 135. S ohledem na úlohy s různými počty vrcholů a spíše opatrnějším odstraňováním hejn bylo nastaveno *distSwarmFromBestThreshlod* = 150 a *maxNumIterForDeleteSwarm* = 14.

6.4 Skutečná adaptace počtu hejn na počet vrcholů

Jak bylo popsáno v kapitole 4.4.11, počet hejn se v průběhu vykonávání algoritmu AHPSO přizpůsobuje počtu vrcholů. AHSPO informaci o počtu vrcholů nepotřebuje, sám si ji pokusí odvodit. Ideální stav je, když se na konci algoritmu rovná počet hejn počtu vrcholů. To platí díky parametru *maxNumOfSwarms* jen do určitého počtu vrcholů (viz kapitola 4.4.2 a 4.4.11). Pro počet vrcholů větších než *maxNumOfSwarms* se již počet hejn nezvyšuje a zůstává *maxNumOfSwarms* = 30 (viz kapitola 6.3.3). Úspěšnost adaptace počtu hejn na počet vrcholů je vidět v tabulce 6.2.

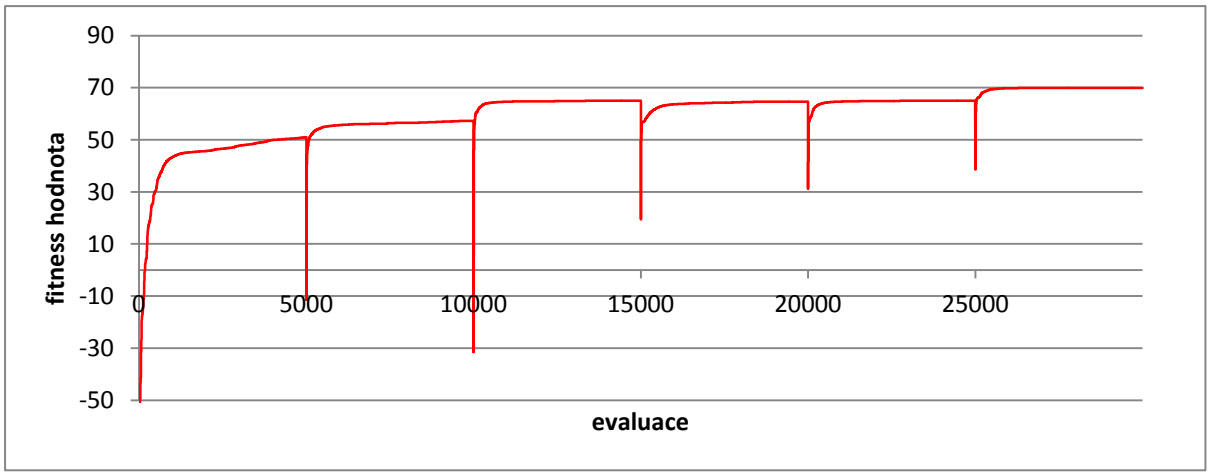
Z tabulky je vidět, že se počet hejn relativně dobře adaptuje na počet vrcholů.

počet vrcholů	1	2	3	4	5	10	15	20	25	30	50
počet hejn	1.00	1.98	3.00	4.00	4.98	9.16	17.26	22.00	25.36	29.06	30.00

Tabulka 6.2 Skutečný počet hejn na konci algoritmu po adaptaci na počet vrcholů (průměr z 50 běhů)

6.5 Průběh fitness hodnoty v čase

Pro ilustraci chování algoritmu AHPSO je v grafu 6.25 zobrazen průběh fitness hodnoty nejlepší částice v čase (průměr z 50 běhů). Průběh algoritmu je očekávaný, každých 5000 evaluací dojde ke změně funkce a tedy ke zhoršení fitness hodnoty. Dynamika optimalizace se tedy neliší od jiných algoritmů na dynamické problémy.



Graf 6.25 Průběh fitness hodnoty nejlepší částice v čase v úloze s 10 vrcholy (prvních 30000 evaluací, průměr z 50 běhů)

7 Porovnání s jinými PSO variantami

Algoritmus AHPSO byl porovnán s několika jinými variantami PSO algoritmu na dynamické problémy. Pro porovnání byly vybrány nejlepší veřejně dostupné varianty PSO: mQPSO 10(5+5^q) (viz kapitola 3.5.18) – více-rojový model, ze kterého AHPSO vycházel, obsahující 10 hejn a v každém hejně 5 normálních a 5 kvantových částic [30]; PSABC (viz kapitola 3.6.7) – hybrid PSO a ABC [44]; FCMPSO – PSO s fuzzy shlukováním [53]; Kamosi – více-rojový přístup s rodičem a potomky [52]; FTMPSO – více-rojový přístup s upravenými rovnicemi rychlosti a pozice [49]. Všechny tyto porovnávané varianty jsou relativně nové (rok 2012 – 2013).

Byly porovnávány offline chyby 8 úloh, každá s jiným počtem vrcholů (1, 5, 10, 20, 30, 50, 100 a 200), s různou rychlostí změny funkce (po 500, 1000, 2500, 5000 a 10000 iteracích). Celkem byl tedy algoritmus AHPSO porovnán s nejlepšími nalezenými algoritmy na 40 dynamických úlohách. Výsledné offline chyby všech algoritmů na jednotlivých úlohách jsou vidět v tabulkách 7.1 – 7.5. Tučně jsou zvýrazněny nejlepší hodnoty. U algoritmu PSABC bylo autory nastaveno *maximalWidth* = 5 (místo standardního 12), což se projevuje mírně lepšími výsledky.

počet vrcholů	AHPSO	mQPSO 10(5+5^q)	PSABC	FCMPSO	Kamosi	FTMPSO
1	**1.14**	33.67	2.77	4.81	5.46	1.76
5	**2.61**	11.91	-	4.95	5.48	2.93
10	3.79	9.62	**3.42**	5.16	5.95	3.91
20	5.40	9.07	**3.12**	5.81	6.45	4.83
30	5.92	8.80	**3.69**	6.03	6.60	5.05
50	5.12	8.72	**3.22**	5.95	7.04	4.98
100	5.45	8.54	**3.01**	6.08	7.39	5.31
200	5.74	8.19	**3.16**	6.20	7.52	5.52

Tabulka 7.1 Offline chyby jednotlivých algoritmů pří změně funkce po 500 evaluacích

počet vrcholů	AHPSO	mQPSO $10(5+5^q)$	PSABC	FCMPSO	Kamosi	FTMPSO
1	**0.69**	18.60	1.68	2.72	2.90	0.89
5	**1.40**	6.56	-	2.99	3.35	1.70
10	2.57	5.71	3.23	3.87	3.94	**2.36**
20	3.33	5.85	3.40	4.13	4.33	**3.01**
30	3.73	5.81	3.28	4.12	4.41	**3.06**
50	3.14	5.87	**2.67**	4.11	4.57	3.29
100	3.32	5.83	**3.08**	4.26	4.77	3.63
200	3.20	5.54	**3.01**	4.21	4.76	3.74

Tabulka 7.2 Offline chyby jednotlivých algoritmů pří změně funkce po 1000 evaluacích

počet vrcholů	AHPSO	mQPSO $10(5+5^q)$	PSABC	FCMPSO	Kamosi	FTMPSO
1	0.31	7.64	**0.01**	1.06	1.10	0.39
5	**0.72**	3.26	-	1.55	1.68	0.91
10	1.48	3.12	**0.18**	2.17	2.33	1.21
20	1.79	3.58	**0.79**	2.51	2.79	1.66
30	2.07	3.63	2.23	2.61	2.88	**1.87**
50	**1.76**	3.63	3.12	2.66	2.97	2.09
100	**1.99**	3.58	3.02	2.62	3.00	2.22
200	**1.84**	3.30	3.14	2.64	2.99	2.22

Tabulka 7.3 Offline chyby jednotlivých algoritmů pří změně funkce po 2500 evaluacích

počet vrcholů	AHPSO	mQPSO 10(5+5^q)	PSABC	FCMPSO	Kamosi	FTMPSO
1	**0.18**	3.82	2.25	0.53	0.56	**0.18**
5	**0.41**	1.90	-	1.05	1.06	0.47
10	1.00	1.91	2.13	1.31	1.51	**0.67**
20	1.08	2.56	2.07	1.69	1.89	**0.93**
30	1.42	2.68	1.88	1.78	2.03	**1.14**
50	**1.27**	2.63	1.91	1.95	2.08	1.32
100	**1.46**	2.52	1.89	1.95	2.14	1.61
200	**1.38**	2.36	1.87	1.90	2.11	1.67

Tabulka 7.4 Offline chyby jednotlivých algoritmů pří změně funkce po 5000 evaluacích

počet vrcholů	AHPSO	mQPSO 10(5+5^q)	PSABC	FCMPSO	Kamosi	FTMPSO
1	0.13	1.90	2.67	0.25	0.27	**0.09**
5	**0.27**	1.03	-	0.57	0.70	0.31
10	0.76	1.10	0.90	0.82	0.97	**0.43**
20	0.79	1.84	0.66	1.23	1.34	**0.56**
30	1.03	2.00	0.77	1.39	1.43	**0.69**
50	1.06	1.99	**0.81**	1.46	1.47	0.86
100	1.32	1.85	**0.83**	1.38	1.50	1.08
200	1.13	1.71	**0.85**	1.36	1.48	1.13

Tabulka 7.5 Offline chyby jednotlivých algoritmů pří změně funkce po 10000 evaluacích

Ze 40 testovaných úloh byl AHPSO nejlepší 14x, PSABC 15x a FTMPSO 12x. Navíc byl AHPSO druhý nejlepší na 13 úlohách a třetí nejlepší rovněž v 13 případech. AHPSO tedy dosahoval podobných hodnot offline chyb jako nejlepší dostupné algoritmy na dynamické problémy.

AHPSO překonal algoritmus QPSO, ze kterého vycházel, na všech testovacích úlohách. Změny provedené na QPSO tedy byly ku prospěchu. AHPSO také překonal na všech úlohách algoritmy FCMPSO a Kamosi. V porovnání s algoritmem FTMPSO jsou výsledky vyrovnané, v 18 případech byl lepší AHPSO, v 18 případech FTMPSO a ve 2 úlohách byly dosaženy stejné výsledky. V porovnání s algoritmem PSABC byl AHPSO lepší 17x, PSABC 18x a u 5 úloh nebylo možné srovnání.

8 Závěr

V této práci byla popsána optimalizace na bázi částicových hejn (PSO). Po nezbytném uvedení do problematiky optimalizace byl detailně vysvětlen princip chování PSO. Především je popsáno, z čeho algoritmus vychází, jak se jedinci pohybují v prostoru kandidátních řešení a jaký je význam jednotlivých nastavitelných parametrů, včetně typů sousedství a topologií. Přiblíženy jsou i jednoduché základní modifikace prvního PSO přidáním váhy nebo omezujícího faktoru.

Je uvedeno, jaký vliv mají tyto parametry, sousedství a základní modifikace na konvergenci algoritmu a udržení diverzity v prohledávacím prostoru. Aby docházelo ke kompromisu mezi konvergencí a diverzitou je potřeba udržovat rovnováhu mezi explorativním a exploitativním chováním algoritmu.

Protože je PSO robustní a flexibilní, dokáže relativně rychle nalézt kvalitní řešení na odlišných typech problémů. Drobnou nevýhodou tohoto přístupu může být přítomnost prvku náhody, který nemůže zaručit správný výsledek v každém běhu algoritmu; to je ale problém všech evolučních výpočetních technik. Řešením je vícenásobné spuštění algoritmu. Na druhou stranu právě náhoda v kombinaci s determinismem, který odráží inteligentní chování celého hejna, vytváří účinný a efektivní algoritmus.

Dále byla vytvořena rešerše variant algoritmu PSO. Byly vybrány a více či méně popsány úspěšné varianty. Některé zahrnují jen drobné úpravy původního algoritmu, jiné naopak drasticky mění celé rovnice a sousedství a zavádějí nové parametry a způsoby chování; nicméně všechny varianty se drží základního principu a idey. Různé varianty byly vyvinuty, aby dosahovaly lepších výsledků na různé typy funkcí. Některé varianty jsou vhodné na multimodální, některé na více-kriteriální a některé na dynamické problémy; jiné varianty slouží k udržení diverzity hejna po celou dobu provádění. Jsou popsány i varianty využívající více hejn, které spolu komunikují.

Také byly popsány některé hybridizace PSO, které kombinují principy PSO s jinými evolučními výpočetními technikami. Cílem kombinací podobných či různých přístupů je zlepšit výkon celého algoritmu.

Praktická část práce se věnuje navržení nové varianty algoritmu PSO na dynamické problémy, který byl pojmenován AHPSO. Jde o více-rojový algoritmus vycházející ze známého algoritmu QPSO. Z původního algoritmu přebírá techniky vyloučení a anti-konvergence, kterou upravuje. Dále zavádí do algoritmu další techniky jako lokální prohledávání, náhodné prohledávání, zvláštní přesun částic a algoritmus SOMA. Některé tyto techniky našly inspiraci v jiných algoritmech (např. v algoritmu ABC).

Významným prvkem AHPSO je adaptace počtu hejn na počet vrcholů. Algoritmus v průběhu svého provádění zvyšuje nebo snižuje počet svých hejn na základě předpokládaného počtu vrcholů. Algoritmus informaci o počtu vrcholů vůbec nepoužívá, tedy nemusí ji znát a prakticky nezná,

a přesto je schopen relativně úspěšně změnit počet hejn tak, aby se přizpůsobil počtu vrcholů. V tom je hlavní rozdíl oproti QPSO, který tento údaj o počtu vrcholů využívá. Díky adaptaci počtu hejn na počet vrcholů dosahuje algoritmus dobré výsledky na úlohách s 1 i např. 100 vrcholy beze změny hodnoty jakéhokoli jeho parametru a bez znalosti tohoto počtu vrcholů.

Nevýhodou algoritmu AHPSO může být relativně velké množství nastavitelných parametrů. Na druhou stranu všechny parametry mají své opodstatnění. Jiné podobně úspěšné algoritmy mají rovněž více nastavitelných parametrů.

Algoritmus byl porovnán s výchozím algoritmem i nejlepšími volně dostupnými (relativně novými) algoritmy na dynamické problémy (celkem porovnán s 5 algoritmy). Jako testovací úlohy byly zvoleny úlohy s pohybujícími se vrcholy (Moving peaks benchmark), které využívaly všechny algoritmy na dynamické problémy pro své testování. Sledované kritérium byla offline chyba, což je standardní metrika k určování kvality algoritmu v dynamických prostředích.

AHPSO dosahoval lepších výsledků na všech testovaných úlohách než algoritmus QPSO, z něhož vzešel, takže jej naprosto překonal. V porovnání s ostatními algoritmy dosahoval AHPSO nejlepších výsledků asi v třetině testovacích úloh. Všechny výsledky AHPSO byly zhruba v těch rozmezích, ve kterých se pohybovaly hodnoty zbylých algoritmů.

Další vývoj by mohl směřovat ke zlepšení algoritmu. Snižování a zvyšování počtu hejn je jen omezené. Algoritmus zatím není schopen reagovat na dynamickou změnu počtu vrcholů v průběhu provádění algoritmu, protože pokud sníží počet hejn, už jej nemůže navýšit (naopak to neplatí). Další možností je zjednodušit algoritmus – snížit počet jeho nastavitelných parametrů odstraněním těch prvků, které mají jen úplně minimální vliv na offline chybu.

Algoritmus je vhodný na dynamické problémy, kdy si udržuje informaci jen o úplně nejlepší pozici. Vývoj by mohl spočívat v udržování informací o všech lokálních extrémech s cílem řešení statických multimodálních problémů. V takovém případě by nemohl být omezen maximální počet hejn a musely by se některé prvky algoritmu odstranit (přesun částic, uspání hejn na lokálních extrémech).

Literatura

[1] ENGELBRECHT, A. P. *Fundamentals of Computional Swarm Intelligence*. Chichester: John Wiley, 2005. 599 s. ISBN 978-0-470-09191-3.

[2] LIANG, J.J., A.K. QIN, P.N. SUGANTHAN a S. BASKAR. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evolutionary Computation* [online]. 2006, vol. 10, issue 3, s. 281-295 [cit. 2014-01-04]. DOI: 10.1109/TEVC.2005.857610. Dostupné z: http://dsp.szu.edu.cn/pso/ispo/download/Comprehensive%20learning%20particle%20swarm%20optimizer%20for%20global%20optimization%20of%20multimodal%20functions.pdf

[3] ZELINKA, I. *Evoluční výpočetní techniky: principy a aplikace*. 1. české vyd. Praha: BEN, 2009. 534 s. ISBN 978-80-7300-218-3.

[4] VESTERSTROM, J. S., J. RIGET. *Particle Swarms: Extensions for Improved Local, Multi-modal, and Dynamic Search in Numerical Optimization*. Master's thesis, EVALife, Dept. of Computer Science, Aarhus Universitet, 2002.

[5] MENDES, R., J. KENNEDY, J. NEVES a S. BASKAR. The Fully Informed Particle Swarm: Simpler, Maybe Better. *IEEE Transactions on Evolutionary Computation* [online]. 2004, vol. 8, issue 3, s. 204-210 [cit. 2014-01-04]. DOI: 10.1109/TEVC.2004.826074. Dostupné z: http://www.cpdee.ufmg.br/~joao/CE/ArtigosPSO/FI_PSO.pdf

[6] ZITZLER, E. *Evolutionary multi-criterion optimization*. Vyd. 1. Berlin: Springer, 2001, 712 s. ISBN 35-404-1745-1.

[7] POLI, R., J. KENNEDY a T. BLACKWELL. Particle swarm optimization. *Swarm Intelligence* [online]. 2007-10-17, vol. 1, issue 1, s. 33-57 [cit. 2014-01-05]. DOI: 10.1007/s11721-007-0002-0. Dostupné z: http://cswww.essex.ac.uk/staff/poli/papers/PoliKennedyBlackwellSI2007.pdf

[8] KENNEDY, J., R. C. EBERHART a Y. SHI. *Swarm intelligence*. San Francisco: Morgan Kaufmann, 2001, xxvii, 512 s. ISBN 15-586-0595-9.

[9] EBERHART, R. a J. KENNEDY. A new optimizer using particle swarm theory. *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science* [online]. IEEE, 1995, s. 39-43 [cit. 2014-01-05]. DOI: 10.1109/MHS.1995.494215. Dostupné z: http://www.ppgia.pucpr.br/~alceu/mestrado/aula3/PSO_2.pdf

[10] CLERC, M. a J. KENNEDY. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*

[online]. 2002, vol. 6, issue 1, s. 58-73 [cit. 2014-01-06]. DOI: 10.1109/4235.985692. Dostupné z: https://bitbucket.org/12er/pso/src/e1371b5bba75fcefcc0fe49b5ed21c82fe223093/doc/literature/Parameter%20Tuning/The%20Particle%20Swarm%20-%20Explosion,%20Stability,%20and%20Convergence%20in%20a%20Multidimensional%20Complex%20Space.pdf

[11] ABIDO, A. A. Particle swarm optimization for multimachine power system stabilizer design. *2001 Power Engineering Society Summer Meeting. Conference Proceedings (Cat. No.01CH37262)* [online]. IEEE, 2001, s. 1346-1351 [cit. 2014-01-06]. DOI: 10.1109/PESS.2001.970272. Dostupné z: http://faculty.kfupm.edu.sa/EE/mabido/Papers/C23.pdf

[12] SHI, Y. a R. EBERHART. A modified particle swarm optimizer. *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)* [online]. IEEE, 1998, s. 69-73 [cit. 2014-01-06]. DOI: 10.1109/ICEC.1998.699146. Dostupné z: https://bitbucket.org/12er/pso/src/852455a777ece12ff6ae9d7b533c2a0f61f808bb/doc/literature/Variants/alter%20particle%20movement/A%20modified%20particle%20swarm%20optimizer.pdf

[13] RATNAWEERA, A., S. K. HALGAMUGE a H. C. WATSON. Self-Organizing Hierarchical Particle Swarm Optimizer With Time-Varying Acceleration Coefficients. *IEEE Transactions on Evolutionary Computation* [online]. 2004, vol. 8, issue 3, s. 240-255 [cit. 2014-01-09]. DOI: 10.1109/TEVC.2004.826071. Dostupné z: http://www.researchgate.net/publication/3418787_Self-organizing_hierarchical_particle_swarm_optimizer_with_time-varying_acceleration_coefficients/file/9fcfd510ae794b17a2.pdf

[14] KADIRKAMANATHAN, V., K. SELVARAJAH a P. J. FLEMING. Stability analysis of the particle dynamics in particle swarm optimizer. *IEEE Transactions on Evolutionary Computation* [online]. 2006, vol. 10, issue 3, s. 245-255 [cit. 2014-01-07]. DOI: 10.1109/TEVC.2005.857077. Dostupné z: http://sci2s.ugr.es/docencia/bioinformatica/bio/ficheros/Apoyo/Tema%2004/A03%20-%20PSO-IEEETEC-2006.pdf

[15] REYNOLDS, C. W. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics* [online]. 1987-08-01, vol. 21, issue 4, s. 25-34 [cit. 2014-01-06]. DOI: 10.1145/37402.37406. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.1261&rep=rep1&type=pdf

[16] DEL VALLE, Y., G. K. VENAYAGAMOORTHY, S. MOHAGHEGHI, J.-C. HERNANDEZ a R. G. HARLEY. Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems. *IEEE Transactions on Evolutionary Computation* [online]. 2008, vol. 12, issue 2, s. 171-195 [cit. 2014-01-07]. DOI: 10.1109/TEVC.2007.896686. Dostupné z: https://mospace.library.umsystem.edu/xmlui/bitstream/handle/10355/30792/ParticleSwarmOptimization.pdf

[17] COELLO, C. A. C. a M. REYES-SIERRA. Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art. *International Journal of Computational Intelligence Research* [online]. 2006, vol. 2, issue 3 [cit. 2014-01-12]. DOI: 10.5019/j.ijcir.2006.68. Dostupné z: http://natcomp.liacs.nl/SWI/papers/particle.swarm.optimization/vol2-issu3-paper5.pdf

[18] KENNEDY, J. a R. C. EBERHART. A discrete binary version of the particle swarm algorithm. *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation* [online]. IEEE, 1997, s. 4104-4108 [cit. 2014-01-09]. DOI: 10.1109/ICSMC.1997.637339. Dostupné z: https://bitbucket.org/12er/pso/src/01a2a80694981894cdffb2c845c7b05cbf478bcc/doc/literature/Variants/different%20domain/Binary.pdf

[19] YE, B., J. SUN a W.-B. XU. Solving the Hard Knapsack Problems with a Binary Particle Swarm Approach. *Computational Intelligence and Bioinformatics* [online]. Springer, 2006, s. 155-163 [cit. 2014-01-09]. DOI: 10.1007/11816102_17. Dostupné z: www.researchgate.net/publication/220776306_Solving_the_Hard_Knapsack_Problems_with_a_Binary_Particle_Swarm_Approach/file/9c960517a76a718f75.pdf

[20] LAPIZCO-ENCINAS, G., C. KINGSFORD a J. REGGIA. A cooperative combinatorial Particle Swarm Optimization algorithm for side-chain packing. *2009 IEEE Swarm Intelligence Symposium* [online]. IEEE, 2009, s. 22-29 [cit. 2014-01-09]. DOI: 10.1109/SIS.2009.4937840. Dostupné z: http://cbcb.umd.edu/~carlk/papers/SCP-CCPSO-preprint.pdf

[21] ZHANG, J., Y. TAN a X. HE. Concentric spatial extension based particle swarm optimization inspired by brood sorting in ant colonies. *2009 IEEE Swarm Intelligence Symposium* [online]. IEEE, 2009, s. 9-15 [cit. 2014-01-09]. DOI: 10.1109/SIS.2009.4937838. Dostupné z: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4937838

[22] SALEHIZADEH, S. M. A., P. YADMELLAT a M. B. MENHAJ. Local Optima Avoidable Particle Swarm Optimization. *2009 IEEE Swarm Intelligence Symposium* [online]. IEEE, 2009, s. 16-21 [cit. 2014-01-08]. DOI: 10.1109/SIS.2009.4937839. Dostupné z: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4937839

[23] VAN DEN BERGH, F. *An analysis of particle swarm optimizers*. Doctoral Dissertation, South Africa, Pretoria, University of Pretoria, 2001. Dostupné z: http://upetd.up.ac.za/thesis/available/etd-05032006-160549/unrestricted/00thesis.pdf

[24] CLERC, M. *Mythical balance or when particle searm optimisation does not exploit* [online]. 2008. 6 s. [cit. 2014-01-10]. Dostupné z: http://clerc.maurice.free.fr/pso/Balanced_PSO/Balanced_PSO.pdf

[25] BANSAL, J. C., K. DEEP, K. VEERAMACHANENI a L. OSADCIW. Information sharing strategy among particles in Particle Swarm Optimization using Laplacian

operator. *2009 IEEE Swarm Intelligence Symposium* [online]. IEEE, 2009, s. 30-36 [cit. 2014-01-08]. DOI: 10.1109/SIS.2009.4937841. Dostupné z: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4937841

[26] CHEN, S.. Particle swarm optimization with pbest crossover. *2012 IEEE Congress on Evolutionary Computation* [online]. IEEE, 2012, s. 1234-1239 [cit. 2014-01-07]. DOI: 10.1109/CEC.2012.6256497. Dostupné z: http://www.academia.edu/1840039/Particle_Swarm_Optimization_with_pbest_Crossover

[27] PARSOPOULOS, K. E. a M. N. VRAHATIS. On the Computation of All Global Minimizers Through Particle Swarm Optimization. *IEEE Transactions on Evolutionary Computation* [online]. 2004, vol. 8, issue 3, s. 211-224 [cit. 2014-01-09]. DOI: 10.1109/TEVC.2004.826076. Dostupné z: http://www.cs.uoi.gr/~kostasp/papers/J05.pdf

[28] CHEN, S. Locust Swarms - A new multi-optima search technique. *2009 IEEE Congress on Evolutionary Computation* [online]. IEEE, 2009, s. 1745-1752 [cit. 2014-01-07]. DOI: 10.1109/CEC.2009.4983152. Dostupné z: http://www.academia.edu/1269109/Locust_Swarms-A_new_multi-optima_search_technique

[29] CHEN, S. a J. MONTGOMERY. A Simple Strategy to Maintain Diversity and Reduce Crowding in Particle Swarm Optimization. *AI 2011: Advances in Artificial Intelligence* [online]. Springer, 2011, s. 281 [cit. 2014-01-07]. DOI: 10.1007/978-3-642-25832-9_29. Dostupné z: https://digitalcollections.anu.edu.au/bitstream/1885/8878/1/Chen_Simple2011.pdf

[30] BLACKWELL, T. a J. BRANKE. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation* [online]. 2006, vol. 10, issue 4, s. 459-472 [cit. 2014-01-10]. DOI: 10.1109/TEVC.2005.857074. Dostupné z: http://ce.aut.ac.ir/~meybodi/paper/Dynamic%20environment/PSO-Dynamic%20Invironmnet+applications/Evlotionary+Dynamic%20Environment/BlBr06_TEC_Multiswarm.pdf

[31] COELLO, C.A.C., G. T. PULIDO a M. S. LECHUGA. Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation* [online]. 2004, vol. 8, issue 3, s. 256-279 [cit. 2014-01-09]. DOI: 10.1109/TEVC.2004.826067. Dostupné z: http://www.cpdee.ufmg.br/~joao/CE/ArtigosPSO/MO_PSO.pdf

[32] HU, X. a R. EBERHART. Multiobjective optimization using dynamic neighborhood particle swarm optimization. *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)* [online]. IEEE, 2002, s. 1677-1681 [cit. 2014-01-12]. DOI: 10.1109/CEC.2002.1004494. Dostupné z: http://www.swarmintelligence.org/papers/CEC2002Multiobjective.pdf

[33] PARSOPOULOS, K. E., D. TASOULIS a M. VRAHATIS. Multiobjective Optimization using Parallel Vector Evaluated Particle Swarm Optimization. *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications*. AIA, 2004, vol. 2 s. 823-828 [cit. 2014-01-12]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=0E012874EA3E245A629ECC5D39A6CB89?doi=10.1.1.10.5636&rep=rep1&type=pdf

[34] LORION, Y., T. BOGON, I. J. TIMM a O. DROBNIK. An Agent Based Parallel Particle Swarm Optimization - APPSO. *2009 IEEE Swarm Intelligence Symposium* [online]. IEEE, 2009, s. 52-59 [cit. 2014-01-08]. DOI: 10.1109/SIS.2009.4937844. Dostupné z: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4937844

[35] TEWOLDE, G. S., D. M. HANNA a R. E. HASKELL. Enhancing performance of PSO with automatic parameter tuning technique. *2009 IEEE Swarm Intelligence Symposium* [online]. IEEE, 2009, s. 67-73 [cit. 2014-01-07]. DOI: 10.1109/SIS.2009.4937846. Dostupné z: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4937846

[36] VAN DEN BERGH, F. a A. P. ENGELBRECHT. A Cooperative Approach to Particle Swarm Optimization. *IEEE Transactions on Evolutionary Computation* [online]. 2004, vol. 8, issue 3, s. 225-239 [cit. 2014-01-10]. DOI: 10.1109/TEVC.2004.826069. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1304845

[37] ZHANG, W., Y. LIU a M. CLERC. An adaptive PSO algorithm for reactive power optimization. *6th International Conference on Advances in Power System Control, Operation and Management. Proceedings. APSCOM 2003* [online]. IEEE, 2003, s. 302-307 [cit. 2014-01-12]. DOI: 10.1049/cp:20030603. Dostupné z: http://www.researchgate.net/publication/4101121_An_adaptive_PSO_algorithm_for_reactive_power_optimization/file/72e7e51d62f9460f57.pdf

[38] EL-DIB, A. A., H. K. M. YOUSSEF, M. M. EL-METWALLY a Z. OSMAN. Load flow solution using hybrid particle swarm optimization. *International Conference on Electrical, Electronic and Computer Engineering, 2004. ICEEC '04* [online]. IEEE, 2004, s. 742-746 [cit. 2014-01-10]. DOI: 10.1109/ICEEC.2004.1374585. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1374585

[39] MIRANDA, V. a N. FONSECA. EPSO - best-of-two-worlds meta-heuristic applied to power system problems. *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)* [online]. IEEE, 2002, s. 1080-1085 [cit. 2014-01-10]. DOI: 10.1109/CEC.2002.1004393. Dostupné z: http://paginas.fe.up.pt/~ee03007/artigo133.pdf

[40] ZHANG, W.-J. a X.-F. XIE. DEPSO: hybrid particle swarm with differential evolution operator. *SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483)* [online]. IEEE, 2003, s. 3816-3821 [cit. 2014-01-10]. DOI: 10.1109/ICSMC.2003.1244483. Dostupné z:

http://www.cs.cmu.edu/~xfxie/paper/SMCC03.pdf

[41] ZHU, W. a J. CURRY. Particle Swarm with graphics hardware acceleration and local pattern search on bound constrained problems. *2009 IEEE Swarm Intelligence Symposium* [online]. IEEE, 2009, s. 1-8 [cit. 2014-01-08]. DOI: 10.1109/SIS.2009.4937837. Dostupné z: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4937837

[42] EL-GAMMAL, D., A. BADR a M. ABD EL AZEIM. New Binary Particle Swarm Optimization With Immunity-Clonal Algorithm. *Journal of Computer Science* [online]. 2013-11-01, vol. 9, issue 11, s. 1534-1542 [cit. 2014-01-10]. DOI: 10.3844/jcssp.2013.1534.1542. Dostupné z: http://thescipub.com/pdf/10.3844/jcssp.2013.1534.1542

[43] BOLUFE ROHLER, A. a S. CHEN. Multi-swarm hybrid for multi-modal optimization. *2012 IEEE Congress on Evolutionary Computation* [online]. IEEE, 2012, s. 1759-1766 [cit. 2014-01-09]. DOI: 10.1109/CEC.2012.6256566. Dostupné z: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6256566

[44] BAKTASH, N. a M. R. MEYBODI. A New Hybrid Model of PSO and ABC Algorithms for Optimization in Dynamic Environment. *International Journal of Computer Theory and Engineering* [online]. 2012, s. 362-364 [cit. 2014-01-12]. DOI: 10.7763/IJCTE.2012.V4.484. Dostupné z: http://www.ijcte.org/papers/484-G1332.pdf

[45] KARABOGA, D. a B. AKAY. A comparative study of Artificial Bee Colony algorithm. *Applied Mathematics and Computation, Volume 214* [online]. Elsevier, 2009, s. 108-132 [cit. 2014-01-12]. DOI: 10.1016/j.amc.2009.03.090. Dostupné z: http://natcomp.liacs.nl/SWI/papers/artificial.bee.colony.algorithm/a.comparative.study.of.abc.algorithm.pdf

[46] BLACKWELL, T. M. a P. J. BENTLEY. A Dynamic Search with Charged Swarms. *GECCO '02 Proceedings of the Genetic and Evolutionary Computation Conference* [online]. San Francisco: Morgan Kaufmann, 2002, s. 19-26 [cit. 2014-05-18]. ISBN:1-55860-878-8. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.9635&rep=rep1&type=pdf

[47] LI, X. a K. H. DAM. Comparing particle swarms for tracking extrema in dynamic environments. *The 2003 Congress on Evolutionary Computation, 2003. CEC '03* [online]. IEEE, 2003, s. 1772-1779 [cit. 2014-05-18]. DOI: 10.1109/CEC.2003.1299887. Dostupné z: http://ro.uow.edu.au/cgi/viewcontent.cgi?article=1456&context=eispapers

[48] JANSON, S. a M. MIDDENDORF. A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* [online]. 2005, vol. 35, issue 6, s. 1272-1282 [cit. 2014-05-18]. DOI: 10.1109/TSMCB.2005.850530. Dostupné z: http://sci2s.ugr.es/EAMHCO/pdf/hierarchicalPSO.pdf

[49] YAZDANI, D., B. NASIRI, A. SEPAS-MOGHADDAM a M. R. MEYBODI. A novel multi-swarm algorithm for optimization in dynamic environments based on particle swarm optimization. *Applied Soft Computing* [online]. 2013, vol. 13, issue 4, s. 2144-2158 [cit. 2014-05-18]. DOI: 10.1016/j.asoc.2012.12.020. Dostupné z: http://ceit.aut.ac.ir/~meybodi/paper/Yazdani-nasiri-Meybodi-2013--2.pdf

[50] PARROTT, D. a X. LI. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Transactions on Evolutionary Computation* [online]. 2006, vol. 10, issue 4, s. 440-458 [cit. 2014-05-18]. DOI: 10.1109/TEVC.2005.859468. Dostupné z: http://researchbank.rmit.edu.au/eserv/rmit:1243/n2006000234.pdf

[51] BRITS, R., A. P. ENGELBRECHT a F. VAN DEN BERGH. A niching Particle Swarm Optimizer. *In Proceedings of the Conference on Simulated Evolution And Learning* [online]. 2002, s. 692-696 [cit. 2014-05-18]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=131F7C59D7D5A3BD84B56F03A15EFD6A?doi=10.1.1.65.6137&rep=rep1&type=pdf

[52] KAMOSI, M., A. B. HASHEMI a M. R. MEYBODI. A New Particle Swarm Optimization Algorithm for Dynamic Environments. *Swarm, Evolutionary, and Memetic Computing* [online]. 2010, s. 129-138 [cit. 2014-05-18]. DOI: 10.1007/978-3-642-17563-3_16. Dostupné z: http://ce.aut.ac.ir/~meybodi/paper/Kamosi-Hashemi-EMCO%202010A%20New%20Particle%20Swarm%20Optimization%20Algorithm%20for%20Dynamic%20Environments.pdf

[53] REZAZADEH, I., S. GHATEI a A. NAEBI. A Modified Particle Swarm Optimization Using FCM for Moving Peaks Benchmark. *Journal of Basic and Applied Scientific Research* [online]. TextRoad Publication, 2012, s. 4266-4274 [cit. 2014-05-18]. ISSN 2090-4304. Dostupné z: http://www.textroad.com/pdf/JBASR/J.%20Basic.%20Appl.%20Sci.%20Res.,%202%284%29%204266-4274,%202012.pdf

[54] BLACKWELL, T., J. BRANKE a X. LI. Particle Swarms for Dynamic Optimization Problems. *Swarm Intelligence* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 193-217 [cit. 2014-05-18]. DOI: 10.1007/978-3-540-74089-6_6. Dostupné z: http://ceit.aut.ac.ir/~meybodi/paper/DL-Harvest%20Search%20Results_files/Dynamic%20environment/PSO%20in%20dynamic%20environment==/Particle%20Swarms%20for%20Dynamic%20Optimization%20Problems.2008.pdf

[55] Moving-peaks: Moving Peaks Benchmark. [online]. [cit. 2014-05-22]. Dostupné z: https://code.google.com/p/moving-peaks/source/browse/#svn%2Ftrunk%2FMPB%2Fsrc%2Fpeaks

[56] Benchmarks. DEAP PROJECT. [online]. 2013 [cit. 2014-05-22]. Dostupné z: http://deap.gel.ulaval.ca/doc/0.8/api/benchmarks.html

Seznam příloh

Příloha A: Obsah přiloženého CD

Na CD, které je přiloženo k této práci, mají adresáře následující obsah:

Dokumenty

- DP.pdf (písemná zpráva)
- DP.docx (zdrojový tvar písemné zprávy)
- Dokumentace.pdf (uživatelská příručka aplikace)
- Data.xlsx (výsledky experimentů v tabulkách)

Programy

- AHPSO_APP.jar (hlavní program)
- run.bat (dávka pro spuštění programu z příkazového řádku)

Zdrojové kódy (všechny zdrojové kódy jsou ve formě projektu vývojového prostředí NetBeans IDE 7.0.1)

- AHPSO_APP (obsahuje všechny zdrojové soubory)