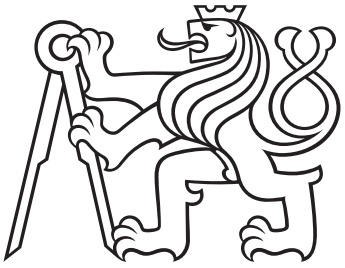
České vysoké učení technické v Praze

Fakulta jaderná
a fyzikálně inženýrská

Katedra softwarového inženýrství
Obor: Aplikace softwarového inženýrství



Algoritmus optimalizace hejnem částic: vývoj a jeho aplikace

Particle Swarm Optimization Algorithm: Development and Applications

BAKALÁŘSKÁ PRÁCE

Vypracoval: Ondřej Pánek
Vedoucí práce: Ing. Quang Van Tran, Ph.D.
Rok: 2018

České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství

Akademický rok 2017/2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	Ondřej Pánek
Studijní program:	Aplikace přírodních věd
Obor:	Aplikace softwarového inženýrství
Název práce česky:	Algoritmus optimalizace hejnem částic: vývoj a jeho aplikace
Název práce anglicky:	Particle Swarm Optimization Algorithm: Development and Applications

Pokyny pro vypracování

1. Literární rešerše vývoje algoritmu optimalizace hejnem částic
2. Implementace algoritmu v MATLABu, příp. v C++
3. Aplikace algoritmu na testovacích funkcích a na reálných příkladech

Doporučená literatura:

[1] EBERHART, R. C., KENNEDY, J. A New Optimizer Using Particle Swarm Theory. Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan, 1995, pp. 39-43.

[2] CLERC, M. Particle Swarm Optimization, 2006. ISBN: 978-1-905209-04-0

[3] PARSOPOULOS, K. E., VRAHATIS, M. N. Particle Swarm Optimization and Intelligence: Advances and Applications, 2010. ISBN: 978-1-61520-667-4

Jméno a pracoviště vedoucího práce:

KSI FJFI ČVUT v Praze

Ing. Quang Van Tran, Ph.D.

.....

vedoucí práce

Datum zadání bakalářské práce: 20.10.2017

Termín odevzdání bakalářské práce: 9.7.2018

Doba platnosti zadání je dva roky od data zadání.

.....

vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská
-1-

.....

děkan

V Praze dne 20.10.2017

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

V Praze dne .....

.....
Ondřej Pánek

Poděkování

Děkuji Ing. Quang Van Tranovi, Ph.D. za ochotné vedení mé bakalářské práce, za vstřícné konzultace a za podnětné návrhy.

Ondřej Pánek

Název práce:
Algoritmus optimalizace hejnem částic: vývoj a jeho aplikace

Autor: Ondřej Pánek

Studijní program: Aplikace přírodních věd
Obor: Aplikace softwarového inženýrství
Druh práce: Bakalářská práce

Vedoucí práce: Ing. Quang Van Tran, Ph.D.
Katedra softwarového inženýrství, Fakulta jaderná
a fyzikálně inženýrská, České vysoké učení technické v Praze

Abstrakt: Tato práce se zabývá přírodou inspirovaným stochastickým algoritmem Optimalizace hejnem částic, který hledá optimální řešení daného problému v globálním měřítku. Je zde popsáno několik jeho modifikací, které díky své robustnosti dokáží řešit širokou škálu problémů a zároveň stanovují požadavky pro nové varianty tohoto algoritmu. Hlavním přínosem této práce je implementace jednotlivých verzí v MATLABu a jejich srovnání na testovacích funkcích. Tyto funkce jsou charakteristické náročností nalezení jejich globálního extrému, a to zejména kvůli dimenzi definičního oboru a počtu lokálních extrémů, ve kterých mohou optimalizační algoritmy uvíznout. Proto jsou na závěr jednotlivé funkce rozděleny do skupin a ke každé této skupině je doporučena varianta algoritmu, která vykazuje nejlepší výsledky.
Klíčová slova: Globální optimalizace, optimalizace hejnem částic,
inteligence hejna, benchmarkové funkce, testování.

Title:
Particle Swarm Optimization Algorithm: Development and Applications

Author: Ondřej Pánek

Abstract: The thesis deals with biologically inspired stochastic Particle Swarm Optimization algorithm, which optimizes a given problem. Several of its modifications are described, which are, thanks to their complexity, able to solve many different types of problems as well as outlining demands for new versions of the algorithm. The MATLAB implementation of these particular versions and then application to benchmark functions is the main contribution of the thesis. The difficulty of optimizing these test functions is given mainly because of a domain's dimension and the amount of local extrema, where there is a tendency for optimization algorithms to stuck in. Thus, these functions are sorted into groups and for each of them the best-performing variant of the algorithm is recommended.
Key words: Global Optimization, Particle Swarm Optimization,
Swarm Intelligence, Benchmark Functions, Testing.

Obsah

Úvod

Touha po pochopení chování zvířecích druhů dala vzniknout nejednomu lidskému dílu. Například při prvních pokusech o let se člověk nechal inspirovat ptactvem a jinak tomu nebylo ani v případě hlavního předmětu této práce - algoritmu Optimalizace hejnem částic. Strategie letu ptáků a jejich vztahy v hejnu byly formovány v důsledku evoluce, a tedy i nutnosti přežít. Zákonitosti, které takové chování popisují, musí tedy tvořit nějaké optimální nastavení systému hejna. Právě tyto zákony začali vědci studovat a popisovat. Zjistili, že synchronní let hejna, které není vedeno žádným vůdcem, je skutečně spojením několika zásad, kterými se každý jedinec v hejnu řídí, aby byla maximalizována šance na přežití. Dá se říci, že maximalizace šance na přežití se skládá jednak ze snahy vyvarovat se ztrátám v důsledku útoku predátora a ze schopností obstarat si co největší množství potravy. Jinak řečeno, hejno jako celek svým pohybem minimalizuje riziko při ohrožení a stejně tak maximalizuje schopnost zabezpečit se dostatečným množstvím potravy. Každý jedinec se tak svým chováním snaží optimalizovat let celého hejna jednoduše proto, že v důsledku evoluce se vyplatí nechat soutěživost stranou a místo toho spolupracovat.

Výsledky prvních počítačových simulací letu hejna ptáků umožnily popsat algoritmus, který je inspirován zákony, které fungují uvnitř takových hejn. Cílem této práce je popsat, implementovat a následně testovat z určitých důvodů nejdůležitější varianty algoritmu Optimalizace hejnem částic.

Popis chování hejna, nebo také inteligence hejna, signalizuje, že tento algoritmus najde uplatnění na poli globální optimalizace. Náplní algoritmů působících v tomto odvětví je hledání nějakého globální extrému, ať už maxima či minima, daného problému. Takový problém bude reprezentován tzv. *účelovými* funkcemi, u kterých nás bude zajímat jejich nejmenší hodnota v globálním měřítku. Charakterizace problému globální optimalizace bude jednou z náplní první kapitoly.

Ve druhé části první kapitoly bude algoritmus Optimalizace hejnem částic (nebo také zkráceně PSO z anglického *Particle Swarm Optimization*) zařazen mezi skupinu stochastických algoritmů, které jsou charakteristické tím, že při svém výpočtu používají prvek náhody. To koresponduje stále s chováním hejna ptáků, neboť i tam působí takové prvky, kterými může být třeba šum v komunikaci mezi jedinci, změna síly větru, útok predátora atd.

Předmětem třetí kapitoly bude charakterizace nejvýznamnějších variant PSO. Nejdříve bude popsána ta zcela první z roku 1995, která slouží jako základní stavební kámen pro všechny ostatní. Dalšími budou tzv. *standardní* verze, které jsou specifické zejména tím, že slouží jako „odrazový můstek“ pro nové varianty PSO.

To znamená, že popíše-li někdo novou verzi tohoto algoritmu a chce-li o ní prohlásit, že je úspěšná, musí vykazovat lepší výsledky než některá z těchto standardních verzí. Jako čtvrté bude popsáno tzv. *bezparametrické* PSO, které, na rozdíl od všech jejích předchůdců, ke své funkčnosti potřebuje méně parametrů. Hlavní myšlenka, která stála u zrodu této verze byla taková, že si algoritmus upravuje své parametry podle potřeby v průběhu výpočtu. Jak vyjde najevo, pro správnou funkčnost algoritmu bude potřeba získávat body rovnoměrně rozdělené uvnitř obecné koule. Proto v poslední části třetí kapitoly budou popsány způsoby, jak takové body získávat.

Poslední část této práce bude zaměřena na implementaci a na testování zmíněných variant PSO. Nejdříve se na základě výsledků prvních testů vyberou nejvhodnější realizace podmínek pro jednotlivé varianty a poté bude probíhat testování na modelových funkcích. Tyto funkce budou rozděleny do charakteristických tříd. Na závěr budou výsledky z testů interpretovány a ke každé třídě funkcí bude doporučena varianta PSO, která je nejlépe minimalizovala.

Kapitola 1

Globální optimalizace

Optimalizace je rozsáhlým polem působnosti aplikované matematiky a obecně se zabývá hledáním nejlepších možných hodnot určité funkce, či vektoru funkcí napříč množinou, na níž jsou tyto funkce definované a na níž chceme tyto hodnoty hledat. K těmto funkcím[1], pro které hledáme množinu optimálních řešení, se zpravidla váže množina určitých omezení. Pokud hledáme optimální řešení na takové množině, která je zároveň definičním oborem účelové funkce, jedná se o globální optimalizaci. Cílem globální optimalizace je tedy nalezení vektoru proměnných, který minimalizuje, resp. maximalizuje, hodnotu účelové funkce vzhledem ke všem omezením problému.

1.1 Obecná definice problému

Matematická formulace problému globální optimalizace může vypadat následovně:

Nechť máme účelovou funkci, resp. vektor účelových funkcí $f_i(\vec{x}) : P \to \mathbb{R}$, kde $P \subset \mathbb{R}^n,\ n \in \mathbb{N},\ P = D_{f_i}, \forall i \in \hat{K},\ K \in \mathbb{N}$.

Nechť dále máme množinu funkcí $u_j(\vec{x}) : P \to \mathbb{R},\ v_k(\vec{x}) : P \to \mathbb{R}$, kde $j \in \hat{L}, k \in \hat{M},\ L, M \in \mathbb{N}$.

Potom definujeme problém globální optimalizace jako:

$$\min_{\vec{x} \in P} f_i(\vec{x}),\ \forall i \in \hat{K}, \tag{1.1}$$

$$\text{vzhledem k } u_j(\vec{x}) = 0,\ \forall j \in \underline{\hat{L}}, \tag{1.2}$$

$$v_k(\vec{x}) \leq 0,\ \forall k \in \underline{\hat{M}}. \tag{1.3}$$

Množina $A \subset P$ se nazývá množinou přípustných řešení, pokud platí, že $\forall \vec{x} \in A$ jsou splněny podmínky (1.2) a zároveň (1.3). Každý vektor z množiny A se tedy nazývá přípustným řešením.

[1] Funkce, které jsou předmětem nějakého optimalizačního problému, se nazývají účelové.

Vektor $\vec{x}^* \in A$ nazveme optimálním řešením problému (1.1), pokud je přípustným řešením a $\forall \vec{x} \in A$ platí:

$$f_i(\vec{x}^*) \leq f_i(\vec{x}),\ \forall i \in \hat{K}.$$

Pokud $K = 1$, jedná se o problém s pouze jednou účelovou funkcí. Je vhodné podotknout, že pokud by bylo předmětem zájmu naopak hledání maxima nějaké funkce $g(\vec{x})$, jednoduchou úpravou se maximalizační úloha převede na minimalizační takto:

$$\max_{\forall \vec{x}} g(\vec{x}) = \min_{\forall \vec{x}} -g(\vec{x}).$$

Podobně se dají převést omezení typu „$\geq$“ na „$\leq$“:

$$u(\vec{x}) \geq 0 \ \Leftrightarrow\ -u(\vec{x}) \leq 0.$$

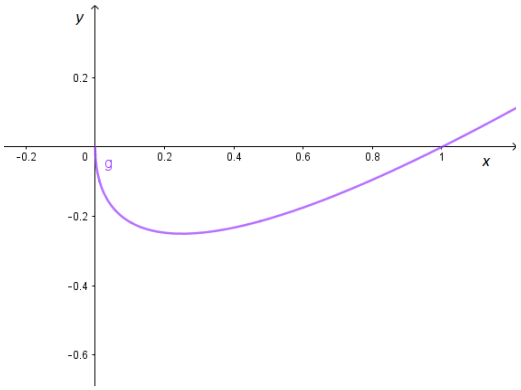
1.2 Rozdělení a typy problémů

Jak již bylo řečeno, problém globální optimalizace může obsahovat pouze jednu účelovou funkci. Stejně tak ale může být cílem optimalizace více funkcí najednou. První charakterizací optimalizačních problémů může být dělení vzhledem k počtu účelových funkcí, které se v problému vyskytují, na úlohy s jednou účelovou funkcí a úlohy s více účelovými funkcemi. Dále je možné klasifikovat optimalizaci podle typu omezení, která jsou kladena na proměnné. Pokud $L = M = 0$, jedná se o úlohu optimalizace bez omezení. Problém může také obsahovat pouze omezení typu „ $=$ “, resp. pouze omezení typu „$\leq$“. Některé formulace problémů optimalizace mohou obsahovat omezení typu „ $=$ “ implicitně, neboť lze takové omezení ekvivalentně převést pouze na soustavu omezení typu „$\leq$“, a to následujícím způsobem:

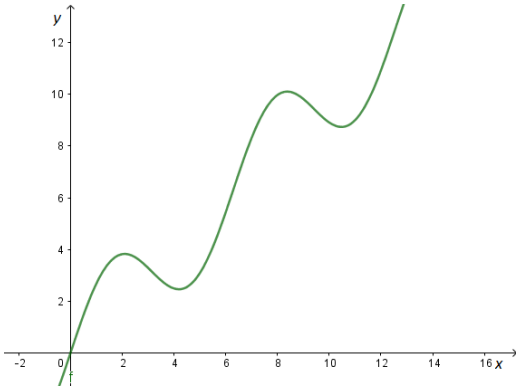
$$u(\vec{x}) = 0 \Leftrightarrow u(\vec{x}) \leq 0 \wedge u(\vec{x}) \geq 0 \Leftrightarrow u(\vec{x}) \leq 0 \wedge -u(\vec{x}) \leq 0.$$

Další dělení bere v potaz počet extrémů účelové funkce, a dělí je na funkce s jediným extrémem (unimodální) a na funkce s několika lokálními extrémy (multimodální). Má-li funkce pouze jedno lokální optimum, potom je toto optimum samozřejmě i globálním optimem(viz Obrázek 1.1).

Optimalizaci lze také pojmenovat podle hodnot, kterých mohou proměnné nabývat. Mohou-li proměnné nabývat pouze diskrétních hodnot, nazýváme optimalizaci diskrétní. Pokud $\vec{x} \in \mathbb{Z}^n$, mluvíme o celočíselném programování, na druhou stranu pokud $\exists i, j \in \hat{n}$ takové, že $x_i \in \mathbb{Z}$ a zároveň x_j může nabývat hodnot z nějakého intervalu, mluvíme o smíšené úloze celočíselného programování. Další charakteristikou problému je typ účelové funkce. V této souvislosti označujeme optimalizaci jako lineární či nelineární. Poslední pohled na členění optimalizace bude rozlišení problémů podle určitosti či neurčitosti. U deterministických úloh jsou účelové funkce a podmínky jednoznačně určeny, tzn. přesně víme, jaký bude výstup např. účelové funkce při konkrétním vstupu. Toto nemusí platit vždy a do optimalizačního problému může vstupovat neurčitost. Ta se projevuje např. při měření fyzikálních veličin, kde s měřením nastává i určitá chyba měření, jejíž hodnota se pohybuje

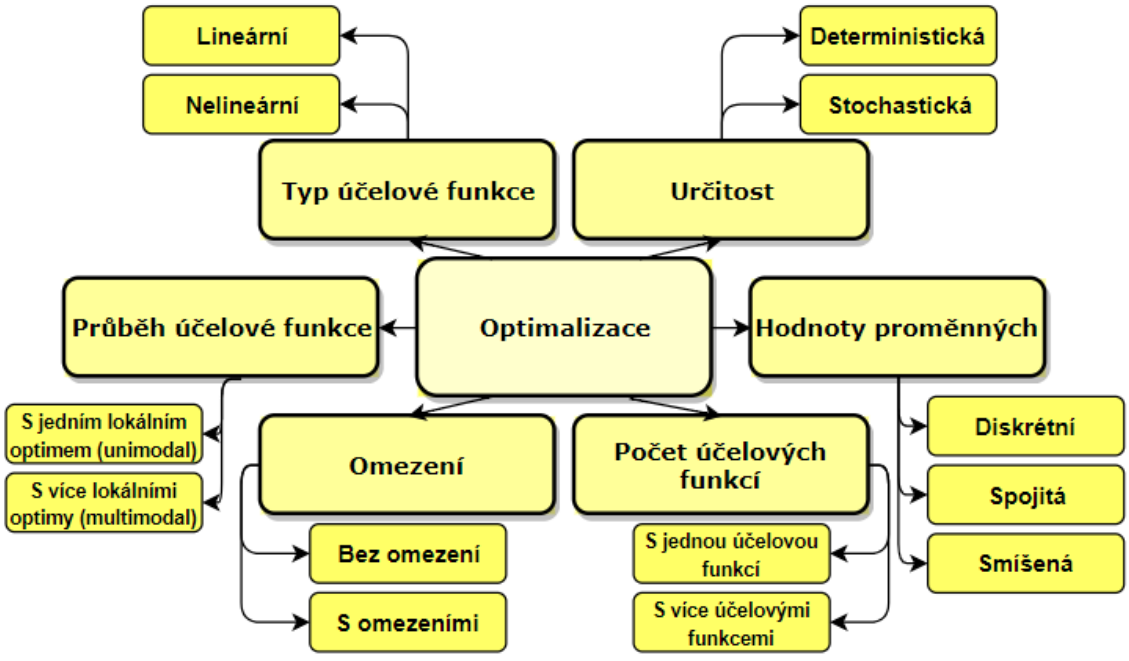


(a) Příklad účelové funkce s jedním optimem. Předpis: $g(x) = -\sqrt{x} + x$.



(b) Příklad účelové funkce s více lokálními optimy. Předpis: $f(x) = x + 2\sin(x)$.

Obrázek 1.1: Příklady funkcí, které spadají do dvou různých skupin vzhledem k počtu extrémů.



Obrázek 1.2: Grafické znázornění členění optimalizace podle jednotlivých charakteristik problému.

v nějakém intervalu nebo rozmezí. Takové problémy se označují jako stochastické. Přehled typů optimalizace je graficky znázorněn na Obrázku 1.2.

Algoritmy optimalizace jsou obecně členěny na algoritmy deterministické, což znamená, že při stejném vstupu budou generovat stále stejný výstup, a na stochastické [18], ve kterých hraje hlavní roli prvek náhody, a tedy se stejným vstupem se výstup může lišit. Mezi klasické deterministické algoritmy patří např. simplexová metoda, gradientní metoda, Newton-Raphsonova metoda a další. Algoritmus optimalizace hejnem částic se řadí mezi metaheuristiky, což je skupina algoritmů spadající mezi stochastické. V následující části bude stručně popsáno další členění této oblasti. Budou tedy zmíněny charakteristiky heuristických, resp. metaheuristických algoritmů.

1.3 Stochastické algoritmy

Stochastické algoritmy se liší od těch deterministických v několika směrech. První rozdíl je v tom, že ve svém výpočtu používají prvek náhody. To znamená, že pokud budeme stochastický algoritmus testovat několikrát na problému se stále stejným zadáním, je možné, že se výsledky jednotlivých testů budou lišit. S tím souvisí další odlišnost od deterministických algoritmů, a to že stochastický algoritmus nezaručuje nalezení globálního optimálního řešení problému, ale může skončit pouze v nějakém lokálním optimu. Avšak toto chování může být někdy žádoucí, neboť může být výhodné nalézt alespoň nějaké optimální řešení v přijatelném čase. Na druhou stranu čas potřebný pro nalezení globálního optima deterministickým algoritmem může neúnosně narůstat s rostoucím počtem dimenzí problému. Stochastické algoritmy lze dělit na dvě větve: heuristiky a metaheuristiky. Jejich hlavní rysy budou popsány v následujícím textu.

1.4 Heuristiky a metaheuristiky

Výraz *heuristika* pochází z řečtiny a ve volném překladu znamená „objevovat metodou pokus-omyl“ nebo „hledat“. Heuristiku používá každý člověk i v běžném životě. Každý někdy řešil nějaký lehčí či obtížnější problém metodou pokus omyl. Další heuristikou, která je frekventovaně využívána, je postup, kdy si nakreslíme problém, který řešíme, pro lepší pochopení celé situace. Často je při studiu matematiky nutné umět si představit znění nějaké obecné věty na konkrétním příkladě, i toto je považováno za heuristický přístup k řešení problému. Takových heuristik je celá řada a lidé je často sami používají, aniž by o tom věděli [20].

Co se týče významu předpony *meta*, tak v tématu optimalizačním algoritmů to znamená „vyšší úroveň“ nebo „vylepšení“ jednodušší heuristiky. To se projevuje tím, že všechny metaheuristické algoritmy používají určitou rovnováhu mezi náhodným hledáním a lokálním prohledáváním. Lokální prohledávání zajišťuje algoritmu „kontrolu“ míst, kde to vypadá, že by mohlo nastávat nějaké optimum, na druhou stranu náhodný prvek by měl zabraňovat situacím, kdy algoritmus skončí v lokálním

optimu, ale posouvá celý výpočet směrem k optimu globálnímu. Správnou kombinací těchto dvou přístupů jsou zkonstruovány všechny úspěšné metaheuristické algoritmy.

Řada metaheuristických algoritmů je inspirována přírodními procesy, neboť v rámci evoluce byly všechny živočišné druhy vystavovány různým překážkám, které musely pro přežití nutně překonat. Protože se takové procesy utvářely miliony let, bylo dobré nechat se jimi poučit, neboť právě ono překonávání vytvořilo dokonalá řešení většiny problémů. Přírodou inspirovanými algoritmy jsou např. genetické algoritmy, optimalizace mravenčí kolonií, optimalizace včelím rojem, optimalizace hejnem světlušek a samozřejmě optimalizace hejnem částic [13]. Naposledy zmíněný algoritmus bude hlavním předmětem celého zbývajícího textu.

Kapitola 2

Optimalizace hejnem částic

Algoritmus optimalizace hejnem částic (anglicky *Particle Swarm Optimization* či zkráceně PSO) je jedním z metaheuristických algoritmů, který byl inspirován přírodními systémy, konkrétně hejny ptáků [9] či ryb. PSO tedy spadá do skupiny algoritmů souhrnně nazývaných *inteligence hejna*. Poprvé byl popsán v roce 1995 Eberhartem a Kennedym [10]. Tento algoritmus je založen na prohledávání D-dimenzionálního prostoru hejny částic, které hledají globální optimální řešení určitého problému v tomto prostoru. Ke každé takové částici se přistupuje jako k nehmotnému D-rozměrnému bodu, který má svoji polohu a rychlost, které se v průběhu výpočtu dynamicky mění. Rychlost částice je přepočítávána na základě znalosti její doposud nejlepší dosažené pozice nebo na základě zkušenosti ostatních členů hejna[1]. Algoritmus PSO dokáže operovat na nejrůznějších typech problémů. Těmi jsou například: Funkce více proměnných, a to jak spojité, tak nespojité, problémy s omezujícími podmínkami, multimodální problémy a další.

Protože se algoritmus optimalizace hejnem částic po svém představení v roce 1995 postupně stával oblíbenějším a začal nacházet své využití v mnoha odvětvích [19], vznikalo stále více jeho variant a dnes jich je už více než sto [23]. Některé vybrané varianty PSO budou v nadcházejících kapitolách popsány. První verzí algoritmu, která bude popsána, bude standardní verze PSO z roku 2006[2,3], neboť tvoří základ pro všechny další modifikace.

2.1 První verze algoritmu

V této části bude popsán vznik a raný vývoj algoritmu optimalizace hejnem částic. Bude zde představena první varianta algoritmu a příčina, díky které tato verze v roce 1995 vznikla. Dále budou popsány další tři modifikace, a to sice dvě standardní

[1] Osobní nejlepší pozice se v literatuře nazývá *fitness* [7].

[2] SPSO - Standard Particle Swarm Optimization. Další standardní verzí je SPSO 2011, která bude taktéž popsána.

[3] Standardní verze algoritmu optimalizace hejnem částic tvoří jakýsi odrazový můstek pro další vylepšování algoritmu. Platí, že pokud je navrhnut nový model PSO, měl by být lepší, tzn. mít o dost lepší výsledky na množině testovacích funkcí, než-li nějaký ze standardních modelů SPSO.

z roku 2006 a 2011, které jsou důležité tím, že nastavují určitou laťku pro vývoj nových verzí algoritmu, a jedna bezparametrická, která demonstruje jiný přístup vzhledem k tomu, že nepotřebuje pro výpočet tolik parametrů jako verze ostatní.

2.1.1 Motivace vzniku a vývoj

Odjakživa se člověk nechává inspirovat přírodními systémy a jejich dokonalou funkčností, a poté s pomocí techniky tyto přírodní jevy napodobuje. Algoritmus optimalizace hejnem částic také vznikl na základě toho, že se člověk nechal inspirovat přírodními procesy. Konkrétně pánové Reynolds [21], Heppner a Grenander [9] simulovali chování ptačích hejn, u kterých je fascinoval naprosto synchronní pohyb celého hejna. Hejno je s to se přeskupovat, rozptylovat se v případě útoku predátora a rychle měnit směr, i bez přítomnosti nějakého vůdce, který by diktoval takový pohyb. Další zajímavým faktem je, že i přes velké množství jedinců v hejně nedochází ke srážkám jednotlivců. První modely Reynoldse a Heppnera byly založeny na tom, že se jedinci snaží udržovat určitou optimální vzdálenost od ostatních členů hejna. Dále bylo zjištěno, že každý jedinec má tendenci směřovat do středu hejna a že se snaží udržovat podobnou rychlost jako jeho sousedé. Základní myšlenka pro vznik algoritmu PSO vzešla z tvrzení Wilsona [10], socio-biologa, který při popisování shlukování ryb, řekl, že jednotlivci dokáží těžit ze zkušenosti a objevů všech ostatních členů hejna a že tato výhoda při hledání potravy převažuje i soutěživost mezi jednotlivci. Z toho lze usuzovat, že shlukování jedinců do hejn vzniklo jako evoluční pokrok.

Algoritmus se poprvé objevil pouze jako simulace chování hejna. Členové hejna (ptáci) se nyní budou vhodněji označovat jako agenti.

První simulace byla založena na srovnávání rychlostí agentů. Ti byli rozmístěni náhodně do určitého prostoru a poté se v každé iteraci přiřadil agentovi vektor rychlosti jeho souseda. Tento jednoduchý způsob začal určitým způsobem připomínat chování hejna, avšak po chvíli se pohyb ustálil, tzn. všichni agenti letěli stejnou rychlostí. Kvůli tomu se do simulace přidal prvek ztřeštěnosti nazvaný „craziness“, což byla vlastně náhodná změna rychlosti v každém kroku. To vizuálně zlepšilo chování simulovaného hejna. Dalším modelem byl Heppnerův Cornfield Vector (vektor kukuřičného pole). Zde se vytvořilo hejno agentů, kteří poletovali kolem jednoho bodu, dokud na něm nepřistáli. Tento přístup měl zásadní nedostatek v tom, že agenti věděli přesně, kde se nachází tento bod. Heppner tedy přidal do simulace vektor kukuřičného pole se souřadnicemi X a Y a každý agent vyhodnocoval svojí pozici podle [10]:

$$Eval = \sqrt{(presentx - 100)^2} + \sqrt{(presenty - 100)^2}$$

To znamená, že kukuřičné pole bylo na souřadnicích $(100, 100)$ a funkce $Eval$ byla vlastně síťová metrika [12], neboť agenti i kukuřičné pole byli reprezentováni pixely. Každý agent si pamatoval svoji nejlepší hodnotu ($pbest[\,]$ [4]) a pozici ($pbestx[\,]$, $pbesty[\,]$), a také měl znalost o nejlepší hodnotě $pbest[gbest]$ a o pozici

[4] Hranaté závorky značí, že se jedná o pole o velikosti S, kde S je velikost hejna.

odpovídající této hodnotě $(pbestx[gbest], pbesty[gbest])$ v rámci celého hejna. Jeho rychlost pak byla upravována jednoduše na základě toho, jestli se agent nachází vlevo či vpravo, za nebo před kukuřičným polem. Byly stanoveny maximální změny rychlosti, které určovaly, jak rychle se může agent přibližovat ke své nejlepší pozici, resp. k nejlepší pozici v rámci celého hejna. Toto vyhodnocování vypadalo následovně [10]:

$$
\begin{gathered}
if\ presentx[i] > pbestx[gbest]\ then\ vx[i] = vx[i] - rand() * g\_increment \\
if\ presentx[i] < pbestx[gbest]\ then\ vx[i] = vx[i] + rand() * g\_increment \\
if\ presenty[i] > pbesty[gbest]\ then\ vy[i] = vy[i] - rand() * g\_increment \\
if\ presenty[i] < pbesty[gbest]\ then\ vy[i] = vy[i] + rand() * g\_increment \\
\forall i \in \hat{S} = \{1, 2, ..., S\}
\end{gathered}
$$

Tedy $g\_increment$ je maximální možná změna rychlosti směrem ke globálnímu optimu, navíc je upravena $rand()$ faktorem, který představuje rovnoměrně rozdělené náhodné číslo z intervalu $(0, 1)$. Stejným způsobem se upravuje rychlost vzhledem k nejlepší osobní pozici agenta, pouze zde vystupuje $p\_increment$. Pomocí těchto pravidel lze již celkem věrohodně simulovat shlukování ptáků.

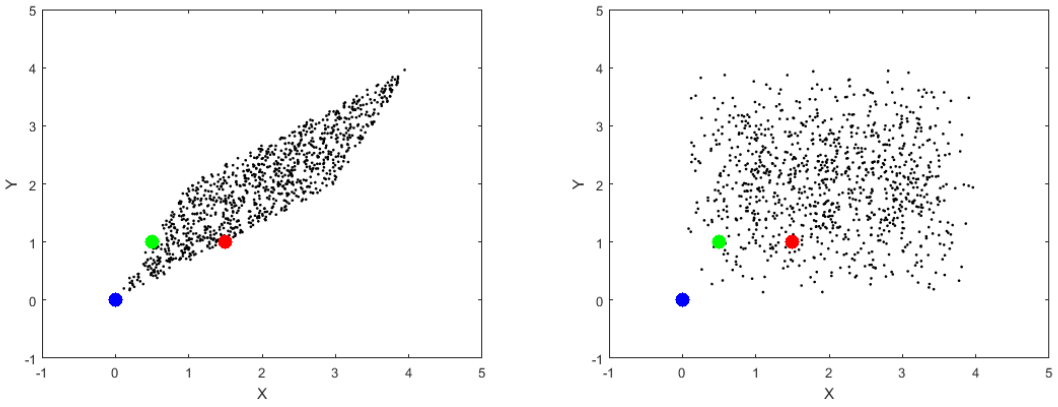
V tuto chvíli se ukázalo, že tento model je schopen optimalizovat lineární 2D funkci. Přirozeným vývojem proto bylo zjednodušit model co nejvíce a současně zachovat jeho účinnost. Díky těmto předchozím modelům bylo ukázáno, že prvek ztřeštěnosti a srovnávání rychlostí s nejbližším sousedem nejsou žádoucí, resp. optimalizace je bez nich rychlejší. Tyto změny vedly k tomu, že se hejno vizuálně začalo chovat spíše jako hejno hmyzu než ptáků. Naopak proměnné reprezentující nejlepší dosaženou osobní i globální pozici se pro simulaci zdály býti nezbytně důležité, a tedy zůstaly zachovány. Jejich reprezentace v modelu je následující: $pbestx[\,], pbesty[\,]$ představují zkušenost a paměť jedince, a tedy díky těmto proměnným vzniká u jedince tendence navracet se na místo, kde se „cítil“ dobře. Vedle toho index $gbest$ poukazuje na určitou skupinovou moudrost nebo znalost pozice, kde se hejnu „dařilo“. Vliv na chování celého hejna měla také samozřejmě velikost jednotlivých přírůstků rychlosti $p\_increment$ (osobní přírůstek) a $g\_increment$ (globální přírůstek). Modely, kde byla velikost osobního přírůstku příliš velká vzhledem ke globálnímu, měly tendenci zůstávat spíše v lokálních minimech, naopak při příliš velké hodnotě globálního přírůstku byl pohyb hejna nepřirozeně rychlý a přímočarý vzhledem ke globálnímu optimu. Z toho bylo usouzeno, že není důvod k tomu, aby hodnoty jednotlivých přírůstků byly různé.

Dalším krokem bylo zobecnění algoritmu, aby pracoval na prostoru obecně o D dimenzích. Nové značení je tedy následující[5,6,7]:

[5] Nyní závorky $[\,][\,]$ představují matici o rozměru $D \times S$, kde S je velikost hejna a D dimenze prohledávaného prostoru.

[6] Písmeno x nyní tedy neznačí o jakou souřadnici se jedná, ale značí, že se jedná o souřadnice obecně.

[7] $pbest[\,]$ je vektor o rozměru $S \times 1$.



(a) *rand*() je číslo. (b) *rand*() je vektor, resp. matice.

Obrázek 2.1: Na obrázcích je znázorněno 1000 možných poloh (černé tečky) jednoho agenta (modrý kruh). Zelený, resp. červený kruh představují osobní, resp. globální optimum tohoto agenta.

$presentx[\,][\,]$	současná pozice hejna
$pbest[\,]$	vektor nejlepších hodnot
$pbestx[\,][\,]$	matice pozic odpovídajících nejlepším hodnotám
$gbest$	index nejlepší pozice v rámci celého hejna
$pbestx[gbest][\,]$	matice rozměru $D \times S$, která má všechny sloupce stejné každý sloupec matice je vektor souřadnic globálního optima
$vx[\,][\,]$	matice rychlostí

Dále se také upustilo od myšlenky srovnávat souřadnice agenta se souřadnicemi nejlepších pozic a vznikla nová pravidla pro určování rychlosti agentů. Rychlost agentů se začala upravovat v závislosti na vzdálenosti od $pbestx[\,][\,]$ a $pbestx[gbest][\,]$. Nové ohodnocení vypadalo takto[8]:

$$\begin{aligned} vx[\,][\,] = vx[\,][\,] &+ p\_increment * rand() \,.* (pbestx[\,][\,] - presentx[\,][\,]) \\ &+ g\_increment * rand() \,.* (pbestx[gbest][\,] - presentx[\,][\,]), \end{aligned}$$

kde *rand*() je matice $D \times S$ náhodných čísel z rovnoměrného rozdělení z intervalu $(0, 1)$. To znamená, že každá složka matice $pbestx[\,][\,] - presentx[\,][\,]$, resp. $pbestx[gbest][\,] - presentx[\,][\,]$, je vynásobena různým náhodným číslem. Častou chybou bývá domněnka, že činitel *rand*() je pouze jedno náhodné číslo [17], jímž se vynásobí celý vektor souřadnic jednoho agenta, resp. matice celého hejna. Výraz *rand*() v modifikaci rychlosti musí představovat vektor, resp. matici různých náhodných čísel z intervalu $(0, 1)$. Rozdíl mezi těmito variantami je interpretován na Obrázku 2.1.

Na Obrázku 2.1a je vidět, že pokud se pro úpravu rychlosti použije jediné číslo pro všechny složky vektoru polohy agenta, prostor, kterým se agent může vydat, se

[8]Máme-li matice A, B o stejných rozměrech $D \times S$, potom operací $A \,.* B$ dostaneme matici C, pro jejíž prvky C_{ij} platí, že $C_{ij} = A_{ij} * B_{ij}$, $\forall i \in \hat{S}$, $\forall j \in \hat{D}$.

omezí na jistý kosodélník. Takový prostor ale jistě není žádoucí. Správné chování je znázorněno na Obrázku 2.1b, kde $rand()$ je již vektor obsahující různá náhodná čísla. Prostor, který je brán v úvahu, se tak značně rozšíří. Rozdíl mezi těmito nastaveními je z obrázků očividný a v implementaci tato chyba nesmí nastat.

Protože nebyl důvod rozlišovat osobní a globální přírůstek, přibyla do tohoto modelu další změna, a to sice stanovení společné hodnoty těchto přírůstků. Tato hodnota byla stanovena na $p\_increment = g\_increment = 2$, což značí, že agent přeletí optimální pozici v polovině případů:

$$\begin{aligned} vx[\,][\,] = vx[\,][\,] &+ 2 * rand() \,.* (pbestx[\,][\,] - presentx[\,][\,]) \\ &+ 2 * rand() \,.* (pbestx[gbest][\,] - presentx[\,][\,]), \end{aligned} \tag{2.1}$$

Pořád se v tomto nastavení ale objevoval jeden problém, tzv. exploze. Exploze (divergence) znamená, že rychlost agentů v průběhu výpočtu neustále narůstá a hejno se pořád vzdaluje od globálního optima. Toto bylo napraveno stanovením maximální rychlosti ($VMAX$). Díky této konstantě se ošetřila velikost rychlosti tak, aby nikdy nepřekročila určitou přípustnou mez.

Tento model podstupoval další modifikace, kterými byly [10]:

- Prostřední bod (Midway Point) - v této metodě byli jedinci hnáni směrem do středu úsečky, jejíž krajní body byly právě osobní a globální optimum.
- Průzkumníci a osadníci (Explorers and Settlers) - zde se populace jedinců dělila na skupinu průzkumníků a skupinu osadníků. Každá z těchto skupin využívala jinou variantu výpočtu rychlosti.
- Metoda bez ohledu na předchozí rychlost - ze vzorce (2.1) zmizela z pravé strany rovnice znalost rychlosti z předešlé iterace $vx[\,][\,]$.

Ovšem žádný z těchto modelů se z určitých důvodů[9] nejevil tak efektivní jako (2.1).

2.1.2 Principy inteligence hejna

Nyní lze říci, že výše popsaný model tvořil první verzi algoritmu optimalizace hejnem částic[10]. Tento algoritmus se řadí do skupiny algoritmů, která se označuje souhrnným názvem „Inteligence hejna“, neboť tento model splňuje pět základních principů inteligence hejna popsané Millonasem v [14]. Těmi jsou:

[9] U prvního z nich to byla konvergence do středu této úsečky bez ohledu na to, bylo-li tam globální optimum. U dalších dvou to byla vyšší doba potřebná pro nalezení optima.

[10] Algoritmus se zjevně mohl jmenovat optimalizace hejnem ptáků nebo spíše hmyzu, nabízí se také název shlukem bodů, ale protože jednotliví agenti disponují určitou rychlostí a zrychlením, hejno částic se nejspíše autorům zdálo nejadekvátnější.

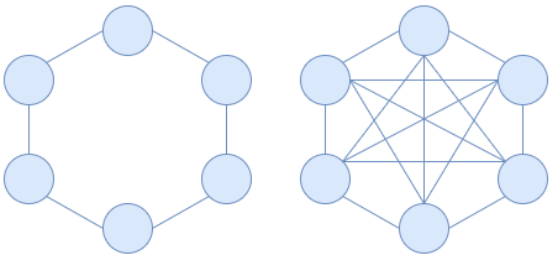
1. Princip blízkosti
   Tento princip říká, že skupina (hejno) by mě měla být schopná provádět elementární rozhodování nebo výpočty v závislosti na čase a prostoru. Tímto rozhodováním jsou myšleny okamžité reakce (pohyby) skupiny na podněty související se změnou času či prostředí. V přírodě takovým chováním může být třeba hledání potravy.
   V algoritmu probíhají iterace, při kterých se skupina pohybuje napříč n-rozměrným prostorem.

2. Princip kvality
   Skupina by měla být s to reagovat na určité faktory kvality. Takovými ukazateli mohou být v přírodě například kvalita potravy či kvalita bezpečnosti prostředí. V případě algoritmu PSO se jedná o body $pbestx[\,][\,]$ a $pbestx[gbest][\,]$. Tyto body určují pomyslnou úroveň kvality prostředí.

3. Princip rozdílné reakce
   Zde se po skupině požaduje, aby reakce od všech jedinců na konkrétní změnu prostředí nebyla zcela stejná. Hejno by se tedy nemělo pohybovat po nepřiměřeně úzkých cestičkách, neboť větší počet jedinců, dobře rozmístěných, umožňuje prohledávat větší prostor. V reálném případě to souvisí s hledáním potravy nebo obranou před nepřáteli.
   U tohoto modelu je tento princip také splněn, protože agenti reagují nejen na společný podnět $pbestx[gbest][\,]$, ale také na svůj osobní $pbestx[\,][\,]$, který může být pro každého jedince různý. Nejednotvárnost odpovědi na změnu prostředí je zajištěna také stochastickým faktorem při aktualizaci rychlosti.

4. Princip stability
   Princip stability poukazuje na to, že skupina by neměla měnit své chování s každou změnou prostředí. Takové přehnané reakce by mohly být pro populaci nevýhodné.
   I toto je ale zajištěno díky tomu, že hejno jako takové mění své chování, mění-li se globální, dosud nalezené, optimum. Zároveň platí, že pokud se tento faktor nemění, skupina nemá tendenci uchylovat se k jiným faktorům kvality. Pouze se mohou měnit sklony jedinců jako důsledek změn jejich osobních optim.

5. Princip adaptace
   Poslední princip mírně protiřečí tomu předchozímu, neboť říká, že skupina by měla být s to měnit svoje chování, pokud se objeví změna, která by byla dostatečně výhodná. Nicméně skloubí-li se čtvrtý a pátý princip dohromady, lze usuzovat, že nejlepší odezva hejna na změnu prostředí se nachází někde na půli cesty mezi zcela uspořádanou odpovědí a totálním chaosem. Jinak řečeno, dostatek náhody umožní produkovat rozdílnou reakci, kdežto příliš mnoho náhody rozbije jakékoli skupinové chování.
   Tento pátý princip je v algoritmu PSO splněn díky tomu, že hejno jako celek opravdu mění své chování při každé změně $pbestx[gbest][\,]$.



Obrázek 2.2: Vlevo je tzv. *prstencová* topologie, tedy každá částice má informace pouze o dvou dalších částicích. Vpravo je znázorněna topologie, kde každá částice má informace o všech ostatních členech hejna.

2.2 Standardní verze

Každá částice[11] se dá považovat za objekt, který je tvořen:

- Momentální pozicí v prohledávaném prostoru.
- Hodnotou účelové funkce na této pozici.
- Rychlostí.
- Znalostí poslední nejlepší dosažené pozice touto částicí.
- Znalostí poslední nejlepší dosažené pozice jejím okolím.

Dohromady částice tvoří tzv. *hejno*, které prohledává určený prostor. Uvnitř hejna existují vztahy mezi částicemi. Aby mohl algoritmus správně fungovat, správně napodobit chování přírody, musí být mezi jednotlivými agenty definované vztahy, na jejichž základě si mohou jednotlivci vyměňovat informace o svých doposud nejlepších pozicích a hodnotách na těchto pozicích. V rámci hejna existují tzv. *sousedství*, což jsou podmnožiny hejna, kde pro každou částici uvnitř jednoho sousedství platí, že zná fitness všech ostatních částic z tohoto sousedství. Můžeme říci, že uvnitř hejna existují *topologie*. Dvě nejčastější topologie jsou znázorněny na Obrázku 2.2 [11].

Prstencová topologie lze vyjádřit následovně: Sousedství S_i i-té částice, pokud je S velikost hejna zapíšeme jako:

$$S_i = \{(i-1)\mathrm{mod}(S), i, (i+1)\mathrm{mod}(S)\}. \tag{2.2}$$

Algoritmus lze rozdělit do dvou částí: inicializace a iterace. Nyní budou tyto dvě části podrobněji popsány v rámci standardu z roku 2006.

2.2.1 SPSO 2006

Jako první bude uveden jednoduchý pseudokód SPSO 2006, který je ovšem základním stavebním kamenem i pro všechny další verze algoritmu (viz Algoritmus 1).

[11] Částice se také označují jako *agenti*.

Algoritmus 1 SPSO 2006

Inicializace:
for Každé částici **do**
  Přiřaď náhodnou pozici uvnitř prohledávaného prostoru.
  Spočítej fitness.
  Poslední nejlepší pozici přiřaď tuto pozici.
  Přiřaď náhodnou rychlost.
end for
Iterace:
while není splněno kritérium pro ukončení výpočtu **do**
  for všechny částice **do**
    Vypočítej novou rychlost.
    Vypočítej novou polohu na základě této nové rychlosti.
    if částice vylétla ven z prohledávaného prostoru **then**
      Proveď vhodné ošetření.
    end if
    if nová fitness je lepší než poslední nejlepší hodnota **then**
      Ulož novou fitness.
      Ulož pozici odpovídající nové fitness.
      if nová fitness je nejlepší z okolí **then**
        Ulož tuto pozici jako fitness celého sousedství částice.
      end if
    end if
  end for
end while

Nyní se přesuneme k přesné definici všech dílčích kroků, které byly načrtnuty v Algoritmus 1.
Jako první zadefinujeme všechny potřebné výrazy:
D Dimenze prostoru, který je prohledáván hejnem.
E Prohledávaný prostor, který je definován následovně:

$$E = \bigtimes_{d=1}^{D} \langle \min_i, \max_i \rangle, \tag{2.3}$$

kde $\min_i$, resp. $\max_i$ značí dolní, resp. horní mez pro i-tou souřadnici částice. Je očividné, že prohledávaný prostor je vlastně hyperkvádr. Často platí, že $\min_i = \max_i$ a jedná se tedy o hyperkrychli.
f Účelová funkce, pro kterou platí, že $E \subset D_f$, a jejíž minimum hledáme.
S Velikost hejna, počet agentů.
t Pořadové číslo iterace, časový krok.
i Index částice v rámci hejna, $i \in \hat{S}$[12].
$\vec{x}_i(t)$ Pozice i-té částice v čase t.
$\vec{v}_i(t)$ Rychlost i-té částice v čase t.
$\vec{p}_i(t)$ Poloha fitness i-té částice v čase t.
$\vec{l}_i(t)$ Poloha nejlepšího fitness z okolí i-té částice v čase t.
Každý z posledních čtyř vektorů má D složek.
$N_i(t)$ Množina indexů, které znázorňují, jaké částice patří do okolí i-té částice.

Inicializace

Nejdříve se zaměřme na topologii, kterou bude tato verze využívat. Na Obrázku 2.2 je znázorněna prstencová topologie, která je využívána v mnoha variantách PSO, i díky její jednoduchosti, nicméně ve standardu z roku 2006 se přistupuje k sofistikovanějšímu modelu sousedství, který se nazývá *adaptivní náhodná topologie*.

Tato topologie spočívá v tom, že se stanoví konstanta $K \in \mathbb{N}$, která udává počet agentů, které bude každá částice informovat o svém fitness. Nutno podotknout, že ta samá částice může být zvolena vícekrát. To znamená, že každá částice informuje nejméně jednu částici, a to sebe samu, a nejvíce $K+1$ částic, neboť informace o sobě samé má vždy. Jednoduchý důsledek takového zadání, je takový, že každá částice může být informována jedním, až S agenty. Jak vyplývá z [4], průměrně je každá částice informována od zhruba K jiných částic. Tento náhodný výběr K částic, které bude daný agent informovat probíhá buď při inicializaci, a nebo po každé iteraci, která nepřinesla zlepšení fitness dané částici.

Samotná inicializace bude probíhat následovně:

1. Stanovení dimenze D a prohledávaného prostoru E, který bude reprezentován maticí $\mathbb{B} \in \mathbb{R}^{D\times 2}$, kde $[\mathbb{B}]_{d\bullet} = (\min_d, \max_d) \subset \overline{\mathbb{R}}, d \in \hat{D}$.
2. $S = 10 + \lfloor 2\sqrt{D} \rfloor$.

[12] Nechť $a \in \mathbb{N}$, potom značíme $\hat{a} = \{1, 2, \ldots, a\}$ a $\underline{\hat{a}} = \{0, 1, 2, \ldots, a\}$

3. Vytvoření matice $\mathbb{N} \in \mathbb{R}^{S \times S}$, která vyjadřuje zvolenou topologii.
   $[\mathbb{N}]_{\bullet i} = N_i(0),\ i \in \hat{S}$.
   Podoba matice $\mathbb{N}$ je znázorněna na příkladu na Obrázku 2.3.
4. Vytvoření matice $\mathbb{X} \in \mathbb{R}^{D \times S}$, která představuje hejno částic.
   $[\mathbb{X}]_{\bullet i} = \vec{x}_i,\ i \in \hat{S}$.
   $x_{i,d}(0) = U(\min_d, \max_d),\ d \in \hat{D}$.
5. Vytvoření matice $\mathbb{V} \in \mathbb{R}^{D \times S}$, jejíž prvky odpovídají rychlosti částic v příslušných směrech.
   $[\mathbb{V}]_{\bullet i} = \vec{v}_i$.
   $v_{i,d}(0) = \frac{U(\min_d, \max_d) - x_{i,d}(0)}{2}$.
6. Vytvoření matice $\mathbb{P} \in \mathbb{R}^{D \times S}$, která je nosičem pozic fitness jednotlivých částic.
   $[\mathbb{P}]_{\bullet i} = \vec{p}_i$.
   $\vec{p}_i(0) = \vec{x}_i(0)$.
7. Řádkový vektor $\vec{p}_{hodnoty} \in \mathbb{R}^S$, do kterého jsou ukládány fitness všech částic.
   $\vec{p}_{hodnoty}(0) = \big(f(\vec{p}_1(0)), ..., f(\vec{p}_S(0))\big)$.
8. Vytvoření matice $\mathbb{L} \in \mathbb{R}^{D \times S}$, která reprezentuje nejlepší pozice, o kterých jsou částice informovány.
   $[\mathbb{L}]_{\bullet i} = \vec{l}_i$.
   $\vec{l}_i(0) = \underset{\substack{j \in N_i(0) \\ j \neq 0}}{\operatorname{argmin}} f(\vec{p}_j(0))$.

Iterace

Iterace budou probíhat tak dlouho, dokud nebude splněno kritérium pro ukončení výpočtu. Takové podmínky jsou zpravidla dvě, a to sice:

- Pokud známe hodnotu účelové funkce v optimu $opt_{hodnota}$, můžeme stanovit minimální přípustnou chybu err, a pokud v průběhu výpočtu bude hodnota nějakého fitness vzdálena od skutečné optimální hodnoty méně než o minimální chybu, výpočet končí. Pokud $\exists i \in \hat{S}\ :\ |opt_{hodnota} - p_{hodnoty,i}| < err\ \Rightarrow$ konec.
- Druhá možnost je stanovení maximálního počtu iterací max_{iter} nebo maximálního počtu vyhodnocení účelové funkce max_{eval}.

Nyní budou definovány úpravy rychlosti a polohy částic. Následující dvě rovnice byly samozřejmě v průběhu času upravovány, ale svůj hlavní tvar dostaly už v roce 1995, kdy byl algoritmus PSO poprvé sestaven.
Nejprve úprava rychlosti:

$$v_{i,d}(t+1) = w v_{i,d}(t) + U_1(0,c)(p_{i,d}(t) - x_{i,d}(t)) + U_2(0,c)(l_{i,d}(t) - x_{i,d}(t)). \quad (2.4)$$

Hodnoty parametrů w a c jsou převzaty z [4] a odvozeny jsou na základě empirických i teoretických poznatků v [3]:

$$w = \frac{1}{2\ln(2)} \quad \approx 0.721 \quad (2.5)$$

$$c = \frac{1}{2} + \ln(2) \ \approx 1.193 \quad (2.6)$$

	1	2	3	4	5	6	7	8	9	10	11	12
1	**1**		1					1			1	
2	2	**2**			2							
3			**3**									
4				**4**	4					4		4
5		5			**5**				5			
6			6			**6**						
7					7		**7**					7
8				8				**8**				
9	9					9			**9**			
10		10			10					**10**		10
11				11			11				**11**	
12					12				12			**12**

Obrázek 2.3: Matice $\mathbb{N}$, zde $S = 12$, $K = 3$. Na i-tém řádku této matice, je znázorněno, jakým částicím jsou poskytovány informace od i-té částice. Naopak, v i-tém sloupci je zachycena množina všech sousedů, od kterých i-tá částice získává informace. Tudíž máme-li např. v pátém řádku a ve druhém sloupci nenulové číslo (políčka bez čísel jsou považována za nulové prvky), které vždy odpovídá číslu řádku, znamená to, že druhá částice může čerpat z informací, které poskytuje částice pátá. Jinak řečeno, pátá částice leží v okolí druhé částice. Diagonála této matice bude vždy tvořena vektorem $(1, 2, ..., S)$, neboť každá částice sama sobě informace poskytuje. Z obrázku lze vidět, že v každém řádku bude minimálně 1 a maximálně $K+1$ nenulových hodnot a v každém sloupci minimálně 1 a maximálně S nenulových hodnot, což přesně odpovídá definici adaptivní náhodné topologie.

Parametr w se nazývá koeficient tření a měl by zabraňovat tomu, aby se rychlost neúměrně zvětšovala. Pokud by se rychlost stále zvětšovala, agent bude vylétávat z definičního oboru účelové funkce, a tedy nebude schopen provádět lokální průzkum. Takový problém se nazývá *exploze* a byl jednou z komplikací při testování prvního modelu PSO [17]. Parametr c určuje, kolik částic zhruba přeletí doposud dosažené nejlepší pozice. Význam parametru c je ilustrován na Obrázku 2.4.

Po aktualizace rychlosti následuje očekávaný krok, a to sice aktualizace polohy:

$$x_{i,d}(t+1) = x_{i,d}(t) + v_{i,d}(t+1). \tag{2.7}$$

Na základě předchozích definic v inicializaci by šla úprava polohy zapsat vektorově, resp. maticově, následovně:

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1), \tag{2.8}$$

$$\mathbb{X}(t+1) = \mathbb{X}(t) + \mathbb{V}(t+1). \tag{2.9}$$

Pokud prohledávaný prostor E je přímo $\mathbb{R}^D$, není potřeba řešit případy, kdy nějaká částice z tohoto prostoru vyletí. Nicméně většinou budou na proměnné účelové funkce vznášena určitá omezení. Proto je potřeba nějakým způsobem ošetřit případy, kdy agent poruší některou z těchto podmínek. Jakým způsobem se takové podmínky realizují bude detailně popsáno v kapitole *Testování*.

Následujícím krokem je porovnání funkčních hodnot nových pozic agentů s jejich osobním, resp. lokálním fitness. Pokud existuje agent, u kterého nedošlo ke zlepšení osobního fitness, tak u tohoto agenta dojde ke změně informovaných částic. Opět je důležité, aby čísla $U_1(0, c)$, resp. $U_2(0, c)$, vystupující v rovnici (2.4), byla různá jak pro všechny souřadnice, tak pro všechny částice.

2.2.2 SPSO 2011

Další variantou, která bude popsána, je standardní verze z roku 2011. Co se týká inicializace, liší se tato verze od té z roku 2006 ve dvou směrech, a to sice:

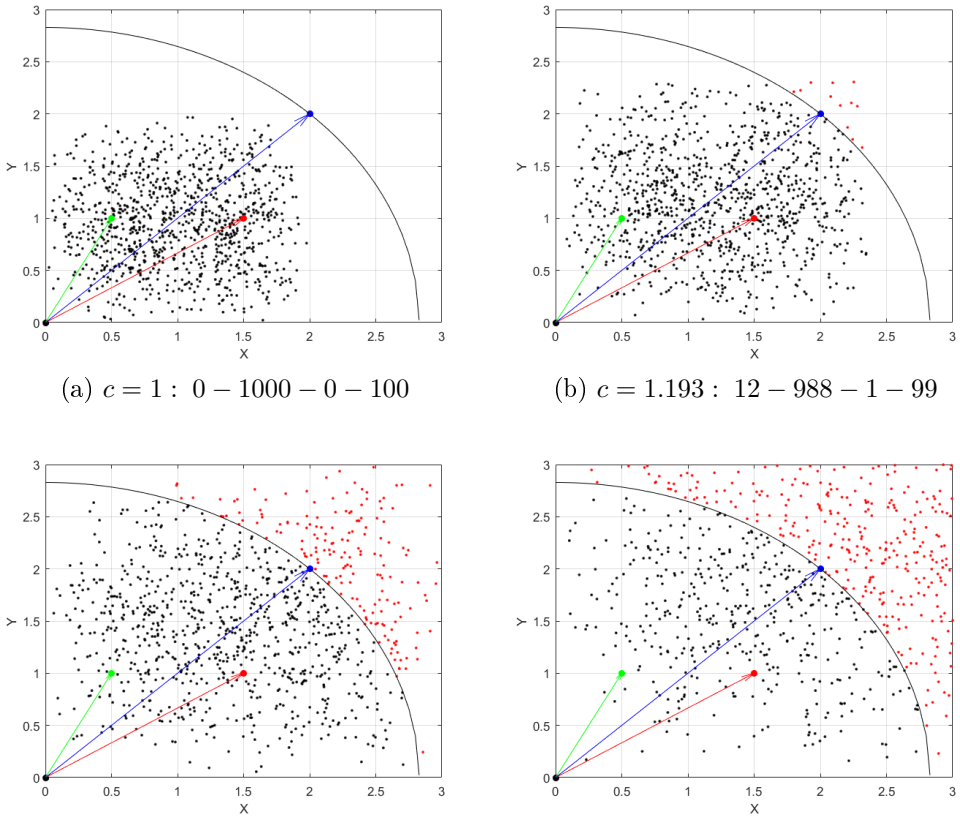
- Velikost hejna není odvozená na základě dimenze ze vzorce $S = 10 + \lfloor 2\sqrt{D} \rfloor$. Existuje doporučená velikost hejna, která odpovídá $S = 40$, ale pevně daná není.
- Počáteční rychlost se stanovuje následovně:

  $$v_{i,d}(0) = U(\min_d, \max_d) - x_{i,d}(0).$$

  Tedy je dvakrát vyšší než počáteční rychlost u SPSO 2006.

Další odlišnost spočívá v úpravě rychlosti a následující pozice. V této verzi se rychlost a pozice odvozuje na základě tzv. *gravitačního centra* $\vec{G}_i$, které je pro každou částici i definováno následovně:

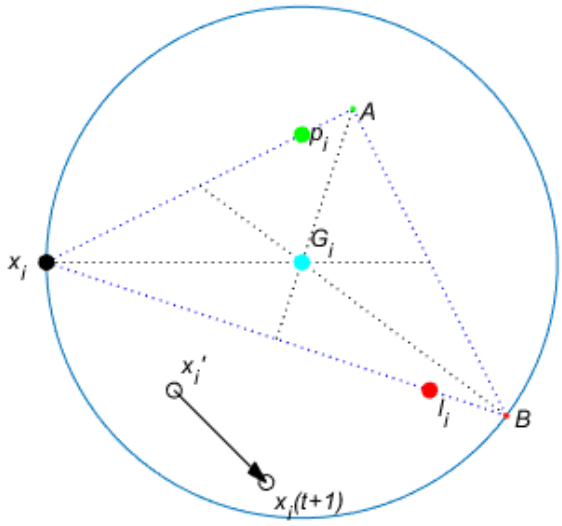
$$\vec{G}_i = \frac{\vec{x}_i + \overbrace{(\vec{x}_i + c(\vec{p}_i - \vec{x}_i))}^{A} + \overbrace{(\vec{x}_i + c(\vec{l}_i - \vec{x}_i))}^{B}}{3} = \vec{x}_i + c\frac{\vec{p}_i + \vec{l}_i - 2\vec{x}_i}{3}, \tag{2.10}$$



(a) $c = 1: \ 0 - 1000 - 0 - 100$ (b) $c = 1.193: \ 12 - 988 - 1 - 99$

(c) $c = 1.5: \ 827 - 173 - 83 - 17$ (d) $c = 2: \ 428 - 572 - 43 - 57$

Obrázek 2.4: Vizualizace parametru c. Na souřadnicích $\vec{x} = (0, 0)$ je agent, jehož osobní fitness (velký zelený puntík) je na souřadnicích $\vec{p} = (\frac{1}{2}, 1)$ a fitness jeho okolí (velký červený puntík) leží v $\vec{l} = (\frac{3}{2}, 1)$. Černé a červené tečky znázorňují celkem 1000 možných pozic, kterých může agent dosáhnout v další iteraci a které jsou spočítány z rovnice (2.4), kde $\vec{v}(0) = (0, 0)$, a tedy souřadnice těchto teček odpovídají přímo složkám vektoru $\vec{v}(1)$. Modrá šipka znázorňuje vektor $(\vec{p} - \vec{x}) + (\vec{l} - \vec{x})$, zde tedy $\vec{p} + \vec{l}$.
Popisky u jednotlivých obrázků jsou ve formátu $out - in - \%out - \%in$. Číslo out, resp. in, udává, kolik agentů „přeletělo“, resp. „nepřeletělo“ bod $\vec{p} + \vec{l}$. Poslední dvě hodnoty jsou procentuálním vyjádřením předchozího v rámci 100 testů, jsou tedy přesnějším zachycením toho, kolik agentů přelétne či nepřelétne bod $\vec{p} + \vec{l}$ při dané hodnotě parametru c.
Z Obr. 2.4a je vidět, že při $c = 1$ převažuje spíše prohledávání nejbližšího okolí (exploatace) oproti průzkumu prostředí, které je více vzdálené (explorace). Nejlepší výsledky, tedy nejlepší rovnováhu mezi těmito dvěma přístupy k prohledávání prostoru, přinesla hodnota $c = \frac{1}{2} + \ln(2) \approx 1.193$. Na obr. 2.4c a 2.4d naopak převažuje explorace až příliš, což může vyústit k nedůslednému prohledání okolí, ve kterém se může nacházet optimum.



Obrázek 2.5: Výpočet nové pozice $i-$té částice.

kde $c > 1$. Tedy vezmeme bod, který je kousek za $\vec{p}_i$, resp. za $\vec{l}_i$, a hledáme těžiště trojúhelníku, který je tvořen body $\vec{x}_i, A, B$.
Poté volíme náhodně bod $\vec{x}_i'$, který se nachází uvnitř hyperkoule $H_i(\|\vec{G}_i - \vec{x}_i\|, \vec{G}_i)$, se středem v bodě $\vec{G}_i$ a o poloměru $\|\vec{G}_i - \vec{x}_i\|$. Úprava rychlosti a souřadnic poté vypadá takto:

- $\vec{v}_i(t+1) = w\vec{v}_i(t) + \vec{x}_i'(t) - \vec{x}_i(t)$
- $\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1) = w\vec{v}_i(t) + \vec{x}_i'(t)$

Varianta z roku 2006 nevěnuje zvláštní pozornost situacím, kdy $\vec{l}_i(t) = \vec{p}_i(t)$. Jinak řečeno, pro částici, která je zároveň „nositelem“ lokálního fitness, se používá stejná pravidla jako pro všechny ostatní. Zde je tento případ ošetřen následujícím způsobem:

$$\vec{G}_i = \frac{\vec{x}_i + (\vec{x}_i + c(\vec{p}_i - \vec{x}_i))}{2} = \vec{x}_i + c\frac{\vec{p}_i - \vec{x}_i}{2}.$$

Zajímavým detailem této verze je to, že díky konstrukci nové pozice, lze jednoduše a zároveň přesně definovat pojmy jako *exploatace* a *explorace*, a to následujícím způsobem:

- Pokud $\exists j \in \hat{S} \ : \ \vec{x}_i(t+1) \in H_j$, potom se jedná o exploataci.
- Jinak jde o exploraci, tedy pokud nová pozice $i-$té částice v čase t nespadá do žádné z hyperkoulí v rámci celého hejna.

Na Obrázku 2.5 je příklad exploatace.

Bylo ukázáno, že u standardní verze SPSO 2006, kde se rychlost každé částice upravuje podle (2.4), má celé hejno nežádoucí sklon konvergovat příliš rychle, pokud se optimum nachází na hlavních osách souřadného systému [25]. Pokud se tedy

optimum nachází dokonce v počátku souřadného systému, je konvergence nejrychlejší. Takové chování hejna může způsobit zkreslení testování algoritmu, kde testovací funkce mají často globální minimum právě v nule.
Výhodou této varianty z roku 2011 je to, že úprava rychlosti je založena na přístupu, který je nezávislý na souřadném systému. Na druhou stranu, v algoritmu SPSO 2006 generování náhodných čísel probíhalo vždy uvnitř nějaké hyperkrychle, což nyní může být problém, pokud chceme generovat rovnoměrně rozložené vektory uvnitř kruhu, koule či obecně u $n-$rozměrné koule. V poslední části této kapitoly je popsáno několik způsobů, jak takové vektory získávat.

2.3 Bezparametrický přístup

Další významnou variantou PSO je tzv. *TRIBES*[13], neboli česky *kmeny*, která vznikla v důsledku vidiny komplexního programu, který by pro svůj běh potřeboval co nejméně vstupních parametrů. Nicméně definice problému (účelové funkce), stanovení definičního oboru a podmínek ukončení[14] běhu programu jsou stále nezbytnou součástí zadání. Ovšem nic víc. Hlavní výsadou TRIBES je schopnost si přizpůsobovat velikost hejna podle průběhu výpočtu. Program tedy vždy začíná s jediným agentem, a podle toho, jestli se daří nalézat dobré či špatné výsledky, přidává či odebírá částice.

Jak název napovídá, v hejně bude definován nový typ topologie, který v jistém smyslu opravdu bude připomínat kmeny částic [2]. Každá skupina, neboli *kmen*, bude tvořena jedinci, kteří spolu sdílejí své zkušenosti a jejichž počet se mění. Společně pak hledají dobré oblasti (globální minima účelové funkce), a tyto informace určitým způsobem vyměňují i do všech ostatních kmenů. V hejnu se tedy budou dynamicky vytvářet skupiny jedinců, které mohou vznikat, měnit svou velikost a také zanikat. To, jakým způsobem bude probíhat utváření takových skupin a jakým způsobem bude probíhat komunikace mezi nimi a mezi jejich jednotlivými členy, bude popsáno v této kapitole.

2.3.1 Definice a vlastnosti kmene

Jako první bude definován tzv. *informátor* [17]. Řekneme, že agent x_i je informátorem, nebo také že *informuje*, agenta x_j, pokud má agent x_j dostupné informace o nejlepší dosažené pozici agenta x_i [2]. Tuto skutečnost budeme značit následovně:

$$\mathrm{INF}(x_i, x_j) = \begin{cases} 1, \text{ pokud } x_i \text{ informuje } x_j. \\ 0, \text{ jinak,} \end{cases}$$

kde $i, j = 1, 2, \ldots, S$.

[13] Poprvé tato varianta byla představena v roce 2003 Maurice Clercem [5]

[14] Těmi jsou: požadovaná přesnost, jištěná maximálním počtem iterací, maximálním počtem vyhodnocení účelové funkce, či omezeným výpočetním časem.

Z této definice je jasné, že $\mathrm{INF}(x_i, x_i) = 1$. U TRIBES se předpokládá, že pokud $\mathrm{INF}(x_i, x_j) = 1 \Rightarrow \mathrm{INF}(x_j, x_i) = 1$, i když v předešlých kapitolách bylo ukázáno, že takovým pravidlem se každá topologie řídit nemusí. Nyní můžeme konečně definovat *kmen*.

Množinu částic $K \subset \hat{S}$, kde S je velikost celého hejna (tedy všech jedinců dohromady), nazveme *kmen*, pokud $\forall i, j \in K = \{i_1, i_2, \ldots, i_k\}$ platí:

$$\mathrm{INF}(x_i, x_j) = 1.$$

V rámci jednoho kmene tedy platí, že každý agent je informován od všech ostatních členů tohoto kmene. Pokud bychom agenty přirovnali k uzlům a skutečnost, že jeden informuje druhého, přirovnali zase k hranám grafu, potom kmen tvoří úplný graf. Dále musí platit, že mezi každými dvěma agenty v rámci celého hejna (nemusí být nutně z jednoho kmene), existuje nějaké spojení. Jinak řečeno, celé hejno tvoří souvislý graf a mezi každými dvěma agenty (uzly) musí existovat cesta.

2.3.2 Vývoj kmene

Abychom mohli popsat způsob, jakým budou kmeny vytvářeny a modifikovány, musíme nejdříve definovat pojmy *kvalita částice* a *kvalita kmene*.

Nejdříve definujeme kvalitu částice. Částici x_i považujeme v čase t za *dobrou*, pokud si zlepšila svoji fitness. Tedy pro její osobní nejlepší dosaženou polohu $\vec{p}_i$ platí: $f(\vec{p}_i(t)) < f(\vec{p}_i(t-1))$. Jinak považujeme částici za *neutrální* [17]. Pro přehlednější zápis definujme pro částici x

$$\mathrm{pg}(x) = \begin{cases} 1, & \text{pokud je částice } x \text{ dobrá.} \\ 0, & \text{pokud je částice } x \text{ neutrální.} \end{cases}$$

Pro další pokračování bude třeba stanovit si nejhorší částici uvnitř kmene. To se provede tak, že se ze všech neutrálních vybere ta částice, která má největší hodnotu fitness (pokud jich je více se stejnou fitness, zvolí se náhodně jedna z nich).

Nyní můžeme definovat kvalitu kmene. Mějme kmen $K = \{x_{i_1}, \ldots, x_{i_k}\}$ o velikosti $k \leq S$. Nechť $r \sim \mathrm{U}\{0, 1, 2, \ldots, k\}$ je rovnoměrně vybrané náhodné číslo a $N_g^K = \sum_{j=1}^{k} \mathrm{pg}(x_{i_j})$ je počet dobrých částic v rámci kmene K. Kmen K považujeme za *špatný*, pokud

$$N_g^K \leq r.$$

Jinak považujeme kmen K za *dobrý*. Na základě kvality kmene a jeho členů bude nyní popsáno, jakým způsobem se kmeny budou vyvíjet.

Kromě vidiny samostatnosti programu je dynamicky se měnící počet částic v hejnu také vylepšením oproti statické velikosti, kde pro určitý problém může být ideální počet částic $S = 20$, ale pro jiný toto číslo může být úplně jiné. Proměnnost velikosti hejna se také snaží určitým způsobem minimalizovat počet vyčíslení účelové funkce. Zároveň se ale nesmí stát, že bychom kvůli odstranění jedné nebo několika částic ztratili optimální řešení. To bude ošetřeno tak, že budeme odstraňovat pouze nejhorší

částici z dobrého kmene. Toto nám zaručuje, že neztratíme důležitou informaci, neboť tu budou držet ostatní členové dobrého kmene. Pokud nastane situace, že kmen, který byl označen jako *dobrý*, obsahuje pouze jednu částici, která je tedy automaticky nejhorší, odstranění této částice proběhne jedině tehdy, pokud některý z jejích informátorů drží lepší informaci než tato částice. Při odstraňování částic z kmenů musí být stále dodržovány pravidla topologie TRIBES. To bude zajištěno tak, že při odstranění nejhorší částice v kmeni převezme její informátory nejlepší částice v rámci tohoto kmene. Pokud je částice v kmeni poslední a splňuje podmínky pro odstranění, budou všichni její informátoři navzájem propojeni.

Na druhou stranu pokud bude kmen klasifikován jako špatný, znamená to, že nemá dostatek informací o prohledávaném prostoru, a je nutné vygenerovat nové částice, které budou přímo spojené s tímto špatným kmenem. Takové částice se budou generovat vždy dvě, a to v každém kmeni. Takto vytvořené částice vždy vytvoří jeden další kmen, tzn. máme-li $m \in \mathbb{N}$ špatných kmenů, potom se vytvoří nový kmen o $2m$ částicích, který bude mít přímou návaznost na oněch m špatných kmenů. Každou z těchto dvou částic budeme dělit na dva typy: *volná* a *omezená*. Volná částice je generována náhodně rovnoměrně napříč celým prohledávaným prostorem, představuje tedy explorační prvek výpočtu. Omezená částice bude naopak představovat exploataci, tedy prohledávání okolí nejlepší částice x z generujícího kmene a okolí jejího nejlepšího informátora y. Tato částice bude generována rovnoměrně uvnitř hyperkoule o poloměru $\|\vec{g}-\vec{p}\|$, kde $\vec{p}$ je fitness x a $\vec{g}$ je fitness y. Často se tedy nová částice bude generovat přímo na pozici částice x, neboť ona sama bude představovat jejího nejlepšího informátora. Spojení bude vždy mezi nejlepší částicí nového kmene a nejlepší částicí kmene špatného.

Jak již bylo řečeno, mezi dvěma částicemi existuje vždy cesta. To znamená, že každá částice bude s nějakou prodlevou informována o nejlepší pozici v rámci celého hejna. Právě kvůli této prodlevě není žádoucí, aby vyhodnocování toho, je-li kmen dobrý či špatný, probíhalo v každé iteraci. Doporučená hodnota prodlevy je $L = NL/2$, kde NL je počet všech informačních spojení v celém hejně po poslední adaptaci, a tedy další adaptace by se měla konat nejpozději po L iteracích [24]. Výpočet čísla

$$NL = \sum_{i=1}^{NT} NP_i^2 + NT(NT-1) = \sum_{i=1}^{S}\sum_{j=1}^{S} \mathrm{INF}(x_i, x_j) + NT(NT-1),$$

kde NT je počet kmenů, NP_i je počet částic v $i-$tém kmeni, S je velikost celého hejna, předpokládá, že každý kmen je přímo spojený se všemi ostatními. Tento odhad je ovšem nadhodnocený, neboť zahrnuje i skutečnost, že $\mathrm{INF}(x,x) = 1$ a obsahuje jak $\mathrm{INF}(x,y)$ tak i $\mathrm{INF}(y,x)$. Číslo L by mohlo být odhadnuto jako

$$L = \sum_{i=1}^{NT} \frac{NP_i(NP_i-1)}{2} + NT(NT-1).$$

Nejvhodnější hodnotou čísla L by byla samozřejmě maximální z minimálních vzdáleností mezi dvěma agenty, nicméně toto by znamenalo před každou adaptací nalézt pro každé dva agenty v hejnu nejkratší cestu mezi nimi, což by mohlo být časově nepřijatelné.

2.3.3 Modifikace polohy agenta

Kromě nutnosti zadání velikosti hejna, mizí v TRIBES také starost o určení koeficientu tření při úpravě rychlosti částice. Částice v TRIBES totiž žádnou rychlostí nedisponuje, mění se pouze její poloha. Dalším vylepšením je to, že si částice pamatuje dvě poslední dosažené pozice. Na základě této historie se částice budou dělit do tří skupin a každá z těchto skupin bude používat jinou úpravu polohy.

Skutečnost, že si agent pohoršil, resp. polepšil, resp. setrvává ve stejném stavu, budeme značit znaménkem „-“, resp. „+“, resp. „=“[15]. Pokud si agent např. pohorší, a poté si naopak polepší, budeme takový sled událostí značit $(-+)$. Protože máme celkem tři možnosti v každém ze dvou stavů, existuje celkem devět případů, jak může historie částice vypadat. Do skupin se dělí následovně:

1. Špatná skupina: $(--)(=-)(+-)(-=)(==)$.
2. Dobrá skupina: $(+=)(-+)$.
3. Excelentní skupina: $(=+)(++)$.

Lze říci, že částice spadající do excelentní skupiny, mají větší právo na prohledávání okolí svého nejlepšího informátora (což bude často právě sama částice), tedy u ní probíhá větší lokální průzkum. Nyní zde budou popsány tři metody určení nové polohy částice, které využívají jednotlivé skupiny [24].

1. **Pivot** (metoda pro špatnou skupinu)

$$\begin{aligned} \vec{x} &= \alpha \mathrm{U}\left(K_n(r,\vec{p})\right) + \beta \mathrm{U}\left(K_n(r,\vec{g})\right), \\ \alpha &= \frac{f(\vec{p})}{f(\vec{p})+f(\vec{g})}, \qquad (2.11) \\ \beta &= \frac{f(\vec{g})}{f(\vec{p})+f(\vec{g})}, \qquad (2.12) \end{aligned}$$

   kde $\mathrm{U}\left(K_n(r,\vec{p})\right)$ je bod rovnoměrně vybraný uvnitř $n-$rozměrné koule se středem v bodě $\vec{p}$ a s poloměrem $r = \|\vec{p}-\vec{g}\|$. Bod $\vec{p}$, resp. $\vec{g}$, je nejlepší osobní poloha, resp. poloha nejlepšího informátora [16].

2. **Pivot se šumem** (metoda pro dobrou skupinu) [1]

$$\begin{aligned} \vec{x} &= \alpha \mathrm{U}\left(K_n(r,\vec{p})\right) + \beta \mathrm{U}\left(K_n(r,\vec{g})\right), \\ b &= \mathrm{N}\left(0, \left|\frac{f(\vec{p})-f(\vec{g})}{f(\vec{p})+f(\vec{g})}\right|\right), \qquad (2.13) \\ x &= (1+b)x. \end{aligned}$$

[15] Pojmem „pohoršil“ rozumíme přemístění částice na místo, kde je hodnota účelové funkce horší než je v jeho fitness. Pokud si agent „polepšil“, nalezl místo s lepší hodnotou než je v jeho současné fitness.

Tato metoda je tedy téměř totožná s první, nicméně díky zařazení parametru b, více částic bude mít možnost provádět lokální průzkum.

3. **Lokální** (metoda pro excelentní skupinu)

$$x_i = g_i + \mathrm{N}\left(\|g_i - x_i\|, g_i - x_i\right),\ i = 1, 2, \ldots, D.$$

Tedy každá souřadnice polohy je upravována nezávisle na ostatních.

Při implementaci takto definovaných parametrů (α, β, b) ovšem narazíme na jeden problém. Chyba v definici těchto tří parametrů nastává v okamžiku, když globální minimum účelové funkce je rovno nule. Potom se ve výrazech (2.11),(2.12),(2.13) bude dělit nulou. Je potřeba tedy tento případ nějak ošetřit.

Nejdříve se podíváme na metodu Pivot. Je zřejmé, že $\beta = 1 - \alpha$. Stačí tedy kontrolovat pouze $f(\vec{p})$. To provedeme tak, že si stanovíme mez ϵ, a pokud $f(\vec{p}) < \epsilon$, potom vybereme α náhodně rovnoměrně z intervalu $\langle 0.5, 1\rangle$. To plyne z toho, že funkce f je nezáporná a tedy výraz (2.11) lze odhadnout jako

$$\frac{1}{2} = \frac{f(\vec{p})}{f(\vec{p}) + f(\vec{p})} \leq \frac{f(\vec{p})}{f(\vec{p}) + f(\vec{g})} \leq \frac{f(\vec{p})}{f(\vec{p}) + 0} = 1.$$

Co se týká metody Pivot se šumem, rozptyl $\sigma^2 = \left|\frac{f(\vec{p}) - f(\vec{g})}{f(\vec{p}) + f(\vec{g})}\right| \in \langle 0, 1\rangle$, což plyne z obdobného odhadu jako pro parametr α. Pokud tedy $f(\vec{p}) < \epsilon \vee f(\vec{g}) < \epsilon$, potom $\sigma^2 \sim \mathrm{U}(0, 1)$.

2.4 Generování náhodných vektorů uvnitř hyperkoule

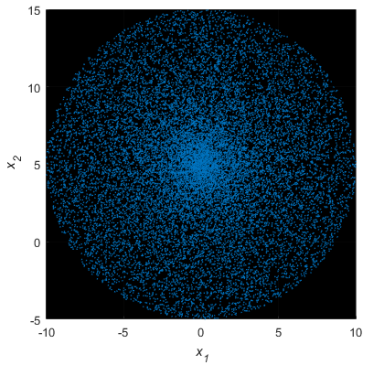
Mějme $n-$rozměrnou kouli[16] $K_n(r, \vec{S})$ s poloměrem r a se středem v bodě $\vec{S} = (s_1, \ldots, s_n)$.
Tedy

$$K_n(r, \vec{S}) = \{\vec{x} \in \mathbb{R}^n \ :\ \|\vec{x} - \vec{S}\| \leq r\}. \tag{2.14}$$

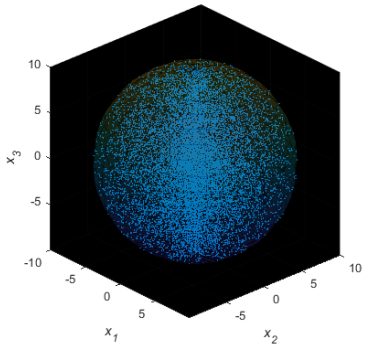
1. Asi nejintuitivnějším přístupem je přechod ke sférickým souřadnicím. Každý bod $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ lze převést do sférických souřadnic $(\rho, \phi_1, \ldots, \phi_{n-1})$ následujícím způsobem [28]:

   - $n = 2$:
   $$x_1 = \rho \sin(\phi_1),$$
   $$x_2 = \rho \cos(\phi_1).$$
   - $n = 3$:
   $$x_1 = \rho \sin(\phi_2) \sin(\phi_1),$$
   $$x_2 = \rho \sin(\phi_2) \cos(\phi_1),$$
   $$x_3 = \rho \cos(\phi_2).$$

[16] Pojmem *hyperkoule* bude dále myšlena vždy obecně $n-$rozměrná koule, pokud nebude řečeno jinak.



(a) Rozložení vektorů uvnitř kruhu.



(b) Rozložení vektorů uvnitř koule.

Obrázek 2.6: Generování náhodných vektorů pomocí převodu na do sférických souřadnic nevytváří rovnoměrné rozložení vektorů.

- $n \geq 4$:

$$\begin{aligned} x_1 &= \rho \sin(\phi_{n-1}) \dots \sin(\phi_2) \sin(\phi_1), \\ x_2 &= \rho \sin(\phi_{n-1}) \dots \sin(\phi_2) \cos(\phi_1), \\ &\dots \\ x_{n-1} &= \rho \sin(\phi_{n-1}) \cos(\phi_{n-2}), \\ x_n &= \rho \cos(\phi_{n-1}), \end{aligned}$$

  kde

$$\begin{aligned} &0 \leq \rho < +\infty, \\ &0 \leq \phi_1 < 2\pi, \\ &0 \leq \phi_k < \pi, \quad k = 2, \dots, n-1. \end{aligned}$$

  První možností je tedy generovat náhodně vektor $(\rho, \phi_1, \dots, \phi_{n-1})$, kde $0 \leq \rho \leq r$ a $\phi_1, \dots, \phi_{n-1}$ je vybíráno na základě (1). Pokud potom sestavíme $\vec{x}$ podle (1), získáme takto vektor, který leží uvnitř hyperkoule $K_n(r, \vec{0})$. Stačí tedy $\vec{x} := \vec{x} + \vec{S}$ a dostaneme náhodně vybraný vektor uvnitř hyperkoule $K_n(r, \vec{S})$. Nicméně takové vektory nebudou rovnoměrně rozloženy uvnitř hyperkoule, což je v případě algoritmu PSO nežádoucí, neboť by mohlo docházet ke zkreslování výsledků při testování na modelových funkcích (viz Obrázek 2.6). Musíme tedy ke generování takových vektorů přistoupit jinak.

2. Druhá metoda se nazývá *metoda zamítnutí* (angl. *acceptance-rejection method)*. Princip je takový, že máme-li hyperkouli $K_n(r, \vec{S})$, vezmeme minimální krychli dimenze n tak, aby se do ní hyperkoule $K_n(r, \vec{S})$ vešla, a budeme generovat vektory uvnitř této krychle. Jak název metody napovídá, z vektorů takto vygenerovaných budeme vybírat ty, které zároveň leží uvnitř hyperkoule, a ty, které jsou mimo, budeme zahazovat. Výběr jednoho takového vektoru je popsán pseudokódem v Algoritmu 2. Takto získané vektory budou rovnoměrně rozložené uvnitř hyperkoule. Tato metoda je ovšem nepoužitelná pro větší n, neboť poměr $k_n = \frac{\text{objem hyperkoule}}{\text{objem hyperkrychle}} = \frac{V_n^{\bigcirc}}{V_n^{\square}}$ bude s rostoucím n rychle klesat k

Algoritmus 2 Metoda zamítnutí

1: Hyperkoule $K_n(r, \vec{S})$
2: Hyperkrychle $G = \prod_{i=1}^{n} <s_i - r, s_i + r>$
3: **for** $i = 1, \ldots, n$ **do**
4: Generuj $x_i \sim \mathrm{U}(-r, r)$.
5: **end for**
6: $\vec{x} := \vec{x} + \vec{S}$.
7: **if** $\vec{x} = (x_1, \ldots, x_n) \in K_n(r, \vec{S})$ **then**
8: Konec.
9: **else**
10: Zpátky na krok 3.
11: **end if**

nule.

Důkaz. Objem hyperkrychle G je roven $V_n^{\square} = (2r)^n$. Objem hyperkoule $K_n(r, \vec{S})$ je roven [6]

$$V_n^{\bigcirc} = \begin{cases} \dfrac{(\pi)^{\frac{n}{2}}}{(\frac{n}{2})!} r^n, \text{ pro } n \text{ sudé.} \\ \\ \dfrac{2(2\pi)^{\frac{1}{2}(n-1)}}{n!!} r^n, \text{ pro } n \text{ liché.} \end{cases}$$

Tedy pro sudé $n = 2m, m \in \mathbb{N}$ je

$$k_n = k_{2m} = \frac{\pi^m}{(2r)^{2m} m!} r^{2m} = \left(\frac{\pi}{4}\right)^m \cdot \frac{1}{m!}.$$

Určitě $\lim_{m \to +\infty} k_{2m} = 0$.
Obdobně pro liché $n = 2m+1, m \in \mathbb{N}_0$ je

$$k_n = k_{2m+1} = \frac{2(2\pi)^m}{(2r)^{2m+1}(2m+1)!!} r^{2m+1} = \left(\frac{\pi}{2}\right)^m \cdot \frac{1}{(2m+1)!!}.$$

Protože pro všechny $m \in \mathbb{N}_0$ je $k_{2m+1} > 0$ a $\lim_{m \to +\infty} \frac{k_{2m+3}}{k_{2m+1}} = \lim_{m \to +\infty} \frac{\pi}{2(2m+3)} = 0$, tak z podílového kritéria pro posloupnosti okamžitě plyne, že $\lim_{m \to +\infty} k_{2m+1} = 0$. □

Některé z hodnot k_n jsou vypsány v Tabulce 2.1.

k_2	k_3	k_4	k_5	k_{10}
0,7854	0,5236	0,3084	0,1645	0,0025

Tabulka 2.1: Konkrétní hodnoty poměru k_n.

Poměr k_n je vlastně pravděpodobnost, se kterou vygenerovaný vektor $\vec{x}$ bude ležet uvnitř hyperkoule, takže pokud bychom potřebovali dostat N bodů uvnitř hyperkoule, v průměru bychom potřebovali $p_N^{(n)} = \frac{N}{k_n}$ pokusů. Je zřejmé, že taková metoda je pro větší n těžko použitelná (viz Tabulka 2.2).

$p_N^{(2)}$	$p_N^{(3)}$	$p_N^{(4)}$	$p_N^{(5)}$	$p_N^{(10)}$
127	191	324	608	40 154

Tabulka 2.2: Potřebné tahy pro generování $N = 100$ vektorů uvnitř hyperkoule metodou zamítnutí.

3. Třetí metoda bude fungovat pouze pro $n = 2$, nicméně zaručuje $K_2(r, \vec{S})$. Postup je následující:

Algoritmus 3 Trojúhelníková metoda

1: Kruh $K_2(r, \vec{S})$
2: $u_1, u_2, u_3 \sim \mathrm{U}(0, 1)$
3: $\phi = 2\pi u_1$
4: $R' = r(u_1 + u_2)$
5: $R = \begin{cases} 2r - R', & \text{pokud } R' > r \\ R', & \text{jinak} \end{cases}$
6: $\vec{x} = R \begin{pmatrix} \cos(\phi) \\ \sin(\phi) \end{pmatrix} + \vec{S}.$

Důkaz. Mějme rovnoramenný trojúhelník ABC, kde $|AB| = |BC|$. Prvním krokem je generovat body rovnoměrně uvnitř tohoto trojúhelníku. Nejdříve vytvoříme kosočtverec $ABCD$, potom náhodně zvolíme bod E na straně a a bod F na straně b. Bod X vytvoříme tak, aby vznikl kosodélník $EBFX$. Pokud takto vytvořený bod neleží uvnitř trojúhelníku ABC, „překlopíme“ jej podle strany AC. Body takto generované budou rovnoměrně rozložené uvnitř ABC.
Nechť máme kružnici $k(r, B)$. Zvolme dva body A, C na této kružnici. Platí $|AB| = |BC| = r$. Uvnitř takového trojúhelníku ABC umíme vytvářet rovnoměrně rozdělené body. V limitním případě, kdy $|AC| \to 0$, bude výška trojúhelníku ABC odpovídat poloměru kružnice r. Protože kruh $K_2(r, B)$ lze vyplnit nekonečně mnoho takovými trojúhelníky, výběr jednoho takového bude odpovídat náhodnému výběru úhlu ϕ. Výběr E, F, resp. X odpovídá $r \cdot u_1, r \cdot u_2$, resp. R', a protože budou ležet na stejné přímce, lze převracení bodu provést tak, jak už bylo napsáno v Algoritmu 3. □

4. Čtvrtá, poslední metoda, která bude zmíněná, je universální pro libovolné $n \in \mathbb{N}$. Krok po kroku vypadá je zobrazená jako Algoritmus 4.
Takto vybírané vektory $\vec{z}$ budou rovnoměrně rozděleny uvnitř hyperkoule, tedy $\vec{z} \sim \mathrm{U}\left(K_n(r, \vec{S})\right)$.

Algoritmus 4 Rovnoměrné generování vektorů uvnitř $K_n(r, \vec{S})$

1: $x_i \sim \mathrm{N}(0,1), \forall i = 1, \ldots, n$
2: $\vec{x} = (x_1, \ldots, x_n)$
3: $\vec{y} = \frac{\vec{x}}{\|\vec{x}\|}$
4: $R = r \cdot U^{\frac{1}{n}},\ U \sim \mathrm{U}(0,1)$
5: $\vec{z} = R\vec{y} + \vec{S}$

Důkaz. Protože $\vec{x} = (x_1, \ldots, x_n), x_i \sim \mathrm{N}(0,1), \forall i \in \hat{n}$, potom hustota pravděpodobnosti $f(\vec{x}) = (2\pi)^{-\frac{n}{2}} \mathrm{e}^{-\frac{\|\vec{x}\|^2}{2}}$.
Takže $f(\vec{y}) = (2\pi)^{-\frac{n}{2}} \mathrm{e}^{-\frac{1}{2}}$ pro libovolné $\vec{y}$. To ale znamená, že vektory $\vec{y}$ jsou rovnoměrně rozloženy na povrchu jednotkové hyperkoule $K_n(1, \vec{0})$.
Aby $\vec{z}$ byly rovnoměrně rozmístěny uvnitř hyperkoule, musí platit, že pokud $R = \|\vec{z}\|$, potom pro $p \in \langle 0, r \rangle$ musí platit:

$$P(R \leq p) = \frac{\text{objem } K_n(p, \vec{S})}{\text{objem } K_n(r, \vec{S})} = \begin{cases} n = 2k & \Rightarrow \quad \dfrac{\pi^k p^{2k}}{k!} \cdot \dfrac{k!}{\pi^k r^{2k}} \\ n = 2k{+}1 & \Rightarrow \quad \dfrac{2(2\pi)^k p^{2k+1}}{(2k{+}1)!!} \cdot \dfrac{(2k{+}1)!!}{2(2\pi)^k r^{2k+1}} \end{cases},$$

$$P(R \leq p) = \left(\frac{p}{r}\right)^n,\ k \in \mathbb{N}.$$

Pokud zvolíme $U \sim \mathrm{U}(0,1)$, potom určitě $P(U \leq u) = u, u \in \langle 0, 1 \rangle$. Nyní zvolme $R = r \cdot U^{\frac{1}{n}}$. Z toho [22]:

$$P(R \leq p) = P(r \cdot U^{\frac{1}{n}} \leq p) = P\left(U \leq \left(\frac{p}{r}\right)^n\right) = \left(\frac{p}{r}\right)^n.$$

Tedy $R\vec{y} \sim \mathrm{U}\left(K_n(r, \vec{0})\right) \Rightarrow R\vec{y} + \vec{S} \sim \mathrm{U}\left(K_n(r, \vec{S})\right)$. □

Kapitola 3

Testování

V této kapitole budou uvedeny výsledky testů jednotlivých variant algoritmu a jejich vzájemné porovnání. Nejdříve bude vybrána nejvhodnější realizace podmínek pro každou z variant, pouze pro verzi z roku 1995 bude rovnou zachována metoda *nejbližší hranice*, jak tomu bylo při vzniku této verze. Budou zde také představeny testovací funkce, které budou později rozděleny do skupin podle jejich charakteristických vlastností.

3.1 Realizace omezujících podmínek

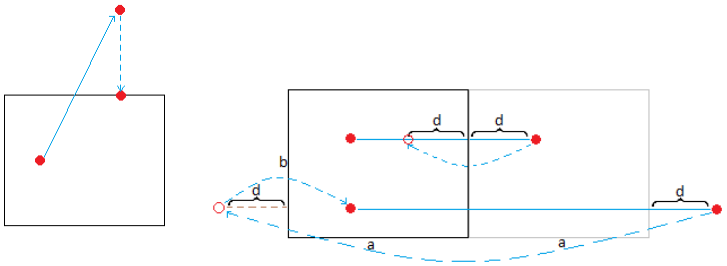
Rozhodování bude probíhat mezi třemi realizacemi omezujících podmínek, a to sice:

1. *Nejbližší hranice* (viz Obrázek 3.1a). Při opuštění prohledávaného prostoru jednoduše navrací agenta na nejbližší hranici. Toto můžeme zapsat takto:

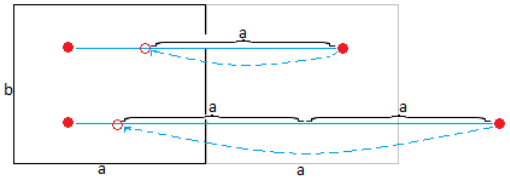
$$\begin{aligned} \text{Pokud } x_{i,d}(t{+}1) < \min_d \;\Rightarrow\; & x_{i,d}(t{+}1) = \min_d, \\ & v_{i,d}(t{+}1) = 0. \\ \text{Pokud } x_{i,d}(t{+}1) > \max_d \;\Rightarrow\; & x_{i,d}(t{+}1) = \max_d, \\ & v_{i,d}(t{+}1) = 0. \end{aligned}$$

2. *Zrcadlení* (viz Obrázek 3.1b). Pokud agent opustí prohledávaný prostor, je jeho poloha „převrácena“ podle hranice prohledávaného prostoru. Souhrnně lze psát:

$$\begin{aligned} & \text{delka} = \max_d - \min_d \\ & \text{Pokud } x_{i,d}(t{+}1) \notin (\min_d, \max_d) \;\Rightarrow\; \text{kolik} = \left\lfloor \frac{(\min_d - x_{i,d})}{\text{delka}} \right\rfloor \\ & \quad \text{Pokud kolik} \equiv 0 \mod 2 \Rightarrow x_{i,d} = \min_d + ((\min_d - \text{kolik} \cdot \text{delka}) - x_{i,d}) \\ & \quad \text{Pokud kolik} \equiv 1 \mod 2 \Rightarrow x_{i,d} = \max_d - ((\min_d - \text{kolik} \cdot \text{delka}) - x_{i,d}) \\ & \quad v_{i,d} = 0 \end{aligned}$$



(a) Nejbližší hranice (b) Zrcadlení



(c) Periodické rozšíření

Obrázek 3.1: Realizace podmínek. Obdélník představuje prohledávaný prostor (nebo také definiční obor), agent je označen červenou tečkou. Nepřerušovaná modrá čára ukazuje, jakým způsobem se agent dostal z prohledávaného prostoru, přerušované šipky pak naznačují, jakým způsobem bude agent přesunut při přeletění definičního oboru.

3. *Periodické rozšíření* (viz Obrázek 3.1c). Při této realizaci podmínek je agent navracen zpět o násobky délky prohledávaného prostoru v daném rozměru. Zapsáno:

$$\text{delka} = \max_d - \min_d$$

$$\text{Pokud } x_{i,d}(t{+}1) \notin (\min_d, \max_d) \;\Rightarrow\; \text{kolik} = \left\lfloor \frac{(\min_d - x_{i,d})}{\text{delka}} \right\rfloor$$

$$x_{i,d} = x_{i,d} + (\text{kolik} + 1) \cdot \text{delka}$$

$$v_{i,d} = 0$$

Porovnání jednotlivých realizací podmínek probíhalo celkem na pěti funkcích. Celkem probíhalo 100 opakování pro každou funkci a v rámci každého z opakování byl stanoven maximální počet iterací na 1000. Velikost hejna pro SPSO 2006, resp. SPSO 2011, byla $S = 10 + \left\lfloor 2\sqrt{D} \right\rfloor$, kde D je dimenze definičního oboru dané funkce, resp. $S = 40$.

Z výsledků uvedených v Tabulce 3.1 můžeme vidět, že jednotlivé realizace si v případě SPSO 2006 vedly dost podobně. Nicméně o $0,8$ % bylo lepší periodické rozšíření. Abychom potvrdili výběr právě této metody, budeme zkoumat ještě průměrnou minimální hodnotu dosaženou v rámci oněch 100 opakování (viz. Tabulka

Nejbližší hranice		úspěšnost [%]		
přesnost	funkce	SPSO 2006	SPSO 2011	TRIBES
0,01	Bukin č.6 2D	28	26	16
0,001	Griewank 30D	40	0	93
1e-10	Levy č.13 2D	100	100	98
0,001	Perm 2D	100	100	100
100	Perm 10D	97	0	69
Průměrně		73,4	45,2	75,2
Zrcadlení		úspěšnost [%]		
0,01	Bukin č.6 2D	25	33	18
0,001	Griewank 30D	42	0	92
1e-10	Levy č.13 2D	100	100	100
0,001	Perm 2D	100	100	100
100	Perm 10D	100	0	97
Průměrně		73,4	46,6	81,4
Periodické rozšíření		úspěšnost [%]		
0,01	Bukin č.6 2D	28	39	20
0,001	Griewank 30D	43	0	91
1e-10	Levy č.13 2D	100	100	98
0,001	Perm 2D	100	100	100
100	Perm 10D	100	0	99
Průměrně		74,2	47,8	81,6

Tabulka 3.1: Procentuální úspěšnost jednotlivých realizací podmínek na daných testovacích funkcích. Úspěchem rozumím dosažení požadované přesnosti před překročením 1000 iterací.

Nejbližší hranice		průměrná optimální hodnota		
přesnost	funkce	SPSO 2006	SPSO 2011	TRIBES
0,01	Bukin č.6 2D	0,0227	0,0242	0,0310
0,001	Griewank 30D	0,0154	323,2276	0,0012
10^{-10}	Levy č.13 2D	0,0000	0,0000	0,0000
0,001	Perm 2D	0,0005	0,0006	0,0006
100	Perm 10D	43,9496	$\approx 2 \cdot 10^{11}$	49,6927
Průměrně		8,7976	$\approx 4 \cdot 10^{10}$	9,9451
Zrcadlení		průměrná optimální hodnota		
0,01	Bukin č.6 2D	0,0276	0,0232	0,0275
0,001	Griewank 30D	0,0151	249,1423	0,0013
10^{-10}	Levy č.13 2D	0,0000	0,0000	0,0000
0,001	Perm 2D	0,0005	0,0005	0,0006
100	Perm 10D	87,8749	$\approx 8 \cdot 10^{9}$	86,1030
Průměrně		17,5836	$\approx 1,6 \cdot 10^{9}$	17,2265
Periodické rozšíření		průměrná optimální hodnota		
0,01	Bukin č.6 2D	0,0208	0,0177	0,0262
0,001	Griewank 30D	0,0119	302,7160	0,0014
10^{-10}	Levy č.13 2D	0,0000	0,0000	0,0000
0,001	Perm 2D	13,5324	0,0006	0,0006
100	Perm 10D	100	$\approx 10^{11}$	82,2058
Průměrně		2,7131	$\approx 2 \cdot 10^{10}$	16,4468

Tabulka 3.2: Průměrné minimální hodnoty nalezené během 100 opakování s 1000 iterací v každém z nich.

Algoritmus	Zvolená real. podmínek
SPSO 2006	periodické rozšíření
SPSO 2011	zrcadlení
TRIBES	periodické rozšíření

Tabulka 3.3: Ke každé verzi algoritmu je přiřazena realizace podmínek, která vykazovala nejlepší výsledky.

3.2). V tomto pohledu periodické rozšíření dává o poznání lepší výsledky než zbývající dvě metody a volíme jej tedy jako realizaci omezujících podmínek pro SPSO 2006.

Co se týká SPSO 2011, periodické rozšíření se opět jevilo nejefektivnější. Tentokrát metoda nejbližší hranice zaostávala už o 2,6 %. Nicméně se rozhodneme znovu až po srovnání průměrných minimálních hodnot. Jak lze vidět v Tabulce 3.2, varianta z roku 2011 si v rámci 1000 iterací vedla špatně v porovnání s ostatními průměrnými hodnotami. Každopádně metoda zrcadlení vykazuje o celý řád lepší průměrnou hodnotu než periodické rozšíření, a tedy ji zvolíme jako výchozí realizaci podmínek pro další testování SPSO 2011.

Stejně jako v předchozích dvou případech se periodické rozšíření, co se procentuální úspěšnosti, dostalo na první příčku. Nyní už metoda nejbližší hranice zaostává o celých 6,4 %. I když nejbližší hranice vykazuje nejlepší průměrné hodnoty, procentuální úspěšnost je o dost horší a pro TRIBES budeme tedy volit realizaci omezujících podmínek periodickým rozšířením. Konečný výběr pro všechny tři varianty je zachycen v Tabulce 3.3.

3.2 Výsledky minimalizace testovacích funkcí

Testovací funkce budou pro lepší přehlednost rozděleny podle počtu lokálních extrémů či podle tvaru. Podle počtu extrémů se funkce dělí na *unimodální* a *multimodální*. Některé funkce jsou charakteristické tím, že do globálního minima vede pouze úzká cesta připomínající rokli. Podle podobnosti k určitému typu terénu budou rozčleněny na *rokle*, *roviny*, *údolí*, *prudké skoky* a *ostatní*. Každá z těchto skupin představuje pro optimalizační algoritmus nějakou komplikaci, která může výpočet výrazně ztížit. Předpisy všech použitých testovacích funkcí jsou k dispozici v poslední části této kapitoly, grafy a argumenty minima potom v [15],[27], [26].

3.2.1 Více lokálních minim

Jednou z hlavních vlastností optimalizačního algoritmu musí být schopnost dostat se z lokálního minima. Jako první budou tedy uvedeny výsledky z testování PSO na multimodálních funkcích (viz Tabulka 3.4 a Tabulka 3.6). Nejdříve testy proběhnou na 2D funkcích s vyšší přesností 10^{-10}.

Jak můžeme vidět z Tabulky 3.4, původní verze z roku 1995 není schopná minimalizovat žádnou z daných funkcí s požadovanou přesností 10^{-10}. Na druhou stranu ostatní tři algoritmy fungovaly velmi dobře, až na funkce Eggholder a Langermann. U těchto dvou funkcí nebyla požadovaná přesnost dosažena ani jednou ze sta opakování. Nicméně pokud se omezíme na maximální odchylku 10^{-3} od skutečné hodnoty minima, pro funkci Langermann všechny verze, kromě PSO 1995, fungovaly na 100 % a průměrný počet iterací nutný k dosažení této přesnosti nepřesáhl 100 (viz Tabulka 3.5). Co se týče funkce Eggholder, rapidní zlepšení o 81 % bylo zaznamenáno pouze u verze SPSO 2011. Ostatní varianty se ve valné většině opakování neblížily ke skutečné hodnotě minima ani v řádu jednoho desetinného místa, u PSO 1995 dokonce ani v řádech stovek (viz Tabulka 3.5). Nejlepší výsledky jsme tak pro 2D multimodální funkce dostali od verze SPSO 2011.

Pokud jde o problémy s vyšší dimenzí, kromě varianty TRIBES, všechny tři ostatní algoritmy v rámci 1000 iterací selhávají (viz Tabulka 3.6). Na druhou stranu bezparametrická verze TRIBES funguje výtečně, pouze u funkce Schwefel 10D selhává stejně jako ostatní. Je možné, že ostatním verzím PSO nestačí maximální počet iterací roven 1000. V Tabulce 3.7 můžeme vidět výsledky SPSO 2006 a SPSO 2011 při počtu iterací nastavených na 10^4. Očividně standardní varianta z roku 2006 zlepšila svou úspěšnost téměř u všech funkcí. Pouze minimalizace funkce Rastrigin 20D zůstává i nadále neúspěšná. SPSO 2011 zřejmě konverguje velmi pomalu a ani desetkrát více iterací nepřineslo žádný úspěch. Varianta TRIBES byla z testování pro vyšší počet iterací vynechána, neboť vykazovala velmi dobré výsledky už při 1000 iteracích. Vyzkoušena byla pouze pro Schwefel 10D, nicméně bezúspěšně.

Pro lepší srovnání můžeme v Tabulce 3.7 vidět průměrné minimální hodnoty nalezené při 1000 a 10000 iteracích. I když procentuální úspěšnost pro SPSO 2006 byla při 1000 iteracích téměř stejná jako pro SPSO 2011, je z této tabulky jasně vidět, že variant z roku 2006 sice nedosáhla požadované přesnosti při 1000 iteracích, nicméně s průměrnou minimální hodnotou 0.0364 nebyla od cíle daleko, kdežto SPSO 2011 se s průměrnou hodnotou 693.8188 ani zdaleka nepřibližovala k minimu. Jediný aspekt, v čem SPSO 2011 předčí SPSO 2006, je u funkce Schwefel 10D, kde dosahuje lepších průměrných hodnot, nicméně ve všech ostatních případech vítězí mladší verze.

Funkce	úspěšnost [%]			
	PSO 1995	SPSO 2006	SPSO 2011	TRIBES
Cross-in-Tray	0	100(71.1)	100(58.4)	100(38.6)
Drop-wave	0	97(226.9)	100(184.7)	88
Eggholder	0	0	0	0
Holder Table	0	92(238.1)	100(71.9)	100(106.9)
Langermann	0	0	0	0
Levy č.13	0	100(107.8)	100(170.6)	99(336.9)
Schaffer č.2	0	100(124)	100(123.5)	100(119.5)
Shubert	0	66	92(194.9)	100(97.1)
Průměrně	0	69.375	74	73.375

Tabulka 3.4: Procentuální úspěšnost jednotlivých variant algoritmu PSO při minimalizaci 2D funkcí. Počet opakování je 100, počet iterací 1000 a požadovaná přesnost 10^{-10}. Číslo v závorce představuje průměrný počet iterací při dosažení požadované přesnosti, který je zveřejněn pouze u úspěšnosti větší nebo rovné než 90 %.

Funkce	úspěšnost [%]			
	PSO 1995	SPSO 2006	SPSO 2011	TRIBES
Eggholder	2	0	81	3
Langermann	8	100(68.8)	100(33.9)	99(68.3)
Funkce	průměrná minimální hodnota [%]			
	PSO 1995	SPSO 2006	SPSO 2011	TRIBES
Eggholder	-758.7065	-955.7257	-959.3654	-951.5044

Tabulka 3.5: Procentuální úspěšnost a průměrná minimální hodnota dosažená jednotlivými variantami algoritmu PSO při minimalizaci 2D funkcí. Počet opakování je 100, počet iterací 1000 a požadovaná přesnost 10^{-3}. Číslo v závorce představuje průměrný počet iterací při dosažení požadované přesnosti, který je zveřejněn pouze u úspěšnosti větší nebo rovné než 90 %.

Funkce	úspěšnost [%]			
	PSO 1995	SPSO 2006	SPSO 2011	TRIBES
Ackley 20D	0	71	0	100(191.8)
Ackley 50D	0	0	0	96(402.6)
Griewank 60D	0	2	0	93(365)
Rastrigin 20D	0	0	0	86
Schwefel 10D	0	0	0	0
Průměrně	0	14.6	0	75

Tabulka 3.6: Procentuální úspěšnost jednotlivých variant algoritmu PSO při minimalizaci multimodálních funkcí. Počet opakování je 100, počet iterací 1000 a požadovaná přesnost 10^{-3}. Číslo v závorce představuje průměrný počet iterací při dosažení požadované přesnosti, který je zveřejněn pouze u úspěšnosti větší nebo rovné než 90 %.

Funkce	úspěšnost [%]	
	SPSO 2006	SPSO 2011
Ackley 20D	78	0
Ackley 50D	6	0
Griewank 60D	40	0
Rastrigin 20D	0	0
Schwefel 10D	17	0
Funkce	min. hodnota pro 10000 iterací	
Ackley 20D	0.3371	16.5561
Ackley 50D	1.8050	19.5157
Griewank 60D	0.0364	693.8188
Rastrigin 20D	19.4315	150.2399
Schwefel 10D	196.4255	178.2948
Funkce	min. hodnota pro 1000 iterací	
Ackley 20D	0.3987	17.5221
Ackley 50D	1.9846	19.7671
Griewank 60D	0.0388	758.4831
Rastrigin 20D	21.6978	168.4645
Schwefel 10D	1127.9479	779.6336

Tabulka 3.7: Procentuální úspěšnost dosažená variantami SPSO 2006 a SPSO 2011 při minimalizaci multimodálních funkcí. Počet opakování je 100, počet iterací 10^4 a požadovaná přesnost 10^{-3}. V poslední části tabulky jsou pro srovnání uvedeny hodnoty z minimalizace při 1000 iteracích.

Funkce	dimenze	definiční obor	glob. min.
Ackley	D	$\langle -32.768, 32.768 \rangle^D$	0
Cross-in-Tray	2	$\langle -10, 10 \rangle^2$	-2.06261
Drop-wave	2	$\langle -5.12, 5.12 \rangle^2$	-1
Eggholder	2	$\langle -512, 512 \rangle^2$	-959.6407
Griewank	D	$\langle -600, 600 \rangle^D$	0
Holder Table	2	$\langle -10, 10 \rangle^2$	-19.2805
Langermann	2	$\langle 0, 10 \rangle^2$	-5.162159
Levy č.13	2	$\langle -10, 10 \rangle^2$	0
Rastrigin	D	$\langle -5.12, 5.12 \rangle^D$	0
Schaffer č.2	2	$\langle -100, 100 \rangle^2$	0
Schwefel	D	$\langle -500, 500 \rangle^D$	0
Shubert	2	$\langle -10, 10 \rangle^2$	-186.7309

Tabulka 3.8: Použité testovací funkce, které mají více lokálních minim.

3.2.2 Rokle

Globální minimum nacházející se v „rokli“ může činit optimalizačním algoritmům potíže. Je tak dáno kvůli tomu, že algoritmus se sice velmi rychle dostane do rokle, nicméně je pro něj velmi obtížné se poté pohybovat touto roklí ke globálnímu minimu. Zde budou testovány všechny čtyři varianty na dvou funkcích, kterými jsou Bukin č.6 a Michalewicz (viz Tabulka 3.10).

Původní verze z roku 1995 byla z dalších testů vynechána, neboť se nemůže při dané požadované přesnosti srovnávat s ostatními verzemi. Jak je vidět z Tabulky 3.9, nejlépe si v této třídě funkcí vedla standardní varianta z roku 2006 s průměrným úspěchem 66.5 %. Algoritmus TRIBES za ní ovšem výrazně zaostal až při posledním problému, kde nestačilo ani navýšení počtu iterací na 10^4.

Funkce	úspěšnost [%]			přesnost
	SPSO 2006	SPSO 2011	TRIBES	
Bukin č.6 2D	29	23	22	0.01
Michalewicz 2D	100(47.7)	100(48.8)	100(73.8)	10^{-10}
Michalewicz 5D	37	0	11	10^{-5}
Michalewicz 10D	100(414.7)	0	29	1
Průměrně	66.5	30.75	40.5	

Tabulka 3.9: Procentuální úspěšnost jednotlivých variant algoritmu PSO při minimalizaci funkcí tvaru rokle. Počet opakování je 100, počet iterací 1000, pouze pro Michalewicz 10D je počet iterací 10^4. Číslo v závorce představuje průměrný počet iterací při dosažení požadované přesnosti, který je zveřejněn pouze u úspěšnosti větší nebo rovné než 90 %.

Funkce	**dimenze**	**definiční obor**	**glob. min.**
Michalewicz	D	$\langle 0, \pi \rangle^D$	$D = 2: \quad -1.8013$ $D = 5: \quad -4.687658$ $D = 10: \quad -9.66015$
Bukin č.6	2	$\langle -15, -5 \rangle \times \langle -3, 3 \rangle$	0

Tabulka 3.10: Použité testovací funkce, které svým tvarem ve 2D připomínají rokli.

3.2.3 Roviny

Funkce tvaru rovin mohou výrazně prodloužit konvergenci algoritmu. Na tomto místě budou zveřejněny výsledky variant SPSO 2006, SPSO 2011 a TRIBES z testů na funkcích Zakharov a Booth (viz Tabulka 3.12).

Z výsledků uvedených v Tabulce 3.11 můžeme usoudit, že kromě verze SPSO 2011, si algoritmy vedly výborně. Standardní verzi nepomohlo ani navýšení iterací na 10^4, nicméně ve všech ostatních případech v rámci těchto testů byl stoprocentní úspěch. Funkce rovinného typu tedy algoritmu PSO nedělají, alespoň v tomto případě, příliš velký problém.

Funkce	úspěšnost [%]			přesnost
	SPSO 2006	SPSO 2011	TRIBES	
Zakharov 20D	100(2175.7)	0	100(747.2)	0.001
Booth 2D	100(108.2)	100(113.9)	100(226.2)	10^{-10}

Tabulka 3.11: Procentuální úspěšnost jednotlivých variant algoritmu PSO při minimalizaci funkcí tvaru roviny. Počet opakování je 100. Pro Zakharov 20D je počet iterací 10^4, pro Booth 2D pak 10^3. Číslo v závorce představuje průměrný počet iterací při dosažení požadované přesnosti, který je zveřejněn pouze u úspěšnosti větší nebo rovné než 90 %.

Funkce	dimenze	definiční obor	glob. min.
Zakharov	D	$\langle -5, 10\rangle^D$	0
Booth	2	$\langle -10, 10\rangle^2$	0

Tabulka 3.12: Použité testovací funkce, které svým tvarem ve 2D připomínají rovinu.

3.2.4 Údolí

Dalšími dvěma funkcemi jsou Dixon-Price a Six-Hump Camel (Tabulka 3.15). Svým tvarem připomínají údolí a do globálního minima se „jde“ velmi pomalu. Tento fakt může při hledání extrému způsobovat značné potíže.

Co se týká funkce Dixon-Price, pouze verze SPSO 2006 dokázala najít v osmi případech globální minimum s požadovanou přesností (viz Tabulka 3.13). Ostatním dvěma verzím nepomohlo ani zvýšení iterací na 10^4. Funkci Six-Hump Camel 2D minimalizovali všechny varianty velmi jednoduše i s vysokou přesností, nicméně na více dimenzionální funkci si vedly špatně. Ovšem jak je vidět z Tabulky 3.14, varianta TRIBES i SPSO 2006 by mohly velmi dobře uspět při snížení přesnosti či zvýšení počtu iterací. SPSO 2011 opět vykazuje nejhorší výsledky a ke globálnímu minimu se ani zdaleka nepřibližuje.

Funkce	úspěšnost [%]			přesnost
	SPSO 2006	SPSO 2011	TRIBES	
Dixon-Price 10D	8	0	0	0.1
Six-Hump Camel 2D	100(50.7)	100(44.5)	100(42.8)	10^{-20}

Tabulka 3.13: Procentuální úspěšnost jednotlivých variant algoritmu PSO při minimalizaci funkcí tvaru údolí. Počet opakování je 100. Pro Dixon-Price 10D je počet iterací 10^4, pro Six-Hump Camel 2D pak 10^3. Číslo v závorce představuje průměrný počet iterací při dosažení požadované přesnosti, který je zveřejněn pouze u úspěšnosti větší nebo rovné než 90 %.

Funkce	Dixon-Price 10D		
	SPSO 2006	SPSO 2011	TRIBES
prům. min. hodnota	0.6174	188.2587	0.6667
nejlepší	0.0750	12.7907	0.6667
nejhorší	0.6667	469.7626	0.6667

Tabulka 3.14: Průměrná minimální nalezená hodnota, absolutně nejmenší, resp. největší nalezená hodnota pro jednotlivé varianty algoritmu PSO při minimalizaci funkce Dixon-Price 10D. Počet opakování je 100, počet iterací 10^4, přesnost 0.1.

Funkce	**dimenze**	**definiční obor**	**glob. min.**
Dixon-price	D	$\langle -10, 10\rangle^D$	0
Six-Hump Camel	2	$\langle -3, 3\rangle \times \langle -2, 2\rangle$	-1.0316

Tabulka 3.15: Použité testovací funkce, které svým tvarem ve 2D připomínají údolí.

3.2.5 Prudký skok

Funkce spadající do této třídy jsou typické tím, že velký spád je pouze v malém okolí minima a v okolí vzdálenějším naopak připomínají rovinu. Funkce De Jong č.5 a Easom 2D budou cílem minimalizace v této části (Tabulka 3.17).

Z výsledků v Tabulce 3.16 je jasné, že minimalizace v rámci této třídy nedělala ani jedné z variant větší problémy a to i s požadovanou přesností 10^{-15}. Algoritmus SPSO 2011 dokonce exceloval stoprocentní úspěšností v obou případech a z těchto výsledků lze usoudit, že se hodí nejlépe na minimalizaci funkcí s prudkým skokem.

Funkce	úspěšnost [%]		
	SPSO 2006	SPSO 2011	TRIBES
De Jong č.5 2D	84	100(137.6)	65(358.3)
Easom 2D	100(181.1)	100(200.3)	100(120.5)

Tabulka 3.16: Procentuální úspěšnost jednotlivých variant algoritmu PSO při minimalizaci funkcí tvaru prudkého skoku. Počet opakování je 100, počet iterací 1000, požadovaná přesnost 10^{-15}. Číslo v závorce představuje průměrný počet iterací při dosažení požadované přesnosti, který je zveřejněn pouze u úspěšnosti větší nebo rovné než 90 %.

Funkce	**dimenze**	**definiční obor**	**glob. min.**
De Jong č.5	2	$\langle -65.536, 65.536 \rangle^2$	0.998004
Easom	2	$\langle -100, 100 \rangle^2$	-1

Tabulka 3.17: Použité testovací funkce, které jsou mimo okolí globálního extrému relativně ploché a minimum se nachází v prudké prohlubni.

3.2.6 Ostatní

V poslední části testování budou provedeny testy ještě na dalších třech funkcích (viz Tabulka 3.19), které nijak nezapadly do výše zmíněných kategorií. O funkcích Beale a Styblinski-Tang by se dalo říci, že svým tvarem jsou napůl cesty mezi údolím a rovinou.

Pro 2D funkci Beale všechny tři varianty fungovaly téměř dokonale, nicméně se zvětšující se dimenzí u ostatních dvou funkcí už velmi rapidně klesala úspěšnost nalezení minima (viz Tabulka 3.18). Pro funkci Styblinski-Tang 30D byla stanovena velmi nízká přesnost rovna 100, nicméně ani toto nevedlo k výraznému zlepšení u SPSO 2011 a TRIBES. Pouze standardní verze PSO 2006 byla s to najít minimum v 94 % případech v rámci této přesnosti. Při míře přesnosti jedna, nestačilo ani $5 \cdot 10^4$ iterací pro úspěšné nalezení optima.

Funkce	úspěšnost [%]			přesnost
	SPSO 2006	SPSO 2011	TRIBES	
Perm 10D	31	0	0	1
Beale 2D	100(138.3)	100(174.7)	99(371.6)	10^{-10}
Styblinski-Tang 30D	94(508.5851)	0	5	100

Tabulka 3.18: Procentuální úspěšnost jednotlivých variant algoritmu PSO při minimalizaci funkcí tvaru roviny. Počet opakování je 100, počet iterací 1000. Číslo v závorce představuje průměrný počet iterací při dosažení požadované přesnosti, který je zveřejněn pouze u úspěšnosti větší nebo rovné než 90 %.

Funkce	**dimenze**	**definiční obor**	**glob. min.**
Perm	D	$\langle -D, D\rangle^D$	0
Beale	2	$\langle -4.5, 4.5\rangle^2$	0
Styblinski-Tang	D	$\langle -5, 5\rangle^D$	$-D \cdot 39.16599$

Tabulka 3.19: Ostatní použité testovací funkce.

3.3 Předpisy testovacích funkcí

1. Ackley: $f(\vec{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi x_i\right)\right) + 20 + \mathrm{e}$
2. Cross-in-Tray: $f(\vec{x}) = -10^{-4}\left(\left|\sin\left(x_1\right)\sin\left(x_2\right)\exp\left(\left|100 - \frac{\sqrt{x_1^2+x_2^2}}{\pi}\right|\right)\right| + 1\right)^{0.1}$

3. Drop-wave: $f(\vec{x}) = -\dfrac{1+\cos\left(12\sqrt{x_1^2+x_2^2}\right)}{0.5(x_1^2+x_2^2)+2}$

4. Eggholder: $f(\vec{x}) = -(x_2+47)\sin\left(\sqrt{\left|x_2+\frac{x_1}{2}+47\right|}\right) - x_1\sin\left(\sqrt{|x_1-(x_2+47)|}\right)$

5. Griewank: $f(\vec{x}) = \sum_{i=1}^{D}\dfrac{x_i^2}{4000} - \prod_{i=1}^{D}\cos\left(\dfrac{x_i}{\sqrt{i}}\right) + 1$

6. Holder Table: $f(\vec{x}) = -\left|\sin(x_1)\cos(x_2)\exp\left(\left|1-\dfrac{\sqrt{x_1^2+x_2^2}}{\pi}\right|\right)\right|$

7. Langermann: $f(\vec{x}) = \sum_{i=1}^{5} c_i \exp\left(-\frac{1}{\pi}\sum_{j=1}^{D}(x_j - A_{ij})^2\right)\cos\left(\pi\sum_{j=1}^{D}(x_j-A_{ij})^2\right)$, kde $\vec{c} = (1,2,5,2,3)$ a $\mathbb{A} = \begin{pmatrix} 3 & 5 \\ 5 & 2 \\ 2 & 1 \\ 1 & 4 \\ 7 & 9 \end{pmatrix}$

8. Levy č.13: $f(\vec{x}) = \sin^2(3\pi x_1) + (x_1-1)^2\left(1+\sin^2(3\pi x_2)\right) + (x_2-1)^2\left(1+\sin^2(2\pi x_2)\right)$

9. Rastrigin: $f(\vec{x}) = 10D + \sum_{i=1}^{D}\left(x_i^2 - 10\cos(2\pi x_i)\right)$

10. Schaffer č.2: $f(\vec{x}) = 0.5 + \dfrac{\sin^2(x_1^2-x_2^2)-0.5}{\left(1+0.001(x_1^2+x_2^2)\right)^2}$

11. Schwefel: $f(\vec{x}) = 418.9829D - \sum_{i=1}^{D} x_i \sin\left(\sqrt{|x_i|}\right)$

12. Shubert: $f(\vec{x}) = \left(\sum_{i=1}^{5} i\cos((i+1)x_1+i)\right)\left(\sum_{i=1}^{5} i\cos((i+1)x_2+i)\right)$

13. Bukin č.6: $f(\vec{x}) = 100\sqrt{|x_2 - 0.01x_1^2|} + 0.01|x_1+10|$

14. Michalewicz: $f(\vec{x}) = -\sum_{i=1}^{D}\sin(x_i)\sin^{2m}\left(\dfrac{ix_i^2}{\pi}\right)$

15. Zakharov: $f(\vec{x}) = \sum_{i=1}^{D} x_i^2 + \left(\sum_{i=1}^{D} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{D} 0.5ix_i\right)^4$

16. Booth: $f(\vec{x}) = (x_1+2x_2-7)^2 + (2x_1+x_2-5)^2$

17. Dixon-Price: $f(\vec{x}) = (x_1-1)^2 + \sum_{i=2}^{D} i(2x_i^2 - x_{i-1})^2$

18. Six-Hump Camel: $f(\vec{x}) = \left(4 - 2.1x_1^2 + \dfrac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (x_2^2-4)x_2^2$

19. De Jong č.5: $f(\vec{x}) = \left(0.002 + \sum_{i=1}^{25} \frac{1}{i + (x_1 - a_{1i})^6 + (x_2 - a_{2i})^6}\right)^{-1}$, kde
$\vec{a}_1 = (-32, -16, 0, 16, 32, -32, -16, \ldots, 0, 16, 32)$ a
$\vec{a}_2 = (-32, -32, -32, -32, -32, -16, -16, -16, \ldots, 32, 32, 32)$

20. Easom: $f(\vec{x}) = -\cos(x_1)\cos(x_2)\exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$

21. Perm: $f(\vec{x}) = \sum_{i=1}^{D} \left(\sum_{j=1}^{D} (j + 10)\left(x_j^i - \frac{1}{j^i}\right)\right)^2$

22. Beale: $f(\vec{x}) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2$

23. Styblinski-Tang: $f(\vec{x}) = \frac{1}{2}\sum_{i=1}^{D}(x_i^4 - 16x_i^2 + 5x_i)$

Závěr

V této práci byly popsány celkem čtyři varianty algoritmu optimalizace hejnem částic. Těmi jsou původní verze PSO 1995, dvě standardní SPSO 2006 a 2011 a bezparametrická TRIBES. Po popisu byly všechny postupně implementovány a testovány. Cílem testování bylo nejprve stanovit nejlépe vyhovující realizaci omezujících podmínek pro každou z variant. Pro standardní verzi z roku 2006 bylo zvoleno *periodické rozšíření*, které předčilo ostatní dvě metody zejména díky lepší průměrné minimální nalezené hodnotě. Pro standardní variantu SPSO 2011 byla vybrána metoda *zrcadlení*, která sice mírně zaostávala v procentuální úspěšnosti, ale v případě minimalizace funkce Perm 10D vracela o celý řád lepší výsledky. Nakonec u verze TRIBES zvítězila realizace podmínek *periodickým rozšířením*, která měla porovnatelnou procentuální úspěšnost s metodou *zrcadlení*, ale vykazovala o kousek lepší průměrné minimální hodnoty. U standardní verze z roku 1995 byla pro autentičnost ponechána metoda *nejbližší hranice*. Po stanovení metod pro obsluhu podmínek byly jednotlivé algoritmy testovány na celkem na třiadvaceti funkcích, které byly podle určitých kritérií rozřazeny do šesti skupin. Mimo doporučení z výsledků testování, může tato práce sloužit také jako návod pro implementaci algoritmů PSO 1995, SPSO 2006, SPSO 2011 a TRIBES.

V rámci první skupiny, do které spadaly *multimodální* problémy, měly při minimalizaci 2D funkcí verze SPSO 2011 a TRIBES srovnatelné procentuální úspěšnosti (74 % a 73.375 %). Každopádně při přechodu na vyšší dimenze úspěšnost verze SPSO 2011 rapidně klesla, ale verze TRIBES si vedla nadále velmi dobře se 75% úspěšností, a tedy se nejlépe hodí na minimalizaci těchto funkcí. I když původní verze PSO 1995 tvoří důležitý stavební kámen pro všechny její další modifikace, neměla šanci se porovnávat s novějšími verzemi v rámci dané přesnosti a počtu iterací, a proto byla z dalších testů vynechána.

Vítěznou variantou se ve druhé skupině stala standardní verze z roku 2006. S úspěšností 65 % předčila TRIBES o celých 26 % v hledání globálního minima u funkcí tvaru *rokle*. Na posledním místě zůstal algoritmus SPSO 2011, který není vhodný pro řešení takovýchto problémů.

Třetí skupina, *roviny*, nepředstavovala většinou velký problém. Pouze SPSO 2011 si nedokázala poradit s vícerozměrným problémem Zakharov 20D. Jak SPSO 2006, tak TRIBES nalezli minimální hodnotu v rámci přesnosti se 100% úspěšností. Každopádně pro tuto skupinu musí být doporučena verze TRIBES, neboť ve vyšších dimenzích potřebovala pro nalezení minima pouze asi jednu třetinu iterací, které musela vykonat SPSO 2006 pro úspěšný běh.

Pro funkce typu *údolí*, které tvořily čtvrtou třídu funkcí při testování, by pro 2D případ mohly být doporučeny všechny tři varianty. Nicméně pokud se dimenze problému zvětšila, ani jeden z algoritmů nebyl schopen dosáhnout dostatečné úspěšnosti. Pouze SPSO 2006 našla řešení v osmi procentech případů. Pokud by ale požadovaná přesnost mohla být o něco vyšší, pak by tato standardní verze minimalizovala funkci Dixon-Price celkem uspokojivě. Každopádně pro přesnost vyšší by měl být pro řešení tohoto problému úplně jiná varianta PSO či algoritmus jako takový.

Ani jedna z variant neměla větší problém minimalizovat 2D funkce s vysokou přesností v páté skupině. Specifický tvar *prudkého skoku* příliš nevadil žádné z verzí, je ale třeba vyzdvihnout standardní variantu z roku 2011, která optimalizovala oba problémy se 100% úspěšností.

Na závěr byly provedeny testy pro další tři funkce, které nebyly zařazeny do žádné ze zmiňovaných tříd. Opět se ukázalo, že 2D funkce není pro žádnou z variant velká překážka. Na druhou stranu funkce Perm 10D a Styblinski-Tang 30D nebyly uspokojivě minimalizovány a je potřeba zvolit jiný optimalizační algoritmus pro hledání jejich globálních extrémů.

Celkově se dá říci, že až na pár výjimek alespoň jedna z verzí SPSO 2006, SPSO 2011 a TRIBES byla schopná optimalizovat daný problém z různorodého spektra testovacích funkcí, a je tedy možné využít doporučený algoritmus pro jednotlivé skupiny funkcí k řešení nového problému.

Literatura

[1] BAGHERZADEH, N., HEIDARI M. a AKBARZADEH-T, M.-R. A Greedy Cluster-Based Tribes Optimization Algorithm. *2014 International Congress on Technology, Communication and Knowledge (ICTCK)* [online]. IEEE, 2014, 2014, , 1-4 [cit. 2018-06-23]. DOI: 10.1109/ICTCK.2014.7033525. ISBN 978-1-4799-8021-5. Dostupné z: http://ieeexplore.ieee.org/document/7033525/

[2] CLERC, M. *Particle Swarm Optimization*. 1. Newport Beach: ISTE, 2006. ISBN 978-1-905209-04-0.

[3] CLERC, M. Stagnation Analysis in Particle Swarm Optimization or What Happens When Nothing Happens [online]. 2006, 17 [cit. 2018-06-24]. Dostupné z: https://hal.archives-ouvertes.fr/hal-00122031/

[4] CLERC, M. *Standard Particle Swarm Optimization*, s. 15, 2012. Dostupné z: https://hal.archives-ouvertes.fr/hal-00764996

[5] CLERC, M. Tribes, ou la coopération de tribus. *Séminaire OEP* [online]. 2003, 14 [cit. 2018-06-23]. Dostupné z: http://www.particleswarm.info/oep_ 2003/

[6] COSCIA, D. R. *The Volume of the N-Sphere*. Suffolk County Community College, Selden, New York, 1973. Dostupné také z: https://www.farmingdale.edu/faculty/donald-coscia/

[7] EBERHART, R. a KENNEDY, J. A new optimizer using particle swarm theory. *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science* [online]. IEEE, 1995, , 39-43 [cit. 2018-06-22]. DOI: 10.1109/MHS.1995.494215. ISBN 0-7803-2676-8. Dostupné z: http://ieeexplore.ieee.org/document/494215/

[8] HELWIG, S. a WANKA, R. Particle Swarm Optimization in High-Dimensional Bounded Search Spaces. *2007 IEEE Swarm Intelligence Symposium* [online]. IEEE, 2007, 2007, , 198-205 [cit. 2018-06-24]. DOI: 10.1109/SIS.2007.368046. ISBN 1-4244-0708-7. Dostupné z: http://ieeexplore.ieee.org/document/4223175/

[9] HEPPNER, F. H. a GRENANDER, U. *A Stochastic Nonlinear Model for Coordinate Bird Flocks* [online]. In: . 1990, s. 233-238 [cit. 2018-06-22].

[10] KENNEDY, J. a EBERHART, R. Particle Swarm Optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks* [online]. IEEE,

1995, 1942-1948 [cit. 2017-12-26]. DOI: 10.1109/ICNN.1995.488968. ISBN 0-7803-2768-3. Dostupné z: http://ieeexplore.ieee.org/document/488968/

[11] KENNEDY, J. a MENDES, R. Population Structure and Particle Swarm Performance. In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)* [online]. IEEE, 2002, s. 1671-1676 [cit. 2018-06-22]. DOI: 10.1109/CEC.2002.1004493. ISBN 0-7803-7282-4. Dostupné z: http://ieeexplore.ieee.org/document/1004493/

[12] KRBÁLEK, M. *Matematická analýza III.* 3., přeprac. vyd. V Praze: České vysoké učení technické, 2011. ISBN 978-80-01-04863-4.

[13] LONES, M. A. Metaheuristics in Nature-Inspired Algorithms. In: *Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion - GECCO Comp '14* [online]. New York, New York, USA: ACM Press, 2014, 2014, s. 1419-1422 [cit. 2018-06-26]. DOI: 10.1145/2598394.2609841. ISBN 9781450328814. Dostupné z: http://dl.acm.org/citation.cfm?doid=2598394.2609841

[14] MILLONAS, M. *Swarms, Phase Transitions, and Collective Intelligence; and a Nonequilibrium Statistical Field Theory of Swarms and Other Spatially Extended Complex Systems.* Santa Fe Institute, 1993. SFI Working Paper. Santa Fe Institute.

[15] MOLGA, M. a SMUTNICKI, C. *Test Functions for Optimization Needs* [online]. Politechnika Wroclawska, 2005, 43 [cit. 2018-06-23]. Dostupné z: http://new.zsd.iiar.pwr.wroc.pl/down.php?option=com_ content& view=category& layout=blog& id=45& Itemid=54& lang=pl

[16] MONSON, C. K. a SEPPI, K. D. Exposing origin-seeking bias in PSO. *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05* [online]. New York, New York, USA: ACM Press, 2005, 2005, 7, 241-248 [cit. 2018-06-23]. DOI: 10.1145/1068009.1068045. ISBN 1595930108. Dostupné z: http://portal.acm.org/citation.cfm?doid=1068009.1068045

[17] PARSOPOULOS, K. E. a VRAHATIS, M. N. *Particle Swarm Optimization and Intelligence: Advances and Applications.* Hershey, PA: Information Science Reference, 2010. ISBN 978-1-61520-666-7.

[18] PARSOPOULOS, K. E. a VRAHATIS, M. N. Recent Approaches to Global Optimization Problems through Particle Swarm Optimization. *Natural Computing.* Nizozemsko: Kluwer Academic Publishers, 2002, 1, 235-306.

[19] POLI, R. An Analysis of Publications on Particle Swarm Optimization. *Technical Report CSM-469.* 2007, 57. ISSN 1744-8050.

[20] POLYA, G. *How to Solve it: A New Aspect of Mathematical Method.* 2d ed. Princeton, N.J.: Princeton University Press, 1957. ISBN 06-910-8097-6.

[21] REYNOLDS, C. W. Flocks, Herds and Schools: A Distributed Behavioral Model. In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH '87* [online]. New York, New York, USA: ACM Press, 1987, 1987, s. 25-34 [cit. 2018-06-24]. DOI: 10.1145/37401.37406. ISBN 0897912276. Dostupné z: http://portal.acm.org/citation.cfm?doid=37401.37406

[22] RUBINSTEIN, R. Y. a KROESE, D. P. *Simulation and the Monte Carlo Method*. 2. Hoboken, New Jersey: Jonh Wiley, 2008. ISBN 978-0-470-17794-5.

[23] SEDIGHIZADEH, D. a MASEHIAN, E. Particle Swarm Optimization Methods, Taxonomy and Applications. *International Journal of Computer Theory and Engineering*. 2009, 1(5), 486-502.

[24] SMAIRI, N. *Optimisation par essaim particulaire: adaptation de tribes à l'optimisation multiobjectif*. Français, 2013. Disertační práce. Université Paris-Est.

[25] SPEARS, W. M., GREEN, D. T. a SPEARS. D. F. Biases in Particle Swarm Optimization. *International Journal of Swarm Intelligence Research* [online]. 2010, 1(2), 34-57 [cit. 2018-06-24]. DOI: 10.4018/jsir.2010040103. ISSN 1947-9263. Dostupné z: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/jsir.2010040103

[26] SURJANOVIC, S., BINGHAM, D. Optimization Test Problems. *Virtual Library for Simulation Experiments: Test Functions and Datasets* [online]. Simon Fraser University, 2103 [cit. 2018-06-27]. Dostupné z: https://www.sfu.ca/ ssurjano/optimization.html

[27] TVRDÍK, J. a KŘIVÝ I. Simple Evolutionary Heuristics for Global Optimization. *Computational Statistics and Data Analysis* [online]. 1999, (30), 345-352 [cit. 2018-06-23]. Dostupné z: http://www1.osu.cz/ tvrdik/?page_ id=18

[28] VILENKIN, N. J. *Translations of Mathematical Monographs: Special Functions and the Theory of Group Representation*. 1. Providence, Rhode Island: American Mathematical Society, 1968. ISBN 978-0821815724.

Příloha A

Zdrojový kód PSO 1995

```
function [opt,fitness,PresentX,iter] = ...
    PSO_1995(fcn, D, B, max_iter, S, max_error, glob_min)

%PSO_1995 Hledá globální optimum (minimum) zadané funkce.
%
%[OPT, FITNESS, PRESENTX, ITER]
%   OPT Dosažená nejlepší hodnota účelové funkce.
%   FITNESS Souřadnice odpovídající nejlepšímu fitness
%           v rámci celého hejna.
%   PRESENTX Matice s poslední dosaženou polohou celého hejna.
%   ITER Iterace potřebné pro dosažení požadované přesnosti.
%
%PSO_1995(FCN,D)
%   FCN účelová funkce
%   D dimenze prohledávaného prostoru
%   B = [-10 10]^D
%   MAX_ITER = 1000
%   S = 20
%
%PSO_1995(FCN,D,B)
%   B matice Dx2 reprezentující prohledávaný prostor
%
%PSO_1995(FCN,D,B,MAX_ITER)
%   MAX_ITER maximální počet iterací
%
%PSO_1995(FCN,D,B,MAX_ITER,S)
%   S velikost hejna, počet agentů
%
%PSO_1995(FCN,D,B,MAX_ITER,S,MAX_ERROR,GLOB_MIN)
%   MAX_ERROR   požadovaná přesnost.
%   GLOB_MIN    skutečná hodnota minimalizované funkce pro testování
%               algoritmu
if(nargin <7)
```

```
    glob_min = -inf;
    if(nargin < 6)
        max_error = +inf;
        if(nargin < 5)
            S = 20; %velikost hejna
            if(nargin < 4)
                max_iter = 1000;
                if(nargin < 3)
                    B = repmat([-10 10],D,1);
                end
            end
        end
    end
end
```

Inicializace

```
PresentX = repmat(B(:,1),1,S) + ...
          (repmat(B(:,2),1,S) - repmat(B(:,1),1,S)).*rand(D,S);
V = zeros(D,S);
ACC_CONST = 2.0;
VMAX = 10.0;
PBESTx = PresentX;
PBEST = fcn(PBESTx);
[~, GBEST] = min(PBEST);
```

Iterace

```
iter = 0;
for i = 1:max_iter
    current_value = fcn(PresentX);
    for j = 1:S
        if(current_value(j) < PBEST(j))
            PBEST(j) = current_value(j);
            PBESTx(:,j) = PresentX(:,j);
            if(current_value(j) < PBEST(GBEST))
                GBEST = j;
            end
        end
    end
    V = V + ACC_CONST*rand(D,S).*(PBESTx - PresentX) + ...
          + ACC_CONST*rand(D,S).*(repmat(PBESTx(:,GBEST),1,S) - PresentX);
    V(V > VMAX) = VMAX;
    V(V < -VMAX) = -VMAX;
    PresentX = PresentX + V;
```

```
    % Zajištění toho, že agenti nevylétnou z prohledávaného prostoru
    lb = B(:,1);
    rb = B(:,2);
    for n = 1:D
        px = PresentX(n,:);
        px(px < lb(n)) = lb(n);
        px(px > rb(n)) = rb(n);
        PresentX(n,:) = px;
    end
    if(PBEST(GBEST) < glob_min + max_error)
        iter = i;
        break;
    end
end
fitness = PBESTx(:,GBEST);
opt = PBEST(GBEST);
```

Příloha B

Zdrojový kód SPSO 2006

Obsah

- Inicializace
- Iterace
- Adaptivní náhodná topologie
- Aktualizace AN topologie
- Aktualizace fitness okolí
- Generování hejna
- Nastavení počáteční rychlosti
- Realizace omezujících podmínek
- Aktualizace osobní fitness

```
function [opt,fitness,X,iter] = SPSO_2006(fcn, D, B,...
    realizace_podminek, max_iter, S, presnost, glob_min)


%SPSO_2006 Hledá globální optimum (minimum) zadané funkce.
%
%[OPT, FITNESS, X, ITER]
%   OPT Dosažená nejlepší hodnota účelové funkce.
%   FITNESS Souřadnice odpovídající nejlepšímu fitness
%           v rámci celého hejna.
%   X Matice s poslední dosaženou polohou celého hejna.
%   ITER Iterace potřebné pro dosažení požadované přesnosti.
%
%SPSO_2006(FCN,D)
%   FCN účelová funkce
%   D dimenze prohledávaného prostoru
%   B = [-10 10]^D
%   REALIZACE_PODMINEK = 'nejblizsi_hranice'
%   MAX_ITER = 1000
%   S = 10 + floor(2*sqrt(D))
%
```

```
%SPSO_2006(FCN,D,B)
%   B matice Dx2 reprezentující prohledávaný prostor
%
%SPSO_2006(FCN,D,B,REALIZACE_PODMINEK)
%   REALIZACE_PODMINEK Řetězec specifikující, jakým způsobem se budou
%   podmínky realizovat. Může nabývat následujících hodnot:
%
%       'nejblizsi_hranice'     Přiřazuje agenta na nejbližší hranici
%                               prohledávaného prostoru.
%       'zrcadleni'             Zrcadlí agenta do prohledávaného prostoru.
%       'periodicke_rozsireni'  Posunuje agenta o násobky délky příslušného
%                               intervalu z prohledávaného prostoru do
%                               prohledávaného prostoru.
%
%SPSO_2006(FCN,D,B,REALIZACE_PODMINEK,MAX_ITER)
%   MAX_ITER maximální počet iterací
%
%SPSO_2006(FCN,D,B,REALIZACE_PODMINEK,MAX_ITER,S)
%   S velikost hejna, počet agentů
%
%SPSO_2006(FCN,D,B,REALIZACE_PODMINEK,MAX_ITER,S,PRESNOST,GLOB_MIN)
%   PRESNOST    požadovaná maximální odchylka od skutečného minima.
%   GLOB_MIN    skutečná hodnota minimalizované funkce pro testování
%               algoritmu
```

Inicializace

```
if(nargin <8)
    glob_min = -inf;
    if(nargin < 7)
        presnost = +inf;
        if(nargin < 6)
            S = 10 + floor(2*sqrt(D)); %velikost hejna
            if(nargin < 5)
                max_iter = 1000;
                if(nargin < 4)
                    realizace_podminek = 'nejblizsi_hranice';
                    if(nargin < 3)
                        B = repmat([-10 10],D,1);
                    end
                end
            end
        end
    end
end
%rng(0,'twister'); %inicializace generátoru náhodných čísel
```

```
N = adaptivni_nahodna_topologie(S);
X = generuj_hejno(D,S,B);
V = nastav_rychlost(D,S,B,X,2006);
P = X; %matice P nese fitness každého agenta
p_hodnoty = fcn(X); %hodnota soucasneho osobniho fitness
L = aktualizuj_fitness_okoli(D,S,N,P,p_hodnoty);
iter = 0;%iterace potřebné pro dosažení požadované přesnosti
```

Iterace

```
w = 1/(2*log(2));
c = 1/2 + log(2);
for i = 1:max_iter
    U1 = c*rand(D,S);
    U2 = c*rand(D,S);
    V = w*V + U1.*(P - X) + U2.*(L - X);
    X = X + V;
    [V, X] = obsluz_podminky(D,S,B,X,V,realizace_podminek);
    [P, p_hodnoty, neni_zmena] = prepocitej_fitness(X,P,p_hodnoty,fcn);
    if(~isempty(neni_zmena))
        %tam, kde není změna, změň informované okolí
        N = aktualizuj_AN_topologii(S, neni_zmena, N);
    end
    L = aktualizuj_fitness_okoli(D,S,N,P,p_hodnoty);
    if(min(p_hodnoty) < glob_min + presnost)
        iter = i;
        break;
    end
end
[opt, idxFitness] = min(p_hodnoty);
fitness = P(:,idxFitness);


end
```

Adaptivní náhodná topologie

```
function [N] = adaptivni_nahodna_topologie(S, K)
%ADAPTIVNI_NAHODNA_TOPOLOGIE   Vytvoří matici reprezentující
%vztahy mezi agenty. Matice N reprezentuje adaptivní náhodnou topologii.
%
%[N]
%   N Vrací matici reprezentující adaptivní náhodnou topologii.
%
%ADAPTIVNI_NAHODNA_TOPOLOGIE(S,K)
%   S Velikost hejna
```

```
%   K Parametr topologie, určuje kolik maximálně agentů bude jeden
%     agent informovat.

if(nargin < 3)
    K = 3;
end
N = diag(1:S); %na diagonále jsou vždy čísla
for i = 1:S
    pocet = randi([0,K]); %náhodný počet agentů, které agent informuje
    if(pocet > 0)
        %náhodní agenti, kteří budou informováni
        informuj = randi([1,S],1,pocet);
        N(i,informuj) = i;
    end
end
end
```

Aktualizace AN topologie

```
function [N] = aktualizuj_AN_topologii(S,neni_zmena,N,K)
%AKTUALIZUJ_AN_TOPOLOGII Aktualizuje matici reprezentující vztahy mezi agenty.
%  Matice N reprezentuje adaptivní náhodnou topologii.
%AKTUALIZUJ_AN_TOPOLOGII(S,NENI_ZMENA,N,K)
%   S Velikost hejna
%   NENI_ZMENA Indexy agentů, u kterých nenastala změna fitness.
%   N Matice současných vztahů mezi agenty.
%   K Parametr adaptivní náhodné topologie.

if(nargin < 4)
    K = 3;
end
for i = neni_zmena
    N(i,:) = 0;%nastavení upravovaného řádku na nulový vektor
    N(i,i) = i;%doplnění diagonály
    pocet = randi([0,K]); %náhodný počet agentů, které agent informuje
    if(pocet > 0)
     %náhodní agenti, kteří budou informováni
        informuj = randi([1,S],1,pocet);
        N(i,informuj) = i;
    end
end
end
```

Aktualizace fitness okolí

```
function [L, g_hodnoty] = aktualizuj_fitness_okoli(D, S, N, P, p_values)
%AKTUALIZUJ_FITNESS_OKOLI  Každému agentovi přiřazuje nejlepší fitness,
%které se nachází v jeho okolí.
%
%[L]
%   Vrací matici L, která představuje fitness okolí všech agentů hejna.
%
%AKTUALIZUJ_FITNESS_OKOLI(D, S, N, p_values)
%   D dimenze
%   S velikost hejna
%   N matice popisující topologii
%   P_VALUES hodnoty odpovídající osobnímu fitness každé částice

L = zeros(D,S);
g_hodnoty = zeros(1,S);
for i = 1:S
    okoli = find(N(:,i));%indexy agentů v okolí i-té částice
    [g,idx] = min(p_values(okoli));%index v pouze v rámci okolí
    idxMin = okoli(idx);%index částice s nejmenším fitness
    L(:,i) = P(:,idxMin);
    g_hodnoty(i) = g;
end
end
```

Generování hejna

```
function X = generuj_hejno(D,S,B)
%GENERUJ_HEJNO Inicializuje hejno agentů.
%
%[X]
%   Vrací matici X o rozměru DxS, která reprezentuje hejno částic.
%
%GENERUJ_HEJNO(D,S,B)
%   D dimenze
%   S velikost hejna
%   B prohledávaný prostor

X = repmat(B(:,1),1,S) + (repmat(B(:,2),1,S) - ...
                  repmat(B(:,1),1,S)).*rand(D,S);
end
```

Nastavení počáteční rychlosti

```
function V = nastav_rychlost(D,S,B,X,verze)
%NASTAV_RYCHLOST Nastaví počáteční rychlost každého agenta.
%
%[V]
%   Vrací matici V o rozměru DxS, která nese informace o rychlosti každého
%   agenta.
%
%NASTAV_RYCHLOST(D,S,B,X)
%   D dimenze
%   S velikost hejna
%   B prohledávaný prostor
%   X hejno

switch verze
    case 2006
        V = (repmat(B(:,1),1,S) + (repmat(B(:,2),1,S) - ...
             repmat(B(:,1),1,S)).*rand(D,S) - X)/2;
    case 2011
        V =  repmat(B(:,1),1,S) + (repmat(B(:,2),1,S) - ...
             repmat(B(:,1),1,S)).*rand(D,S) - X;
end
end
```

Realizace omezujících podmínek

```
function [V,X] = obsluz_podminky(D,S,B,X,V,realizace_podminek)
%OBSLUZ_PODMINKY Upravuje polohu agentů tak, aby nevyletěli z
%prohledávaného prostoru.
%
%[V,X]
%   V Modifikovaná matice rychlostí.
%   X Modifikované polohy agentů.
%
%OBSLUZ_PODMINKY(D,S,B,X,V,OBSLUHA_PODMINEK)
%   D Dimenze
%   S Velikost hejna
%   B Prohledávaný prostor
%   X Aktuální pozice hejna
%   V Matice rychlostí hejna.
%   OBSLUHA_PODMINEK Řetězec specifikující, jakým způsobem se budou
%   podmínky obsluhovat. Může nabývat následujících hodnot:
%
%       'nejblizsi_hranice'     Přiřazuje agenta na nejbližší hranici
%                               prohledávaného prostoru.
```

```
%       'zrcadleni'             Zrcadlí agenta do prohledávaného prostoru.
%       'periodicke_rozsireni'  Posunuje agenta o násobky délky příslušného
%                               intervalu z prohledávaného prostoru do
%                               prohledávaného prostoru.

switch realizace_podminek
    case 'nejblizsi_hranice'
        for i = 1:S
            for j = 1:D
                if(X(j,i) < B(j,1))
                    X(j,i) = B(j,1);
                    V(j,i) = 0;
                elseif(X(j,i) > B(j,2))
                    X(j,i) = B(j,2);
                    V(j,i) = 0;
                end
            end
        end
    case 'zrcadleni'
        d = B(:,2) - B(:,1);%délky intervalů
        for i = 1:S
            for j = 1:D
                if(X(j,i) < B(j,1))
                    %násobky délky intervalu d
                    kolik = floor((B(j,1) - X(j,i))/d(j));
                    if(mod(kolik,2) == 0)%analogicky jako níže
                        X(j,i) = B(j,1) + ((B(j,1) - kolik*d(j)) - X(j,i));
                    else
                        X(j,i) = B(j,2) - ((B(j,1) - kolik*d(j)) - X(j,i));
                    end
                    V(j,i) = 0;
                elseif(X(j,i) > B(j,2))
                    kolik = floor((X(j,i) - B(j,2))/d(j));
                    if(mod(kolik,2) == 0)
                        %pokud je sudý násobek,
                        %odečítá se od pravého kraje intervalu
                        X(j,i) = B(j,2) - (X(j,i) - (B(j,2) + kolik*d(j)));
                    else%pokud je lichý, přičítá se k levé straně
                        X(j,i) = B(j,1) + (X(j,i) - (B(j,2) + kolik*d(j)));
                    end
                    V(j,i) = 0;
                end
            end
        end
    case 'periodicke_rozsireni'
        d = B(:,2) - B(:,1);
        for i = 1:S
```

```
            for j = 1:D
                if(X(j,i) < B(j,1))
                    %násobky délky intervalu d
                    kolik = floor((B(j,1) - X(j,i))/d(j));
                    X(j,i) = X(j,i) + (kolik+1)*d(j);
                    V(j,i) = 0;
                elseif(X(j,i) > B(j,2))
                    kolik = floor((X(j,i) - B(j,2))/d(j));
                    X(j,i) = X(j,i) - (kolik+1)*d(j);
                    V(j,i) = 0;
                end
            end
        end
    otherwise
        error('Byla zadána neplatná obsluha podmínek!')
end
end
```

Aktualizace osobní fitness

```
function [P, p_values, neni_zmena, fx] = prepocitej_fitness(X,P,p_values,fcn)
%PREPOCITEJ_FITNESS Aktualizuje osobní fitness a odpovídající hodnoty
%účelové funkce. Také kontroluje, u kterých agentů se osobní fitness
%nezměnilo, a tedy je nutné na to reagovat v aktualizaci okolí částic.
%
%[P, P_VALUES, NENI_ZMENA]
%   P Matice s fitness celého hejna.
%   P_VALUES Vektor hodnot, které odpovídají fitness jednotlivých agentů.
%   NENI_ZMENA Vektor indexů, které odpovídají agentům, u kterých nedošlo
%   ke zlepšení fitness. Pouze pro verzi SPSO 2006 a 2011
%
%PREPOCITEJ_FITNESS(X,P,P_VALUES,FCN)
%   X hejno
%   P matice se současnými fitness
%   P_VALUES vektor se současnými hodnotami účelové funkce ve fitness
%   FCN účelová funkce

fx = fcn(X);
mensi = fx < p_values;
neni_zmena = find(~mensi);
if(sum(mensi) ~= 0)
    kde = find(mensi);
    p_values(kde) = fx(kde);
    P(:,kde) = X(:,kde);
end
end
```

Příloha C

Zdrojový kód SPSO 2011

Obsah

- Inicializace
- Iterace
- Generování bodů v kouli

```
function [opt,fitness,X,iter] = SPSO_2011(fcn, D, B, realizace_podminek,...
                                    max_iter, S, presnost, glob_min)


%SPSO_2011 Hledá globální optimum (minimum) zadané funkce.
%
%[OPT, FITNESS, X, ITER]
%   OPT Dosažená nejlepší hodnota účelové funkce.
%   FITNESS Souřadnice odpovídající nejlepšímu fitness v rámci celého hejna.
%   X Matice s poslední dosaženou polohou celého hejna.
%   ITER Iterace potřebné pro dosažení požadované přesnosti.
%
%SPSO_2011(FCN,D)
%   FCN účelová funkce
%   D dimenze prohledávaného prostoru
%   B = [-10 10]^D
%   REALIZACE_PODMINEK = 'nejblizsi_hranice'
%   MAX_ITER = 1000
%   S = 10 + floor(2*sqrt(D))
%
%SPSO_2011(FCN,D,B)
%   B matice Dx2 reprezentující prohledávaný prostor
%
%SPSO_2011(FCN,D,B,REALIZACE_PODMINEK)
%   REALIZACE_PODMINEK Řetězec specifikující, jakým způsobem se budou
%   podmínky realizovat. Může nabývat následujících hodnot:
%
```

```
%       'nejblizsi_hranice'     Přiřazuje agenta na nejbližší hranici
%                               prohledávaného prostoru.
%       'zrcadleni'             Zrcadlí agenta do prohledávaného prostoru.
%       'periodicke_rozsireni'  Posunuje agenta o násobky délky příslušného
%                               intervalu z prohledávaného prostoru do
%                               prohledávaného prostoru.
%
%SPSO_2011(FCN,D,B,REALIZACE_PODMINEK,MAX_ITER)
%   MAX_ITER maximální počet iterací
%
%SPSO_2011(FCN,D,B,REALIZACE_PODMINEK,MAX_ITER,S)
%   S velikost hejna, počet agentů
%
%SPSO_2011(FCN,D,B,REALIZACE_PODMINEK,MAX_ITER,S,PRESNOST,GLOB_MIN)
%   PRESNOST    požadovaná maximální odchylka od skutečného minima.
%   GLOB_MIN    skutečná hodnota minimalizované funkce pro testování
%               algoritmu
```

Inicializace

```
if(nargin < 8)
    glob_min = -inf;
    if(nargin < 7)
        presnost = +inf;
        if(nargin < 6)
            S = 40; %velikost hejna
            if(nargin < 5)
                max_iter = 1000;
                if(nargin < 4)
                    realizace_podminek = 'nejblizsi_hranice';
                    if(nargin < 3)
                        B = repmat([-10 10],D,1);
                    end
                end
            end
        end
    end
end
N = adaptivni_nahodna_topologie(S);
X = generuj_hejno(D,S,B);
V = nastav_rychlost(D,S,B,X,2011);
P = X; %matice P nese fitness každého agenta
p_hodnoty = fcn(X); %hodnota současného osobního fitness
L = aktualizuj_fitness_okoli(D,S,N,P,p_hodnoty);
iter = 0;%počet iterací, které byly potřebné pro splnění zadané přesnosti
```

Iterace

```
w = 1/(2*log(2));
c = 1/2 + log(2);
for i = 1:max_iter
    G = X + c*(P+L-2*X)/3;%gravitační centrum
    r = zeros(1,S);%poloměry
    for j = 1:S
        r(j) = norm(G(:,j) - X(:,j));
    end
    Xp = randKn(S,D,r,G');
    V = w*V + Xp - X;
    X = X + V;
    [V, X] = obsluz_podminky(D,S,B,X,V,realizace_podminek);
    [P, p_hodnoty, neni_zmena] = prepocitej_fitness(X,P,p_hodnoty,fcn);
    if(~isempty(neni_zmena))%tam, kde není změna, změň informované okolí
        N = aktualizuj_AN_topologii(S, neni_zmena, N);
    end
    L = aktualizuj_fitness_okoli(D,S,N,P,p_hodnoty);
    if(min(p_hodnoty) < glob_min + presnost)
        iter = i;
        break;
    end
end

[opt, idxFitness] = min(p_hodnoty);
fitness = P(:,idxFitness);


end
```

Generování bodů v kouli

```
function Z = randKn(S,D,r,C)
%RANDKN generuje body náhodně rovnoměrně uvnitř D-rozměrné koule
%S počet bodů, D dimenze, r poloměry, C středy
X = randn(D,S);
Y = X;
R = repmat(r.*(rand(1,S).^(1/D)),D,1);
for i = 1:S
    Y(:,i) = Y(:,i)/norm(Y(:,i));
end
Z = R.*Y + C';
end
```

Příloha D

Zdrojový kód TRIBES

Obsah

- Inicializace
- Iterace
- Aktualizace hejna
- Určení nejlepších a nejhorších částic v každém z kmenů
  - Odstranění nejhorších částic z dobrých kmenů a přepojení informačních spojení
  - Generování nových částic ze špatných kmenů
- Určení nejlepší částice uvnitř kmene
- Aktualizace stavu částice
- Ohodnocení kmenů
- Pohyb hejna
- Určení typu pohybu částice

```
function [opt,fitness,X,iter] = TRIBES(fcn, D, B, realizace_podminek,...
                                       max_iter, presnost, glob_min)


%TRIBES Hledá globální optimum (minimum) zadané funkce.
%
%[OPT, FITNESS, X, ITER]
%   OPT Dosažená nejlepší hodnota účelové funkce.
%   FITNESS Souřadnice odpovídající nejlepšímu fitness v rámci celého hejna.
%   X Matice s poslední dosaženou polohou celého hejna.
%   ITER Iterace potřebné pro dosažení požadované přesnosti.
%
%TRIBES(FCN,D)
%   FCN účelová funkce
%   D dimenze prohledávaného prostoru
%   B = [-10 10]^D
%   REALIZACE_PODMINEK = 'nejblizsi_hranice'
```

```
%   MAX_ITER = 1000
%
%TRIBES(FCN,D,B)
%   B matice Dx2 reprezentující prohledávaný prostor
%
%TRIBES(FCN,D,B,REALIZACE_PODMINEK)
%   REALIZACE_PODMINEK Řetězec specifikující, jakým způsobem se budou
%   podmínky realizovat. Může nabývat následujících hodnot:
%
%       'nejblizsi_hranice'     Přiřazuje agenta na nejbližší hranici
%                               prohledávaného prostoru.
%       'zrcadleni'             Zrcadlí agenta do prohledávaného prostoru.
%       'periodicke_rozsireni'  Posunuje agenta o násobky délky příslušného
%                               intervalu z prohledávaného prostoru do
%                               prohledávaného prostoru.
%
%TRIBES(FCN,D,B,REALIZACE_PODMINEK,MAX_ITER)
%   MAX_ITER maximální počet iterací
%
%TRIBES(FCN,D,B,REALIZACE_PODMINEK,MAX_ITER,PRESNOST,GLOB_MIN)
%   PRESNOST    požadovaná maximální odchylka od skutečného minima.
%   GLOB_MIN    skutečná hodnota minimalizované funkce pro testování
%               algoritmu
```

Inicializace

```
if(nargin < 7)
    glob_min = -inf;
    if(nargin < 6)
        presnost = +inf;
        if(nargin < 5)
            max_iter = 100;
            if(nargin < 4)
                realizace_podminek = 'nejblizsi_hranice';
                if(nargin < 3)
                    B = repmat([-10 10],D,1);
                end
            end
        end
    end
end
S = 1;
X = B(:,1) + (B(:,2) - B(:,1)).*rand(D,1);%jedna částice
T = ones(S);%matice sousednosti, T(i,j) = j, pokud INF(x_i,x_j) == 1
H = ones(2,S);%historie částic, H(1,:) ... t, H(2,:) ... t-1
              %1 <=> "+", 0 <=> "=", -1 <=> "-"
```

```
G = X;%fitness okolí všech částic
P = X;%osobní fitness
fx = fcn(X);
p_hodnoty = fx;%hodnoty odpovídající osobnímu fitness
g_hodnoty = fx;%lokální fitness
kmeny = ones(1,S);%kmeny(i) == j, pokud i-tá částice náleží do j-tého kmene
[C,~,ic] = unique(kmeny);%C je sesortěný vektor kmeny, obsahující každou
                         %hodnotu pouze jednou. C(ic) == kmeny,
                         %kmeny("~") == C
pocet_kmenu = length(C);
velikosti_kmenu = hist(ic,1:max(ic));% == četnosti jednotlivých indexů
                                     %v ic == četnosti prvků v kmeny
skupina = {'excelent'};%špatná, dobrá, excelentní,
                       %pro rozřazení částic pro pohyb
n = 1;%parametr pro reorganizaci hejna
L = 1;%odhad počtu iterací potřebných pro
      %přenos informace od částice x_i k x_j
iter = 0;%iterace potřebné pro dosažení požadované přesnosti
```

Iterace

```
for i = 1:max_iter
    %vektory indexů
    [spatna, dobra, excelent] = vyber_modifikaci_polohy(skupina);
    %pohyb podle skupin
    X = pohyb_hejna(X,P,G,spatna,dobra,excelent,p_hodnoty,g_hodnoty);
    %zeros(D,S) pouze jako záplata místo V, nepoužije se
    [~, X] = obsluz_podminky(D,S,B,X,zeros(D,S),realizace_podminek);
    p_hodnoty_pred = p_hodnoty;%osobní fitness o 1 zpět
    [P, p_hodnoty, ~, fx] = prepocitej_fitness(X,P,p_hodnoty,fcn);
    [G, g_hodnoty] = aktualizuj_fitness_okoli(D,S,T,P,p_hodnoty);
    [H, skupina] = aktualizuj_stav_castice(H, skupina,...
        p_hodnoty, p_hodnoty_pred, fx);
    if(n == L)%je třeba aktualizovat hejno
        klasifikace_kmenu = ...
            klasifikuj_kmeny(kmeny, pocet_kmenu, velikosti_kmenu, H(1,:));
            %1 <=> dobrý kmen, 0 <=> špatný kmen
        [X,P,G,T,H,skupina,p_hodnoty,g_hodnoty,kmeny] = ...
            aktualizuj_hejno(klasifikace_kmenu, kmeny, velikosti_kmenu, ...
            X,P,G,T,H,skupina,p_hodnoty,g_hodnoty,fcn,B);
        S = size(X,2);
        [C,~,ic] = unique(kmeny);
        pocet_kmenu = length(C);
        velikosti_kmenu = hist(ic,1:max(ic));
        n = 0;
        %další adaptace až za L iterací
```

```
        L = S*(S-1)/2 + pocet_kmenu*(pocet_kmenu-1);
    end
    n = n + 1;
    if(min(p_hodnoty) < glob_min + presnost)
        iter = i;
        break;
    end
end
[opt, idx_opt] = min(p_hodnoty);
fitness = P(:,idx_opt);


end
```

Aktualizace hejna

```
function [X,P,G,T,H,skupina,p_hodnoty,g_hodnoty,kmeny] = ...
    aktualizuj_hejno(klasifikace_kmenu, kmeny, velikosti_kmenu, ...
    X,P,G,T,H,skupina,p_hodnoty,g_hodnoty,fcn,B)


pocet_kmenu = length(klasifikace_kmenu);
S = size(X,2);
D = size(X,1);
```

Určení nejlepších a nejhorších částic v každém z kmenů

```
[nejhorsi_castice, nejlepsi_castice] = ...
    nastav_nej_v_kmeni(S,pocet_kmenu,kmeny,p_hodnoty);
```

Odstranění nejhorších částic z dobrých kmenů a přepojení informačních spojení

```
%kanálů na nejlepší částice v jejich kmenu
idx_dobre = find(klasifikace_kmenu);%indexy dobrých kmenů
idx_spatne = 1:pocet_kmenu;
idx_spatne(idx_dobre) = [];%indexy špatných
poc_dobrych = length(idx_dobre);
kmenu_odstraneno = 0;%kolik bude odstraněno jednoprvkových kmenů
if(poc_dobrych > 0)%pokud je vůbec co odstraňovat
    odstranit = zeros(1,S);
    %odstranit(i) == j : bude odstraněna j-tá částice
    for k = idx_dobre
        nejhorsi = nejhorsi_castice(k);
        %není jediná ve kmeni => určitě bude odstraněna
```

```
    if(velikosti_kmenu(k) > 1)
        %ukládám si její index pro odstranění,
        %které bude probíhat až na konci
        odstranit(k) = nejhorsi;
        %přepopuji pouze tam, kde je spojení
        for j = T(T(:,nejhorsi)>0,nejhorsi)
            if(kmeny(j) ~= k)%a není ve stejném kmeni
                %přepojení na nejlepší z kmene
                T(j,nejlepsi_castice(k)) = nejlepsi_castice(k);
            end
        end
    else%pokud je částice poslední v kmeni
        %informátoři nejhorší částice
        T_nejhorsi = T(T(:,nejhorsi)>0,nejhorsi);
        for j = T_nejhorsi
            %pokud nějaký informátor nese lepší informaci
            if(g_hodnoty(j) < g_hodnoty(nejhorsi))
                vsechny_spoje = kombinace(T_nejhorsi);
                for i = size(vsechny_spoje,2)
                    %je potřeba pospojovat všechny částice (mimo kmen),
                    %které ta nejhorší informovala
                    T(vsechny_spoje(1,i),vsechny_spoje(2,i)) = ...
                        vsechny_spoje(2,i);
                    T(vsechny_spoje(2,i),vsechny_spoje(1,i)) = ...
                        vsechny_spoje(1,i);
                end
                kmenu_odstraneno = kmenu_odstraneno + 1;
                odstranit(k) = nejhorsi;
                break;%stačí aby pouze jeden nesl lepší informaci
            end
        end
    end
    if(odstranit(nejhorsi) > 0)
        %nuluji sloupec i řádek, abych odpojil částici od hejna
        T(:,nejhorsi) = 0;
        T(nejhorsi,:) = 0;
    end
end
%konečně odstranění
pryc = odstranit(odstranit > 0);
X(:,pryc) = [];
P(:,pryc) = [];
%G(:,pryc) = [];%po aktualizuj_fitness_okoli se opraví samo
H(:,pryc) = [];
T(:,pryc) = [];
T(pryc,:) = [];%řádky i sloupce
skupina(pryc) = [];
```

```
    p_hodnoty(pryc) = [];
    %g_hodnoty(pryc) = [];%po aktualizuj_fitness_okoli se opraví samo
    kmeny(pryc) = [];
    S = S - length(pryc);
    pocet_kmenu = pocet_kmenu - kmenu_odstraneno;
    %musím znovu určit nejlepší ve kmeni, pokud byla nějaký agent odstraněn
    [~, nejlepsi_castice] = nastav_nej_v_kmeni(S,pocet_kmenu,...
                                               kmeny,p_hodnoty);
    %oprava matice T
    for j = 1:S
        T(T(:,j)>0,j) = j;%vsechny prvky v j-tém sloupci musí být == j
    end
    %mohla být odstraněna částice,
    %která nesla g_hodnotu pro nějakou částici ve špatném kmenu
    [G, g_hodnoty] = aktualizuj_fitness_okoli(D,S,T,P,p_hodnoty);
end
```

Generování nových částic ze špatných kmenů

```
poc_spatnych = length(idx_spatne);
%z každého špatného kmene vzejdou dvě nové částice
poc_novych_castic = 2*poc_spatnych;
S_stare = S;%stará velikost hejna
S = S_stare + poc_novych_castic;%nová velikost hejna
if(poc_novych_castic > 0)%pokud je vůbec možné přidávat nové
    idx_noveho_kmene = max(kmeny)+1;
    %přidání nových částic do hejna
    %nové volné částice
    X_volne = B(:,1) + (B(:,2) - B(:,1)).*rand(D,poc_novych_castic/2);
    X_omezene = zeros(D,poc_novych_castic/2);
    for i = 1:poc_novych_castic/2
        %fitness v okolí nejlepší částice z i-tého špatného kmene
        g_fitness_i = G(:,nejlepsi_castice(idx_spatne(i)));
        r = norm(P(:,nejlepsi_castice(idx_spatne(i))) - g_fitness_i);
        X_omezene(:,i) = randKn(1,D,r,g_fitness_i');
        %pokud se vygenerovaly mimo prohledávaný prostor,
        %přiřadí se na hranici
        X_omezene(X_omezene(:,i) < B(:,1),i) = ...
            B(B(:,1) > X_omezene(:,i),1);
        X_omezene(X_omezene(:,i) > B(:,2),i) = ...
            B(B(:,2) < X_omezene(:,i),2);
    end
    X_nove = [X_volne X_omezene];
    P_nove = X_nove;
    G_nove = X_nove;
    H_nove = ones(2,poc_novych_castic);
```

```
    skupina_nove = cell(1,poc_novych_castic);
    for i = 1:poc_novych_castic
        skupina_nove{i} = 'excelent';
    end
    fx = fcn(X_nove);
    p_hodnoty_nove = fx;
    %index nejlepší nové částice v rámci nového kmene
    [g_hodnoty_nove, idx_nejlepsi_nove] = min(fx);
    g_hodnoty_nove = g_hodnoty_nove*ones(1,poc_novych_castic);
    kmeny_nove = idx_noveho_kmene*ones(1,poc_novych_castic);
    T_nove = zeros(S);%rozšíření matice
    T_nove(1:S_stare,1:S_stare) = T;
    T_novy_kmen = zeros(poc_novych_castic);
    for i = 1:poc_novych_castic
        %submatice reprezentující vztahy uvnitř kmene
        T_novy_kmen(:,i) = i+S_stare;
    end
    T_nove(S_stare+1:S,S_stare+1:S) = T_novy_kmen;
    %index nejlepší nové částice už v rámci hejna
    idx_nejlepsi_nove = idx_nejlepsi_nove + S_stare;
    for i = 1:poc_spatnych
        %symetricky se vyplní T_nove tak, aby nejlepší částice v každém
        %špatném kmeni měla spojení s nejlepší částicí v nově
        %utvořeném kmeni
        T_nove(nejlepsi_castice(idx_spatne(i)),idx_nejlepsi_nove) = ...
            idx_nejlepsi_nove;
        T_nove(idx_nejlepsi_nove,nejlepsi_castice(idx_spatne(i))) = ...
            nejlepsi_castice(idx_spatne(i));
    end
    T = T_nove;
    p_hodnoty = [p_hodnoty p_hodnoty_nove];
    g_hodnoty = [g_hodnoty g_hodnoty_nove];
    X = [X X_nove];
    P = [P P_nove];
    G = [G G_nove];
    H = [H H_nove];
    skupina = [skupina skupina_nove];
    kmeny = [kmeny kmeny_nove];
end


end
```

Určení nejlepší částice uvnitř kmene

```
function [nejhorsi_castice, nejlepsi_castice] = ...
    nastav_nej_v_kmeni(S,pocet_kmenu,kmeny,p_hodnoty)
```

```
%nejhorsi_castice(j) = i, v j-tém kmenu je i-tá částice nejhorší
nejhorsi_castice = zeros(1,pocet_kmenu);
%nejlepsi_castice(j) = i, v j-tém kmenu je i-tá částice nejlepší
nejlepsi_castice = zeros(1,pocet_kmenu);
for i = 1:S
    for j = 1:pocet_kmenu
        if(kmeny(i) == j)%pokud i-tá částice leží v j-tém kmeni
            if(nejhorsi_castice(j) == 0)%pokud ještě není nastavena
                nejhorsi_castice(j) = i;%nastav první, na kterou narazíš
            end
            if(nejlepsi_castice(j) == 0)
                nejlepsi_castice(j) = i;
            end
            %i-tá částice je horší než současná nejhorší
            if(p_hodnoty(nejhorsi_castice(j)) < p_hodnoty(i))
                nejhorsi_castice(j) = i;
            end
            if(p_hodnoty(nejlepsi_castice(j)) > p_hodnoty(i))
                nejlepsi_castice(j) = i;
            end
        break;%každá částice je pouze v jednom kmeni
        end
    end
end
end
```

Aktualizace stavu částice

```
function [H, skupina] = aktualizuj_stav_castice(H, skupina,...
    p_hodnoty, p_hodnoty_pred, fx)
    H(2,:) = H(1,:);%historii o dvě zpět už zahazuji
    for i = 1:length(skupina)%pro všechny částice
        if(p_hodnoty(i) < p_hodnoty_pred(i))%zlepšení
            H(1,i) = 1;
        elseif(fx(i) > p_hodnoty_pred(i))%zhoršení
            H(1,i) = -1;
        elseif(fx(i) == p_hodnoty_pred(i))%stagnace
            H(1,i) = 0;
        end
    end
    for i = 1:length(skupina)
        if((H(2,i) == 1 && H(1,i) == 1) || ...
                (H(2,i) == 0 && H(1,i) == 1))
            skupina{i} = 'excelent';
        elseif((H(2,i) == 1 && H(1,i) == 0) || ...
                (H(2,i) == -1 && H(1,i) == 1))
```

```
            skupina{i} = 'dobra';
        else
            skupina{i} = 'spatna';
        end
    end
end
```

Ohodnocení kmenů

```
function kk = klasifikuj_kmeny(kmeny, pocet_kmenu, velikosti_kmenu, H)
kk = ones(1,pocet_kmenu);%všechny kmeny nastaveny jako dobré
for i = 1:pocet_kmenu
    %náhodně určená hranice pro klasifikaci kmenu
    r = randi([0,velikosti_kmenu(i)],1);
    Ngi = 0;%počet dobrých částic v i-tém kmenu
    for j = find(kmeny == i)%pro každou částici v i-tém kmenu
        if(H(j) == 1)%zde je H pouze jednořádková
            Ngi = Ngi + 1;
        end
    end
    if(Ngi <= r)
        kk(i) = 0;%i-tý kmen je špatný
    end
end
end
```

Pohyb hejna

```
function X = pohyb_hejna(X,P,G,spatna,dobra,excelent,p_hodnoty,g_hodnoty)
    D = size(X,1);
    if(~isempty(spatna))
        X(:,spatna) = pivot(X(:,spatna),P(:,spatna),G(:,spatna),D,...
                            p_hodnoty(:,spatna),g_hodnoty(:,spatna));
    end
    if(~isempty(dobra))
        X(:,dobra) = pivot_sum(X(:,dobra),P(:,dobra),G(:,dobra),D,...
                               p_hodnoty(:,dobra),g_hodnoty(:,dobra));
    end
    if(~isempty(excelent))
        X(:,excelent) = lokalni(X(:,excelent),G(:,excelent),D);
    end
end

function X = pivot(X,P,G,D,p_hodnoty,g_hodnoty)%pro špatnou skupinu
    s = size(X,2);%počet částic ve špatné skupině
```

```
    alpha = zeros(1,s);
    for i = 1:s
        if(abs(p_hodnoty(i)) > 1e-15)%jinak může vzniknout 0/0 == NaN
            alpha(i) = p_hodnoty(i) / (p_hodnoty(i) + g_hodnoty(i));
        else
            alpha(i) = 0.5 + 0.5*rand();
        end
    end
    beta = 1 - alpha;
    r = zeros(1,s);
    Up = zeros(D,s);
    Ug = zeros(D,s);
    for i = 1:s
        r(i) = norm(P(:,i) - G(:,i));%"poloměry"
        Up(:,i) = randKn(1,D,r(i),P(:,i)');%rovnoměrně uvnitř koule
        Ug(:,i) = randKn(1,D,r(i),G(:,i)');
    end
    alpha = repmat(alpha,D,1);
    beta = repmat(beta,D,1);
    X = alpha.*Up + beta.*Ug;
end

function X = pivot_sum(X,P,G,D,p_hodnoty,g_hodnoty)
    s = size(X,2);
    X = pivot(X,P,G,D,p_hodnoty,g_hodnoty);
    b = zeros(1,s);
    for i = 1:s
        if(abs(p_hodnoty(i)) > 1e-15 || abs(g_hodnoty(i)) > 1e-15)
            %jinak může vzniknout 0/0 == NaN
            vari = (p_hodnoty(i) - g_hodnoty(i)) / ...
                   (p_hodnoty(i) + g_hodnoty(i));
        else
            vari = rand();
        end
        b(i) = sqrt(abs(vari))*randn();%musí být absolutní hodnota
    end
    b = repmat(b,D,1);
    X = (1+b).*X;
end

function X = lokalni(X,G,D)
    s = size(X,2);
    N = zeros(D,s);
    for i = 1:s
        if(G(:,i) == X(:,i))
            continue;
        end
```

```
        for j = 1:D
        mu = G(j,i)-X(j,i);
        vari = abs(G(j,i)-X(j,i));
        N(j,i) = mu + sqrt(vari)*randn();
        end
    end
    X = X + N;
end
```

Určení typu pohybu částice

```
function [spatna, dobra, excelent] = vyber_modifikaci_polohy(skupina)
%spatna, dobra, excelent jsou vektory indexu v rámci hejna, určující, která
%částice bude využívat metodu pivot, resp. pivot se šumem, resp. lokální
[C,~,ic] = unique(skupina);
c = hist(ic,1:max(ic));
poc_spatna = 0;
poc_dobra = 0;
poc_excelent = 0;
for i = 1:length(C)%určení kolik částic je špatných, dobrých, excelentních
    switch C{i}
        case 'spatna'
            poc_spatna = c(i);
        case 'dobra'
            poc_dobra = c(i);
        case 'excelent'
            poc_excelent = c(i);
    end
end
isp = 1; ido = 1; iex = 1; %indexy v rámci skupin
spatna = zeros(1,poc_spatna);
dobra = zeros(1,poc_dobra);
excelent = zeros(1,poc_excelent);
for i = 1:length(skupina)%pro každou částici
    switch skupina{i}
        case 'spatna'
            spatna(isp) = i;
            isp = isp + 1;
        case 'dobra'
            dobra(ido) = i;
            ido = ido + 1;
        case 'excelent'
            excelent(iex) = i;
            iex = iex + 1;
    end
end
```

```
end
```

Příloha E

Ukázka zdrojového kódu testovacího skriptu

```
function [proc_uspech, stats_opt, stats_iter] = hromadne_testy_pr()
%rng(0,'twister'); %inicializace generátoru náhodných čísel
%rng(7);
D = 2;
presnost = 1e-2;
fcn = @(x) bukin6_vec(x);
B = [-15 -5; -3 3];
glob_min = 0; %argmin = [-10; 1]

pocet_opakovani = 100;
rp_n = 'nejblizsi_hranice';
rp_z = 'zrcadleni';
rp_p = 'periodicke_rozsireni';
rp_akt = rp_z;
max_iter = 1000;
S_95 = 20;
S_6 = 10 + floor(2*sqrt(D));%pro SPSO 2006
S_11 = 40;%pro SPSO 2011

uspech_95 = zeros(pocet_opakovani,1);
uspech_6  = zeros(pocet_opakovani,1);
uspech_11 = zeros(pocet_opakovani,1);
uspech_T  = zeros(pocet_opakovani,1);

opt_95 = zeros(pocet_opakovani,1);
opt_6  = zeros(pocet_opakovani,1);
opt_11 = zeros(pocet_opakovani,1);
opt_T  = zeros(pocet_opakovani,1);

iter_95 = zeros(pocet_opakovani,1);
iter_6  = zeros(pocet_opakovani,1);
```

```
iter_11 = zeros(pocet_opakovani,1);
iter_T  = zeros(pocet_opakovani,1);
tic
parfor i = 1:pocet_opakovani
    [opt_95(i),~,~,iter_95(i)] = PSO_1995(fcn,D,B,max_iter,S_95,...
                                          presnost,glob_min);
    [opt_6(i),~,~,iter_6(i)]   = SPSO_2006(fcn,D,B,rp_p,max_iter,S_6,...
                                           presnost,glob_min);
    [opt_11(i),~,~,iter_11(i)] = SPSO_2011(fcn,D,B,rp_z,max_iter,S_11,...
                                           presnost,glob_min);
    [opt_T(i),~,~,iter_T(i)]   = TRIBES(fcn,D,B,rp_p,max_iter,...
                                           presnost,glob_min);

    if(iter_95(i) > 0)
        uspech_95(i)  = 1; end
    if(iter_6(i) > 0)
        uspech_6(i)  = 1; end
    if(iter_11(i) > 0)
        uspech_11(i) = 1; end
    if(iter_T(i) > 0)
        uspech_T(i)  = 1; end

end
toc
proc_uspech_95 = sum(uspech_95) / pocet_opakovani;
proc_uspech_6  = sum(uspech_6)  / pocet_opakovani;
proc_uspech_11 = sum(uspech_11) / pocet_opakovani;
proc_uspech_T  = sum(uspech_T)  / pocet_opakovani;


stats_opt_95 = [mean(opt_95)  min(opt_95)  max(opt_95) ];
stats_opt_6  = [mean(opt_6)  min(opt_6)  max(opt_6) ];
stats_opt_11 = [mean(opt_11) min(opt_11) max(opt_11)];
stats_opt_T  = [mean(opt_T)  min(opt_T)  max(opt_T) ];

stats_iter_95 = mean(iter_95(iter_95 > 0));
stats_iter_6  = mean(iter_6(iter_6 > 0));
stats_iter_11 = mean(iter_11(iter_11 > 0));
stats_iter_T  = mean(iter_T(iter_T > 0));

proc_uspech = [proc_uspech_95; proc_uspech_6; proc_uspech_11; proc_uspech_T];
stats_opt   = [stats_opt_95; stats_opt_6; stats_opt_11; stats_opt_T];
stats_iter  = [stats_iter_95; stats_iter_6; stats_iter_11; stats_iter_T];
```