VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

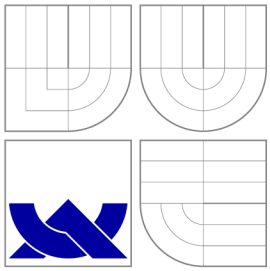
ROJENÍ ČÁSTIC JAKO VÍCEKRITERIÁLNÍ OPTIMALIZAČNÍ METODA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

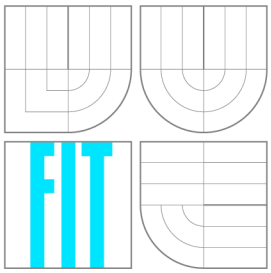
AUTOR PRÁCE
AUTHOR

TOMÁŠ BENČOK

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ROJENÍ ČÁSTIC JAKO VÍCEKRITERIÁLNÍ OPTIMALIZAČNÍ METODA

MULTI-OBJECTIVE PARTICLE SWARM OPTIMIZATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ BENČOK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. LUKÁŠ SEKANINA, Ph.D.

BRNO 2011

Abstrakt

Tato práce se zabývá problematikou multikriteriální optimalizace složitých matematických funkcí a netradičními přístupy při hledání jejich extrémů. Hlavní pozornost je věnována algoritmu PSO (Particle Swarn Optimalization), jehož implementace je také součástí práce. Algoritmus je upraven tak, aby automaticky vytvářel Paretovu množinu (nedominantních) řešení. Jeho výkonnost je pak porovnána na několika úlohách s genetickým algoritmem NSGA-II.

Abstract

This bachelor work deals with the difficulties of multi-objective optimization of certain hard mathematical functions and non-traditional approaches for discovering their extremes. Main attention is given to the PSO (Particle Swarm Optimalization) algorithm, implementation of which is also part of the bachelor work. The algorithm is modified in a such way, that it automatically builds Pareto-Optimal Set of the solutions. It's performance is compared with NSGA-II genetic algortihm on several test tasks.

Klíčová slova

Paretova množina, optimalizace, algoritmus, funkce, parametr, roj částic.

Keywords

Pareto-optimal set, optimalization, algortihm, function, parameter, particle swarm.

Citace

Tomáš Benčok: Rojení částic jako vícekriteriální
optimalizační metoda, bakalářská práce, Brno, FIT VUT v Brně, 2011

Rojení částic jako vícekriteriální optimalizační metoda

Prohlášení

Prehlasujem, že túto bakalársku prácu som vypracoval samostatne pod vedením pána Doc. Sekaniny. Uviedol som všetky literárne pramene a publikácie, z ktorých som počas písania tejto práce čerpal.

.....
Tomáš Benčok
17. mája 2011

Poděkování

Ďakujem pánovi Doc. Ing. Lukášovi Sekaninovi, Ph.D. za jeho dobrý prístup pri vedení tejto práce a za poskytnutie odbornej pomoci a konzultácií pri riešení niektorých problémov, ktoré sa počas jej písania vyskytli. Taktiež ďakujem svojej rodine a kamarátom, za poskytnutú podporu.

Obsah

Kapitola 1

Úvod

Optimalizácia je jedným z mnohých termínov, ktorý môže pre rôznych ľudí znamenať rôzne veci. Vo všeobecnosti však tento termín zodpovedá nastaveniu parametrov systému do takého stavu, aby produkoval čo najlepší možný výstup [4].

V dnešnej dobe už väčšina reálnych optimalizačných problémov zahŕňa optimalizáciu aj viacerých zložitých matematických funkcií súčasne, čo môže vyžadovať väčšie množstvo počítačového času stráveného pri ich výpočte. Tieto funkcie môžu obsahovať niekoľko lokálnych extrémov a úlohou optimalizačných algoritmov je nájsť ich – podľa možnosti čo najrýchlejšie a najefektívnejšie. Výstupom optimalizácie môže byť viacero rovnako dobrých (navzájom nedominantných) riešení ležiacich na tzv. Paretovej množine. Preto na takéto úlohy nie je vhodné ani jednoduché aplikovať štandardné deterministické postupy, ale namiesto toho sa používajú metódy stochastické [16]. Tie sa delia predovšetkým na techniky hard computing a soft computing.

Hard computing techniky sú akousi heterogénnou kolekciou tradičných výpočtových metód, kde sú presne definované požiadavky, dáta, vstupy, výstupy a ďalšie parametre systému. Je pre ne typická presnosť, určitosť, rigoróznosť a na výpočet potrebujú mať zostrojený presný analytický model problému. Reálne problémy z praxe sú však často zložité, je ťažké ich definovať presne a preto je pri konštrukcii takýchto modelov potrebná určitá miera abstrakcie. Práve z tohto dôvodu nie je vhodné tieto techniky použiť na problémy, ktoré už vo svojej podstate vyžadujú istú mieru neurčitosti.

Na druhú stranu, soft computing techniky sa od tých hard computing značne odlišujú – už len kvôli tomu, že mnohé z nich hľadajú svoju inšpiráciu na výpočet v prírode. Pojmom soft computing sa označuje skupina metód určených na riešenie výpočtovo náročných úloh, akými sú napr. NP-úplné problémy, ktorých presné riešenie nemôže byť nájdené v polynomiálnom čase [2]. Patria medzi ne:

- neurónové siete,
- fuzzy systémy,
- evolučné výpočtové techniky,
- swarm inteligence,
- teória chaosu ai.

Algoritmus PSO (Particle Swarm Optimization 2) patrí do skupiny označenej ako Swarm intelligence, a je predmetom tejto práce. Techniky soft computingu pracujú s množinou

kandidátnych riešení, teda ich výstupom nie je (resp. nemusí byť) jedno konkrétne presné riešenie, ale niekoľko približných riešení. No na druhú stranu si dokážu poradiť so širším spektrom problémových úloh, a niektoré sú dokonca schopné vyvinúť kompletne nové riešenia. Patria sem algoritmy ako napríklad NSGA-II, SPEA2, PAES, MicroGA, PSO a niekoľko ďalších.

Mnoho problémov z inžinierskej praxe môže byť definovaných ako optimalizačná úloha, napr. nájdenie optimálnej trajektórie robota či optimálnej hrúbky steny tlakovej nádoby, optimálne nastavenie parametrov regulátoru atď. Inými slovami, riešený problém je možné previesť na matematickú úlohu danú vhodným funkčným predpisom, ktorého optimalizácia vedie k nájdeniu argumentov účelovej funkcie, čo je aj jej cieľom [16].

Príkladov existuje mnoho. Riešenie takýchto problémov zvyčajne vyžaduje prácu s argumentmi optimalizovaných funkcií, pričom ich definičné obory môžu byť rôznorodého charakteru, ako napr. obor celých, reálnych alebo komplexných čísiel.

Cieľom práce je pochopiť a vysvetliť problematiku multikriteriálnej optimalizácie, implementovať upravenú verziu algoritmu PSO, vyskúšať ju na niekoľkých testovacích funkciách a zhodnotiť dosiahnuté výsledky.

Text práce je členený nasledovne do 7 kapitol. V kapitole 1 sa nachádza vysvetlenie, prečo je optimalizácia potrebná a prečo má zmysel sa ňou zaoberať. V kapitole 2 je opísaný vznik základného algoritmu PSO, sú v nej vysvetlené základné pojmy, ktoré je potrebné v tomto kontexte vedieť, a taktiež sa v nej nachádza zjednodušený pseudokód tohto algoritmu. Ďalšia kapitola (3) je teoretická a zaoberá sa problematikou multikriteriálnej optimalizácie. Je v nej definované, čo sa označuje pod pojmom Paretova množina, čo všetko treba zohľadniť, ak sa z jednokriteriálnej PSO verzie prechádza na viackriteriálnu, a tiež sa v nej nachádza jednodušený popis algoritmu NSGA-II, s ktorým bude na konci porovnaná výkonnosť algoritmu PSO. Kapitola 4 je výhradne venovaná implementáčným detailom. Nachádza sa tu pseudokód modifikovanej verzie algoritmu PSO pre riešenie multikriteriálnych problémov. Je v nej vysvetlené, ako celý algoritmus funguje, okrem toho je v nej možné nájsť popis mutácie, ďalej na čo slúži parameter *crowding distance* a je v nej opísaný aj spôsob výberu vodcu pre jednotlivé častice. V kapitole 5 sú definované všetky testovacie funkcie, na ktorých bola testovaná výkonnosť algoritmu a je v nej vysvetlená IGD metrika. Tiež sa tu nachádzajú grafy optimálnych Paretových množín pre tieto funkcie. Kapitola 6 obsahuje popis experimentov a zhŕňa všetky dosiahnute výsledky, a nadobudnuté poznatky. A nakoniec v poslednej 7. kapitole je napísaný záverečný komentár.

Kapitola 2

Particle Swarm Optimalization (PSO)

2.1 Vznik a myšlienka

Algoritmus PSO vynašli James Kennedy a Russel C. Ebehart v roku 1995. Autori sa inšpirovali správaním kŕdľov vtákov, húfov rýb a rojmi včiel. Zaujímalo ich, ako je možné, že kŕdeľ vtákov dokáže zostať pohromade a to napriek tomu, že neexistuje žiadny reálny vodca, ďalej prečo jedinci do seba nikdy nenarazia, prečo má takéto zoskupenie väčšiu šancu na nájdenie potravy a tým pádom aj na prežitie atď. Pomocou simulácie na celulárnych automatoch bolo zistené, že takéto globálne správanie je jednoducho možné vytvoriť na základe niekoľkých pravidiel, ktoré závisia len od lokálnych interakcií jedincov v ich najbližšom okolí.

Koníčkom autorov bola a taktiež aj je sociálna psychológia a aj vďaka nej sa im podarilo tento algoritmus vytvoriť. Vychádzali z myšlienky, že človek ako taký by sa nikdy nestal tým čím je dnes, keby nežil v spoločnosti a nebol medzi ostatnými ľuďmi. Vo svojej knihe [8] tvrdia, že každý z nás sa chce stať lepším a úspešnejším jedincom a každý z nás má tendenciu vyhodnocovať svoje úspechy a porovnávať svoje vlastnosti a schopnosti s vlastnosťami ostatných ľudí. Ak zistíme, že niekomu sa darí lepšie než nám, je pravdepodobné, že ak by sme napodnili jeho správanie, mohli by sme tiež dosiahnuť lepšie výsledky. Dokonca ani nie je potrebné chápať, prečo niečo funguje tak, ako to funguje (nepotrebujeme to vedieť do najmenších detailov) a náš výkon sa napriek tomu môže zlepšiť. Napodobňovanie spávania niekoho iného je jednou z najrýchlejších foriem učenia [8]. Imitáciu správania možno pozorovať nielen medzi ľuďmi, ale aj v živočíšnej ríši.

Práve kvôli tomuto je možné tvrdiť, že akýsi druh globálnej optimalizácie na základe lokálnych interakcií prebieha priamo vo vnútri našej spoločnosti (a formuje tak vyšší celok - kultúru). Ľudia spolu komunikujú, vymieňajú si názory, skúsenosti a pokiaľ sa im to hodí, prijímajú tieto myšlienky za svoje vlastné – učia sa jeden od druhého.

Autori algoritmu sa snažili implementovať čosi podobné. Na n-rozmernú funkciu, ktorej extrém chceli nájsť, poslali roj častíc, v ktorom každá častica predstavuje jedno kandidátne riešenie problému. Častica pozná svoju polohu a doposiaľ najlepšiu nájdenú hodnotu, má určitú formu pamäte. Okrem toho komunikuje s časticami v jej najbližšom okolí, takže vie, ako sa darilo ostatným časticiam (prípadne môže poznať polohu globálne najlepšieho riešenia – záleží od implementácie). Častica je teda schopná podobne ako jedinec v spoločnosti vyhodnotiť svoju úspešnosť, porovnať ju s úspešnosťou ostatných častíc a nakoniec upraviť

svoje správanie (smer a rýchlosť letu) tak, aby sa čo najviac priblížila úspešnosti ostatných častíc. Počas tohto hľadania sa môže aj ona sama stať najlepšou a začať k sebe priťahovať ostatné častice.

2.2 Porovnanie s evolučnými výpočtovými technikami

PSO je na populáciách založená stochastická optimalizačná metóda, ktorá je vo viacerých ohľadoch podobná evolučným výpočtovým technikám. Je veľmi užitočná a efektívna pri optimalizácii parametrov u spojitých, viacrozmerných funkcií. Bola úspešne aplikovaná v niekoľkých oblastiach, ako napríklad pri optimalizácii funkcií, tréningu neurónových sietí, pri fuzzy systémoch a mnohých ďalších problémoch z reálneho sveta.

Podobnosti s evolučnými algoritmami:

- populácia niekoľkých kandidátnych riešení,
- náhodné rozmiestnenie častíc,
- výpočet fitness hodnoty,
- pravdepodobnostne orientované prehľadávanie priestoru a
- negarantuje úspech.

Niektoré **rozdiely** voči evolučným algoritmom:

- PSO v sebe nemá žiadnu funkciu na výber – selekciu rodičov do ďalšej generácie – tento mechanizmus je nahradený použitím vodcov pri prehľadávaní priestoru. V podstate sa dá povedať, že PSO neprodukuje žiadnych nových potomkov - stále pracuje s tými istými jedincami, ktorým sa len stále mení vektor rýchlosti (anglicky velocita – spojený termín pre rýchlosť a smer letu).
- Ďalší rozdiel je v spôsobe, akým sa používa operátor mutácie. Kým evolučné algoritmy používajú typ mutácie, ktorý môže potenciálne riešenie nasmerovať akýmkoľvek smerom (teda aj do oblastí, ktoré majú menšiu pravdepodobnosť úspešnosti), PSO používa na úpravu velocity operátor, ktorý nasmeruje časticu špecifickým smerom. Tento smer je závislý od tzv. p_{Best} a g_{Best} hodnôt. Niekedy to však môže spôsobovať problémy, pretože ak je smer p_{Best} podobný tomu g_{Best}, uhol potenciálnych smerov ktoré by mohli vzniknúť bude malý, čo v konečnom dôsledku zabráni objavovaniu širších regiónov priestoru, a môže spôsobiť uviaznutie častice v lokálnom optime.

2.3 Základné pojmy

V tejto podkapitole sú vysvetlené všetky pojmy, ktoré sa vyskytujú v algoritme PSO. Slová v zátvorke sú ich štandardným anglickým ekvivalentom.

- *Roj* (swarm) – Populácia, s ktorou algoritmus pracuje. 2.1
- *Častica* (particle) – Je súčasťou roja. Každá individuálna častica predstavuje jedno potenciálne riešenie problému. Jej pozícia je určená riešením, ktoré momentálne reprezentuje.
- *pbest* (personal best) – Najlepšia pozícia, v akej sa častica doposiaľ nachádzala, t.j. pozícia, ktorá vrátila zatiaľ najlepšiu hodnotu pri riešení problému.
- *lbest* (local best) – Pozícia najlepšej častice v okolí danej častice.
- *gbest* (global best) – Pozícia globálne najlepšej častice v celom roji.
- *Vodca* (leader) – Častica, ktorá je použitá k navádzaniu iných častíc k lepším oblastiam prehľadávaného priestoru.
- *Vektor rýchlosti* (velocity) – Je to vektor, ktorý riadi celý optimalizačný proces – určuje smer, akým bude častica v ďalšom kroku letieť, v snahe zlepšiť súčasnú pozíciu.
- *Zotrvačnosť* (intertia weight) – Úlohou tejto premennej je kontrolovať dopad predošlých rýchlostí na súčasnú rýchlosť danej častice. Väčšia hodnota umožňuje častici prehľadávať rozľahlejšie časti priestoru, nižšia zase podporuje lokálne prehľadávanie.
- *Faktory učenia* – Konštanty C1 a C2 – C1 zodpovedá vôli častice dôverovať vlastnému úspechu, C2 zodpovedá vôli častice dôverovať radšej úspechu častíc v jej okolí.
- *Topológia okolia* (neigborhood topology) – Určuje množinu častíc, ktoré sa podieľajú na výpočte lbest hodnoty danej častice. Ak sa pri výpočte použije parameter crowding distance, okolie častice ako také nie je potrebné zavádzať.

Obrázok 2.1: Húf rýb, prevzaté z [1].

2.4 Priebeh optimalizácie

V algoritme PSO častice *letia* hyper-dimenzionálnym priestorom, v ktorom sa snažia nájsť optimum. Zmeny pozície častice v rámci tohto priestoru sú založené na myšlienke pochádzajúcej zo sociálnej psychológie, a síce že každý jednotlivec sa snaží napodobniť úspech iného (úspešnejšieho) jednotlivca. Každá z nich má tri atribúty – pozíciu, vektor rýchlosti a doposiaľ najlepšiu vypočítanú hodnotu pbest. Pozícia sa mení na základe vlastných skúseností častice a jej okolia.

Nech vektor $x_i(t)$ značí pozíciu častice π v čase t. Pozícia π je potom zmenená pripočítaním velocity $v_i(t)$ k súčasnej pozícii, tak že:

$$x_i(t) \quad = \quad x_i(t-1) + v_i(t) \tag{2.1}$$

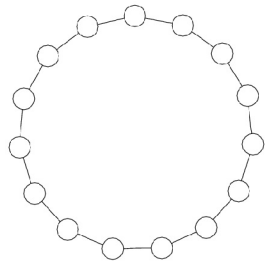
Vektor rýchlosti zodpovedá výmene informácií v sociálnom prostredí a vo všeobecnosti je definovaný nasledovným spôsobom:

$$v_i(t) \quad = \quad Wv_i(t-1) + C_1r_1(x_{pBest} - x_i(t)) + C_2r_2(x_{gBest} - x_i(t)) \tag{2.2}$$

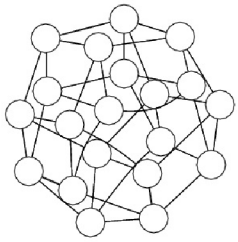
kde r_1 a r_2 sú náhodné čísla patriace do intervalu $[0, 1]$. Častice majú tendenciu byť ovplyvnené úspechom ktorejkoľvek častice, ku ktorej sú pripojené. Tieto častice sa nutne nemusia nachádzať v jej najbližšom okolí, môžu to byť častice, ktoré sú blízko seba aj na základe definovanej topológie. Táto topológia potom vytvára sociálnu štruktúru roja [8].

2.5 Topológie okolia častice

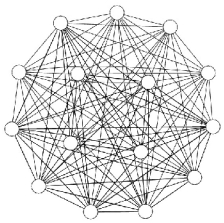
V tejto podkapitole sú opísané rôzne typ okolí, v akých sa častica môže nachádzať. Ako už bolo spomenuté vyššie, okolie častice sa podieľa na výpočte hodnoty l_{Best} a existuje niekoľko všeobecne známych typov, ktoré sa pre tento účel používajú. Obrázky 2.2 - 2.6 zobrazujú najčastejšie prípady.



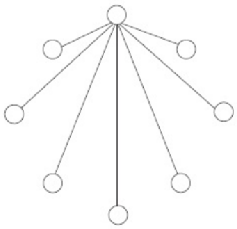
Obrázok 2.2: Kruhová topoógia – topológia reprezentuje l_{Best} schému, kde okolie i-tej častice $k_i = 2$. Táto topológia sa používa celkom bežne a častica je ovplyvnená len jej 2 najbližšími susedmi.



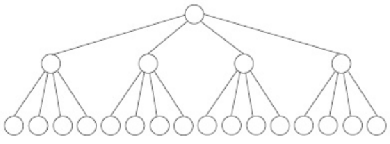
Obrázok 2.3: Ukážka formy kruhovej topológie, kde $k_i = 4$. Častice je napojená na 4 iné častice v jej okolí.



Obrázok 2.4: Plne prepojený graf – každá častica je prepojená s každou. Táto schéma prepojení sa väčšinou používa pri riešení funkcií s jedným globálnym optimom.



Obrázok 2.5: Hviezdicová topológia – častice sú napojené na vedúcu (tzv. ohniskovú) časticu a tá je spojená s ostatnými.



Obrázok 2.6: Stromová toplógia - všetky častice sú usporiadané do stromu. Každá častica v takomto zapojení je ovplyvnená zatiaľ jej najlepšou pozíciou (p_{Best}) a najlepšou pozíciou častice, ktorá je v strome priamo nad ňou.

2.6 Základný algoritmus

V tejto kapitole sa nachádza pesudokód algoritmu PSO aj s vysvetlivkami. Ako už bolo spomenuté vyššie, samotný algoritmus nie je zložitý a stojí na 3 pilieroch sociálno-kognitívnej psychológie:

- vyhodnotenie,
- porovnanie,
- imitácia.

Toto je len základný pseudokód algoritmu PSO (prevzatý z [8]), ktorý sa používa pri výpočtoch funkcií s jedným globálnym optimom a v tejto práci je uvedený len kvôli úplnosti. Celý algoritmus PSO je postavený na myšlienke zmeny vektora rýchlosti každej z častíc. Táto zmena môže smerovať buď hodnote p_{Best} a podporovať tak lokálne prehľadávanie v blízkom okolí častice, alebo môže byť nasmerované ku globálne najlepšej hodnote g_{Best}. Nová hodnota rýchlosti je vypočítaná v každom ďalšom kroku algoritmu (každá ďalšia generácia ju má inú).

Algoritmus použitý na multi-kriteriálnu optimalizáciu vznikne jeho miernou modifikáciou, je o niečo zložitejší a bude prezentovaný v ďalšej kapitole.

```
1  Loop
2        For i = 1 to number of individuals
3        if G(x_i) > G(p_i) then do       //G() vyhodnocuje vhodnosť
4                 For d= 1 to dimensions
5                 p_id = x_id      //p_id je zatiaľ najlepšia hodnota
6                 Next d
7        End do
8
9        g = i //arbiter
10       For j = 1 to indexes of neighbours
11                if G(p_j) > G(p_g) than g = j //g je index najlepšej
12                                              //častice v~okolí
13
14       next j
15       For d = 1 to number of dimensions
16                compute v_id     //podľa vyššie prezentovaného vzorca
17                check v_id       //kontrola, či častica neodletela
18                                 //príliš ďaleko od oblasti záujmu
19
20                compute x_id     //výpočet novej polohy
21       Next d
22                Next i
23 Until termination criterion is satisfied
```

Na začiatku je potrebná inicializácia častíc všetkých častíc. Pre každú časticu sa vypočíta fitness hodnota danej funkcie pre daný počet rozmerov. Častica si súčasnú hodnotu porovná so svojou doposiaľ najlepšou hodnotou a ak je súčasná hodnota lepšia, bude uložená do p_{Best}. Potom sa všetky získané hodnoty vzájomne porovnajú a určí sa globálne najlepšia pozícia častice, tá sa stane vodcom. Ďalej sa pre každú časticu vypočíta nová

hodnota vektoru rýchlosti podľa vyššie uvedenej rovnice a tiež sa určí jej nová pozícia. Celý priebeh sa opakuje až do splnenia ukončovacej podmienky – tou môže byť napríklad dostatočne dobrá fitness hodnota, no väčšinou sa algoritmus necháva bežať pre vopred daný počet iterácií (generacií).

2.7 Inicializácia častíc a definícia parametrov

Pozície a velocity častíc sú obvykle inicializované náhodne, pričom pozície sú rovnomerne rozdistribuované každým rozmerom a vektory rýchlosti sú zvolené náhodne v intervale $[-v_{max}, v_{max}]$.

Ďalšou otázkou môže byť, aký počet častíc sa má pri výpočte použiť. Odpoveď na túto otázku nie je jednoznačná, a však experimentmi bolo zistené, že populácia o veľkosti 20–50 častíc pracuje obvykle dobre [8].

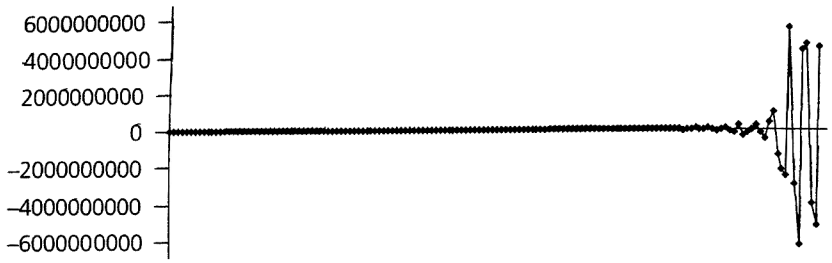
2.7.1 Vektor rýchlosti v_{max}

Algoritmus PSO vykonáva v každej iterácii výpočet na základe modifikácie vzdialenosti, ktorú častice prejdú cez každý rozmer. Zmeny rýchlostí častíc sú stochastické a kvôli tomuto môže niekedy nastať nechcená situácia, že rýchlosť sa priblíži až k nekonečnu a častica sa príliš vzdiali od riešenia problému. Každá častica teda vykonáva akýsi kmitavý pohyb, a práve preto je tam takéto obmedzenie velocity potrebné.

$$if \quad v_i \quad > \quad v_{max} \quad than \quad v_i = v_{max} \tag{2.3}$$

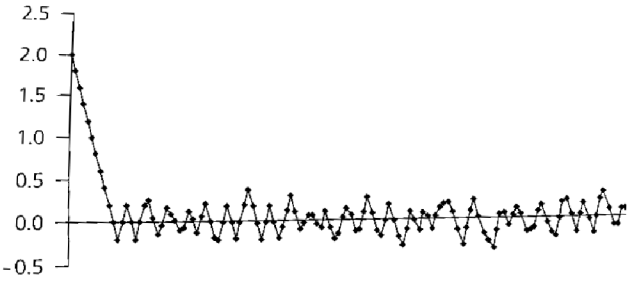
$$else \quad if \quad v_i \quad < \quad -v_{max} \quad than \quad v_i = -v_{max} \tag{2.4}$$

Ak by algoritmus nemal nastavené tieto obmedzovacie podmienky, môže sa stať to, čo je zobrazené na obrázku 2.7 - a síce, že častica rýchlo *odletí* preč do oblastí mimo záujmu.



Obrázok 2.7: Závislosť pozície častice od parametru rýchlosti v_{max}. Obrázok ukazuje stav po 150. iterácii. (Prevzaté z [8]).

No na druhú stranu, obmedzenie rýchlosti častice v rámci prehľadávaného priestoru spôsobí, že častica bude s väčšou pravdepodobnosťou prehľadávať danú oblasť, ktorá nás zaujíma – a síce v okolí optima, tak ako je to ukázané na obrázku 2.8.



Obrázok 2.8: Závislosť pozície častice od parametru v_{max}. Je vidieť, že častica sa už tentokrát nevzdialila od oblasti záujmu. Prevzané z [8].

2.7.2 Zotrvačnosť W

Tento parameter slúži na kontrolu prehľadávacích schopností algoritmu. Jeho analógiu by mohla byť metafora chladnutia teploty pri simulovanom žíhaní. Čím je jeho hodnota vyššia, tým môžu častice prehľadávať väčší priestor (exploration) a čím je nižšia, častice viac prehľadávajú lokálne oblasti (exploitation). Do výpočtového vzorca pre úpravu velocity častice bol vložený až v roku 1998 a pôvodne bol navrhnutý kvôli tomu, aby bolo možné eliminovať parameter v_{max} – ale nepodarilo sa mu to úplne. Je to multiplikačný parameter a upravuje hybnosť častice tým, že ohodnotí príspevok predošlej velocity [10]. Existujú dva prístupy pri jeho použití – môže byť buď konštantný, alebo dynamický. Dynamický môže buď to narastať alebo klesať (či už lineárne alebo nelineárne). Shi navrhuje, aby jeho hodnota začala lineárne klesať od hodnoty 0.9 do 0.4, zatiaľ čo Y. Zheng v [17] navrhuje presne opačné použitie. Podľa experimentov Y. Zhenga dosahuje algoritmus PSO lepšie výsledky.

2.7.3 Ďalšie konštanty

Konštanty C_1, C_2 sú nezáporné konštantné čísla, ktoré sú vždy použité a vynásobené s pseudonáhodnými číslami r_1 a r_2. C_1 je kognitívny parameter, ktorý reprezentuje *vlastnú vôľu častice* veriť svojim doterajším úspechom a pokračovať v prehľadávaní priestoru viacmenej tak, ako začala. C_2 je naopak sociálny parameter, ktorý hovorí o tom, ako veľmi sa dá konkrétna častica ovplyvniť úspechmi iných častíc.

Hodnoty oboch parametrov sú väčšinou nastavená na 1, aby bola v populácii zachovaná rovnorodosť medzi jedincami, ktorý veria radšej sebe a tými, ktorý radšej sledujú správanie ostatných. Samotné ovplyvnenie či už tým alebo oným smerom je potom dané práve ich vynásobením pseudonáhodným číslami r_1 a r_2.

Kapitola 3

Multikriteriálna optimalizácia

V tejto kapitole je uvedený popis definície matematických úloh, ktoré sa bude snažiť algoritmus vyriešiť. Ďalej je v nej možné nájsť definíciu Paretovej množiny, upravený pseudokód PSO pre multikriteriálne úlohy a vysvetlenie, prečo je potrebné počítať hodnotu *crowding distance*.

3.1 Definícia problému

Predmetom záujmu je riešenie problému takéhoto typu:

$$minimalizcia\ \vec{f}(\vec{x}) \quad := \quad [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})] \tag{3.1}$$

vzhľadom k:

$$g_i(\vec{x}) \quad \leq \quad 0 \quad i = 1, 2, \ldots, m \tag{3.2}$$

$$h_i(\vec{x}) \quad = \quad 0 \quad i = 1, 2, \ldots, p \tag{3.3}$$

kde $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ je vektor rozhodovacích premenných, $f_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, k$ sú objektívne funkcie a $g_i, h_j : \mathbb{R}^n \to \mathbb{R},\ i = 1, \ldots m, j = 1, \ldots, p$ sú obmedzovacie podmienky (constraints) pre dané funkcie.

3.2 Paretova množina

Ak sa pri optimalizácii problémovej úlohy snažíme dosiahnuť viaceré ciele, v zásade nám z toho môže vzniknúť množina viacerých optimálnych riešení (všeobecne známa ako Paretova množina), pričom nedostaneme len jediné riešenie. Ak nemáme k dispozícii žiadne dodatočné informácie, nemôžeme povedať, že jedno riešenie nachádzajúce sa v tejto množine je lepšie než druhé – všetky sú rovnako dobré vzhľadom na obmedzenia parametrov, ktoré boli na začiatku stanovené. V praxi obvykle bývajú tieto ciele dosť protichodné, hľadáme akýsi kompromis medzi určitými vlastnosťami a kvalitami – snahou preto je nájsť čo najviac takýchto riešení aby sme sa mohli rozhodnúť, ktoré by pre nás bolo najefektívnejšie.

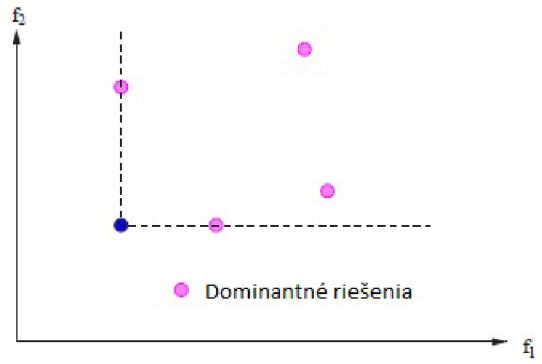
Multiobjektívna optimalizácia sa okrem iného zaoberá aj generovaním Paretovej množiny, čo je v podstate množina nedominantných riešení, ktoré majú viac než jeden cieľ.

Riešenie sa nazýva nedominantným vtedy, keď už nie je možné zlepšiť hodnotu žiadneho parametru riešenia bez toho, aby sa nezhoršila hodnota nejakého iného parametru.

Cieľom multikiteriálnej optimalizácie je teda viesť výpočet (resp. prehľadávanie priestoru) tak, aby bola nájdená aspoň približná časť Paretovej množiny, a taktiež aby boli riešenia v tejto množine rovnomerne rozložené [12]. Rozptyl riešení je jedným z kritérií na vyhodncovanie kvality získaného riešenia a efektívnosti algoritmu.

Na opis matematického konceptu optimálnosti, ktorému sa práca venuje, budú predstavené niekoľké definície (podľa [13]).

Definícia 1. Sú dané dva vektory $\vec{x}, \quad \vec{y} \in \mathbb{R}^k$ a $\vec{x} \neq \vec{y}$. Hovoríme, že $\vec{x}$ **dominuje** $\vec{y}$ (značíme $\vec{x} \prec \vec{y}$) pokiaľ platí $x_i \leq y_i$ pre $i = 1, ..., k$, a súčasne existuje také i, že $x_i < y_i$.



Obrázok 3.1: Príklad dvojrozmernej optimalizácie. [13]

Definícia 2. Hovoríme, že vektor premenných $\vec{x}^* \in \mathcal{F} \subset \mathbb{R}^n$ ($\mathcal{F}$ je vhodnosť (angl. feasibility) leží na **optimálnej Paretovej množine**, ak je nedominantný voči $\mathcal{F}$.

Definícia 3. Optimálna Paretova množina je $\mathcal{P}^*$ je definovaná nasledovne:
$\mathcal{P}^* = \{ \vec{x} \in \mathcal{F} \mid \vec{x}$ má v sebe Pareto optiálne riešenia $\}$

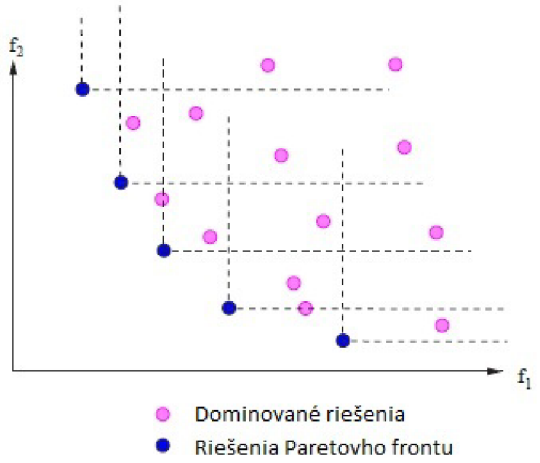
Definícia 4. Obyčajná Paretova množina je $\mathcal{PF}^*$ je definovaná nasledovne:

$$\mathcal{PF}^* \quad = \quad \{\vec{f}(\vec{x}) \in \mathbb{R}^k | \vec{x} \in \mathcal{P}^*\} \tag{3.4}$$

Optimálna Paretova množina je zobrazená na obrázku 3.2.

3.3 Multikriterálny PSO

Ako už bolo spomenuté v kapitole 2, aby bol PSO schopný vyriešiť mutlikriteriálne problémy, musí byť jeho schéma mierne pozmenená. Pri riešení multi-objekívnych problémov sa vo všeobecnosti snažíme dosiahnuť 3 ciele [13]:



Obrázok 3.2: Paretova množina niekoľkých riešení v priestore dvojkriteriálnych funkcií [13].

- Maximalizovať počet prvkov v nájdenej Paretovej množine.
- Minimalizovať vzdialenosť Paretovho frontu vytreného naším algoritmom s rešpektovaním hlavnej Paretovej množiny.
- Maximalizovať rozptyl nájdených riešení.

Pri použití PSO sa snažíme dostať čo najviac (rozdielnych) riešení počas jedného priechodu algoritmu. Takže, ak chceme PSO rozšíriť o multikriteriálnu optimalizáciu, je potrebné zaoberať sa 3 hlavnými problémami [13] (podobne ako u evolučných algoritmov):

- Ako vybrať častice (vodcov), aby produkovali nedominantné riešenia pred tými dominovanými?
- Ako udržiavať nedominantné riešenia počas hľadania tak, aby bolo možné nahlásiť ďalšie riešenia, ktoré budú nedominantné voči všetkým ostatným nájdeným riešeniam a nie len voči tomu terajšiemu. Taktiež by bolo dobré, keby boli tieto riešenia dobre rozdistribuované po Paretových množinách.
- Ako udržiavať rôznorodosť častíc v roji tak, aby celý výpočet nekonvergoval len k jedinému riešeniu?

Pri optimalizácii úloh s jedným extrémom je vodca častíc určený prevažne topológiou ich okolia. Ale pri multi-kriteriálnej optimalizácii si každá častica môže vybrať jedného vodcu z množiny viacerých (rôznych) vodcov. Takáto množina je obvykle uložená na inom mieste, než je roj a nazýva sa **externý archív**. Je to repozitár, v ktorom sú uložené zatiaľ najlepšie nájdené nedominantné riešenia. Vodca je vybraný vždy, keď je chce častica zmeniť svoju pozíciu (takže v každom cykle výpočtu). Veľkosť archívu je pevne nastavená na začiatku, v rôznych štúdiách väčšinou bývajú použité archívy, ktoré majú kapacitu na 500 až 1000 prvkov. Obsah tohto archívu sa počas behu algoritmu modifikuje, pretože postupne sú nachádzané čoraz lepšie riešenia a v konečnom dôsledku je aj výstupom algoritmu.

3.4 NSGA-II

NSGA-II je viackriteriálny genetický algoritmus, ktorý vznikol modifikáciou algoritmu NSGA. Je založený na nedominantnom výbere členov populácie. Je to rýchly, efektívny radiaci algoritmus, využívajúci elitizmus a nepotrebuje špecifické nastavenie žiadnych zdieľaných parametrov ako potreboval jeho predchodca [7].

Populácia je inicializovaná na základe rozsahu problému a jeho obmedzovacích podmienok, ak tam nejaké existujú. Potom je táto populácia na základe nedominancie roztriedená do rôznych frontov alebo vĺn. Prvá (najlepšia) množina je tá množina jedincov, ktorej v súčasnej populácii nedominuje žiadna iná. Druhej množine dominujú jedinci z prvej atď. Každý jedinec v každej množine je ohodnotený fitness hodnotou, ktorá je odvodená z množiny, v ktorej sa nachádza. Napríklad, jedincom z najlepšej množiny bude pridelená hodnota 1, z druhej hodnota 2 atď [7].

Ďalej je každému jedincovi vypočítaný parameter zvaný crowding distance. Tento parameter meria, ako blízko je jedinec k svojim susedom. Čím vyššia je jeho priemerná hodnota, tým je väčšia rôznorodosť (diverzita) v populácii.

Rodičia sú vybraní z populácie v binárnom turnaji, ktorý vychádza z hodnoty fitness alebo z parametru crowding distance. Jedinec bude vybraný, ak má nižšiu hodnotu fitness, alebo ak je jeho parameter crowding distance vyšší. Na vybraných jedincov sú potom aplikované operátori kríženia a mutácie. Populácia zložená zo súčasnej populácie a súčasných potomkov bude opäť zoradená na základe nedominantných kritérií a n najlepších jedincov bude vybraných do ďalšej generácie [7].

Kapitola 4

Implementácia

V tejto kapitole sú opísané implementačné detaily algoritmu PSO upraveného na multikriteriálnu optimalizáciu. Nachádza sa tu jeho pseudokód, popis mutácie, ďalej je tu vysvetlená dôležitosť parametru *crowding distance* a sú tu spomenuté aj niektoré ďalšie implementačné detaily.

4.1 Pseudokód MOPSO

V tejto podkapitole sa nachádza základný pseudokód algoritmu MOPSO [13]. V tejto práci bol na implementáciu použtý veľmi podobný kód, ktorý bude podrobnejšie rozobraný v sekcii zaoberajúcej sa implementáciou.

```
1  Begin
2       Initialize swarm                          //náhodná incializácia
3       Initialize leaders in external archive    //1. generácia
4                                                 //náhodní vodcovia
5       Quality(leaders)
6       g = 0
7       While g < g_max
8       For each particle
9               Select leader           //výber vodcu z množiny viacerých
10              Update Position         //let častíc
11              Mutation                //aplikácia mutačného operátora
12              Evaluation              //výpočet fitness hodonty
13              Update p_best
14      End for
15      Update leaders
16      Quality (leaders)               //crodinig distance
17      g++                             //ďalšia generácia
18      End While
19      Return archive                  //archív obsahuje nájdenú
20                                      //Paretovu množinu
21 End
```

Pseudokód MOPSO sa od toho pôvodného až tak veľmi nelíši, no už na prvý pohľad je zrejmé, že je o niečo zložitejší, pretože sem pribudli nové veci, ktoré potrebujú dodatočnú réžiu. Je potrebné sa zaoberať externým archívom, ktorý obsahuje najlepšie riešenia,

usporiadať ho a tieto riešenia ohodnotiť. Tiež treba vyriešiť spôsob akým sa vyberie nový vodca pre niektorú z častíc. A pribudla aj mutácia.

4.2 Mutácia

Operátor mutácie (niekde sa uvádza pod názvom turbulencia) bol do MOPSO zavedený kvôli tomu, že niektoré funkcie obsahujú niekoľko Paretových množín (lokálnych) a má teda zabrániť predčasnej konvergencii algoritmu do lokálneho optima [4]. Pri rôznych verziách MOPSO je jej pravdepodobnosť aplikácie tohto operátora obvykle nastavená na 60%, čo v praxi znamená, že mutácia sa uplatní len počas prvej polovice generácií (spomedzi všetkých), potom už nie. Zmutovaním sa myslí mierne vychýlenie častice zo svojej súčasnej pozície.

Operátor mutácie bol prevzatý z evolučných algoritmov, pri ktorých sa používajú dva prístupy - rovnorodá (uniform) a nerovnorodá (non-uniform) mutácia. Pri rovnorodej je povolený rozsah vychýlenia častice od danej pozície konštantný počas všetkých generácií. Pri nerovnorodej mutácii sa tento rozsah s pribúdajúcimi generáciami znižuje. V implementácii bol použitý **neuniformný prístup**.

4.3 Crowding Distance

Crowding distance je mechanizmus, ktorý spolu s mutačným operátorom udržiava rôznorodosť nedominantných riešení uložených v externom archíve. Jej hodnota poskytuje dobrý odhad hustoty iných riešení nachádzajúcich sa okolí daného riešenia [6]. Každou novou generáciou častíc sa nájdené riešenia približujú ku globálnej Paretovej množine a to tak, že aktualizujú svoj vektor rýchlosti a nasledujú najúspešnejšie častice vo svojom okolí.

Vypočíta sa nasledovne: Najprv sa vzostupne zoradia hodnoty objektívnych funkcií v externom archíve. Hodnota *crowding distance* pre konkrétne riešenie je priemerná vzdialenosť od 2 susedných riešení. Okrajovým riešeniam, ktoré majú najvyššiu alebo najnižšiu hodnotu objektívnych funkcií je pridelená najvyššia možná hodnota, takže budú vždy vybrané. Proces je takto spravený pre každú objektívnu funkciu. Finálna hodnota *crowding distance* pre dané riešenie je daná súčtom všetkých týchto hodnôt pre každú objektívnu funkciu.

4.4 Výber vodcu

Už by malo byť zrejmé, že ak sa PSO rozšíri viac-kriteriálnu optimalizáciu, jedným z hlavných problémov bude, akým spôsobom vybrať vodcu pre časticu, ak existuje viac než jedno rovnako dobré riešenie problému. Najpriamočiarejším prístupom by mohlo byť, ak by sa za nového vodcu považovalo každé nové nájdené riešenie uložené v externom archíve. No tento prístup má tú nevýhodu, že veľkosť archívu pri ňom rastie veľmi rýchlo.

Pri implementácii algoritmu použitého v tejto práci bola braná do úvahy hodnota *crowding distance* ako jedno z ďalších diskriminačných kritérií (za normálnych okolností je archív naplnený len na základe Paretovej dominancie). Tento prístup tiež určuje, ktorých vodcov by bolo dobré udržiavať počas viacerých generácií, ak by mala byť alebo už bola prekročená veľkosť archívu. Keďže ľubovoľná častica v archíve by sa nemala stať vodcom, je potrebné definovať akúsi podmnožinu z tých najlepších častíc v archíve. Podmnožina bude určená na základe **percentuálnej veľkosti** archívu tak, že napríklad 10% z tých najlepších

riešení bude možné v nasledujúcom kroku označiť za vodcov. Pri výpočte nového vektoru rýchlosti pre danú časticu bude síce vodca zvolený **náhodne**, no vybrané riešenia majú lepšiu charakteristiku *crowding distance* , takže zároveň majú vyššiu pravdepodobnosť že nájdu lepšie riešenie. Lepšiu predstavu, akým spôsobom je to urobené ukazuje nasledujúci pseudokód, ktorý bol prevzatý z [3].

```
1 Begin
2       For i = 0 to archive length      //náhodná incializácia
3       compute crowding distance        //CD
4       End For
5       Sort archive members according to CD //usporiadanie
6       Create potencial leader set according to given percentage
7       Choose particle in random way
8       Make this particle the leader  //vodca bol vybraný
9 End
```

4.5 Ďalšie implementačné detaily

Algoritmus bol naprogramovaný v jazyku C. Implementácia vychádza z kódu prezentovaného v [12], ktorý bol dostupný pod *free licenciou* určenou na akademické účely z http://www.particleswarm.info. Kód algoritmu slúžil len na inšpiráciu a bol do značnej miery pozmenený – boli vytvorené štruktúry reprezentujúce časticu aj archív, niektoré funkcie boli upravené a boli pridané nové obslužné funkcie, medzi nimi aj funkcie na výpočet štatistík, tiež pribudlo rozhranie na spracovanie parametrov ai.

Celý program v podstate funguje na základe práce s dvojrozmernými poliami. Častica je reprezentovaná štrukturou *particle*, ktorá v sebe zahŕňa jej aktuálnu pozíciu a fitness hodnotu, jej doteraz najlešiu dosiahnutú pozíciu a fitness hodnotu a vektor rýchlosti. Potom je vytvrené pole takýchto štruktúr o počte častíc. Externý archív je tiež reprezentovaný štruktúrou a obsahuje v sebe doteraz najlepšie fiteness hodnoty a pozície. Napríklad pre fitness hodnoty je vytvorené pole *Fit[i]* a jeho parametrom je počet objektívnych funkcií. Pre počet rozmerov (resp. premenných n, pre ktoré sa funkcia počíta) je vytvorené pole *Var[i]*, ktorého parametrom je počet premenných. Priebeh algoritmu vyzerá nasledovne:

Na začiatku sú pomocou generátora pseudonáhodných čísiel inicializované pozície a rýchlosti všetkých častíc v poliach *Var* a *Vel*. O to, v akom rozsahu budú rozmiestnené po prehľadávanom priestore, sa postará funkcia **get_ranges()**, ktorá pozná definičné obory všetkých funkcií použitých v programe. Funkcia **eval_particles()** dosadí pozície do objektívnych funkcií, tie sa vyhodnotia a tým naplnia pole *Fit*. V tomto mieste sú už dostupné prvé pozície aj hodnoty, ktoré sa na týchto miestach vyskytujú. Funkcia **save_best()** naplní polia *BestFit* a *BestVar*, aby bolo možné v ďalších krokoch porovnávanie a funkcia **insert_pareto()** vloží niektoré riešenia do externého archívu. Nasleduje hlavná smyčka programu.
Funkcia **eval_velocity()** vypočíta nové rýchlosti a nové pozície. V nej sa tiež nachádza mechanizmus na náhodný výber vodcu z podmnožiny tých najlepších (ako bolo opísané v 4.4), aj keď o samotné usporiadanie hodnôt polí sa zvlášť starajú funkcie **qsortFit()** a **qsortCrowd()**, obe volané z funkcie **crowding()**, ktorá okrem toho vypočíta aj hodnoty parametrov *crorwding distance*. Nasleduje test, či sa častice príliš nevzdialili od oblasti záujmu – funkcia **route_particles()** v prípade potreby zmení vektor rýchlosti na opačný.

Potom prichádza na rad funkcia **mutation()** ktorá v závislosti od generácie mierne rozptýli častice po priestore. Jej úroveň je nastavená na 60%, čo znamená že, ak ubehne trochu viac než polovica z definovaného počtu generácií, táto funkcia sa už viacej volať nebude. Znovu sa zavolá funkcia **eval_particles()**, čím sa získajú nové hodnoty. **Update_archive()** vloží časticu z populácie do archívu ak je lepšia - to určí zase funkcia **check_nondom()**. Ostáva už len skontrolovať, či sa boli dosiahnuté lepšie hodnoty p_{Best} (**new_pBest()**) a celý algoritmus môže pokračovať až dovtedy, kým sa nedosiahne zvolený počet generácií. Výsledky sú zapísané do súborov, jeden z nich obsahuje priebeh výpočtu (pozície a ich fitness hodnoty) tlačený po nastavenom počte generácií a ďalší má v sebe len poslednú generáciu fitness hodnôt z archívu, takže je ho vhodné použiť na vykreslenie výsledkov.

Kapitola 5

Testovacie funkcie a metriky

V tejto kapitole sú definované testovacie funkcie, na ktorých bol MOPSO vyskúšaný a vysvetlenie použitých metrík, podľa ktorých boli vyhodnotené výsledky.

5.1 Testovacie funkcie

Algoritmus bol vyskúšaný na 7 testovacích funkciách, z toho 1 mala ešte dodatočné obmedzovacie podmienky. Boli vybrané nasledovné funkcie: Kita, Kursawre [9], ZDT1, ZDT2, ZDT3, ZDT6 [18] a funkcia zo Schafferovej štúdie [15]. Tieto funkcie boli vybrané najmä preto, lebo mnoho štúdií zaoberajúcich sa algoritmom NSGA-II testuje jeho výkon práve na nich.

5.1.1 Kitova funkcia

Kita je jediná z funkcií, ktorá má dodatočné obmedzujúce podmienky a ako u jedinej spomedzi všetkých funkcií bolo hľadané globálne maximum. Pri všetkých ostatných funkciách sa hľadalo minimum. Je to 2-rozmerná funkcia a optimálne riešenie je možné vyčítať z obrázku 5.1.

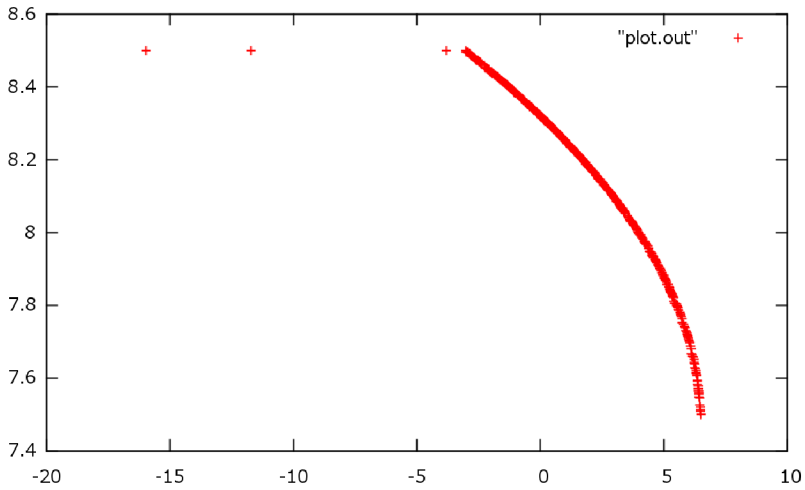
$$n = 2 \tag{5.1}$$
$$f_1(x) = -1 \cdot x_i^2 + x_{(i+1)} \tag{5.2}$$
$$f_2(x) = \frac{x_i}{2} + x_{(i+1)} + 1 \tag{5.3}$$

Obmedzujúce podmienky:

$$g_1(x) = \frac{x_1}{6} + x_2 - \frac{13}{2} \tag{5.4}$$
$$g_2(x) = \frac{x_i}{2} + x_1 - \frac{15}{2} \tag{5.5}$$
$$g_3(x) = 5 \cdot x_1 + x_2 - 30 \tag{5.6}$$

5.1.2 Kursawreho funkcia

Ďalšia z optimalizovaných funkcií bola Kursawreho funkcia, viz. obrázok 5.2

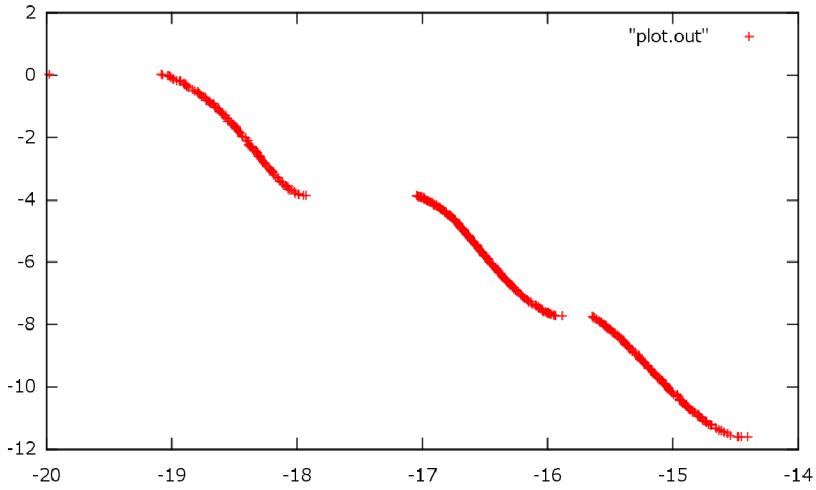


Obrázok 5.1: Optimálna Paretova množina riešení pre Kitovu funkciu.

$$n = 2 \tag{5.7}$$

$$f_1(x) = \sum_{i=1}^{n-1} \left(-10 \cdot exp\left(-0.2\sqrt{x_i^2 + x_{i+1}^2} \right) \right) \tag{5.8}$$

$$f_2(x) = \sum_{i=1}^{n} |x_i|^{0.8} + 5 \cdot \sin(x_i^3) \tag{5.9}$$

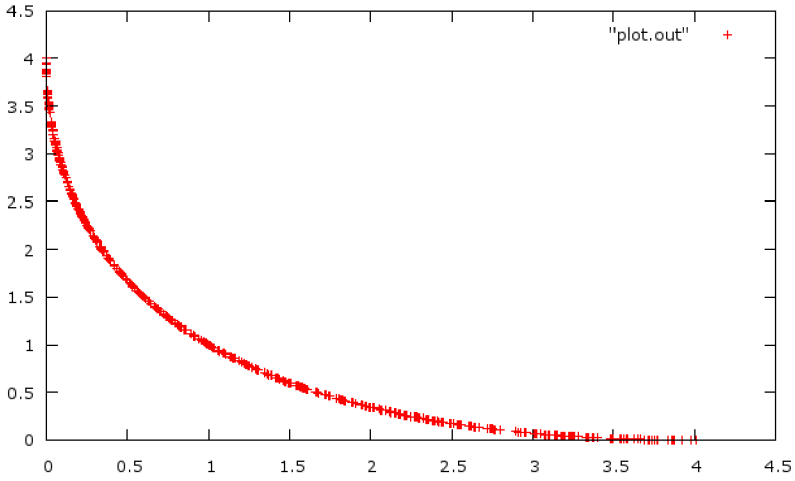


Obrázok 5.2: Optimálna Paretova množina riešení pre Kursawreho funkciu.

5.1.3 Shafferova funkcia

Najjednoduchšia funkcia z pomedzi všetkých ostatných objektívnych funkcií. Má konvexnú Paretovu množinu a jej riešenie sa hľadá iba pre jeden rozmer.

$$n = 1 \tag{5.10}$$
$$f_1(x) = x^2 \tag{5.11}$$
$$f_2(x) = (x-2)^2 \tag{5.12}$$

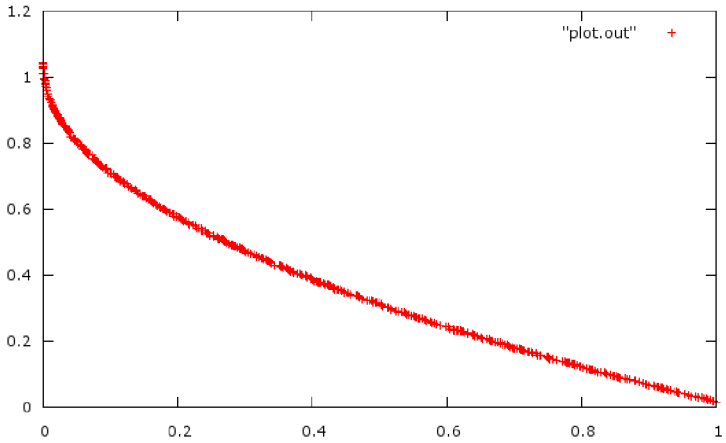


Obrázok 5.3: Paretova množina riešení pre Schafferovu funkciu.

5.1.4 ZDT1

Všetky ZDT funkcie použité v tejto bakalárskej práci boli počítané pre 30 rozmerov a ich optimum bolo nájdené ak $\vec{x_i} \in [0,1]$ pre $\forall\ i$. Tento testovací problém má konvexnú Paretovu množinu [11] a jeho optimálne riešenie je na obrázku 5.4. Predpis funkcie je definovaný nasledovne:

$$n = 30 \tag{5.13}$$
$$f_1(x) = x_1 \tag{5.14}$$
$$f_2(x) = g(x)\left[1 - \sqrt{\frac{x_1}{g(x)}}\right] \tag{5.15}$$
$$g(x) = 1 + 9 \cdot \left(\frac{\sum_{i=1}^{n} x_i}{(n-1)}\right) \tag{5.16}$$



Obrázok 5.4: Optimálna Paretova množina pre funkciu ZDT1

5.1.5 ZDT2

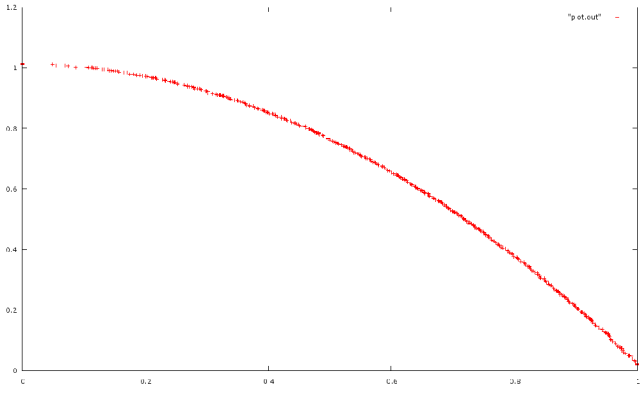
Objektívna funkcia ZDT2 má konkávnu Paretovu množinu [11] (viditeľnú na obrázku 5.5) a má definovaný nasledujúci predpis:

$$n = 30 \tag{5.17}$$

$$f_1(x) = x_1 \tag{5.18}$$

$$f_2(x) = g(x)[1 - (\frac{x_1}{g(x)})^2] \tag{5.19}$$

$$g(x) = 1 + 9 \cdot \left(\frac{\sum_{i=1}^{n} x_i}{(n-1)} \right) \tag{5.20}$$



Obrázok 5.5: Optimálna Paretova množina pre funkciu ZDT2

5.1.6 ZDT3

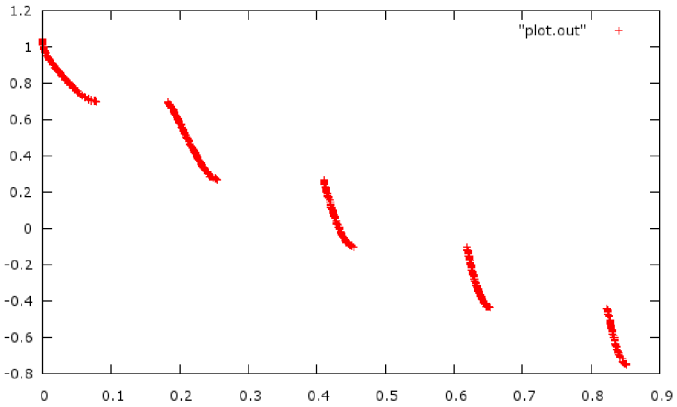
Tento problém generuje niekoľko nespojitých Paretových množín[11] (viz. 5.6) a má nasledujúcu definíciu:

$$n = 30 \tag{5.21}$$

$$f_1(x) = x_1 \tag{5.22}$$

$$f_2(x) = g(x)\left[1 - \sqrt{\frac{x_1}{g(x)}} - \frac{x_1}{g(x)}\sin(10\pi x_1)\right] \tag{5.23}$$

$$g(x) = 1 + 9 \cdot \left(\frac{\sum_{i=1}^{n} x_i}{(n-1)}\right) \tag{5.24}$$



Obrázok 5.6: Optimálna Paretova množina pre funkciu ZDT3

5.1.7 ZDT6

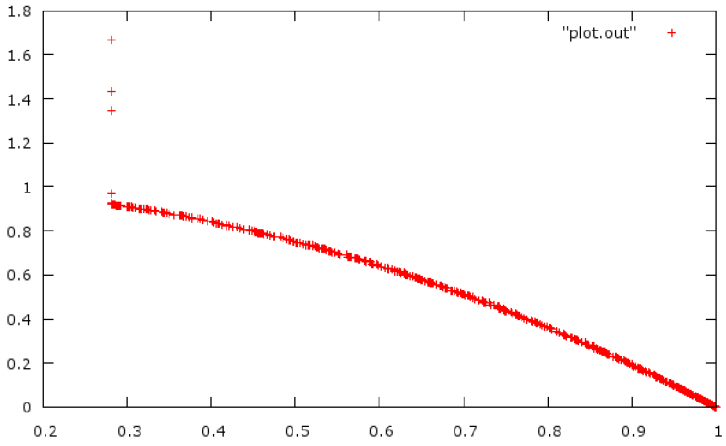
Tento testovací problém v sebe zahŕňa 2 problémy spôsobené nerovnomerným rozložením riešení v prehľadávanom priestore. Za prvé, riešenia z $\mathcal{P}_{true}$ sú nerovnomerne rozložené okolo $\mathcal{PF}_{true}$ a za druhé, hustota riešení je najnižšia v okolí $\mathcal{PF}_{true}$ a najvyššia ďaleko od neho [11]. Tieto problémy a jej nekonvexná podstata spôsobujú, že mnoho algoritmov určených na multikriterálnu optimalizáciu má problém s jej riešením.

$$n = 30 \tag{5.25}$$

$$f_1(x) = 1 - \exp(-4x_1)\sin^6(6\pi x_1) \tag{5.26}$$

$$f_2(x) = g(x) \cdot \left[1 - \left(\frac{f_1(x)}{g(x)}\right)^2\right] \tag{5.27}$$

$$g(x) = 1 + 9 \cdot \left[\frac{\sum_{i=2}^{n}(n-1)}{x_i}\right]^{0.25} \tag{5.28}$$



Obrázok 5.7: Optimálna Paretova množina pre funkciu ZDT6

Nekonvexný PF_{true} je vytvorený práve funkciou g(x) [5].

5.2 Použitá metrika

Na to, aby bolo možné kvantitatívne vyhodnotiť a porovnať výsledky urobenej optimalizácie, je nutné použiť istú, vopred formálne definovanú metriku. V tejto práci bola použitá metrika IGD, ktorá slúži na zistenie, či sú získané hodnoty vo finálnej Paretovej množine rozdistribuované rovnomerne. Jej bližší popis sa nachádza nižšie v texte.

5.2.1 IGD metrika (Inverted Generational Distance)

Van Veldhuizen a Lamont navrhli GD metriku (Generational Distance) ako spôsob odhadnutia ako ďaleko od seba sú prvky v Paretovej množine vytvorenej algoritmom od tých, ktoré sa nachádzajú v pravej Paretovej množine. Matematicky to vyzerá nasledovne:

$$IGD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n} \tag{5.29}$$

kde n je počet nedominantných riešení nájdených algoritmom ktoré skúmame a d_i je Euklidova vzdialenosť medzi každým z nich a najbližším členom v pravej Paretovej množine. Hodnota nula indikuje, že všetky prvky nájdené algoritmom sa nachádzajú na skutočnej Paretovej množine daného riešenia. A preto každá iná nenulová hodnota hovorí o vzdialenosti od skutočnej Paretovej množiny. Variantou GD metriky je IGD, kde sa so skutočnou Paretovou množinou porovnávajú aproximované riešenia z niekoľkých behov vyprodukované algoritmom. Tento spôsob poskytuje lepší odhad hodnôt a v niektorých prípadoch tiež zabraňuje slabej konvergncii algoritmu [14]. Vypočítané hodnoty tejto metriky pre určené množstvo častíc, sú uvedené v tabľkách 6.1, 6.2 a 6.3. Potom bol z nich pre každú z objektívnych funkcií vypočítaný **priemer** a **rozptyl**.

Kapitola 6

Získané výsledky a ich vyhodnotenie

V tejto kapitole sa nachádza popis experimentov vykonaných s algoritmom a vyhodnotenie dosiahnutých výsledkov.

6.1 Základné nastavenia

Všetky experimeny pre všetky funkcie boli robené pre 10.000 evaluácií pri konštantnej veľkosti externého archívu obmedzenej na 500 prvkov. Bola testovaná kvalita riešenia, v závislosti od počtu použitých častíc a od počtu ich generácií. Inými slovami – zisťovalo sa, či sa oplatí použiť väčšie množstvo častíc a nechať ich prehľadávanť daný priestor kratšiu dobu (po menší počet generácií), alebo sa použije menej častíc, no priestor sa im dovolí prehľadávať o niečo dlhšie.

Na každej funkcii boli urobené 3 experimentny pre 10, 50 a 100 častíc. Každý z nich bol kvôli štatistickej významnosti zopakovaný 5 krát pre daný počet častíc. Zo získaných výsledkov boli vypočítané metriky opísané v 5.2 a porovnané s výsledkami metrík algoritmu NSGA-II. Prehľad výsledkov metrík je urobený v tabuľke 6.5 uvedenej nižšie v texte. Všetky experimenty boli robené na osobnom počítači ThinkPad R500, ktorý má tieto parametre: procesor Intel Core 2 Duo P8700 s frekvenciou 2,53 GHz; 2GB DDR3; 64-bitový operačný systém Kubuntu 11.04.

	priemer	rozptyl
KITA	0.0009	0.0001
KUR	0.0036	0.0002
SCH	0.0033	x
ZDT1	0.0097	0.0019
ZDT2	0.0223	0.0064
ZDT3	0.0155	0.0020
ZDT6	0.0420	0.0041

Tabuľka 6.1: Výsledky metriky IGD pre algoritmus NSGA-II, prevzaté z [14].

Funkcia/Beh	1	2	3	4	5
KITA	0.00090	0.00642	0.00417	0.00138	0.00222
KUR	0.00474	0.00479	0.00489	0.00631	0.00478
SCH	0.00043	0.00040	0.00039	0.00040	0.00041
ZDT1	0.00882	0.00654	0.01304	0.00544	0.01459
ZDT2	0.00953	0.02671	0.02107	0.00883	0.03152
ZDT3	0.05913	0.04591	0.03664	0.03913	0.03238
ZDT6	0.75308	0.08734	0.88884	0.65209	0.877616

Tabuľka 6.2: Namerané hodnoty jednotlivých funkcií pre metriku IGD. Priestor prehľadávalo 10 častíc.

Funkcia/Beh	1	2	3	4	5
KITA	0.00094	0.00440	0.00099	0.00081	0.00080
KUR	0.00454	0.00537	0.00419	0.00500	0.00451
SCH	0.00038	0.00040	0.00036	0.00039	0.00042
ZDT1	0.03270	0.01940	0.01780	0.02200	0.01660
ZDT2	0.06499	0.05801	0.05395	0.03238	0.04539
ZDT3	0.05630	0.05906	0.05637	0.04689	0.04898
ZDT6	0.62823	0.65194	0.71800	0.84918	0.82147

Tabuľka 6.3: Namerané hodnoty jednotlivých funkcií pre metriku IGD. Priestor prehľadávalo 50 častíc.

Funkcia/Beh	1	2	3	4	5
KITA	0.00078	0.00123	0.00088	0.00310	0.00106
KUR	0.00510	0.00463	0.00490	0.00466	0.00516
SCH	0.00039	0.00041	0.00035	0.00040	0.00041
ZDT1	0.03089	0.03124	0.03265	0.03483	0.02452
ZDT2	0.07538	0.09485	0.08481	0.07068	0.07267
ZDT3	0.05700	0.05637	0.06698	0.07449	0.05724
ZDT6	0.90437	0.80943	0.86531	0.95045	0.93706

Tabuľka 6.4: Namerané hodnoty jednotlivých funkcií pre metriku IGD. Priestor prehľadávalo 100 častíc.

I G D	10 castic priemer	50 castic priemer	100 castic priemer	10 castic rozptyl	50 castic rozptyl	100 castic rozptyl
KITA	0.0030	0.000796	0.00705	0.00203	0.00001	0.00529
KUR	0.0051	0.004722	0.00487	0.00006	0.00041	0.00021
SCH	0.0004	0.00039	0.00039	0.00000	0.00000	0.00000
ZDT1	0.0331	0.0217	0.03082	0.00721	0.00578	0.00334
ZDT2	0.0195	0.05094	0.0796	0.00249	0.01124	0.03078
ZDT3	0.0426	0.05353	0.06241	0.00249	0.00471	0.00720
ZDT6	0.65067	0.73376	0.89332	0.29529	0.09438	0.06224

Tabuľka 6.5: Metrika IGD pre implementovaný algoritmus - získané výsledky.

Jeden experiment pre 10.000 evaluácií netrval dlho, rádovo iba niekoľko sekund. Záviselo to však od konkrétnej funkcie. Schafferova, Kursawreho a Kitova funkcia boli vypčítané o niekoľko sekúnd rýchlejšie a dokonca aj kvalitnejšie, než boli funkcie ZDT. Kým prvým trom spomínaným stačilo približne 10.000–15.000 evaluácií na nájdenie optimálneho riešenia, funkcie ZDT1, ZDT2 a ZDT3 ich potrebovali minimálne 25.000–30.000. Funkcia ZDT6 je zo všetkých najnáročnejšia a častice mali tendenciu zaseknúť sa v lokálnom optime, ktoré ťažko prekonávali. Na výpočet optimálnej Paretovej množiny tejto funkcie bolo potrebných približne 6 minút.

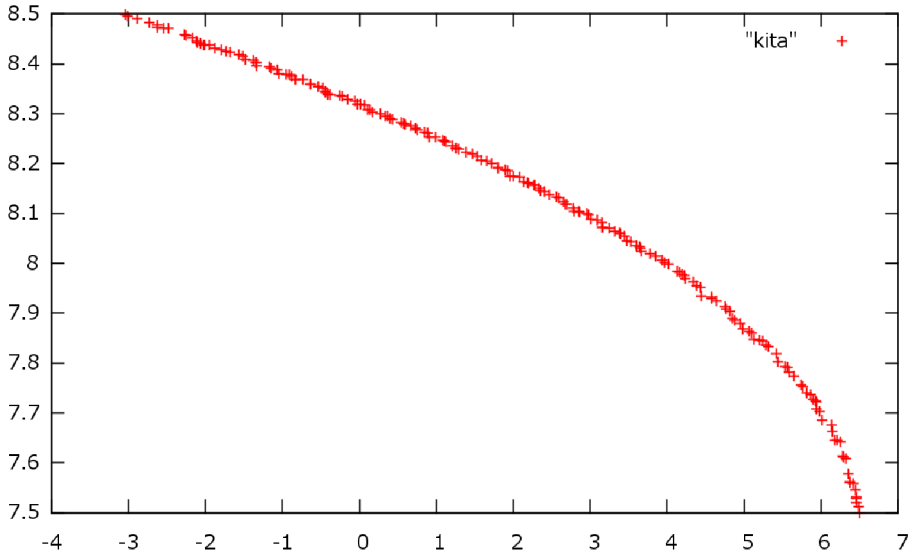
Z uvedených metrík vyplýva, že táto verzia algoritmu **nie je taká efektívna** ako je algoritmus NSGA-II, ale je rádovo horšia. 10.000 evalácií je príliš málo na to, aby našiel optimálnu Paretovu množinu. Pre jednoduchšie a menej rozmerné úlohy akými sú Kita, Kursawre a Schafferova funkcia funguje dobre, ale zlyháva na funkciách ZDT. Algoritmus je schopný vypočítať optimálnu Paretovu množinu – veď všetky testovacie dáta pre optimálne Paretove množiny boli vypočítané týmto algoritmom, ale potrebuje na to väčší počet evaluácií. Pri funkciách ZDT1, ZDT2 a ZDT3 dosahuje relatívne dobré výsledky po 25.000, až 30.000 evaluáciách. Dokáže vypočítať aj funkciu ZDT6, ale potrebuje na to dokonca ešte väčšie množstvo času.

6.2 Vplyv počtu generácií na kvalitu riešenia

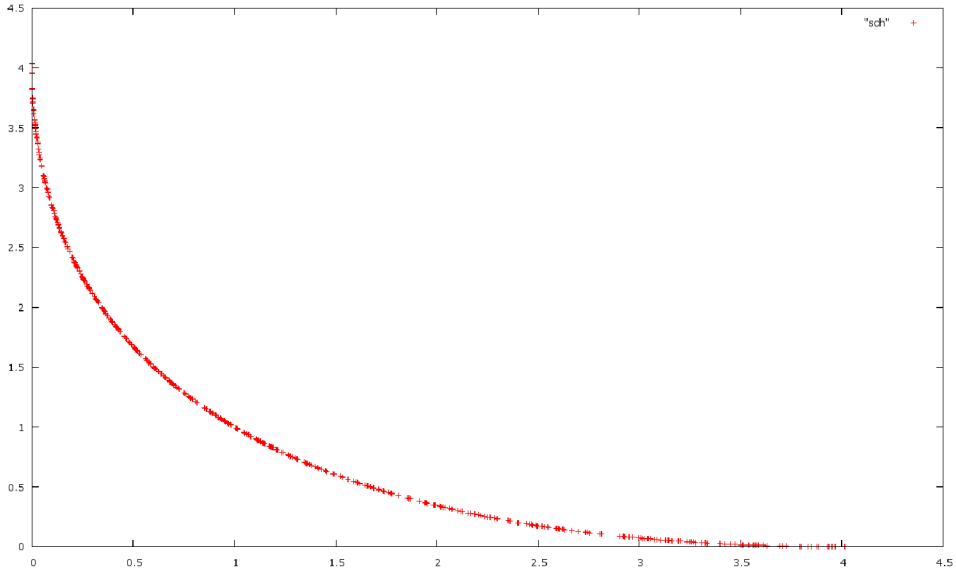
Z obrázkov výsledných Paretových množín (priložených na externom DVD nosiči) vyplýva, že ak sa aj menšiemu množstvu častíc umožní daný priestor prehľadávať dlhšie (po väčší počet generácií), sú schopné nájsť kvalitnejšie riešenie, než by našlo väčšie množstvo častíc. Je to dobre viditeľné najmä na funkciách ZDT, ktoré sú zložitejšie než Kitova, Schafferova a Kursawreho funkcia. V závislosti na pevne danom počte evaluácií, väčšie množstvo častíc prehľadáva danú oblasť kratší čas a to, či tieto častice nájdu lepšie riešenie alebo nie, je tiež závislé inicializácie ich počiatočných pozícií. Z experimentov teda vyplýva, že ak sa časticiam (a nezáleží na ich počte) dovolí prehľadávať priestor objektívnej funkcie dlhšie, po väčšie množstvo generácií, sú schopné nájsť lepšie riešenie.

6.3 Grafy získaných Paretových množín

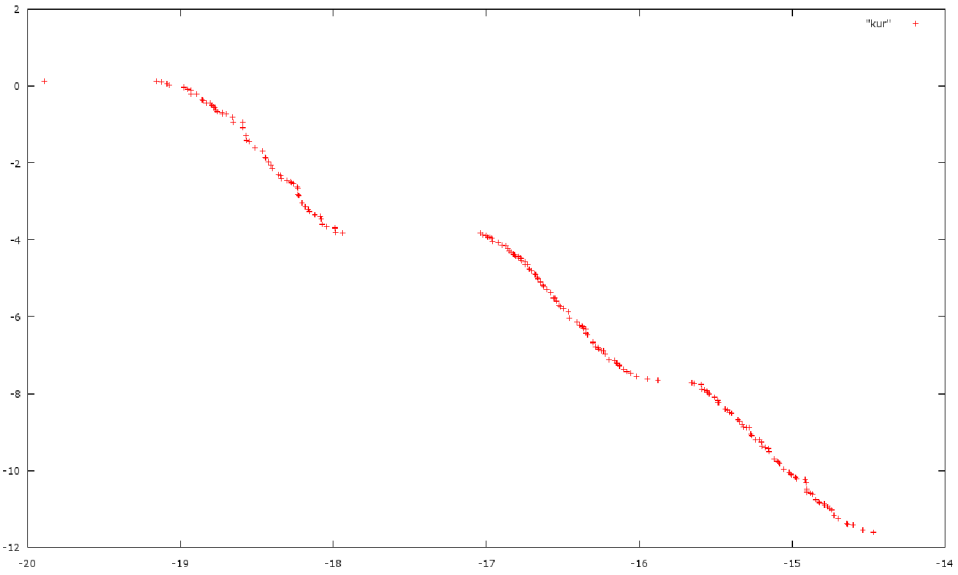
V tejto podkapitole sú zobrazené získané dáta (fitness hodnoty) objektívnych funkcií vyprodukované implementovaným algoritmom po 10.000 evaluáciách. Na vykreslenie výsledkov bol pužitý voľne šíriteľný program *GNU plot*. Z priestorových dôvodov tu sú uvedené iba grafy pre 10 častíc. Na obrázkoch 6.1–6.7 je vždy zobrazený jeden (náhodne vybraný) testovací beh. Kvalitu výpočtu možno porovnať s grafmi uvedenými v kapitole 5. Ostatné grafy je možné nájsť na priloženom DVD.



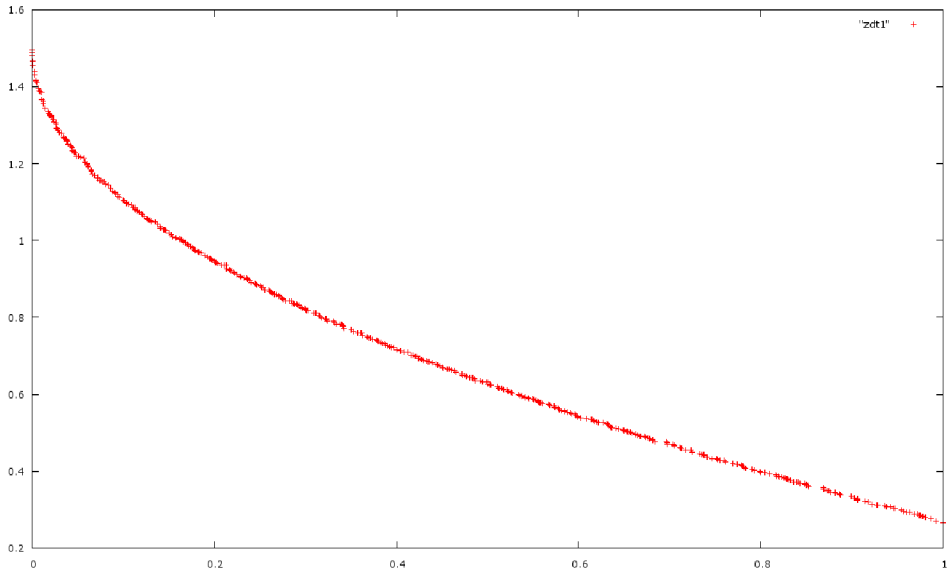
Obrázok 6.1: Získané riešenie na funkcii Kita.



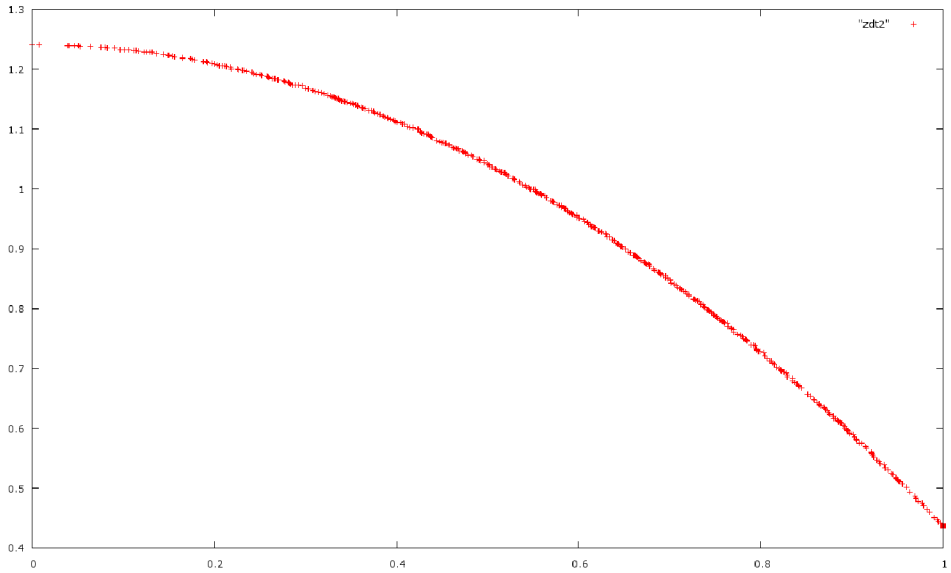
Obrázok 6.2: Získané riešenie na Schaferovej funkcii.



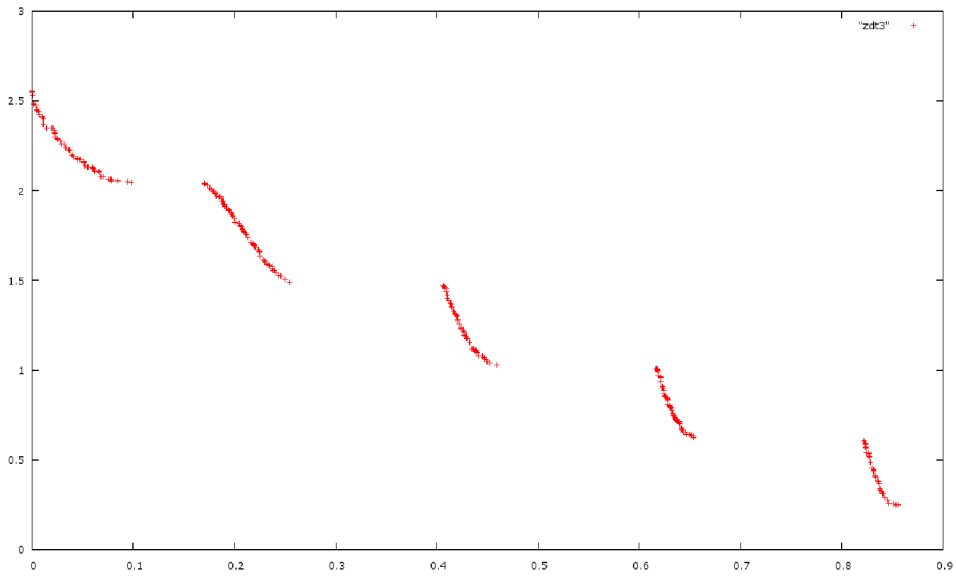
Obrázok 6.3: Získané riešenie pre 10 častíc na Kursawreho funkcii.



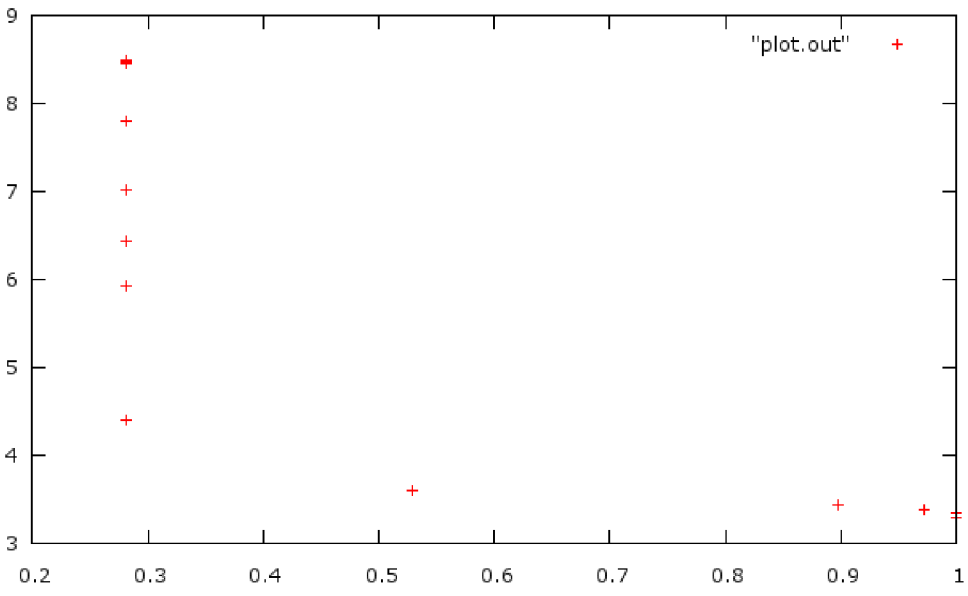
Obrázok 6.4: Získané riešenie na funkcii ZDT1. Optimum je dosiahnuté, ak $\forall$ vypočítané body x_i ležia v intervale $[0, 1]$ pre obe funkcie.



Obrázok 6.5: Získané riešenie pre funkciu ZDT2 . Optimum je ako pri obrázku 6.4.



Obrázok 6.6: Riešenie pre ZDT3. Optimum opäť leží v intervale $[0, 1]$. Je vidieť, že odchýlka už začína narastať.



Obrázok 6.7: ZDT6 pre 10 častíc. Optimum leží tiež v intervale $[0, 1]$. Paretova množina nebola v tomto prípade vytvorená – algoritmus nenašiel optimálne riešenie v stanovenom čase.

6.4 Vplyv počtu častíc na veľkosť výslednej Paretovej množiny

Tento experiment bol vykonaný pre 10.000 evaluácií. Z experimentu vyplynulo, že počet častíc nemal skoro žiadny vplyv na veľkosť vygenerovaného archívu, teda na veľkosť získanej výslednej Paretovej množiny. Výsledky v tabuľke 6.6 jasne ukazujú, že nezáleží na tom, koľko častíc sa v skutočnosti použije. Pre 10, 50 aj 100 častíc bola veľkosť archívu približne rovnaká. Z toho vyplýva, že jeho veľkosť priamoúmerne závisí od počtu vykonaných evaluácií. Pri Schafferovej funkcii (viz. 5.1.3), ktorá je spomedzi všetkých funkcií použitých v tejto práci najjednoduchšia, bol archív naplnený až do jeho plnej kapacity. Desať častíc síce našlo trochu menej riešení, no tento rozdiel nie je štatisticky významný.

Na základe získaných údajov možno tvrdiť, že za použitia 10–50 častíc funguje algoritmus dobre. Ak sa použije 100 častíc, množstvo nájdených riešení sa nezvýši, no nároky ná výkon počítača budú zbytočne vyššie. Dokonca už existuje modifikovaná verzia nazvaná Micro-MOPSO, ktorá pre svoj výpočet používa nízky počet častíc – iba 5 [3].

funkcia/častice	10	50	100
KITA	165.8	180.0	182.8
KUR	177.2	163.4	185.2
SCH	500.0	500.0	500.0

Tabuľka 6.6: Závislosť veľkosti výsledného archívu od počtu častíc.

Kapitola 7

Záver

Algoritmus PSO bol objavený približne pred 15 rokmi a je novým prírastkom do rodiny algoritmov zaoberajúcich sa optimalizáciou. Za túto krátku dobu však ukázal, že má naozaj veľký potenciál na riešenie ťažkých úloh, a uplatní sa v rôznych inžinierskych disciplínach.

Cieľom tejto práce bolo zoznámiť sa s metódou PSO a použiť ju na optimalizáciu zložitých matematických funkcií. Problematika bola zároveň rozšírená o multi-kriteriálnu optimalizáciu, pri ktorej do procesu hľadania optima vstupuje viacej objektívnych funkcií, prípadne aj viacej premenných. S multi-kriteriálnou optimalizáciou súvisí mnoho ďalších vecí, akými sú napr. tvorba Paretovej množiny priamo počas behu algoritmu, usporiadavanie získaných hodnôt v externom archíve a výpočet parametru crowding distnace, ďalej spôsob výberu vodcu pre danú časticu z tejto množiny pri prehľadávaní priestoru a mnoho ďalších, ktoré bolo nutné pri jeho implementácii zohľadniť.

Výsledky mali byť a aj boli porovnané s ďalším netriviálnym algoritmom z oblasti soft-computingu, s algoritmom NSGA–II. Na tento účel bol zvolený preto, lebo sa v podstate jedná o jeden z najlepších algoritmov pre výpočet podobných funkcií a je mu venované dostatočné množstvo rôznych štúdií. Skoro každá štúdia s ním konfrontuje svoje výsledky.

S algoritmom boli urobené experimenty pri konštantnom množstve evaluácií berúce do úvahy počet použitých častíc a generácií a následne boli vyhodnotené podľa metriky IGD opísanej vyššie. Experimenty potvrdili, že modifikovanej verzii algoritmu PSO pre multikriterálne problémy sa podarilo vypočítať Paretovu množinu pre väčšinu z testivacích funkcií v relatívne krátkom čase (rádovo v minútach) a teda, že tento algoritmus je vhodným nástrojom pri riešení podobných úloh. No implementovaná verzia nie je až taká efektívna, lebo podľa metrík dosiahol algoritmus NSGA-II lepšie výsledky. Môže to súvisieť s nastavením parametrov algoritmu, no rovnako aj s jeho implementačnými detailami. Výpočet optimálnej Paretovej množiny mu trvá síce dlhšie, ale nakoniec dokázal nájsť tieto riešenia pre všetky testovacie funkcie.

Bolo tiež zistené, že v prípade použitia takejto verzie algoritmu (ktorý počíta crowding distance) je ideálne, ak množstvo častíc hľadajúcich riešenie nepresiahne počet 50, pretože jeho zvýšením sa už kvalita získaného riešenia nezvýši, iba by to zbytočne zaťažilo výpočtový výkon.

Nakoľko ma táto problematika zaujala, v blízkej budúcnosti by som sa chcel zoznámiť aj s ďalším optimalizačnými algoritmami (či už z množiny PSO alebo inými) a vyskúšať si ich implementáciu.

Literatúra

[1] [Online].
URL <http://lesleehorner.files.wordpress.com/2010/10/schoolfish.jpg>

[2] Soft Computing. 2011, online.
URL <http://en.wikipedia.org/wiki/Soft_computing>

[3] Cabrera, J.; Coello, C.: Micro-MOPSO: A Multi-Objective Particle Swarm Optimizer That Uses a Very Small Population Size. In *Multi-Objective Swarm Intelligent Systems*, *Studies in Computational Intelligence*, ročník 261, Springer Berlin / Heidelberg, 2010, s. 83–104.

[4] Coello, C.; Pulido, G.; Lechuga, M.: Handling multiple objectives with particle swarm optimization. *Evolutionary Computation, IEEE Transactions on*, ročník 8, č. 3, 2004: s. 256 – 279, ISSN 1089-778X.

[5] Coello, C. A. C.; Lamont, G. B.; Veldhuizen, D. A. V.: *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer Science + Business Media, LLC, 2007, ISBN 978-0-387-33254-3, online.

[6] Deb, K.; Agrawal, S.; Pratap, A.; aj.: A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In *Parallel Problem Solving from Nature PPSN VI*, *Lecture Notes in Computer Science*, ročník 1917, Springer Berlin / Heidelberg, 2000, s. 849–858.
URL <http://dx.doi.org/10.1007/3-540-45356-3_83>

[7] Deb, K.; Pratap, A.; Agarwal, S.; aj.: A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II. 2000.
URL <http://sci2s.ugr.es/docencia/doctobio/2002-6-2-DEB-NSGA-II.pdf>

[8] Kennedy, J.; Eberhart, R. C.; Shi, Y.: *Swarm Intelligence*, ročník 1. Academic Press, 2001, ISBN 978-1-55860-595-4, 497 s.

[9] Kursawe, F.; Schwefel, H.-P.; Männer, R.: A Variant of Evolution Strategies for Vector Optimization. In *Parallel Problem Solving from Nature. 1st Workshop, PPSN I, volume 496 of Lecture Notes in Computer Science*, Springer-Verlag, 1991, s. 193–197.

[10] Malik, R. F.; Rahman, T. A.; Hashim, S. Z. M.; aj.: New Particle Swarm Optimizer with Sigmoid Increasing Inertia Weight. 2007: s. 35 – 44.

[11] Rangaiah, G. P.: *Multi-Objective Optimalization: Techniques and Applications in Chemical Engineering*. World Scientific Publishing Co. Pte. Ltd., 2009, ISBN 978-981-283-651-9, online.

[12] Raquel, C. R.; Prospero C. Naval, J.: An Effective Use of Crowding Distance in Multiobjective Particle Swarm Optimalization. 2005.

[13] Reyes-sierra, M.; Coello, C. A. C.: Multi-Objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Reasearch*, ročník 2, č. 3, 2006: s. 287–308.

[14] Santana-Quintero, L. V.; Ramírez-Santiago, N.; Coello, C. A. C.: Towards a More Efficient Multi-Objective Particle Swarm Optimizer. Technická správa, Evolutionary Computation Group (EVOCINV)Departamento de Computación, Mexico, 2008.
URL `<http://delta.cs.cinvestav.mx/~ccoello/chapters/chapter-bui-final.pdf.gz>`

[15] Schaffer, J. D.: Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. 1985: s. 93–100.
URL `<http://portal.acm.org/citation.cfm?id=645511.657079>`

[16] Zelinka, I.; Oplatková, Z.; Šeda, M.; aj.: *Evoluční výpočetní techniky - Principy a aplikace*. BEN - technická literatura, 2009, ISBN 978-80-7300-218-3, 536 s.

[17] ling Zheng, Y.; hua Ma, L.; yan Zhang, L.; aj.: Empirical study of particle swarm optimizer with an increasing inertia weight. Technická správa, 2003.

[18] Zitzler, E.; Deb, K.; Thiele, L.: Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, ročník 8, 2000: s. 173–195.

Dodatok A

Obsah DVD

- **source** - zložka obsahuje zdrojové kódy konzolovej aplikácie *mypso* na výpočet Paretovej množiny 7 rôznych objektívnych funkcií – jedná sa konkrétne of funkcie Kita, Kursawre, Schaffer, ZDT1, ZDT2, ZDT3 a ZDT6.
- **thesis** - tu sa nachádzajú všetky súbory (obrázky, grafy, zdrojové kódy) potrebné na vygenerovanie tejto bakalárskej práce, napísanej v začkovacom jazyku LaTeX.
- **DATA** - zložka obsahuje vypočítané dáta pre jednotlivé počty častíc (10, 50 a 100) a jednotlivé funkcie z piatich rôznzch behov programu. Ťiež je v nej možné nájsť vykreslené grafy Paretových množín.
- **readme_sk** - textový súbor obsahujúci popis adresárovej štruktúry DVD nosiča v anglickom jazyku.
- **readme_en** - textový súbor obsahujúci popis adresárovej štruktúry DVD nosiča v anglickom jazyku.

Program je možné preložiť príkazom *make*, pričom je nutné mať nainštalovaný prekladač *gcc*. Program sa potom spustí nasledovným príkazom: **./mypso -f** [**funkcia**] **-c** [**počet_častíc**] **-g** [**počet_generácií**]. Parameter *-f je povinný*, ostatné sú voliteľné. Bez ich zadania sa program spustí s prednastavenými hodnotami na počet_častíc = 10 a počet_generácií = 1000. Ak sa spustí s parametrom *-h*, na štandartný výstup bude vypísaná jednoduchá nápoveda.

Príklad spustenia: ./mypso -f KUR alebo ./mypso -f ZDT3 -c 10 -g 2500.