



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY

A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

PARALELNÍ OPTIMALIZAČNÍ METODY

PARALLEL OPTIMIZATION METHODS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

Tomáš Marcina

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. Michal Kozubík

BRNO 2023

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Tomáš Marcina

ID: 230124

Ročník: 3

Akademický rok: 2022/23

NÁZEV TÉMATU:

Paralelní optimalizační metody

POKYNY PRO VYPRACOVÁNÍ:

1. Proveďte rešerši paralelních optimalizačních metod.
2. Vytvořte knihovnu paralelních optimalizačních metod pro Matlab a následně v CUDA.
3. Srovnajte výsledky zvolených metod na benchmarkových funkcích.
4. Srovnajte zvolené metody z hlediska výpočetní náročnosti.

DOPORUČENÁ LITERATURA:

Kochenderfer, M. J., & Wheeler, T. A. (2019). Algorithms for optimization. Mit Press, 2019.

Termín zadání: 6.2.2023

Termín odevzdání: 22.5.2023

Vedoucí práce: Ing. Michal Kozubík

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Práca je zameraná na porovnanie rôznych algoritmov určených pre optimalizáciu viacrozmerných funkcií. Konkrétne sa zaoberá populačnými metódami optimalizácií, ktoré sú vo všeobecnosti vhodné pre paralelizovanie výpočtu optimálnej hodnoty funkcie. Paralelizovanie výpočtu vedie k výraznému zrýchleniu výpočtov. Porovnáva niekoľko rôznych algoritmov na to vhodných. Na záver sú zhrnuté výhody a nevýhody jednotlivých algoritmov.

KĽÚČOVÉ SLOVÁ

Optimalizácie, paralelné optimalizácie, diferenčná evolúcia, optimaizácia rojenia častíc, algoritmus svätlušiek, algoritmus umelých všelých kolónií, optimalizácia sworky šedých vlkov.

ABSTRACT

The work is concerned on the comparison of different algorithms designed for the optimization of multidimensional functions. Specifically, it deals with population optimization methods, which are generally suitable for parallelizing the calculation of the optimal function value. The parallelization of the calculation leads to a significant acceleration of the calculations. It compares several different algorithms suitable for this. At the end, the advantages and disadvantages of individual algorithms are summarized.

KEYWORDS

Optimalization, parallel optimizations, differential Evolution, particle swarm optimization, firefly algorithm, artificial bee colony algorithm, grey wolf optimization.

MARCINA, Tomáš. *Paralelní optimalizační metody*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2023, 40 s. Bakalářská práce. Vedúci práce: Ing. Michal Kozubík

Vyhlásenie autora o pôvodnosti diela

Meno a priezvisko autora: Tomáš Marcina
VUT ID autora: 230124
Typ práce: Bakalárska práca
Akademický rok: 2022/23
Téma závěrečnéj práce: Paralelní optimalizační metody

Vyhlasujem, že svoju záverečnú prácu som vypracoval samostatne pod vedením vedúcej/cého záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávných dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno
.....
podpis autora*

*Autor podpisuje iba v tlačenej verzii.

POĎAKOVANIE

Rád by som sa poďakoval vedúcemu práce pánovi Ing. Michalovi Kozubíkovi za odborné vedenie, konzultácie, trpezlivosť a jeho jedinečné podnety motivujúce k práci.

Obsah

Úvod	11
1 Optimalizácie	12
1.1 Formulácia základnej optimalizačnej úlohy	12
1.2 Obmedzenia optimalizácie	13
1.3 Metódy optimalizácie	13
1.3.1 Deterministické a stochastické metódy	13
1.3.2 Heuristické a metaheuristické algoritmy	14
2 Populačné metódy	15
2.1 Inicializácia	15
2.1.1 Náhodné rozloženie agentov	15
2.1.2 Rozloženie agentov po intervaloch	16
2.2 Evolučné algoritmy	17
2.2.1 Diferenčná evolúcia	17
2.3 Algoritmy rojovej inteligencie	18
2.3.1 Optimalizácia rojením častíc	18
2.3.2 Algoritmus svetlušiek	18
2.3.3 Algoritmus umelých včelých kolónií	19
2.3.4 Optimalizácia svorkou šedých vlkov	20
3 NVIDIA CUDA	21
3.1 Kvalifikácia funkcií a premenných	21
3.2 Paralelizácia	21
4 Implementácia v programe Matlab	23
4.1 Inicializácia populácie	23
4.2 Algoritmy pre optimalizácie	23
5 Implementácia na platforme CUDA	24
6 Testovacie funkcie	25
6.0.1 Easom's function	25
6.0.2 Hölder table function	25
6.0.3 Beale function	26
6.0.4 Bukin function 6	26
6.0.5 Layeb10 function	27
6.0.6 Schaffer function	28

7	Testovanie implementovaných algoritmov	29
7.1	Porovnanie verzií implementovaných algoritmov	29
7.1.1	Porovnanie pôsobu rekombinácie agentov	29
7.1.2	Testovanie optimalizácie svorkou šedých vlkov	30
7.2	Časová náročnosť algoritmov	30
7.3	Závislosť presnosti výpočtu na počte iterácií	31
7.4	Závislosť presnosti výpočtu na počte agentov	32
7.5	Porovnanie konvergenzie jednotlivých algoritmov	33
	Záver	36
	Literatúra	37
	Zoznam príloh	38
A	Volanie pomocných funkcií	39

Zoznam obrázkov

1.1	Príklad optimalizácie v jeden rozmernom priestore	12
2.1	Inicializácia náhodnej populácie	16
2.2	Inicializácia populácie po intervaloch	17
3.1	Príklad konfigurácie kernel funkcie	22
6.1	Graf Easom's function	25
6.2	Graf Hölder table function	26
6.3	Graf Beale function	26
6.4	Graf Bukin function	27
6.5	Graf Layeb function	27
6.6	Graf Schaffer function	28
7.1	Porovnanie dvoch verzií diferenčnej evolúcie za rôznych okolností . . .	29
7.2	Porovnanie dvoch verzií diferenčnej evolúcie za rôznych okolností . . .	30
7.3	Závislosť času na počte agentov	31
7.4	Závislosť trvania jednej iterácie algoritmu svetlušiek na počte agentov	31
7.5	Priemerná hodnota algoritmov na Easom function	34
7.6	Priemerná hodnota algoritmov na Schaffer function	34
7.7	Priemerná hodnota algoritmov na Bukin function	35
7.8	Priemerná hodnota algoritmov na Beale function	35

Zoznam výpisov

A.1	Priklad volania funkcie s účelovými funkciami	39
A.2	Priklad volania funkcie s účelovými funkciami	39
A.3	Priklad volania funkcie pre nájdenie najlepšieho riešenia v populácií .	40



Úvod

Optimalizácia je proces hľadania optimálnej hodnoty funkcie, teda jej extrémov (minima, respektíve maxima). Optimalizácia predstavuje základ pre využitie matematických modelov, štatistiky pri rozhodovaní. Snaha zrýchliť tento proces vedie k paralelizovaniu výpočtov, ku čomu sú vhodné populačné metódy optimalizácie.

Prvá kapitola sa zaoberá všeobecnou problematikou optimalizácií a postupmi hľadania riešenia.


V druhej kapitole je popísaný princíp jednotlivých algoritmov a tým čo vlastne populačné metódy pre optimalizácie sú.

V tretej kapitole sa venujem implementáciám pomocných funkcií, ako aj funkciám, na ktorých som testoval algoritmy pre optimalizácie.

Nakoniec v poslednej kapitole porovnávam jednotlivé algoritmy, ktoré som implementoval.

1 Optimalizácie

1.1 Formulácia základnej optimalizačnej úlohy

Cieľom každej optimalizačnej úlohy je nájsť optimálne riešenie účelovej funkcie. .
Čo možno zapísať ako

$$\min_x f(\mathbf{x}) \quad (1.1)$$

za podmienok $x \in \mathcal{X}$,

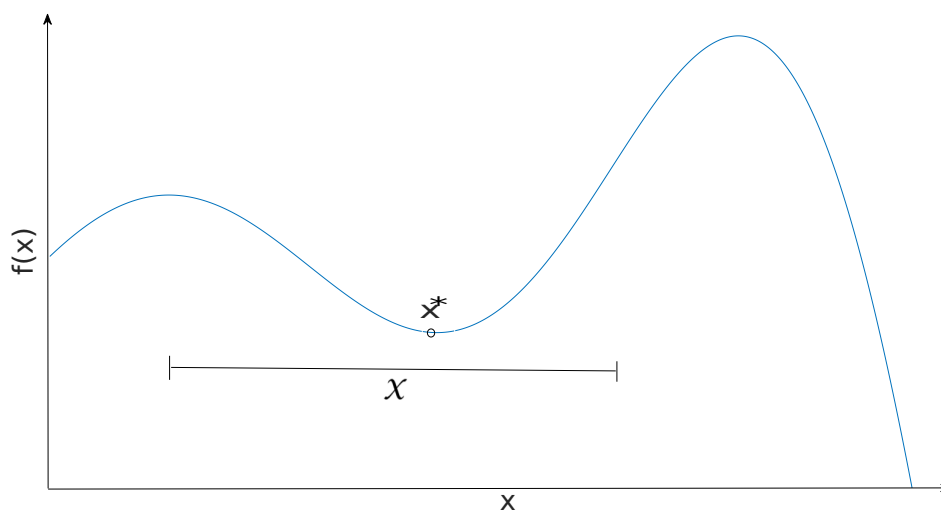
kde

\mathbf{x} je vektor $[x_1, x_2, \dots, x_n]$, ktorý definuje bod v N-rozmernom priestore,
 $f(\mathbf{x})$ je účelová funkcia,
 \mathcal{X} je definičný obor účelovej funkcie.

Potom prvky vo vektore \mathbf{x} možno upraviť, tak aby sa minimalizovala účelová funkcia. Akúkoľvek hodnotu \mathbf{x} v prehľadávanom priestore \mathcal{X} , ktorá minimalizuje účelovú funkciu nazveme riešenie, ktoré možno zapísať ako [2]

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) = \{\mathbf{x}^* \in \mathcal{X} : f(\mathbf{x}^*) \leq f(\mathbf{x}), \text{ pre } \forall \mathbf{x} \in \mathcal{X}\} \quad (1.2)$$

Na obrázku možno vidieť príklad optimalizácie v jeden rozmernom priestore s riešením x^* , ktoré je lokálne minimum.



Obr. 1.1: Príklad optimalizácie v jeden rozmernom priestore

Toto je všeobecná definícia pre hľadanie minima funkcie. Pre hľadanie maxima funkcie možno rovnicu predefinovať na

$$\max_x f(\mathbf{x}), \quad (1.3)$$

riešením bude potom

$$\mathbf{x}^* = \arg \max_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) = \{\mathbf{x}^* \in \mathcal{X} : f(\mathbf{x}^*) \geq f(\mathbf{x}), \text{ pre } \forall \mathbf{x} \in \mathcal{X}\} \quad (1.4)$$

1.2 Obmedzenia optimalizácie

Pre väčšinu optimalizačných úloh je definičný obor \mathcal{X} zužovaný definovanými funkciami

$$g_i(\mathbf{x}) \geq 0, i = 1, 2, \dots, m, \quad (1.5)$$

ktoré priamo určujú priestor pre optimalizáciu. Obmedzenia sú typicky definované s operátormi $\geq, \leq, =, \neq$, prípadne $>, <$. [1]

Napríklad priestor \mathbf{R}^n , možno obmedziť

$$g_1(x_i) \geq -5, \quad (1.6)$$




$$g_2(x_i) \leq 5, \quad (1.7)$$


potom definičný obor \mathcal{X} bude v intervale $< -5; 5 >$ pre všetky x_i , pričom $i \in [1, 2, \dots, n]$ je index rozmeru v prípade optimalizácie v n-rozmernom priestore.

1.3 Metódy optimalizácie


1.3.1 Deterministické a stochastické metódy

Optimalizačné metódy môžeme rôzne rozlišovať, deterministické optimalizácie zaručujú nájdenie optimálneho výsledku.

Gradientové metódy sú príkladom tých  optimalizácií. Vychádzajú z toho, že sa určí nejaký počiatočný bod  \mathbf{x} , z ktorého sa sekvenčne pohybujeme v opačnom smere gradientu do bodu, ktorý nazveme riešením.  Gradient je strmosť funkcie v danom bode, určená vektorom prvých derivácií jednotlivých rozmerov. V prípade týchto metód je nutné zvoliť si správnu veľkosť kroku, pri malej dĺžke kroku bude výsledok presný na úkor viacej výpočtov a naopak pri veľkej dĺžke kroku sa môže stať, že riešenie preskočíme. Keď sa gradient bude rovnať nule našli sme riešenie. [8]

 Avšak s rastúcim rozsahom problému sa výrazne zvyšuje ich časová náročnosť. Preto vznikli stochastické algoritmy, ktoré zaviedli do výpočtu prvok náhody,

postupov na základe skúseností, prípadné intuícií, ktorý výrazne zrýchli proces hľadania výsledku. [3]

Z toho vyplýva, že stochastické algoritmy neprehľadávajú celý priestor riešení účelovej funkcie, ale len jeho časť. Zároveň ich opakované spúšťanie nezaručuje zakaždým rovnaký výsledok, ale len výsledok blížiaci sa optimálnemu riešeniu. 

1.3.2 Heuristické a metaheuristické algoritmy

Heuristické algoritmy sú navrhnuté tak, aby riešili problém efektívnejšie a rýchlejšie ako tradičné metódy výmenou za to, že výsledok nebude úplne presný, optimálny. V princípe sa teda jedná o stochastickú metódu výpočtu.

Metaheuristické algoritmy boli koncipované, aby koordinovali spoluprácu medzi ostatnými vyhľadávacími metódami a riešili komplikované problémy. Hlavnou myšlienkou bolo poskytnúť stratégiu, ktorá sa nezameriava na jeden špecifický problém, ale je použiteľná na mnoho problémov. [4] Zvyčajne napodobňujú správanie zvierat v prírode, prirodzený vývoj situácií alebo fyzikálne princípy.

2 Populačné metódy

Základnou myšlienkou populačných metód je rozmiestnenie súboru návrhových bodov (agentov) s počtom N po celom prehľadávanom priestore. Agenti sa následne vzájomne podľa vopred definovaného princípu ovplyvňujú, tento princíp je priamo definovaný v špecifickom algoritme a iteračne sa pohybujú smerom ku optimu účelovej funkcie. [1] Vyšší počet agentov pomáha algoritmu vyhnúť sa uviaznutiu v lokálnom minime účelovej funkcie. Väčšina populačných metód má stochastickú povahu výpočtov.

Vzhľadom na to, že poloha agenta sa aktualizuje vždy na konci iterácie a nové hodnoty jednotlivých rozmerov závisia len od ich hodnoty v prechádzajúcej iterácii je vo väčšine prípadov ľahké paralelizovať ich výpočet.

2.1 Inicializácia

Výpočet začína generovaním počiatočnej populácie, pod čím rozumieme rozloženie agentov v prehľadávanom priestore. Prehľadávaný priestor je často definovaný dolnými a hornými hranicami \mathbf{a} a \mathbf{b} . Počiatočnú populáciu možno generovať rôznymi spôsobmi.

2.1.1 Náhodné rozloženie agentov

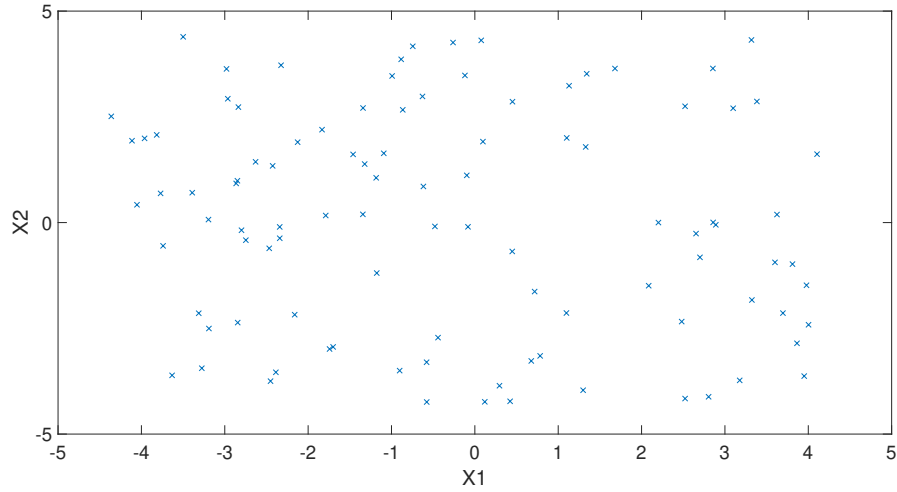
Náhodné rozloženie agentov vychádza zo vzorca

$$x_i^{(m)} = a_i + \phi(b_i - a_i), \quad (2.1)$$

kde

- $\mathbf{x}^{(m)}$ je agent s indexom m
- i je index aktuálnej dimenzie $i \in [1, 2, ..n]$ v n -rozmernom priestore
- ϕ je nahodnné číslo v intervale $< 0; 1 >$
- a je dolná hranica prehľadávaného priestoru
- b je horná hranica prehľadávaného priestoru.

~~Na obrázku~~ možno vidieť výsledné rozloženie pre dvojrozmerný priestor s počtom agentov $N = 100$ a hornou hranicou $b_{1,2} = 4, 5$, dolnou hranicou $a_{1,2} = -4, 5$.



Obr. 2.1: Inicializácia náhodnej populácie

2.1.2 Rozloženie agentov po intervaloch

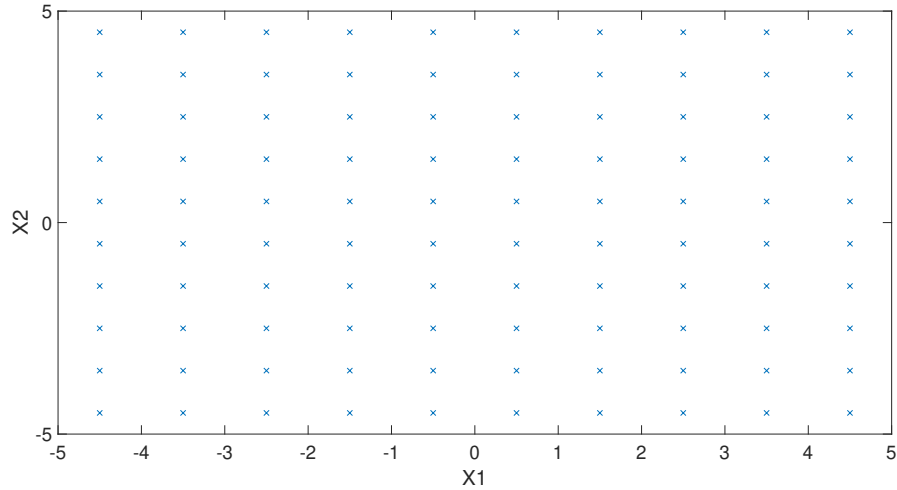
Agenti sú rovnomerne rozložené vo všetkých osiach. Pre jednoduchosť povedzme, že počet agentov N bude druhá mocnina celého čísla, potom

$$x_i^{(m)} = a_i + m \cdot (b_i - a_i) / (m_{max} - 1), \quad (2.2)$$

kde

- $\mathbf{x}^{(m)}$ je agent s indexom m ,
- i je index aktuálnej dimenzie $i \in [1, 2, ..n]$ v n -rozmernom priestore
- m_{max} je druhá odmocnina z N ,
- m je celé číslo z intervalu $< 0; m_{max} >$, stúpajúce od 0 s krokom 1,
- a je dolná hranica prehľadávaného priestoru,
- b je horná hranica prehľadávaného priestoru.

Na obrázku možno vidieť výsledné rozloženie pre dvojrozmerný priestor s počtom agentov $N = 100$ a hornou hranicou $b_{1,2} = 4, 5$, dolnou hranicou $a_{1,2} = -4, 5$.



Obr. 2.2: Inicializácia populácie po intervaloch

2.2 Evolučné algoritmy

Vo svojej podstate sú evolučné algoritmy jednoduchými modelmi Darwinovej evolučnej teórie. [2] Jedná sa o metaheuristické, populačné algoritmy, ktorých princípy sú inšpirované biologickou evolúciou, mutáciou, selekciou najlepších jedincov, a iných.

2.2.1 Diferenčná evolúcia

Diferenčnej evolúcia (DE) sa pokúša optimalizovať každého agenta v populácii rekombináciou iných agentov. Vstupnými parametrami sú pravdepodobnosť kríženia p v intervale $< 0; 1 >$ a rozdielová váha w , ktorá je typicky číslo z intervalu $< 0, 4; 1 >$.

Priebeh algoritmu vyzerá nasledovne, pre každého jedinca $\mathbf{x}^{(m)}$

1. skombinuje náhodných agentov, vzhľadom na to, že princíp kombinácie sa môže líšiť tu sú dva príklady

- z populácie sa vyberú traja náhodný agenti $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$ a skombinujú sa podľa vzorca [1]

$$\mathbf{u}^{(m)} = \mathbf{r}_1 + w(\mathbf{r}_2 - \mathbf{r}_3), \quad (2.3)$$

- v druhom prípade sa z populácie vyberú až štyria náhodný agenti $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \mathbf{r}_4$ a doteraz najlepšie riešenie \mathbf{x}_{best} , následne sa skombinujú podľa vzorca [2]

$$\mathbf{u}^{(m)} = \mathbf{x}_{\text{best}} + w(\mathbf{r}_1 + \mathbf{r}_2 - \mathbf{r}_3 - \mathbf{r}_4), \quad (2.4)$$

2. vyberie sa náhodná dimenzia $j \in [1, 2, \dots, n]$ v n -rozmernom priestore,
3. skonštruuje sa dočasný vektor $\mathbf{y}^{(m)}$ krížením, pre ktorého každý rozmer i platí

$$y_i^{(m)} = u_i^{(m)} \text{ ak } i=j, \text{ alebo s pravdepodobnosťou } \frac{1}{n} \text{ v opačnom prípade } y_i^{(m)} = x_i^{(m)} \quad (2.5)$$

4. pre ďalšiu generáciu sa vyberie riešenie bližšie optimu medzi $\mathbf{x}^{(m)}$ a $\mathbf{y}^{(m)}$

2.3 Algoritmy rojovej inteligencie

Algoritmy sú založené na kolektívnom správaní decentralizovaných a samoorganizovaných systémov. Tieto systémy tvoria agenti, ktorí interagujú medzi sebou a s prostredím, podobne ako to robia skutočné roje v prírode.

2.3.1 Optimalizácia rojením častíc

Pre urýchlenie konvergenzie smerom k minimám účelovej funkcie zavádza optimalizácia rojením častíc (PSO) hybnosť pohybu agenta. Každý agent sleduje svoju aktuálnu polohu, rýchlosť a najlepšiu nájdenú polohu. V každej iterácii sa pohybuje smerom ku najlepšej pozícii, v ktorej sa kedy nachádzal a zároveň smerom ku doteraz nájdenej najlepšej pozícii v rámci celej populácie. [1] Pohyb smerom ku najlepšej pozícii je násobený číslom w z intervalu $< 0; 1 >$.

Potom sa nová poloha agenta vypočíta ako

$$\mathbf{x}(k+1)^{(m)} = \mathbf{x}(k)^{(m)} + \mathbf{v}(k)^{(m)}, \quad (2.6)$$

a hodnota vektoru $\mathbf{v}^{(m)}$ pre ďalšiu iteráciu ako

$$\mathbf{v}(k+1)^{(m)} = w\mathbf{v}(k)^{(m)} + c_1r_1(\mathbf{x}(k)_{best}^{(m)} - \mathbf{x}(k)^{(m)}) + c_2r_2(\mathbf{x}(k)_{best} - \mathbf{x}(k)^{(m)}), \quad (2.7)$$

kde

- w, c_1, c_2 sú voliteľné parametre, pričom $c_1, c_2 \in \mathbf{R}$,
- r_1, r_2 sú náhodné hodnoty z intervalu $< 0; 1 >$,
- $\mathbf{x}_{best}^{(m)}$ je doteraz najlepšia hodnota v akej sa agent nachádzal,
- \mathbf{x}_{best} je doteraz najlepšia nájdená hodnota.

2.3.2 Algoritmus svetlušiek

Algoritmus svetlušiek (FF) sa inšpiruje chovaním týchto zvierat v období párenia, kedy sa svetielkovaním snažia prilákať jedincov opačného pohlavia. Svetlušky sa iteratívne pohybujú smerom ku tým s vyššou intenzitou svietenia. [1]

Intenzita svietenia je funkciou vzdialenosti medzi dvomi príslušnými agentami, ktorú možno zapísať ako

$$I(r) = e^{-\gamma r^2}, \quad (2.8)$$

kde γ je voliteľný parameter blízky nule s kladným znamienkom.

Pohyb agenta $\mathbf{x}^{(m)}$ smerom ku agentovi $\mathbf{y}^{(m)}$ zapíšeme ako

$$\mathbf{x}(k+1)^{(m)} = \mathbf{x}(k)^{(m)} + \beta I(\|\mathbf{y}(k)^{(m)} - \mathbf{x}(k)^{(m)}\|)(\mathbf{y}(k)^{(m)} - \mathbf{x}(k)^{(m)}) + \alpha \epsilon, \quad (2.9)$$

kde

- α, β sú voliteľné parametre, ktoré patria do intervalu $< 0; 1 >$,
- ϵ je náhodný pohyb, ten sa určí normálnym rozdelením okolo nuly s rozptylom 1.

2.3.3 Algoritmus umelých včelých kolónií

Algoritmus sa skladá z troch skupín včiel

- robotnice, ktoré aplikujú štruktúru náhodného susedstva pre každého agenta a vypočítajú hodnotu spôsobilosti riešenia,
- sledovatelia, ktorí vyberú agenta na základe spôsobilosti riešenia a opäť aplikujú štruktúru náhodného susedstva,
- prieskumníci, ktorí nahradia riešenie, ktoré sa už nedá vylepšiť novým riešením, to či sa už nedá vylepšiť určuje premenná d , ktorej maximálna hodnota d_{max} je voliteľný parameter, $d_{max} \in \mathbf{Z}$.

Tieto tri skupiny sa snažia nájsť optimálne riešenie účelovej funkcie. [5]

Algoritmus umelých včelých kolónií (ABC) bol pôvodne určený pre numerickú optimalizáciu.

Štruktúra náhodného susedstva pre agenta $\mathbf{x}^{(m)}$ spočíva v troch krokoch:

1. Vyberie sa náhodný agent $\mathbf{x}^{(n)}$, pričom $m \neq n$,
2. následne sa pre každý rozmer $i \in [1, 2, \dots, n]$ vypočíta

$$x_i(k+1)^{(m)} = x_i(k)^{(m)} + \Phi_k(x_i(k)^{(m)} - x_i(k)^{(n)}), \quad (2.10)$$

Φ je náhodné číslo z voliteľného intervalu $< -c, c >$, pričom $c \in \mathbf{R}$

3. pokiaľ nové riešenie nie je kvalitnejšie voči starému pripočíta sa 1 ku premennej $d^{(m)}$, v opačnom prípade sa nové riešenie uloží.

Prieskumníci kontrolujú každého agenta $\mathbf{x}^{(m)}$, pokiaľ jeho hodnota stúpne

na

$d^{(m)} = d_{max}$, potom toto riešenie nahradia novým, napríklad podľa vzorca [6]

$$\mathbf{x}(k+1)^{(m)} = \mathbf{r}_{j1}\mathbf{x}(k)^{(m)} + \mathbf{r}_{j2}\mathbf{x}(k)_{min} + \mathbf{r}_{j3}(\mathbf{x}(k)_{r1} - \mathbf{x}(k)_{r2}), \quad (2.11)$$



kde

- všetky \mathbf{r}_j sú navzájom rôzne hodnoty určené náhodne v intervale $< 0; 1 >$,
- všetky \mathbf{x}_r sú navzájom rôznych agentů určený náhodne,
- \mathbf{x}_{\min} je doteraz najlepšie nájdené riešenie.

2.3.4 Optimalizácia svorkou šedých vlkov

Algoritmus je založený na správaní sa šedého vlka pri love v prírode. Vyberú sa tri najlepšie riešenia, ktoré nazveme α, β a γ . Následne sa vypočíta koeficient a , ktorého počiatočná hodnota je 2 a s každou iteráciou klesá k 0 podľa vzorca [7]

$$a = 2 - 2k/k_{\max}, \quad (2.12)$$

kde

- k je aktuálna iterácia
- k_{\max} je maximálny počet iterácii.

Potom sa pre každého agenta $\mathbf{x}^{(m)}$ vypočítajú tri vektory \mathbf{X}_P ako

$$\mathbf{X}_P = \mathbf{P} - A \times \mathbf{D}_P, \quad (2.13)$$

$$A = 2ar_1 - a, \quad (2.14)$$

$$\mathbf{D}_P = |2r_2\mathbf{P} - \mathbf{X}|, \quad (2.15)$$

kde

- \mathbf{P} sú nájdené riešenia α, β, γ
- $r_{1,2}$ sú náhodné čísla z intervalu $0; 1$
- $\mathbf{x}^{(m)}$ je poloha aktuálneho agenta.

Nová poloha sa nakoniec určí z priemeru \mathbf{X}_P

$$\mathbf{x}^{(m)} = \frac{\mathbf{X}_\alpha + \mathbf{X}_\beta + \mathbf{X}_\gamma}{3}, \quad (2.16)$$

Algoritmus možno upraviť aby sa znížila pravdepodobnosť uviaznutia v lokálnom minime. Vylepšená verzia spočíva v tom, že sa spočíta vzdialenosť R medzi minulou a vypočítanou polohou agenta, v tejto vzdialenosti R sa náhodne vyberie riešenie $\mathbf{X}^{(n)}$. [7] Následne sa pre každý rozmer vypočíta nová hodnota podľa vzorca

$$X_i(k+1)^{(m)} = X_i(k)^{(m)} + r(X_i(k)^{(n)} - X_i(k)^{(o)}), \quad (2.17)$$

kde

- $\mathbf{X}^{(o)}$ je náhodný agent z populácie,
- r je náhodné číslo z intervalu $<0;1>$
- i je index počítaný rozmer, $i \in [1, 2, \dots, n]$ v n -rozmernom priestore.

3 NVIDIA CUDA



Snaha zrýchliť komplikované výpočty vedie k paralizácii výpočtov, ku čomu je určená práve platforma CUDA. Túto platformu podporujú určité typy grafických procesorov (GPU) NVIDIA. CUDA predstavuje nadstavbu pre programovacie jazyky, ako sú C, C++, Python, Matlab, a tak ďalej.

3.1 Kvalifikácia funkcií a premenných

Vzhľadom na to, že kód beží zároveň na CPU ako aj GPU, sa kód rozdeľuje do dvoch skupín:

- host-code - beží na CPU
- device-code - beží na GPU

Kvalifikátor `__host__` je voliteľný a určuje, že funkciu je možné volať len z CPU a beží ako host-code. V prípade, že sa nachádza pred premennou, je premenná definovaná na CPU.

V prípade kvalifikátoru `__device__` určuje, že funkciu je možné volať len z GPU a beží ako device-code. V prípade, že sa nachádza pred premennou, je premenná definovaná na GPU.

V prípade, že sa tieto dva kvalifikátory použijú súčasne funkcia alebo premenná bude mať oba atribúty.

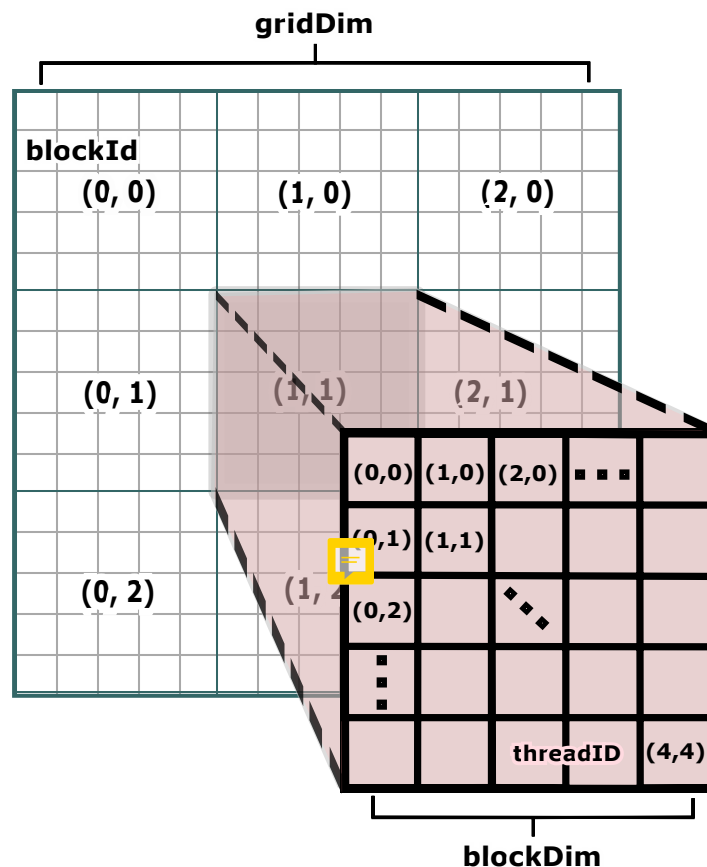
Kvalifikátor `__global__` definuje funkciu volanú z CPU, ktorá beží ako device-code. Tieto funkcie sa volajú spolu s konfiguráciou pre paralelizáciu. Kernel je synonymum pre `__global__` funkciu.

Nakoniec kvalifikátor `__shared__`, ktorý definuje premennú na GPU s dátami dostupnými len v rámci jedného bloku vlákien.

3.2 Paralelizácia

Určuje ju konfigurácia pri volaní kernel funkcie. Táto konfigurácia určí koľko procesov bude prebiehať paralelne. Píše sa pred parametre, ktoré predávame funkcií ako `<<< blocksPerGrid, threadsPerBlock >>>`, tieto parametre sú buď dátového typu `int` alebo `dim3`. Prvý parameter predstavuje počet blokov a druhý počet vlákien na jeden blok, ktoré budú bežať paralelne. Dátový typ `dim3` predstavuje štruktúru s tromi premennými typu `int` (`x`, `y`, `z`) určujúcimi veľkosť jednotlivých rozmerov, či už blokov alebo vlákien. Tieto rozmery sú obmedzené vzhľadom na jednotlivé GPU.

Na 3.1 môžeme vidieť vizualizáciu konfigurácie kernel funkcie pre počet blokov 3x3 a počet vlákien na blok 5x5.



Obr. 3.1: Príklad konfigurácie kernel funkcie

Ku jednotlivým vláknám následne pristupujeme pomocou vstavaných premenných, ktoré predstavujú:

- `threadIdx.x` - číslo vlákna v aktuálnom bloku
- `blockIdx.x` - číslo aktuálneho bloku
- `blockDim.x` - počet vlákien na blok
- `gridDim.x` - počet všetkých blokov dohromady

Následne napríklad číslo aktuálneho vlákna v rámci rozmeru `x` zistíme nasledovne :

$$\text{threadIdx.x} + \text{blockIdx.x} * \text{blockDim.x},$$

v prípade, že chceme pristupovať ku medzivýpočtom iných vlákien je dobré ich synchronizovať pomocou funkcie `__syncthreads()`.

4 Implementácia v programe Matlab



Pre ~~implementáciu~~ algoritmov bol použitý program matlab. Implementovaných bolo päť populačných algoritmov pre optimalizácie. Tieto algoritmy sú popísané v kapitole 2. V prípade diferenčnej evolúcie a optimalizácie svorkou šedých vlkov je k dispozícii základná a vylepšená verzia. Vylepšená verzia diferenčnej evolúcie by mala konvergovať rýchlejšie ku minimu účelovej funkcie, vylepšená optimalizácia svorkou šedých vlkov upravuje algoritmus aby sa zamedzilo jeho uviaznutiu v lokálnom minime.

4.1 Inicializácia populácie

Výpočet počiatočnej populácie je možný dvoma spôsobmi. V prvom prípade sa jedná o náhodné rozloženie agentov, v druhom o rozloženie po intervaloch. Inicializácia prebieha vo funkcii **pop_init**, ktorej prvý parameter určuje počet agentov, druhý funkciu, ktorá sa bude optimalizovať, tretí parameter určuje ako sa bude poloha agentov počítat. Pokiaľ sa tretí parameter rovná 1 inicializácia prebehne náhodne, v opačnom prípade po intervaloch. Príklad volania funkcie je vo výpise kódu A.2.

4.2 Algoritmy pre optimalizácie

Každý algoritmus začína inicializáciou premenných, po ktorom nasleduje samotný algoritmus. Ich výstupom je pole funkčných hodnôt najbližších optimu funkcie pre každú iteráciu, súradnice najlepšieho nájdeného riešenia a pole obsahujúce trvanie každej iterácie v sekundách. O hľadanie najlepšieho riešenia v populácii sa stará funkcia **find_best_sol**. Volanie funkcie je vo výpise kódu A.3.

5 Implementácia na platforme CUDA

6 Testovacie funkcie

Výpočet hodnoty funkcie v bode, ako aj obor, v ktorom je definovaná sú výstupmi funkcie **input_func**. Jej vstupnými parametrami sú poloha agenta a celočíselný parameter, ktorý vyjadruje vybranú testovaciu funkciu.

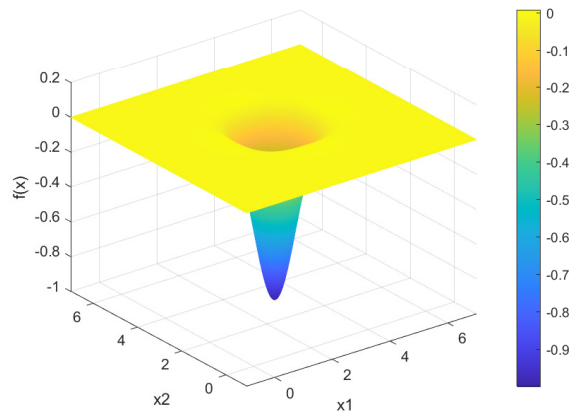
Sú to funkcie pre testovanie algoritmov pre optimalizácie. Oblasti, v ktorých sa testujú sú zvyčajne rýchlosť konvergenencie, presnosť výpočtu, robustnosť, a tak ďalej. Príklad volania funkcie je vo výpise kódu A.1.

6.0.1 Easom's function

Funkcia, ktorej minimum sa nachádza na malom priestore oproti prehľadávanej oblasti. Jej hodnotu v danom bode určuje vzorec:

$$f(x_1, x_2) = -\cos(x_1) \cos(x_2) \exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2), \quad (6.1)$$

v prehľadávanom priestore $-100 \leq x_1, x_2 \leq 100$ sa minimum nachádza v bode $f(\pi, \pi) = -1$ [9]. Graf funkcie sa nachádza na grafe na obrázku 6.1.



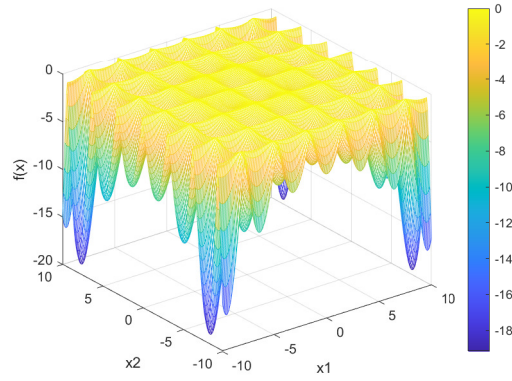
Obr. 6.1: Graf Easom's function

6.0.2 Hölder table function

Funkcia má 4 body s rovnakým minimom na kraji prehľadávaného priestoru $-10 \leq x_1, x_2 \leq 10$ v bodoch $x_1 = \pm 8,05502$ a $x_2 = \pm 9,66459$ s hodnotou $-19,2085$, funkčnú hodnotu určuje [10]:

$$f(x_1, x_2) = -\left| \sin(x_1) \cos(x_2) \exp\left(\left|1 - \frac{\sqrt{x_1^2 + x_2^2}}{\pi}\right|\right) \right|. \quad (6.2)$$

Graf funkcie sa nachádza na grafe na obrázku 6.2.



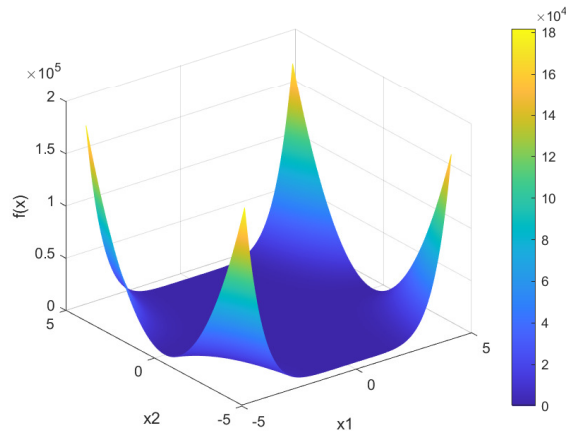
Obr. 6.2: Graf Hölder table function

6.0.3 Beale function

Funkcia, ktorá má veľa hodnôt blízkyh minimu na veľej ploche. Jej vzorec je nasledovný:

$$f(x_1, x_2) = (1,5 - x_1 + x_1x_2)^2 + (2,25 - x_1 + x_1x_2^2)^2 + (2,625 - x_1 + x_1x_2^3)^2, \quad (6.3)$$

v prehľadávanom priestore $-4,5 \leq x_1, x_2 \leq 4,5$ sa minimum nachádza v bode $f(3; 0,5) = 0$ [10]. Graf funkcie sa nachádza na grafe na obrázku 6.3.



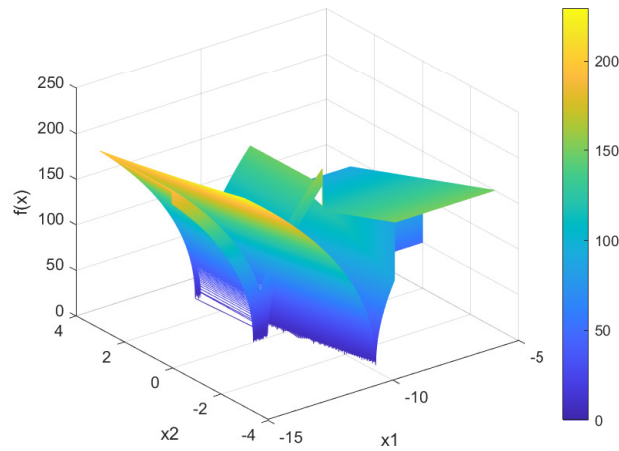
Obr. 6.3: Graf Beale function

6.0.4 Bukin function 6

Funkcia, ktorá veľa hodnôt blízkyh minimu v jednom páse naprieč rozmeru x_1 . Funkčnú hodnotu popisuje vzorec:

$$f(x_1, x_2) = 100\sqrt{|x_2 - 0,01x_1^2|} + 0,01|x_1 + 10|, \quad (6.4)$$

v prehľadávanom priestore $-15 \leq x_1 \leq -5$ a $-3 \leq x_2 \leq 3$ sa minimum nachádza v bode $f(-10; 1) = 0$ [10]. Graf funkcie sa nachádza na grafe na obrázku 6.4.



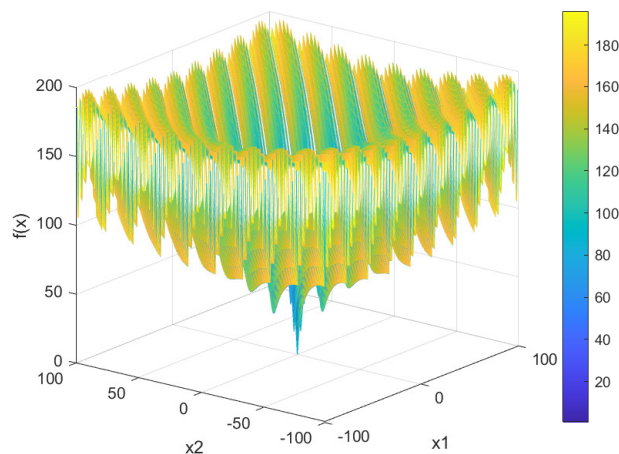
Obr. 6.4: Graf Bukin function

6.0.5 Layeb10 function

Multimodálna funkcia s mnoho lokálnymi minimami v celom prehľadávanom priestore. Funkčnú hodnotu pre n-roznerov popisuje vzorec:

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} (((\ln(x_i^2 + x_{i+1}^2 + 0,5))^2 + |(100 \sin(x_i - x_{i+1}))|), \quad (6.5)$$

v prehľadávanom priestore $-100 \leq x_i \leq 100$ sa minimum nachádza v bode $f(\mathbf{x}^*) = 0$, $\mathbf{x}^* = [0, 5; 0, 5 \dots 0, 5]$ [11]. Graf funkcie sa nachádza na grafe na obrázku 6.5.



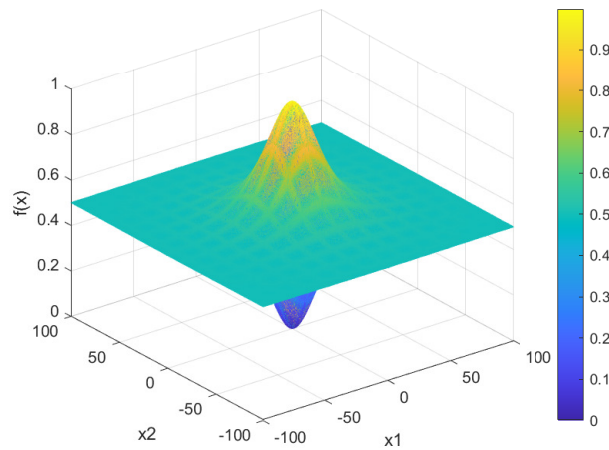
Obr. 6.5: Graf Layeb function

6.0.6 Schaffer function

V okolí minima má funkcia mnoho symetricky rozložených hodnôt. Funkčnú hodnotu pre n -rozmerov popisuje vzorec:

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} 0.5 + \frac{\sin^2(x_i^2 - x_{i+1}^2) - 0.5}{[1 + 0.001(x_i^2 + x_{i+1}^2)]^2}, \quad (6.6)$$

v prehľadávanom priestore $-100 \leq x_i \leq 100$ sa minimum nachádza v bode $f(\mathbf{x}^*) = 0$, $\mathbf{x}^* = [0; 0 \dots 0]$ [10]. Graf funkcie sa nachádza na grafe na obrázku 6.6.



Obr. 6.6: Graf Schaffer function

7 Testovanie implementovaných algoritmov

V tejto kapitole sa budem venovať testovaniu a porovnávaniu implementovaných algoritmov pre optimalizácie. Jedná sa o 5 algoritmov, ktorými sú Diferenčná evolúcia, Optimalizácia rojením častíc, Algoritmus svetlušiek, Algoritmus umelých včelích kolónií a Optimalizácia svorkou šedých vlkov. V prípade Diferenčnej evolúcie a Optimalizácie svorkou šedých vlkov sú k dispozícii dve verzie.

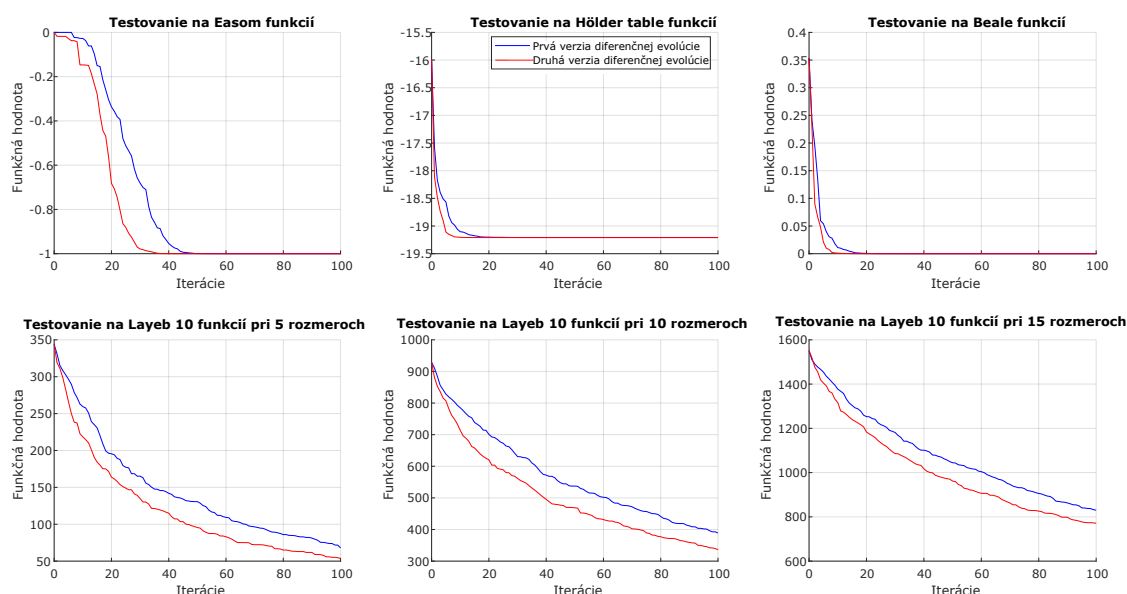
7.1 Porovnanie verzií implementovaných algoritmov

7.1.1 Porovnanie pôsobu rekombinácie agentov

Spôsob rekombinácie agentov je jediným rozdielom dvoch variant tohto algoritmu. Prvá verzia rekombinuje agentov podľa vzorca 2.3, druhá podľa vzorca 2.4.

Vzhľadom na to, že sa líšia len v jednom výpočte časová náročnosť algoritmu zostáva rovnaká. Pri testovaní na rôznych funkciách je zjavné, že druhá verzia algoritmu konverguje ku optimálnemu riešeniu rýchlejšie ako prvá pri rovnakej počítateľnej populácii. Testy som robil s počtom agentov 100, vykreslená je priemerná hodnota na iteráciu pri 20 spusteniach algoritmov.

V grafe na obrázku 7.1 je vidieť príklad konvergenzie algoritmov k optimálnemu riešeniu na rôznych funkciách. Pri testovacej funkcii Layeb10 som zvyšoval počet rozmerov, výsledok ostal rovnaký. Druhá verzia diferenčnej evolúcie konverguje rýchlejšie.

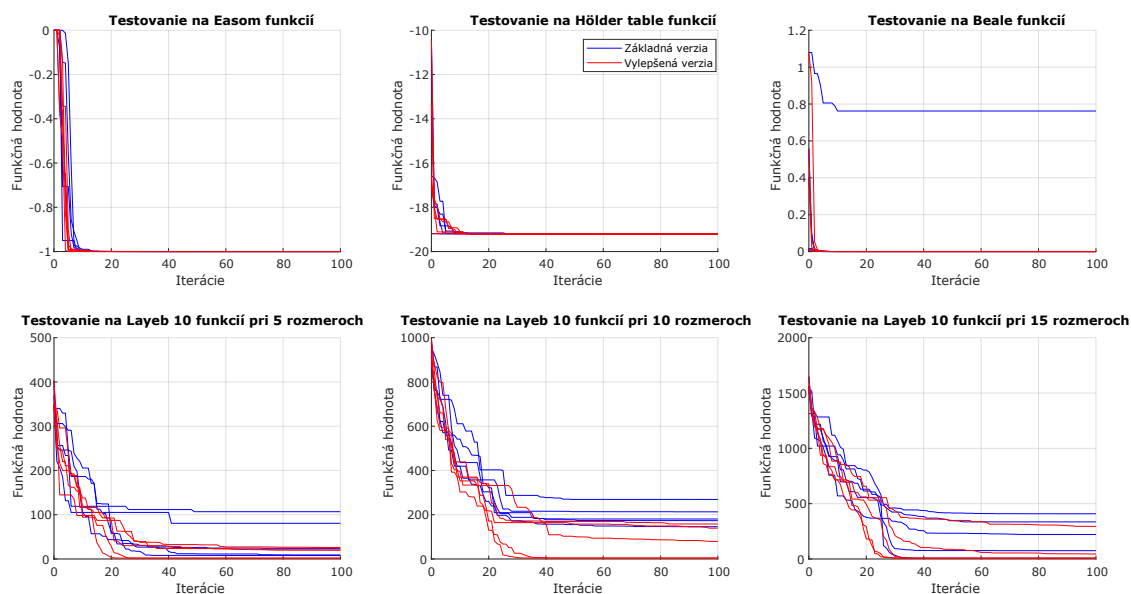


Obr. 7.1: Porovnanie dvoch verzií diferenčnej evolúcie za rôznych okolností

7.1.2 Testovanie optimalizácie svorkou šedých vlkov

Pre optimalizáciu svorkou šedých vlkov je k dispozícii základná a vylepšená verzia, ktorá po výpočte základnej verzie vytvorí v susedstve aktuálneho agenta nového podľa vzorca 2.17 a vyberie sa vhodnejšie riešenie. Tento prístup zníži pravdepodobnosť uviaznutia algoritmu v lokálnom minime funkcie. [7]

V grafe na obrázku 7.2 môžeme vidieť príklad toho ako vylepšená verzia nemá tendenciu ostať v lokálnom minime funkcie, teda je spoľahlivejšia, pri rovnakej počiatkovej populácii. Testy som opäť robil s počtom agentov 1000 a vykreslených je 5 sputení.

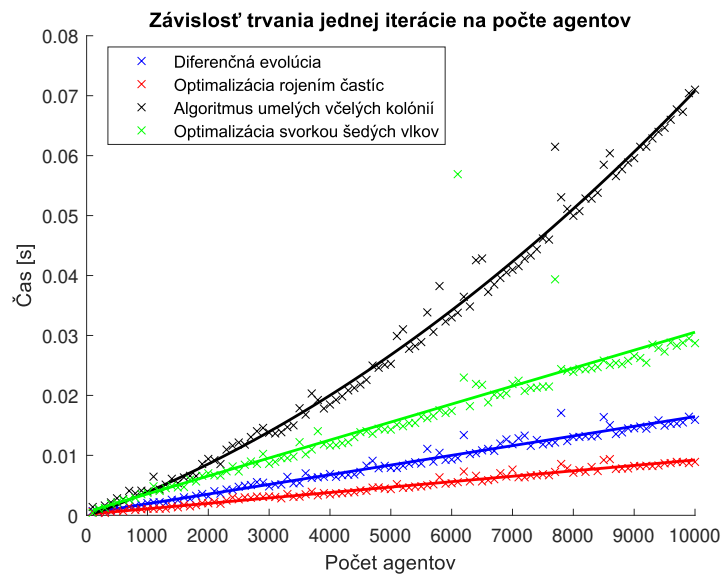


Obr. 7.2: Porovnanie dvoch verzií diferenčnej evolúcie za rôznych okolností

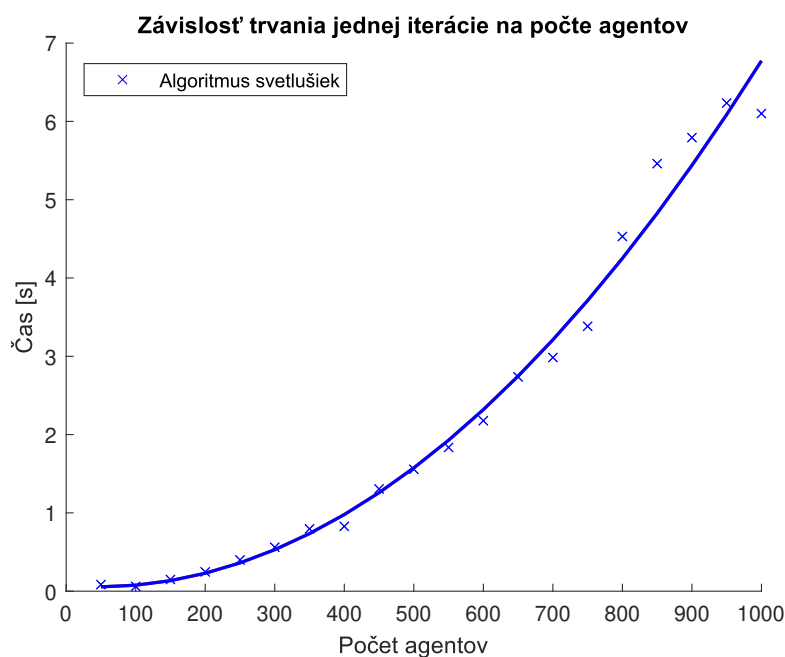
7.2 Časová náročnosť algoritmov

V grafoch na obrázkoch 7.3 a 7.4 je vidieť časová závislosť implementovaných algoritmov na počte agentov. V grafoch je vždy zobrazená priemerná hodnota z 50 iterácií.

Priemerný čas jednej iterácie algoritmu svetlušiek stúpal exponenciálne, to isté platí pre umelé včelie kolónie ale výrazne pomalšie. V prípade algoritmu svetlušiek je to spôsobené vnoreným cyklom for. Časová náročnosť pri zvyšných algoritmoch stúpala lineárne.



Obr. 7.3: Závislosť času na počte agentov



Obr. 7.4: Závislosť trvania jednej iterácie algoritmu svetlušiek na počte agentov

7.3 Závislosť presnosti výpočtu na počte iterácií

V tabuľke 7.1 je zobrazená priemerná hodnota z 10 spustení algoritmov pri 50 agentoch.

Všetky algoritmy už pri počte iterácií 10 dosiahli výsledok dosť blízky optimálnemu riešeniu vzhľadom ku maximálnej funkčnej hodnote, s výnimkou Layeb10 funkcie,

ktorá je náročná na optimalizáciu. Pri počte iterácií 50 boli výnimkou už len optimalizácia rojením častíc a Algoritmus svetlušiek. Pre počet iterácií nevyhovovala už len Optimalizácia rojením častíc, ktorá je pravdepodobne náchyľnejšia na uviaznutie v lokálnom minime.

Tab. 7.1: Závislosť presnosti výpočtu na počte iterácií

Algoritmus	Testovacia funkcia	f_{min}	Počet iterácií			
			10	50	100	500
DE	Beale	0	2,35E-05	7,62E-02	2,29E-01	0,00E+00
	Bukin	0	7,32E-01	2,47E-02	2,96E-02	2,78E-02
	Layeb10	0	6,73E+00	3,78E-05	2,31E-08	0,00E+00
	Schaffer N.2	0	8,22E-03	0,00E+00	0,00E+00	0,00E+00
PSO	Beale	0	8,39E-02	2,37E-01	2,42E-01	7,62E-02
	Bukin	0	3,66E-01	3,01E-02	2,43E-02	2,69E-02
	Layeb10	0	1,73E+01	5,10E+00	7,25E+00	5,16E+00
	Schaffer N.2	0	4,56E-03	2,08E-10	0,00E+00	0,00E+00
FF	Beale	0	5,66E-05	2,08E-05	7,64E-02	2,33E-06
	Bukin	0	6,38E-01	2,98E-01	2,57E-01	8,26E-02
	Layeb10	0	9,06E+00	2,59E+00	3,06E-01	4,74E-03
	Schaffer N.2	0	1,38E-07	1,57E-08	5,19E-09	1,28E-09
ABC	Beale	0	6,18E-03	7,36E-10	4,71E-18	0,00E+00
	Bukin	0	7,32E-01	2,47E-02	2,96E-02	2,78E-02
	Layeb10	0	4,97E+00	1,25E-01	6,58E-03	0,00E+00
	Schaffer N.2	0	7,14E-03	1,37E-08	1,59E-12	0,00E+00
GWO	Beale	0	1,09E-05	1,71E-07	1,10E-08	3,98E-10
	Bukin	0	1,83E+00	5,69E-01	5,52E-01	2,14E-01
	Layeb10	0	3,27E+00	8,61E-01	5,74E-01	3,89E-01
	Schaffer N.2	0	4,56E-03	2,08E-10	0,00E+00	0,00E+00

7.4 Závislosť presnosti výpočtu na počte agentov

V tabuľke 7.2 je zobrazená priemerná hodnota z 10 spustení algoritmov pri 50 iteráciach.

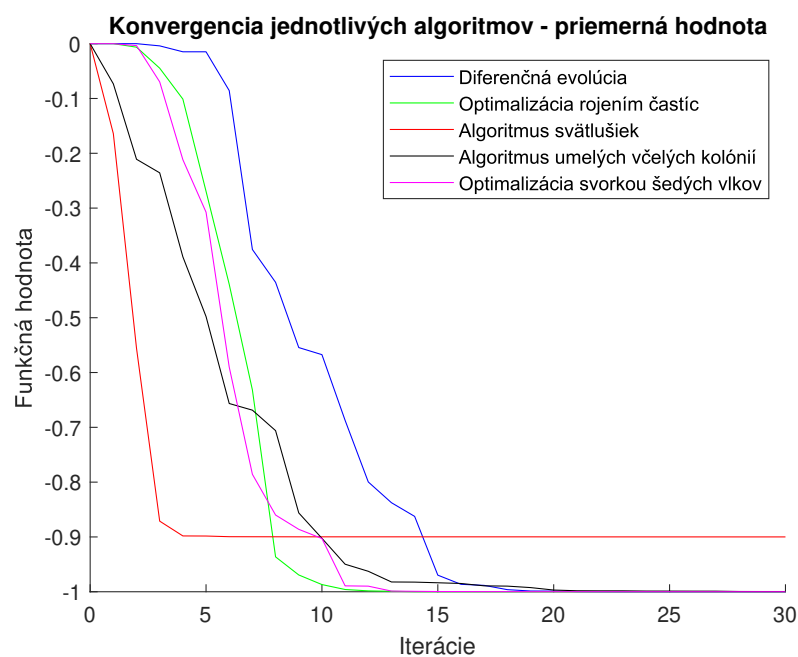
S počtom agentov 10 algoritmy dosiahli horšie výsledky ako pri 10 iteráciach s počtom agentov 50. Pri vyšších počtoch agentov (100, 500) sú výsledky robustnejšie.

Tab. 7.2: Závislosť presnosti výpočtu na počte agentov

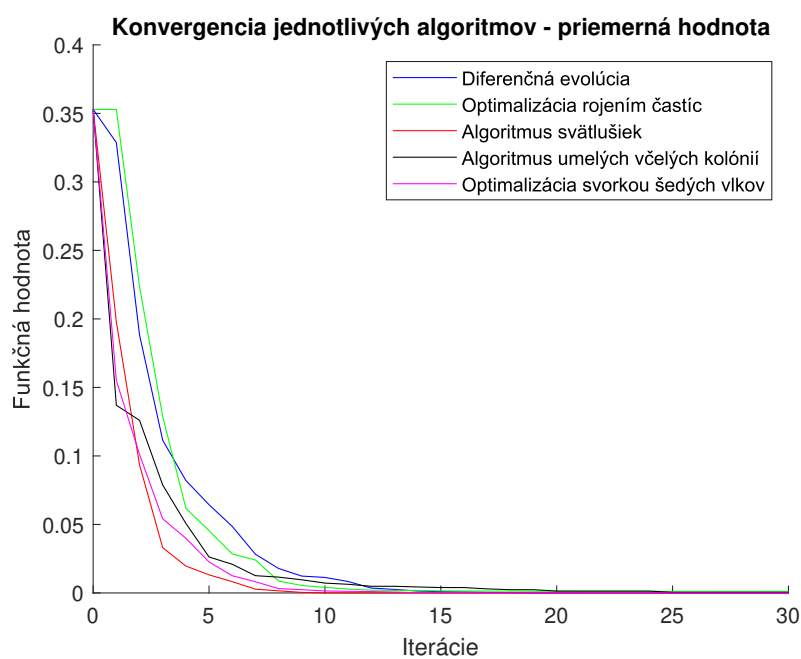
Algoritmus	Testovacia funkcia	f_{min}	Počet agentov			
			10	50	100	500
DE	Beale	0	1,10E+00	7,62E-02	7,62E-02	5,62E-22
	Bukin	0	1,97E-02	1,92E-02	1,75E-02	2,30E-02
	Layeb10	0	7,03E+00	6,20E-05	9,45E-06	2,00E-06
	Schaffer N.2	0	1,33E+01	2,77E-05	5,71E-06	2,60E-06
PSO	Beale	0	1,44E+00	1,63E-01	1,63E-27	1,11E-32
	Bukin	0	2,23E-02	1,62E-02	1,64E-02	2,33E-02
	Layeb10	0	2,84E+01	4,20E+00	2,87E-01	2,87E-01
	Schaffer N.2	0	1,73E-01	1,04E-03	0,00E+00	0,00E+00
FF	Beale	0	2,34E-01	2,88E-05	3,81E-06	1,45E-06
	Bukin	0	1,72E+00	2,91E-01	1,68E-01	8,96E-02
	Layeb10	0	3,74E+01	3,07E-01	8,18E-03	3,25E-03
	Schaffer N.2	0	2,51E-01	1,49E-08	5,02E-09	6,92E-10
ABC	Beale	0	7,62E-02	5,00E-11	2,17E-11	3,37E-12
	Bukin	0	4,86E-02	5,34E-02	4,78E-02	4,34E-02
	Layeb10	0	4,95E+00	8,30E-01	7,51E-02	2,04E-02
	Schaffer N.2	0	2,11E-03	1,79E-08	1,58E-08	2,43E-09
GWO	Beale	0	7,64E-02	1,20E-07	2,79E-08	5,60E-09
	Bukin	0	1,60E+00	6,78E-01	4,94E-01	3,39E-01
	Layeb10	0	2,31E+00	9,75E-01	5,73E-01	2,87E-01
	Schaffer N.2	0	3,92E-04	1,72E-07	0,00E+00	0,00E+00

7.5 Porovnanie konvergenzie jednotlivých algoritmov

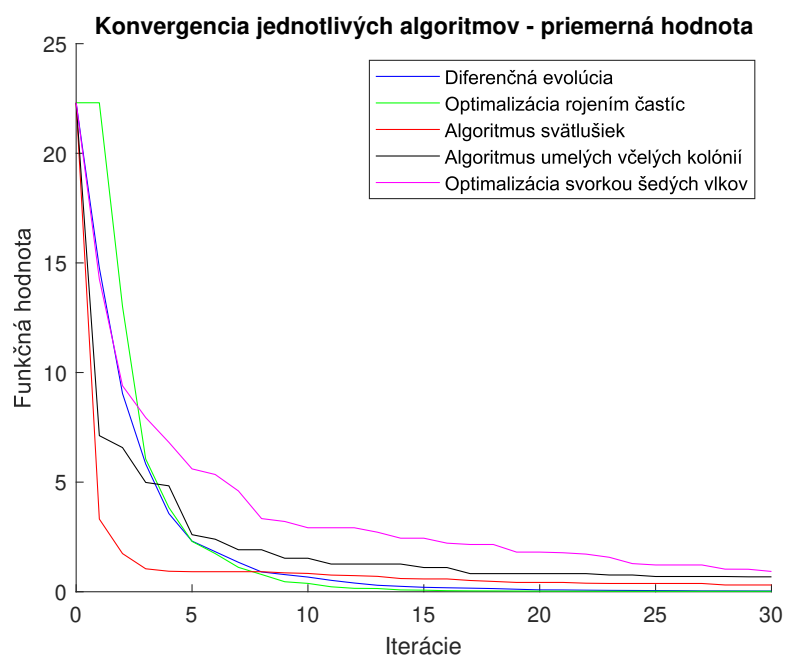
V grafoch na obrázkoch 7.5, 7.6, 7.7, 7.8 su zobrazené priemerné hodnoty z 10 spustení pri 50 agentoch a 50 iteráciách. Algoritmus svetlušiek má najlepšiu konvergenciu, ale je náchylný na uviaznutie v lokálnom minime. Pri zvyšných algoritmoch sa javí, že ich konvergencia závisí skôr od testovanej funkcie.



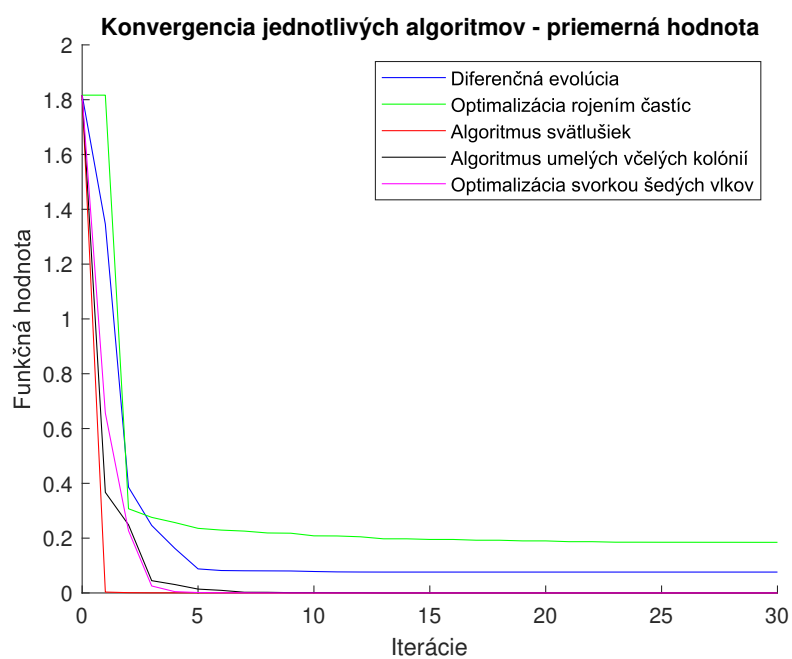
Obr. 7.5: Priemerná hodnota algoritmov na Easom function



Obr. 7.6: Priemerná hodnota algoritmov na Schaffer function



Obr. 7.7: Priemerná hodnota algoritmov na Bukin function



Obr. 7.8: Priemerná hodnota algoritmov na Beale function

Závěr

V práci som skúmal možnosti optimalizácií komplikovaných funkcií. V prostredí matlab som následne implementoval jeden evolučný algoritmus a štyri algoritmy rojovej inteligencie.

V prípade algoritmov, ktoré mali viacej verzií, tá druhá vždy splnila účel aký mala. Pre diferenčnú evolúciu, komplikovanejšia metóda kríženia konvergovala rýchlejšie. Základná verzia algoritmu svorkou šedých vlkov je dosť nachylná na uviaznutie v lokálnom minime, čo jej vylepšená verzia kompenzovala.

Čo sa týka rýchlosti konvergenzie je najlepší algoritmus svetlušiek, avšak jednoducho uviazne v lokálnom minime a vzhľadom na počet agentov má dosť veľkú časovú náročnosť. Najlepšiu časovú závislosť má optimalizácia rojenia častíc, konverguje rovnako ako zvyšné algoritmy ale nie je tak robustná. Z pohľadu robustnosti, časovej náročnosti sú určite najlepšie diferenčná evolúcia a optimalizácia svorkou šedých vlkov. Jedinou nevýhodou algoritmu umelých včelích kolónií je nelineárna časová závislosť na počte agentov, ktorá ale nie je tak výrazná.

V rámci práce som mal ešte v pláne porovnať konvergenciu pri rôznych typoch inicializácie populácie ale vzhľadom na to, že pri inicializácií po intervaloch, v prípade niektorých funkcií sa mnohokrát agent inicioval do globálneho minima neprišlo mi to zaujímavé.

Literatúra

- [1] Kochenderfer, M.;Wheeler T.: *Algorithms for optimization*. The MIT Press Cambridge, 2019, ISBN 9780262039420.
- [2] Tvrdlík, J.: *Evoluční algoritmy*. Ostravská univerzita, Přírodovědecká fakulta, 2004: s. 4-48.
- [3] Curtis, F. E.; Robinson, D. P.: Exploiting negative curvature in deterministic and stochastic optimization. *Mathematical Programming*, č. 176, 2019, ISSN 14364646.
- [4] Akinola, O. O.; Ezugwu, A. E.; Agushaka, J. O.;Zitar, R. A.; Abualigah, L.: Multiclass feature selection with metaheuristic optimization algorithms: a review. *Neural Computing and Applications*, č. 34, 2022, ISSN 14333058.
- [5] Ye, T.; Wang, W.; Wang, H.; Cui, Z.; Wang, Y.; Zhao, J.; Hu, M.: Artificial bee colony algorithm with efficient search strategy based on random neighborhood structure. *Knowledge-Based Systems*, č. 241, 2022, ISSN 0950-7051.
- [6] Peng, H.; Deng, C.; Wu, Z.: Best neighbor-guided artificial bee colony algorithm for continuous optimization problems. *Soft Computing*, č. 23, 2019, ISSN 14337479.
- [7] Nadimi-Shahraki, M. H.; Taghian, S.; Mirjalili, S.: An improved grey wolf optimizer for solving engineering problems. *Expert Systems with Applications*, č. 166, 2021, ISSN 09574174.
- [8] Antoniou, A.; Lu, W. S.: Algorithms and Engineering Applications. *Practical optimization*, Springer New York, 2007, ISBN 978-0-387-71107-2.
- [9] Mloga, M.; Smutnicki, C.: Test functions for optimization needs. *Test functions for optimization needs*, č. 101, 2005.
- [10] Jamil, M.; Yang, X. S.: A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, č. 4, 2013, ISSN 20403615.
- [11] Layeb, A.: New hard benchmark functions for global optimization. <https://arxiv.org/abs/2202.04606>, 2023, DOI 10.48550/ARXIV.2202.04606.

Zoznam príloh

A Volanie pomocných funkcií

39

A Volanie pomocných funkcií

Výpis A.1: Příklad volania funkcie s účelovými funkciami

```
1 [z,a,b] = input_func(x_i,input_f);
2 % x_i      - vektor s polohou agenta
3 % input_f  - číslo priradené funkcií
4
5 % z        - hodnota funkcie v bode
6 % a        - dolné obmedzenia funkcie
7 % b        - horné obmedzenia funkcie
```

Výpis A.2: Příklad volania funkcie s účelovými funkciami

```
1 [s] = pop_init(N_f,inp_f,sw)
2 % s      - štruktúra s
3           % P      - Poloha agenta
4           % a      - dolné obmedzenia funkcie
5           % b      - horné obmedzenia funkcie
6
7 % N_f    - Počet agentov
8 % inp_f  - Testovaná funkcia
9 % sw     - 1      -> náhodná populácia,
10 %       - inak   -> inicializácia po intervaloch
```

Výpis A.3: Příklad volání funkce pro nájdenie najlepšieho riešenia v populácii

```
1      [best_sol] = find_best_sol(N, best_sol, P, input)
2      % N          - počet agentov
3      % best_sol   - doteraz najlepšie riešenie
4      % P          - polohy agentov
5      % input      - testovaná funkcia
6
7      % best_sol   - najlepšie riešenie
```