Mendelova univerzita v Brně
Provozně ekonomická fakulta

Particle Swarm Optimization: Implementace a testování biologicky inspirované optimalizační metody

Diplomová práce

Vedoucí práce:

doc. Ing. Jan Žižka, CSc.

Bc. Tomáš Przybek

Brno 2016

List zadání

Na tomto místě bych chtěl poděkovat panu doc. Ing. Janu Žižkovi, CSc. za vedení diplomové práce, odborné rady a cenná doporučení.

Čestné prohlášení

Prohlašuji, že jsem práci: **Particle Swarm Optimization: Implementace a testování biologicky inspirované optimalizační metody** vypracoval samostatně a veškeré použité prameny a informace uvádím v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací*.

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 30. prosince 2015 _____

Abstract

Przybek, T. Particle Swarm Optimization: Implementation and testing of bioinspired optimization method. Diploma thesis. Brno, 2016.

This thesis analyzes the implementation of a testing algorithm, Particle Swarm Optimization, biologically inspired optimization method. Introduce us briefly with evolutionary algorithms, analyzes in detail the PSO algorithm and its parameters. Testing is performed on numerical, nominal, and binary data. The application contains graphical user interface. The algorithm is compared with genetic algorithm at the end and results are appropriately discussed.

Keywords

Particle Swarm Optimization, Optimization, C#, Evolutionary Algorithms

Abstrakt

Przybek, T. Particle Swarm Optimization: Implementace a testování biologicky inspirované optimalizační metody. Diplomová práce. Brno, 2016.

Tato diplomová práce rozebírá problematiku implementace a testování algoritmu Particle Swarm Optimization, biologicky inspirovanou optimalizační metodu. Stručně nás seznámí s evolučními algoritmy, podrobně rozebírá PSO algoritmus a jeho parametrizaci. Testování probíhá na numerických, nominálních a binárních datech. Aplikace obsahuje přehledné grafické uživatelské rozhraní. Algoritmus je na konci porovnán s Genetickým algoritmem a výsledky jsou náležitě diskutovány.

Klíčová slova

Particle Swarm Optimization, Optimalizace, C#, Evoluční algoritmy

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod a cíl práce

1.1 Úvod

Optimalizace je hledání hodnot pro množinu proměnných, díky kterým maximalizujeme nebo minimalizujeme nákladovou funkci, $f(\vec{x})$. Nákladová funkce je mnohorozměrná, pokud závisí na více než jedné proměnné, jako je tomu běžně i ve skutečných vztazích. Hledání optima způsobem pokus omyl není nejvhodnější, proto používáme optimalizační techniky, díky kterým můžeme optimum najít efektivněji.

První optimalizační úlohy můžeme zařadit již do období antiky. Podle Zelinky (2002) již v této době byly formulovány úlohy, u kterých byla snaha nalézt největší nebo nejmenší hodnoty. Mezi známé klasické úlohy patří Didonina úloha, kde hledáme maximální možnou výměru pozemku ohraničenou volskou kůží. K vývoji v oblasti optimalizace přispěl i Isaac Newton s publikací Matematické základy přírodní filozofie. Dalším podnětem pro rozvoj optimalizace po druhé světové válce byly úlohy z oblasti ekonomie. Snaha byla optimalizovat výrobní programy, kde při určitém výrobním programu maximalizujeme účelovou funkci neboli nejčastěji zisk podniku a přitom dodržujeme stanovená omezení jako například množství materiálu. Z této oblasti patří k optimalizačním úlohám také dopravní problémy, kde se snažíme zjistit minimum, buď pro celkové dopravní náklady, nebo čas pro veškerou přepravu. V alokačních problémech zjišťujeme optimální umístění spojovacího bodu při zohledňování dopravních a kooperačních nákladů. Využití optimalizace zasahuje i do dalších oblastí jako je chemie, strojírenství, stavebnictví a další. Složitost optimalizačních úloh neustále stoupá. V minulosti se složité úlohy omezovaly, aby je bylo možné řešit pomocí deterministických metod. Dnes, z důvodu zvyšující se poptávky po co nejoptimálnějších informacích, se hledají metody, pomocí kterých řešíme úlohy bez jejich zjednodušování.

Problematika optimalizace je již mnoho let tématem mnohých matematických a technických publikací a výzkumů. Při optimalizaci byl poměrně dlouho využíván klasický matematický aparát založený na infinitezimálním počtu, variačních

metodách aplikovaných ve funkcionálních prostorech či numerických metodách (Zelinka, 2004). Tímto způsobem bylo možné nalézt globální extrém u prostších úloh, případně lokálního extrému u komplexnějších úloh. Výpočetní složitost některých problémů znemožňuje najít deterministickým způsobem řešení, které by bylo nalezeno v rozumném čase. Proto v 60. letech 20. století začínají vznikat nové optimalizační algoritmy spadající do skupiny tzv. evolučních algoritmů. Jejich název byl převzat z evolučních procesů odehrávajících se v přírodě a popsaných Charlesem Darwinem v 19. století (Ježek, 2005).

V posledních desetiletích bylo vytvořeno mnoho převratných a v praxi často využívaných typů evolučních algoritmů. Obsahují několik zvláštností, díky kterým jsou široce použitelnými a také často používanými. Vyžadována je pouze efektivně sestavená účelová (cenová) funkce a znalost optimalizované problematiky. Podstatnou výhodou je i snaha o nalezení globálního extrému, který se principiálně evoluční algoritmy snaží zjistit. Naproti tomu běžné optimalizační metody se často zaseknou pouze v lokálních extrémech. Pro práci s algoritmy vycházíme ze zkušeností nabytých při používání různých typů dat (Zelenka, 2002).

1.2 Cíl

Náplní této práce bude seznámení se s evolučními algoritmy, konkrétně s algoritmem Particle Swarm Optimization a vytvořením efektivní implementace algoritmu Particle Swarm Optimization (optimalizace rojem částic) včetně grafického uživatelského rozhraní. Následně bude algoritmus otestován pomocí numerických, binárních a nominálních typů dat. Výsledky budou porovnány s jiným typem evolučního algoritmu a to s genetickým algoritmem.

2 Teoretická část

2.1 Optimalizační algoritmy

Optimalizační algoritmy se využívají při řešení mnoha problémů v nejrůznějších oborech. Většinu řešených problémů lze převést na matematický problém definovaný tzv. účelovou funkcí. Můžeme tedy optimalizaci považovat za metodu hledající nejvhodnější řešení daného problému. Cílem optimalizace je nalézt ideální parametry, jejichž funkce dosahuje maxima nebo minima. Zkoušíme tedy měnit parametry řešeného problému a zkoumáme jejich výsledný vliv na daný optimalizovaný problém. Optimalizační metody členíme na lokální a globální. Lokální metody zlepšují parametry jen v blízkém okolí, zatímco globální v celém prostoru. Optimalizace může být jednorozměrná, případně multikriteriální. O jednorozměrné optimalizaci hovoříme, měníme-li pouze jediný parametr. Multikriteriální optimalizace nastává, když měníme více parametrů zároveň (Truhlář, 2008).

Příkladů z praxe při řešení optimalizačního problému je celá řada. Můžeme sem řadit například nalezení optimální trajektorie robota, optimální tloušťky stěny tlakové nádoby, optimální nastavení parametrů regulátoru, optimální relace mezi fuzzy množinami. Těchto příkladů můžeme nalézt nekonečně mnoho. Při práci s parametry účelových funkcí musíme brát na zřetel i jejich definiční obor, který může nabývat různorodého charakteru. Mohou to být například obory celočíselné, reálné, komplexní nebo diskrétní. V celém intervalu povolených hodnot také může parametr účelové funkce nabývat, v určitých subintervalech, různých typů hodnot (například celočíselné, reálné, komplexní nebo diskrétní). Při optimalizaci často využíváme i penalizací a omezení nejen na dané parametry, ale i na funkční hodnotu optimalizované funkce. Nalezení řešení takových optimalizačních problémů je analytickou cestou častokrát možné, nicméně velmi výpočetně a časově náročné (Zelinka, 2004).

Pro nalezení řešení těchto optimalizačních problémů bylo vyvinuto v posledních desetiletích množství velmi účinných a výkonných algoritmů. Tyto

algoritmy dovolují hledat řešení velmi rychlým a efektivním způsobem. Tato třída algoritmů se nazývá „evoluční algoritmy“.

Níže v této kapitole jsou uvedeny základní a zároveň nejvyužívanější typy evolučních algoritmů a nástin principů, na kterém pracují.

2.2 Evoluční algoritmy

Tento typ algoritmů dokáže řešit i velmi složité problémy rychle a elegantně. Evoluční algoritmy pochází z Darwinovy teorie o vzniku druhů, neboli evoluční teorie. Pojetí evoluce z pohledu informačních technologií je tedy inspirováno Darwinovým pojetím evoluce. Tato teorie je povětšinou akceptována jako vysvětlení vzniku a adaptace přírodních druhů. Teorie se opírá o to, že přizpůsobení jedinci mají větší šanci na přežití v daném prostředí a mohou dále předávat své geny. Jedinci se od sebe mírně liší svojí genetickou výbavo, kdy občasné, obvykle náhodné změny tvoří evoluci u daného druhu. Další a další generace generují nové změny v genetickém kódu, čímž se přizpůsobují prostředí s cílem přežít a množit se. Hlavním rozdílem evolučních algoritmů oproti přirozené evoluci je účel a cíl, které určuje fitness funkce. V přírodě probíhá evoluce postupně a trvale bez konce a cíle, tedy nemá žádný koncový stav. Jejich hlavním prvkem je tzv. populace možných řešení, která se nadále vyvíjí a mění. V dalších iteracích nastává průběžné vylepšování jejích vlastností. Na tomto principu vznikla spousta algoritmů, které se inspirují přírodními jevy nebo chováním některých živočišných druhů. Patří mezi ně například netopýří algoritmus, mravenčí kolonie, kukaččí algoritmus, proudění vody a další. Tyto typy algoritmů fungují s různými heuristikami, které následně mění svoji populaci. V populaci mohou nastávat vylepšení, kdy jsme našli optimálnější řešení. V takovém případě nová generace nahrazuje generaci předchozí. Při vytváření následujících generací často hraje důležitou roli náhoda a její implementace do navrhovaných řešení. Jak už bylo zmíněno, rozvoj nastal v posledních desetiletích hlavně z důvodu pokroku v oblasti výpočetní techniky. Tyto metody se používají tam, kde by klasické optimalizační metody byly nedostatečné z důvodu přílišné složitosti daného problému (Jurčík, 2014; Dostál, 2007).

Mezi hlavní výhody evolučních algoritmů patří, že se algoritmy snaží nalézt globální a nikoliv pouze lokální optimum jako tomu bývá u klasických metod. Také nám, na rozdíl od klasických numerických metod, po ukončení jejich činnosti zpravidla poskytují více možných řešení. Další výhodou je, že řešitel zkoumaného problému nemusí znát nutně optimalizační algoritmus, nýbrž stačí pouze velmi dobrá znalost zkoumaného optimalizovaného problému a schopnost tuto problematiku popsat. Nevýhody evolučních algoritmů jsou zřejmé, z důvodu práce s pseudonáhodnou veličinou můžeme jen velice těžko předpovídat chování algoritmu a je obtížné sestavovat komplikované matematické důkazy těchto algoritmů. Navzdory těmto nevýhodám jsou v praxi velmi využívané, protože často naleznou výhodnější řešení než klasické metody, a také kvůli jejich vyšší efektivitě (Ježek, 2005).

Problémy, které se objevují při optimalizaci některých typů úloh, podle Zelinky (2004):

- Prostor možných řešení je příliš veliký na to, abychom ho mohli klasickými metodami prohledat. Například enumerativní přístup, který prohledává celý prostor postupně a hledá nejlepší řešení.
- Řešený problém je natolik složitý, že vytvořený matematický model použitý při řešení tohoto problému vrací výsledky, které řeší tento problém u modelu, avšak není řešením pro skutečný řešený problém. Důvodem je přílišné zjednodušení při tvorbě matematického modelu, který neodpovídá skutečnosti.
- Měření kvality pomocí účelové funkce může podléhat šumu nebo se měnit v čase. Tady můžeme například použít množinu řešení zobrazující vývoj nejlepších řešení v čase.
- Možná řešení jsou podrobena tak přísným podmínkám, že je prohledávaný prostor možných řešení rozdroben na spoustu izolovaných podmnožin možných řešení, která reprezentují tzv. „akceptovatelná řešení“.

2.3 Základní pojmy

V této podkapitole jsou obsaženy základní pojmy používané v oblasti evolučních algoritmů. Vysvětlení těchto pojmů usnadňuje pochopení podrobnějšího popisu evolučních algoritmů.

Jedinec – je část programu, která představuje možné řešení a je to prvek zajišťující prohledávání stanoveného prostoru (Dostál, 2007).

populace – představuje množinu jedinců. První generace jedinců je nejčastěji náhodně vygenerovaná. Při vytváření jedinců jsou stanoveny pouze omezující podmínky jako například hloubka stromu, vyvážení stromu a podobně (Dostál, 2007).

Generace – představuje současnou populaci jedinců. Vždy po každém cyklu mutací, křížení a selekcí vznikne nová generace (Dostál, 2007).

genotyp – obsahuje genetické informace jedince (Dostál, 2007).

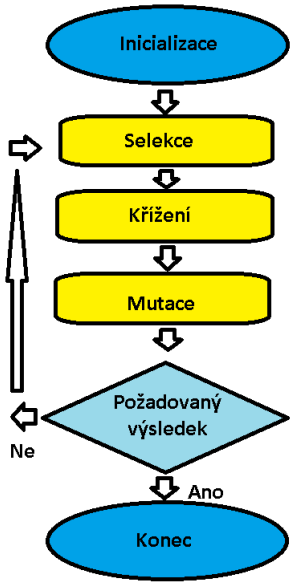
fenotyp – jedná se o interpretaci genotypu, tedy jak se konkrétní gen projevuje. Je to například barva vlasů nebo barva očí. U genetických algoritmů reprezentuje dekódování genotypu, např. dekódování řetězce jako dekadického čísla (Dostál, 2007).

Operátory – užívají se k úpravám všech jedinců a tím se mění celá populace a nabývá nových vlastností. Mezi tyto operátory řadíme mutace, křížení, inverze apod. (Jurčík, 2014).

Fitness – jedná se o účelovou funkci, která se využívá v evolučních algoritmech a určuje kvalitu daného jedince (Jurčík, 2014).

2.4 Přehled operátorů

Následující text popisuje operátory, pomocí kterých evoluční algoritmy pracují. Tyto operátory upravují původní generaci jedinců, tedy vytvářejí generaci nových jedinců. Obrázek 1 popisuje obecný princip evolučních algoritmů. Pro každý z popsaných operátorů existuje spousta různých variant, jejichž podrobnější popis tato práce neobsahuje, jelikož to není jejím cílem a přesahuje to její rámec.



Obrázek 1: Princip evolučních algoritmů. Zdroj: Dostál, 2008

2.4.1 Rychlost

Tento operátor používají algoritmy Rojení částic (např. PSO). Tento operátor mění pozici jedince v prostoru. Vstupem je tedy jedinec a na základě změny pozice částice v prostoru hledáme extrém.

2.4.2 Selekce

Tento operátor stanovuje vhodnost daného řešení z populace k další reprodukci, operátor tedy nevytváří, ale pouze posuzuje vhodnost jedinců. I řešení, která nedosahují nejlepších výsledků, se v evolučním programování zcela nezavrhují. Slabší jedinci s nižší fitness funkcí, kteří se ocitnou ve skupině slabých jedinců, mohou postoupit do další generace a vylepšit řešení problému. Jako vstup dostáváme určitou populaci jedinců a pomocí účelové funkce dochází k selekci vybraných jedinců. Do další generace se dostávají jen vybraní jedinci, kteří se mohou dále reprodukovat a podílet se na dalších generacích (Metelková, 2006). Mezi algoritmy používající selekci patří Genetický algoritmus, Evoluční strategie, Evoluční programování, Greedy algoritmus, Scatter search, SOMA, Diferenciální evoluce, Umělý imunitní systém, Mravenčí kolonie (ACO) a Tabu search (Metelková, 2006).

Přehled různých druhů selekcí:

Ruleta: Jednotlivé výseče na ruletě představují možná řešení, která podle výsledku fitness funkce zabírají poměrově výseče na ruletě. Dále probíhá výběr náhodně (Metelková, 2006).

Matice: Do matice řešení jsou nejprve vložena ta s nejvyšší selekcí, která zabírají i vyšší počet políček než řešení s nižší selekcí. Po vyplnění matice řešení jsou pářením vytvářeni noví potomci (Metelková, 2006).

Klasifikace: Tato selekce je podobná Ruletě. Ovšem jednotlivá řešení dostávají dle svých fitness ohodnocení pravděpodobnost výběru. Čím vyšší je fitness ohodnocení, tím je vyšší šance výběru (Metelková, 2006).

Dalšími typy selekcí jsou například Turnaj, Boltzmanův operátor, Soft brood, Rozklad a konkurence (Metelková, 2006).

2.4.3 Mutace

Vstupem je jedinec, u nějž bude mutace probíhat. Mutace je náhodně prováděná operace, kdy se vybere náhodné číslo, které se přičte k parametrům mutujícího jedince. Toto číslo má u evolučních algoritmů obvykle rozdělení podle Gaussovy křivky. Genetický algoritmus používá inverzi vybraných bitů v čísle. Mutace nám dává nové informace. Jedná se tedy o proces, kdy dochází k náhodným změnám vlastností jedince. Evoluční algoritmy, na rozdíl od genetických algoritmů, vždy používají mutaci. Přestože je vliv na populaci obvykle nízký, tento operátor zlepšuje různorodost populace a snižuje riziko uvíznutí v lokálním extrému (Jurčík, 2014).

Mezi algoritmy používající mutaci řadíme Horolezecký algoritmus, Evoluční programování, Genetický algoritmus, Simulované žíhání, SOMA, Diferenciální evoluce nebo Umělý imunitní systém (Metelková, 2006).

2.4.4 Křížení

Vstupem je jedinec, avšak každý problém definuje proces křížení různě. U jedince definovaného diskrétními daty je pro proces křížení používán výběr dat z jednoho rodiče. U jedince definovaného reálnými daty se obvykle pro proces křížení používá průměr obou rodičů (Jurčík, 2014).

Všeobecně je křížení definováno jako výměna informací mezi dvěma jedinci. Zkřížením dvou jedinců se zrodí dva noví jedinci (potomci). Kompletní proces křížení probíhá tak, že u každého jedince v populaci proběhne náhodný pokus, jehož výsledek může jedince přiřadit do množiny vybraných jedinců ke křížení. Z této množiny vytváříme páry ke křížení, buď náhodným výběrem, nebo po sobě jdoucími jedinci (Kvasnička, 2000).

Algoritmy používající křížení jsou Genetický algoritmus, Genetické programování, Paralelní simulované žíhání, SOMA, Diferenciální evoluce a Umělý imunitní systém (Metelková, 2006).

2.4.5 Ostatní operátory

- Operátor reprodukce využívá operátory mutace a křížení a jejich kombinaci při vytváření nových potomků.
- Dalším operátorem je například inverze, kde dochází ke změně binárního řetězce na jiný řetězec.
- Stochastický operátor využívá náhodných změn stavové proměnné.
- Operátor feromonová matice požívá algoritmus ACO (Mravenčí kolonie). Tato matice nám pomáhá v hledání možných řešení například při řešení problému obchodního cestujícího.

2.5 Popis účelové funkce

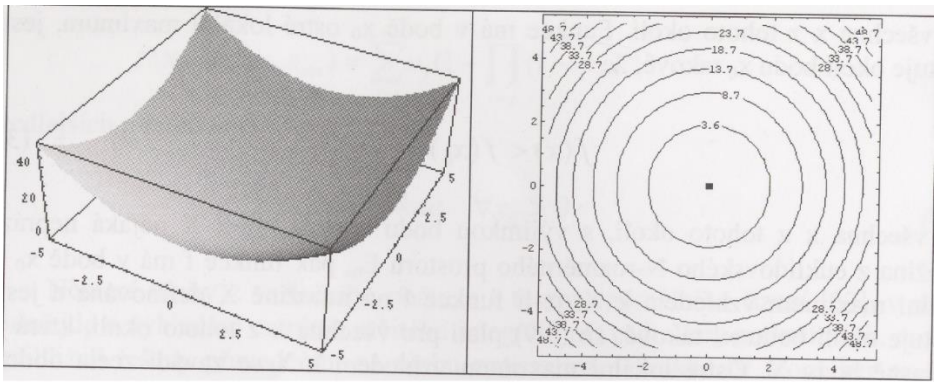
Účelová funkce, také zvaná fitness funkce, slouží u optimalizačních algoritmů k ohodnocení kvality možných řešení z jejich přípustné množiny. Argumenty optimalizovaných funkcí mohou nabývat různých hodnot od celých čísel po čísla komplexní a jiná. Abychom mohli dosáhnout požadovaných výsledků při optimalizaci, je velmi důležitý vhodný návrh účelové funkce pro daný typ problému. Čím jsou jedinci silnější, tím lepšího výsledku dosahují v účelové funkci. I tento princip je inspirován Darwinovým popisem evoluce, tedy že lépe vybavení jedinci mají vyšší šanci v dalším vývoji. Zajištění zjištění objektivní kvality jedinců je esenciální pro postupnou konvergenci řešení k vhodnějším výsledkům. Ohodnocení bývá často problematickou částí, protože objektivní ohodnocení bývá u algoritmů

obtížné. Účelová funkce pro ohodnocení se používá u genetických algoritmů, genetického programování nebo v algoritmech založených na evolučním programování. Účelová funkce je navržená tvůrcem algoritmu a jako taková většinou nepodléhá evoluci, jako je tomu u populace jedinců představujících možná řešení. Ovšem například u tzv. autokonstruktivní evoluce dochází k evoluci i u samotných evolučních mechanismů (Dostál, 2007).

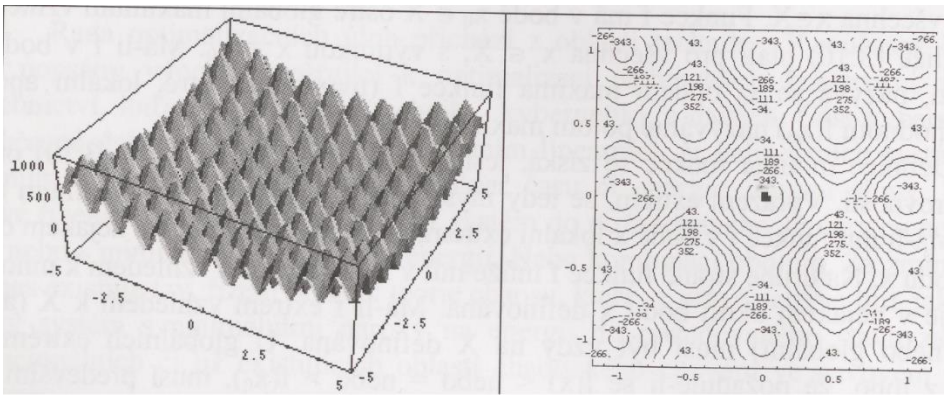
Optimalizace účelové funkce nás tedy vede ke zjištění optimálních hodnot jejich argumentů. Účelovou funkci budeme standardně značit $f(x)$, případně $f_{cost}(x)$. Účelová funkce bývá nazývána i jako funkce cenová, proto označení „cost“. U účelové funkce platí, že má lokální maximum v bodě x_0, pokud platí $f(x) \leq f(x_0)$, kde $x \in X$ a X je množina bodů symbolizujících okolí bodu x_0. Dále rozlišujeme ostré lokální maximum, kde platí $f(x) < f(x_0)$ (Zelinka, 2002). Globální extrém nastává tehdy, pokud v bodě $x_0 \in X$, kde X je neprázdná množina z euklidovského N-rozměrného prostoru E_n definujícího všechny možné body z definičního oboru, nastává $f(x) \leq f(x_0)$, kdy platí pro všechna $x \in X$. Případně ostré globální maximum $f(x) < f(x_0)$ v bodě $x_0 \in X$ pro všechny $x \in X$ s výjimkou $x = x_0$. Zjištění extrému nemusí vždy znamenat zjišťování maxima, ale i minima. V tomto případě stačí obrátit smysl nerovnosti na $f(x) > f(x_0)$, případně $f(x) \geq f(x_0)$ (Zelinka, 2002).

Účelovou funkci můžeme chápat i jako geometrický problém, kde hledáme dosaženou nejnižší, případně nejvyšší pozici na N-rozměrné ploše. Tuto plochu nazýváme hyperplocha či prostor možných řešení tohoto problému. Hodnota N je počet dimenzí dané účelové funkce, který je dán počtem optimalizovaných argumentů této funkce. V případě, že hledáme optimum účelové funkce, která má pět na sobě nezávislých argumentů, snažíme se najít extrém na pětirozměrné hyperploše v šestirozměrném prostoru. Šestý rozměr nám dává výsledná návratová hodnota účelové funkce. Obrázek 2 i 3 zobrazují funkci s jedním globálním

extrémem, liší se však v tom, že funkce na obrázku 3 je obklopena lokálními extrémy. Přestože se jedná pouze o trojrozměrnou projekci funkce, princip je stejný jako u funkcí vícerozměrných. Návratová hodnota je udávána na ose z, ostatní rozměry dimenzí jsou stejné jako osy *x* a *y.*



Obrázek 2 Účelová funkce jako geometrický problém Zdroj: Zelinka, 2002

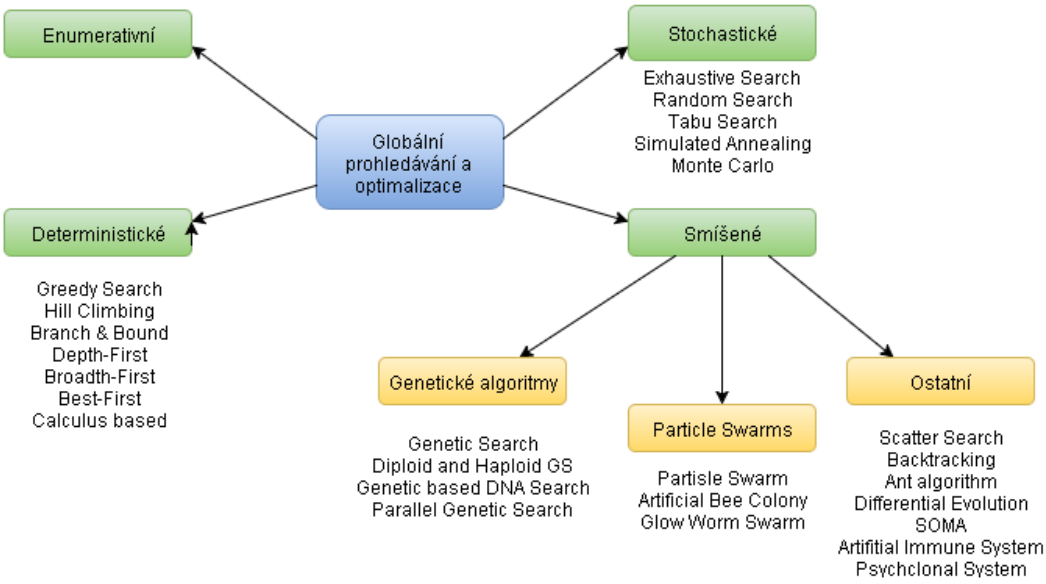


Obrázek 3 Účelová funkce jako geometrický problém Zdroj: Zelinka, 2002

Korektní vytvoření účelové funkce je jednou z nejnáročnějších a nejkritičtějších částí optimalizačního procesu. Možných optimalizačních problémů je nekonečně mnoho, a proto neexistuje jednotný postup pro tvorbu účelové funkce. Je nutné vědět, čeho musíme dosáhnout a z čeho můžeme vycházet (Zelinka, 2002).

2.6 Rozčlenění optimalizačních algoritmů

V následující kapitole je uvedeno rozdělení optimalizačních algoritmů a jejich popis. Existuje více druhů rozdělení, vybrané uspořádání však uspokojivě charakterizuje současnou situaci a člení algoritmy dle jejich funkčních principů, viz obrázek 4.



Obrázek 4 Optimalizační algoritmy Zdroj: podle Zelinka (2002) a Dostál (2012)

2.6.1 Enumerativní

Jedná se o nejjednodušší optimalizační algoritmy. Pracují s daty bez předpokladů a dosazují celý prostor možných řešení postupně do účelové funkce, čímž vždy naleznou optimum. Tyto metody jsou vhodné pro argumenty, kde dosazujeme diskrétní hodnoty a obecně relativně malé množství dat. Praktické nasazení pro řešení složitých problémů není možné, jelikož by prohledávání bylo časově příliš náročné. Časová náročnost problému exponenciálně roste s množstvím dosazovaných argumentů a velikostí dat (Antošovský, 2013).

2.6.2 Deterministické

Pro nalezení optima používají stejně jako Enumerativní metody zadaná data, ale neprovádí výpočet všech možných řešení. Výsledkem tedy může být uvíznutí v lokálním extrému a nenalezení skutečného optima. Deterministické metody používají zadané předběžné předpoklady a prohledávají jen klíčové hodnoty v datech. Výstupem je jeden nalezený extrém. Předpoklady mohou být například: problém je lineární, problém je konvexní, prohledávaný prostor možných řešení je spojitý nebo účelová funkce je unimodální. Možných předpokladů je mnoho (Zelinka, 2002).

2.6.3 Stochastické

Tyto algoritmy, na rozdíl od výše uvedených, přináší používání kromě reálných dat i pravděpodobnosti a náhody při výběru možných řešení. Výstupem je nejlepší řešení nalezené během náhodného hledání. Tyto algoritmy neuvíznou v lokálním extrému a na rozdíl od deterministických metod mohou bezproblémově prohledávat i nespojitý prostor. Vhodné jsou pro přibližné odhady a prohledávání malých prostorů.

2.6.4 Smíšené

Představují kombinaci metod deterministických a stochastických. V současné době se jedná o nejefektivnější optimalizační metody. Mohou pracovat s libovolnými daty bez omezení velikosti prohledávaného prostoru.

2.7 Popis principů vybraných optimalizačních algoritmů

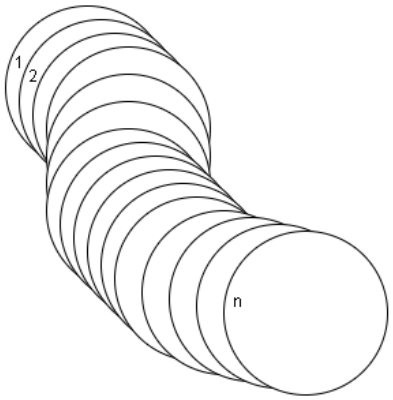
2.7.1 Hill-Climbing (Horolezecký algoritmus)

Algoritmus řadíme do skupiny tzv. gradientních algoritmů (procházejí prostor možných řešení ve směru nejvyššího spádu). Jedná se o nejjednodušší informovanou metodu pro prohledávání stavového prostoru. Předpokládejme řešení ve tvaru vektoru (1.1) a kriteriální funkci (1.2).

$$x = (x_1, x_2 \dots x_n) \tag{1.1}$$

$$f = f(x) \tag{1.2}$$

Na začátku se vygeneruje náhodný startovací bod, který je ohodnocen pomocí kriteriální funkce. Toto řešení je rozšířeno, čímž se zjistí ohodnocení všech sousedů okolo aktuálního řešení. Aktuálním řešením se stává to s nejvyšším ohodnocením. Postup se opakuje, dokud v okolí nalézáme řešení s vyšším ohodnocením *f* než je to aktuální, případně dokud nedosáhneme maximálního počtu iterací, viz obrázek 5. Algoritmus je náchylný k uvíznutí v lokálních extrémech, což je dáno principem, na kterém funguje. Je vhodný k prohledávání prostoru bez lokálních extrémů, neboť jeho uvíznutí by záleželo na místě vygenerování startovacího bodu. Výhodami jsou jednoduchost, rychlost a paměťová náročnost algoritmu. Pro jeho běh a funkčnost není nutné ukládání historie navštívených bodů.



Obrázek 5 Princip horolezeckého algoritmu Zdroj: podle Zelinka (2002)

2.7.2 Stochastic Hill-Climbing (Stochastický horolezecký algoritmus)

Jedná se o rozšíření původního algoritmu Hill-Climbing a přidáváme k ní stochastickou část. Vycházíme z principu Hill-Climbing algoritmu a ten mnohokrát zopakujeme z náhodně vygenerovaných startovacích bodů. Za specifických podmínek opět dochází k zacyklení a uvíznutí v lokálním extrému.

2.7.3 Greedy Search (Hltavé prohledávání)

Vstupem je n-prvková množina. Snažíme se nalézt její podmnožinu splňující podmínky odrážející charakter problému. Výsledkem je řešení optimalizující účelovou funkci. Algoritmus pracuje v iteracích, kdy v každém kroku zvolíme nejvhodnější bod pro zařazení do optimálního řešení a body, které nebyly zvoleny v tomto kroku, vyřadíme (Metelková, 2006).

2.7.4 Depth-First Search (Prohledávání do hloubky)

Prohledávání do hloubky nám negarantuje, že je nalezeno nejoptimálnější možné řešení. Algoritmus je podobný jako prohledávání do šířky s rozdílem, že neprohledáváme sousedy uzlů, nýbrž jejich následníky, kteří splňují dané podmínky. Algoritmus se zanořuje do hloubky a tím poměrně rychle nalezne nějaké řešení. Princip je tedy prostá expanze prvního následníka a splňuje-li podmínky, pokračuje dále v expanzi. Pokud první následník kritéria nesplňuje, pokračujeme v expanzi dalších následníků. Pokud ani ti nesplňují kritéria, vracíme se o uzel v hierarchii výše. Pro správnou funkčnost používáme tzv. zarážku, která zastaví zanořování. V opačném případě by zanořování pokračovalo do té doby, než by

přetekl zásobník nebo by bylo nalezeno nějaké řešení. Zarážkou bývá obvykle hodnota označující maximální možnou délku cesty.

2.7.5 Breath-First Search (Prohledávání do šířky)

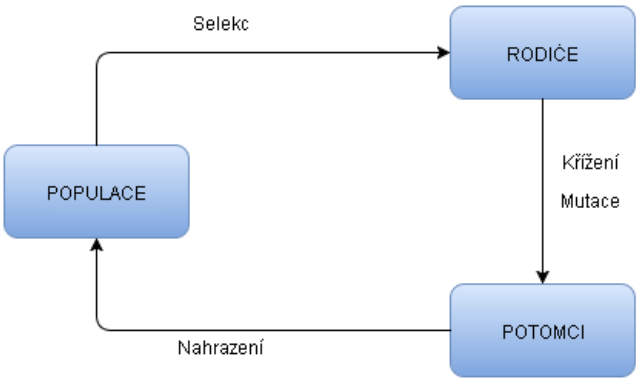
Můžeme jej přirovnat k lavině z vrcholu hory nebo k vlně způsobené objektem vrženým proti hladině vody. Algoritmus prohledává sousedy v jednotlivých vrstvách pomocí FIFO fronty a tím dostáváme optimální řešení, jestliže nějaké existuje. Algoritmus funguje následovně: Zařadíme startovní uzel do fronty. Tento uzel vezmeme z fronty a prozkoumáme, jestli je řešení nalezeno - tím bychom ukončili hledání a dostali výsledek. Není-li řešení nalezeno, zařadíme do fronty neprozkoumané následníky tohoto uzlu. Je-li fronta prázdná, není řešení nalezeno. V opačném případě prozkoumáme první uzel ve frontě a tento postup se opakuje.

2.7.6 Genetický algoritmus

Patří mezi úspěšné optimalizační techniky, u kterých je nevýhodou poměrně náročný výpočetní proces a je potřeba volit přiměřeně velkou populaci. Vývoj populací běží v diskrétních krocích. Posloupnost populací nazýváme generacemi. Na počátku se vygenerují na náhodně zvolených, ale rozprostřených souřadnicích jedinci. Množství jedinců musí být dostatečné kvůli degeneraci populace, kdy větší populace obvykle obsadí větší množství lokálních extrémů a je větší šance na dosažení optima. Všichni jedinci se pomocí fitness funkce ohodnotí a podle výsledné kvality jedince se vyberou kandidáti pro výrobu nové generace jedinců. Nižší ohodnocení ovšem neznamená úplné vyloučení daného jedince z další možné reprodukce. Zmíněný jedinec nemusí být v celku tak kvalitní jako nejlepší jedinci v současné generaci, ale některé části slabšího jedince mohou být vhodné pro další reprodukci, čímž zlepší následující generaci. Každý jedinec je definován svým chromozomem složeným z genů, tedy podmnožin chromozomu. Geny určují vlastnosti jedinců a tím pádem i jejich kvalitu. Rozmnožování vybraných jedinců probíhá tak, že se chromozomy zkříží, čímž vytvoří chromozom nový.

Jedinci z množiny jedinců vybraných pro další reprodukci jsou ke křížení do dvojic vybíráni buď náhodně, nebo jsou jako pár bráni jedinci po sobě jdoucí. Pokud má tato množina lichý počet jedinců, je jeden z nich vyřazen. Následuje mutace,

která náhodně mění bitovou hodnotu genu. Mutace nesmí být příliš rozsáhlá, aby jedinec unikl možnému uvíznutí v lokálním extrému a zároveň byl zachován prvek dědičnosti. Jedinci by neměli být pouze kvalitní, ale i odlišní od průměru. Vhodné křížení zachovává co nejvyšší různorodost jedinců, kdy právě jejich diverzita zabraňuje uvíznutí jedinců v lokálním extrému. Konec nastává, jakmile se populace nadále nezlepšuje. Nalezený výsledek se považuje za globální optimum, které však ve skutečnosti optimem být nemusí. Na obrázku 6 je zobrazen cyklus běhu genetického algoritmu (Žižka, 2014).



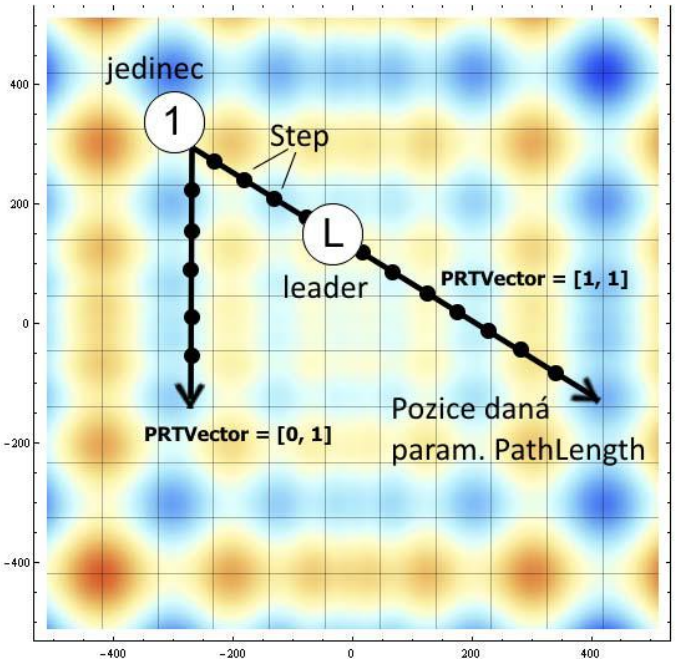
Obrázek 6 Princip genetického algoritmu

2.7.7 SOMA (samoorganizující se migrační algoritmus)

Spadá do skupiny smíšených algoritmů a je inspirován inteligentními jedinci, kteří spolupracují na vyřešení společného problému jako například vlci lovící potravu. V podstatě se každý jedinec snaží najít nejlepší řešení, která si v průběhu hledání sdělují a na jejichž základě mění své další chování. Migrace jedinců probíhá vždy za nejlepším řešením.

Při běhu algoritmu nedochází k vytváření nových jedinců, jako je tomu například u výše popsaného Genetického algoritmu. Jedinci pouze migrují v prostoru možných řešení a jsou ovlivňováni jinými jedinci, tedy samoorganizují se. Před tvorbou populace a spuštěním algoritmu je nutné nadefinovat řídící a ukončovací parametry spolu s fitness funkcí. Na počátku vygenerujeme ve stavovém prostoru na náhodných pozicích populaci. Stanovíme Leadera, který má nejvyšší ohodnocení fitness funkce. Následuje další migrační kolo, kdy se všichni jedinci přesunují směrem k leaderovy po předdefinovaných krocích a pokud některý

z jedinců nalezne řešení lepší než jeho původní, řešení si zapamatuje. Jedinci se nepohybují po přímce, ale mírně se od ní odchylují, což zlepšuje diverzitu populace a robustnost algoritmu. Na obrázku 7 vidíme, že před migrací jedince se nejprve vygeneruje prázdný PRTVector o velikosti D, který je naplněn hodnotami z intervalu $< 0,1 >$, které jsou porovnávány s parametrem PRT. Pokud je n-té číslo vyšší než PRT parametr, nastavíme v PRTVectoru na n-té pozici 0, jinak nastavíme 1. Parametry s ohodnocením 0 se nepřičítají, čímž se snižuje počet stupňů volnosti pohybu jedince v prostoru. Poté, co se jedinci usadí na svých nejlepších pozicích, zkontrolujeme rozdíl mezi nejhůře a nejlépe ohodnoceným jedincem. Pokud je rozdíl vyšší než výše povolené chyby, nebo nebyl naplněn minimální počet migračních kol, migraci opakujeme (Volná, 2012).



Obrázek 7 Proces migrace jedince

Vyskytuje se více typu SOMA algoritmu:

- AllToOne

  Zakladní typ SOMA algoritmu je popsán výše.
- AllToAll

  Nepoužívá se Leader a všichni směřují ke všem.

- AllToAllAdaptive

  Shodná s AllToAll s rozdílem, že migrace aktuálního jedince probíhá okamžitě na nalezenou pozici.

- AllToOneRand

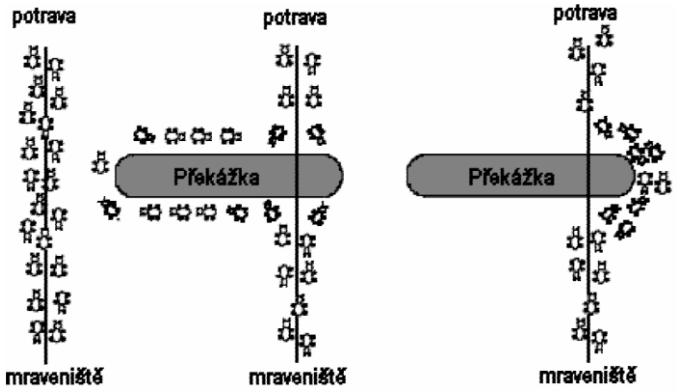
  Podobná se základním typem SOMA, ale Leader není vybírán podle nejvyššího ohodnocení fitness funkce.

2.7.8 Immunology system method (Metoda imunologického systému)

Jak název napovídá, tento algoritmus je inspirovaný principem imunologického systému v živém organismu. Tento systém obsahuje mnoho typů různých agentů a každý z agentů zajišťuje řešení od sebe odlišných úloh. Tento multiagentní systém je schopen řešit mnoho komplikovaných úloh, jako například komplexní antivirová ochrana rozsáhlých počítačových sítí. Klíčová je dovednost vzájemné komunikace agentů, kdy má každý agent odlišnou pravomoc. Právě pomocí vzájemných interakcí a určité svobody rozhodování zvládají tyto systémy řešení komplexních úloh (Zelinka, 2004).

2.7.9 Ant Colony Optimization (ACO, Optimalizace mravenčích kolonií)

Inspiruje se chováním kolonie mravenců při hledání potravy. Z mraveniště, které představuje zdroj mravenců, náhodnými směry putují mravenci hledající potravu, která představuje cíl jejich cesty. Jakmile mravenec nalezne nějaké řešení, vypouští při zpáteční cestě feromon, který značkuje cestu. Feromon napomáhá mravencům nasměrovat se k cíli. Mravenci se vydají tou cestou, kde je feromon silnější. Jak bylo zmíněno výše, bez feromonu jsou jejich rozhodnutí náhodná. Na obrázku 8 je patrné, že čím je cesta kratší, tím bude její označení intenzivnější.



Obrázek 8 Hledání optimální cesty Zdroj: Metelková (2006)

Podstatnou vlastností algoritmu je postupné odpařování feromonů. Tím je zajištěno odstranění méně optimálních cest, což zvyšuje robustnost algoritmu při hledání globálního optima a snižuje možnost uvíznutí v lokálním extrému. Tento algoritmus se často používá při řešení problému obchodního cestujícího nebo návrhu telekomunikačních sítí. Následuje pseudokód řešící problém obchodního cestujícího.

```
Ohodnocení všech hran defaultním množstvím feromonu
Náhodné vygenerování mravenců v uzlech
while (i < max_pocet_cyklu)
while (!mravenec_v _cili)
      for j to max_ant do
            Přesun mravence[j ] k dalšímu uzlu na základě ohodnocení
      end
      end
      for k to max_ant do
      Vypařování feromonu
      Označení feromony
      If (delka(ant[k ]) < best) best = delka(ant[k ])
      end
end
```

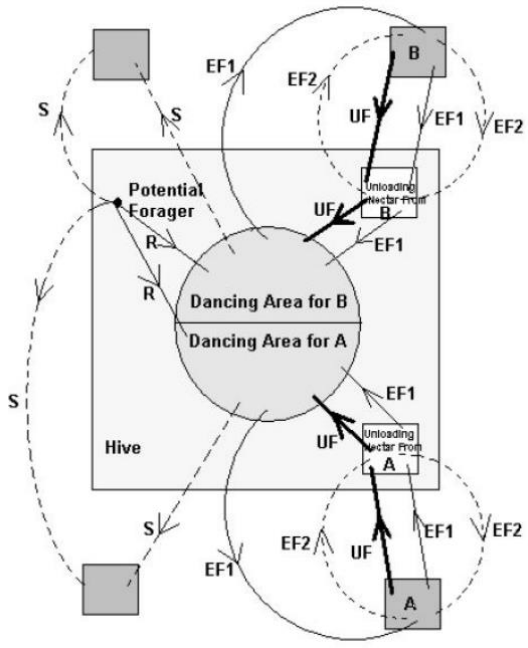
Program 1 : Pseudokód algoritmu ACO Zdroj: upraveno podle Dorigo (2006)

2.7.10 Artificial Bee Colony Algorithm (ABC, Optimalizace včelím rojem)

Byl poprvé publikován v roce 2005 a spadá pod algoritmický přístup optimalizace včelím rojem. Inspiruje se chováním biologických včel a přebírá některé jejich charakteristické znaky. Další používané algoritmy inspirované tímto přístupem jsou Honeybee Mating Optimisation Algorithm (včelí pářící optimalizační algoritmus),

Bee System algorithm (algoritmus včelího systému), Bees Algorithm (včelí algoritmus) aj.

Včelí roj se skládá ze tří částí, a to ze zdroje potravy, pracujících včel a nepracujících včel. Kvalita zdroje potravy se stanovuje podle vzdálenosti od úlu, konzistence energie a obtížnosti tuto energií získat z potravy. Pracující včely se snaží najít na konkrétním místě ve stavovém prostoru co nejlepší řešení. Pracující včely rovněž přenáší informace o vzdálenosti a směru řešení od úlu a o jeho kvalitě. Nepracující včely se dělí na dvě části, a to průzkumné včely a diváky. Průzkumné včely prohledávají prostor možných řešení a snaží se najít náhodně co nejlepší řešení. Diváci čekají v úlu a stanovují kvalitu řešení pomocí sdílených informací. Sdílení informací probíhá na tzv. taneční ploše, kde jednotlivé včely představují nalezená řešení a všechno podstatné o nich. Následně ostatní včely pomáhají zlepšit (lokální prohledávání) nejlepší (ruletová selekce) nalezená řešení (Machytka, 2013).



Obrázek 9 Princip ABC algoritmu Zdroj: (Machytka, 2013)

Zobrazený princip na obrázku 9 ukazuje dva zdroje potravy, a to řešení A a B. Dále je zobrazena potencionální dělnice, která neobsahuje žádné informace. Z této včely

se může stát průzkumná včela, která hledá nová řešení (možnost S), nebo půjde na taneční plochu a zjistí informace o již známých řešeních (možnost R). Po nalezení řešení se z potencionální včely stává pracující včela. Ta může nalezené řešení opustit (možnost UF), získat další včely sdílením řešení na taneční ploše (možnost EF1), nebo přenést řešení bez získání dalších včel (možnost EF2).

2.7.11 Diferenciální evoluce

Nový a hojně využívaný algoritmus, který často rychleji konverguje k řešení než jiné podobné algoritmy. Principiálně je podobný evolučním algoritmům. Pro každého jedince ze staré generace A vytvoří konkurenčního jedince, kdy do nové populace bude vybrán ten s nižší funkční hodnotou. Pro vytvoření jedince nového potřebujeme čtyři rodiče. Ze tří náhodně vybraných nestejných jedinců vytvoříme jejich kombinací tzv. šumový vektor, který je následně použit při křížení se čtvrtým rodičem. Následuje porovnání a soutěž čtvrtého rodiče a potomka o místo v nové generaci.

Následuje Diferenciální evoluce zapsaná v pseudokódu.

```
Generuj P (stará generace P, velikost N náhodně v D)
while ukončující podmínka
      for i = 1 to N do
            Generuj vektor u
            Vytvoř vektor Y křížením u a x_i
            If f(y) < f(x_i) then Q += y
                        else Q += x_i
      P = Q
end
```

Program 2 : Pseudokód algoritmu Diferenciální evoluce Zdroj: upraveno podle Volná (2012)

2.8 Particle Swarm Optimization (PSO, Optimalizace hejnem částic)

Tato výpočetní technika má původ v umělé inteligenci a sociální psychologii. Byla vyvinuta roku 1995 elektroinženýrem Russelem Eberhartem a sociálním psychologem Jamesem Kennedym a inspiruje se sociálním chováním rybích a ptačích hejn. Algoritmus PSO pochází z pokusných modelů vytvářených biologem

Frankem Heppnerem roku 1990. Ten zkoušel vytvářet algoritmy inspirované chováním hejn (Volná, 2012).

Tyto modely byly velmi podobné již známým modelům a chování hejn věrně kopírovaly. Navíc byla přidána možnost přistání v hnízdišti, ke kterému byla částice přitahována. Takové hnízdiště může reprezentovat i cíl částic. V modelu se částice pohybovaly v hejnu, kdy byl jejich pohyb dán vektory rychlosti a pozicemi v prostoru. Pravidla pohybu definovala, že každá částice se snažila dostat mezi ostatní, ale zároveň se nesměla s žádnou přímo střetnout. Jakmile se nějaká částice vyskytla v místě hnízdiště a pokud byla nastavená hodnota přistání v hnízdišti vyšší než hodnota udržující částici v hejnu, částice se odpoutala od hejna a zůstala na místě. Čím více částic zůstalo v hnízdišti, tím více částic bylo k hnízdišti přitahováno. Tímto principem nakonec celé hejno zůstalo v hnízdišti (Ježek, 2005).

Níže v této kapitole je popsán princip fungování algoritmu PSO, kde můžeme inspiraci tímto modelem vidět. Hledání řešení ve stavovém prostoru je velmi podobné hledání hnízdiště a inteligence hejna, tedy sociokognitivní myšlení jedinců při hledání řešení a ovlivňování okolních jedinců je také velmi podobné. Změny se nachází u procházení jedinců stavovým prostorem a hledání jejich hnízdiště. Problémem je zjištění optimálního hnízdiště neboli řešení, které jedince na náhodných pozicích přitáhne k tomuto optimu.

Sociální pojetí lze vidět v chování částic, například učení se od ostatních jedinců. Většina inteligentních živočichů se učí od jiných živočichů a inspiruje se z jejich úspěchů. Snažíme se vzájemně porovnávat a imitujeme chování úspěšných lidí v pro nás významných záležitostech. Pouhé napodobování úspěšných jedinců nás k optimálnímu řešení nedovede a je nutné zkoušet nová řešení. Vhodné je aplikovat přístupy oba a najít jejich správný poměr. Příliš málo nových zkušeností nás k příliš dobrému výsledku nedovede, protože zůstaneme uzavřeni určitém kruhu (lokální extrém) a dále se nedostaneme. Naopak ignorováním úspěšných jedinců nedosáhneme relativně optimálního výsledku a určitě nedosáhneme globálního extrému. Toto přirovnání nás vede k tomu, že by částice měli mít nějaký druh inteligence, ale výstižnější popis by byl nemyslící částice létající podle určitých

pravidel n-rozměrným prostorem možných řešení. Tato optimalizační technika se ukázala být nejefektivnější při řešení vícerozměrných problémů a ukázala univerzálnost při řešení různých typů problémů.

Výhodou této optimalizační metody je nízká ovlivnitelnost velikostí prostoru možných řešení. Další výhodou je, že dokáže řešit i velmi nelineární problémy. Metoda poměrně rychle konverguje k optimálnímu řešení, tam, kde mnoho standardních metod zklame. Lepší je i oproti obdobným optimalizačním metodám jako jsou genetické algoritmy. U genetických algoritmů špatná řešení vypadávají v následující iteraci, a proto PSO účinněji zachovává diferenciaci jedinců, kde jednotlivé částice v následující iteraci nevypadávají, ale řídí se nejlepšími nalezenými řešeními. Pokud jsou si řešení velmi blízká, můžou částice uvíznout v lokálním extrému. Výhodami jsou i relativně snazší implementace algoritmu a nižší množství parametrů.

2.8.1 Používané termíny

Particle (agent, částice) jedná se o jednotlivce z hejna. Každá částice se pohybuje individuálně, ale jejich řídící princip je u všech stejný. Pohybují se směrem ke svému PBest a zároveň směrem ke GBest a během toho ověřují pozici, na níž se nachází.

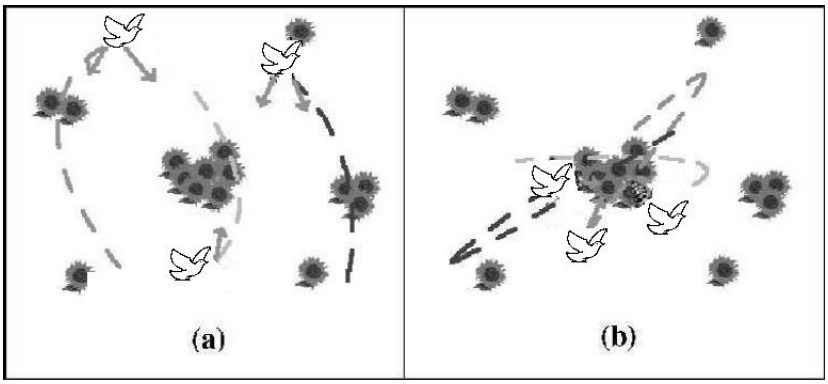
Position je reprezentována polem hodnot, kde každá jednotlivá hodnota definuje pozici v dané dimenzi. U řešeného problému o n parametrech budeme mít pozici reprezentovanou polem o délce n a optimální řešení budeme hledat v n-dimenzionálním prostoru možných řešení.

PBest v níže uvedeném příkladu je to maximální množství drobků nalezených daným jedincem. Hodnota, která je definována účelovou funkcí, je pro každou částici jedinečná. Tedy každá částice má své vlastní PBest a tuto hodnotu při každé iteraci porovnává s aktuálním PBest a případně ji zaměňuje.

GBest je maximální ohodnocení účelovou funkcí pro konkrétní řešení nalezené celým hejnem. Při každé iteraci porovnává každá částice aktuální ohodnocenou pozici s GBest a případně ji zamění. Všechny částice jsou k této hodnotě přitahovány.

2.8.2 Princip PSO

Existuje hejno holubů o n jedincích a jejich úkolem je najít na náměstí místo s nejvyšším množstvím drobků. Jednotliví holubi jsou náhodně vygenerováni na různých pozicích na náměstí a mají náhodnou rychlost. Každý z holubů při vygenerování nezná své okolí, ale dokáže si zapamatovat místo s nejvyšší koncentrací drobků a zároveň zná místo se zatím nejvyšší koncentrací drobků objevenou hejnem. Každý holub má svoji setrvačnost pohybu při hledání, ale pohyb usměrňuje zpět k nejlepšímu řešení, které během svého hledání nalezl a také směrem k nejlepšímu řešení nalezenému hejnem. Výsledná trajektorie bývá někde mezi těmito třemi vektory. Vliv na výslednou rychlost holuba má vliv sociální a individuální konstanty a také konstanta váhy.



Obrázek 10 Princip algoritmu PSO Zdroj: upraveno podle Robinson (2004)

V levé části obrázku 10 jsou zobrazeny částice, které jsou přitahovány k nejlepšímu řešení nalezenému celým hejnem a zároveň přitahovány k jejich individuálnímu maximu. V pravé části mají částice jistou setrvačnost a pohybují se kolem lokálního extrému. Pokud jiná částice nenalezne lepší řešení, zůstanou částice v tomto extrému.

Každá částice je považována za bod v n-rozměrném prostoru a tato pozice je stanovena jeho polohou X_i a rychlostí V_i, které jsou definovány n-rozměrným vektorem. Níže uvedené vektory definují vektor pozici i-té částice v iteraci *t* a vektor rychlosti *i*-té částice *v* iteraci *t*.

$$X_i(t) = \{x_{i1}(t), x_{i2}(t), \dots, x_{in}(t)\} \quad (1.3)$$

$$V_i(t) = \{v_{i1}(t), v_{i2}(t), \dots, v_{in}(t)\} \quad (1.4)$$

Rychlost každé částice je reprezentována sociální a individuální hodnotou. Důležitost těchto dvou hodnot je proměnlivá a na každou z nich je užívána jiná náhodná váha. Individuální hodnota je chápána jako vektor rovnající se bodu v prostoru možných řešení, kde má částice nejlepší osobní řešení a budeme jej označovat X_i^{PBest}. Nejlepší řešení nalezené celým rojem (GBest) budeme označovat X^{GBest}.

Následující rovnice popisuje pohyb částice do nově vypočítané lokality

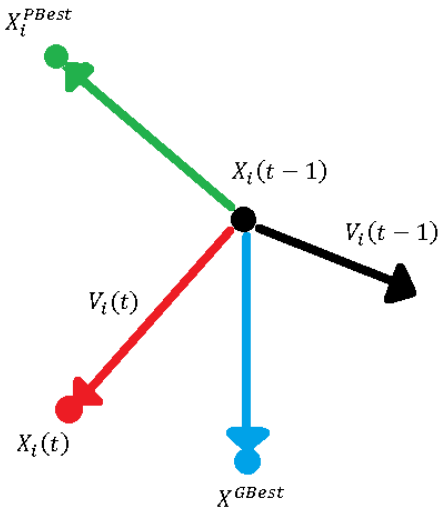
$$X_i(t) = X_i(t-1) + V_i(t) \quad (1.5)$$

Mechanismus fungování pohybu částic je jeden z hlavních prvků celého algoritmu a podrobné seznámení s ním je podstatné pro porozumění principu pohybu částic. Stanovení rychlosti částic popisuje následující rovnice.

$$V_i(t) = w(t) \times V_i(t-1) + c_1 \times rand_1 \times \left(X_i^{PBest} - X_i(t-1)\right) + c_2 \times rand_2 \times (X^{GBest} - X_i(t-1)) \quad (1.6)$$

Rovnice se skládá ze tří částí. Hodnoty $i = \{1, 2, \dots, n\}$ označují jednotlivé částice a proměnná n představuje velikost populace. Proměnná *t* představuje počet iterací. První část rovnice $w(t) \times V_i(t-1)$ popisuje setrvačnost pohybu částice (viz obrázek 11), tedy snahu pokračovat stále stejným směrem. Často bývá násobena nějakou vahou, která omezuje vliv předcházející rychlosti na rychlost současnou. V druhé části ($c_1 \times rand_1 \times \left(X_i^{PBest} - X_i(t-1)\right)$) je popsán přitažlivý pohyb k zatím nejlepšímu nalezenému řešení X_i^{PBest} částicí X_i, hodnotu účelové funkce na tomto místě označujeme PBest a je násobena váhami c_1 a $rand_1$. Hodnoty c_1 a $rand_1$ budou podrobněji popsány v následujících částech textu. Zatím stačí pouze vědět, že c_1 je kladná konstanta a proměnná $rand_1$ nabývá náhodné hodnoty

z intervalu $< 0{,}1 >$. V poslední části rovnice $(c_2 \times rand_2 \times (X^{GBest} - X_i(t-1)))$ je definována přitažlivost k nejlepšímu zatím nalezenému bodu v prostoru X^{GBest} a jeho vypočítanou hodnotu označujeme GBest.



Obrázek 11 Vektorové zobrazení pohybu částice 2D prostorem

2.8.3 Průběh PSO algoritmu

V této části je popsán průběh algoritmu v jednotlivých krocích:

1. Nastavení parametrů algoritmu, které určují jeho funkčnost. Nejdříve vymezíme rozsah prostoru řešení, tedy vybereme parametry, které chceme optimalizovat a přiřadíme rozumné hodnoty, ve kterých chceme hledat optimální řešení. Stanovíme X_{max} a X_{min} hodnotu omezující každou dimenzi. Dále nastavíme hodnoty V_{max} a V_{min} omezující velikost rychlosti částic (v další části textu je uveden důvod tohoto omezení). Určíme počet částic *n* v hejnu, hodnoty c_1 a c_2, vliv individuálního a sociálního chování částic a hodnotu *w,* vliv rychlosti z minulé iterace.
2. Definujeme účelovou funkci, která představuje propojení mezi algoritmem a reálným světem. Optimalizace účelové funkce představuje řešený problém a její definování je klíčovým prvkem optimalizace, pro každý problém musí být vytvořena individuálně. Více v kapitole o účelové funkci.

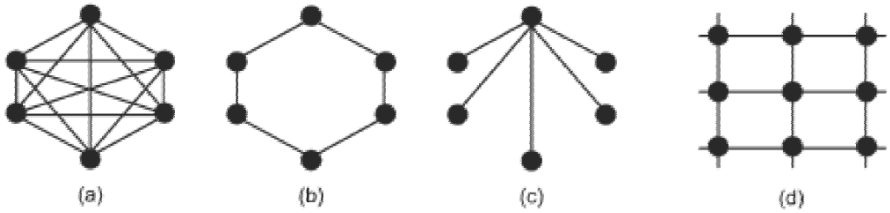
3. Inicializuje se hejno částic, ke každé částici se vygeneruje náhodná startovací pozice na intervalu $< X_{max},\ X_{min} >$ pro každou dimenzi a startovací rychlost pro každou částici, hodnota každého parametru bude v intervalu $< V_{max}, V_{min} >$. Poté se vyhodnotí účelová funkce u všech částic a stanovíme X^{GBest} a X_i^{PBest}.
4. Cyklus hledání optima je následující:
    a. Stanovení rychlostí částic $V_i(t)$ a korekci podle $< V_{max}, V_{min} >$.
    b. Stanovení pozic částic $X_i(t)$ a korekci podle $< X_{max}, X_{min} >$.
    c. Aktualizace nových hodnot X^{GBest} a X_i^{PBest}.
5. Opakování předchozího kroku dokud nenalezneme dostatečně dobrou hodnotu účelové funkce nebo dokud není dosažen maximální počet iterací.

2.8.4 Topologie PSO

Algoritmus PSO existuje v mnoha provedeních. Mohou se lišit např. v předávání informací mezi částicemi. Bylo zjištěno, že některé z optimalizačních problémů tíhnou k jinému než optimálnímu řešení. Příčinou je topologie PSO, kdy všichni jedinci znají nejlepší řešení celého hejna a tím pádem každý jedinec může ovlivňovat všechny.

Algoritmus PSO a jeho topologie vlivu částic může nabývat různých typů sousedství. Sousedství nemá nic společného se vzájemnou prostorovou vzdáleností částic od sebe v prohledávaném prostoru. Význam sousedství se vztahuje na určitý vliv mezi částicemi na jejich vzájemný pohyb. Rozdělujeme je na dva hlavní typy. Globální sousedství, jehož částice znají nejlepší řešení z celého hejna, jedná se tedy o úplný graf (viz obrázek 12a). Druhým typem je částečně propojený graf (viz 12b, c, d), kde jsou částice vzájemně ovlivňovány podle určitých pravidel. Obrázek 12b zobrazuje topologii částic uspořádaných do kruhu, kdy se vzájemně ovlivňují pouze částice sousední. Obrázek 12c zobrazuje topologii hvězdicovou, kde jsou částice od sebe odtrženy a informace se shromažďují v jedné

hlavní částici. Topologie von Neumanna (viz obrázek 10d) bývá často označována za nejlepší.

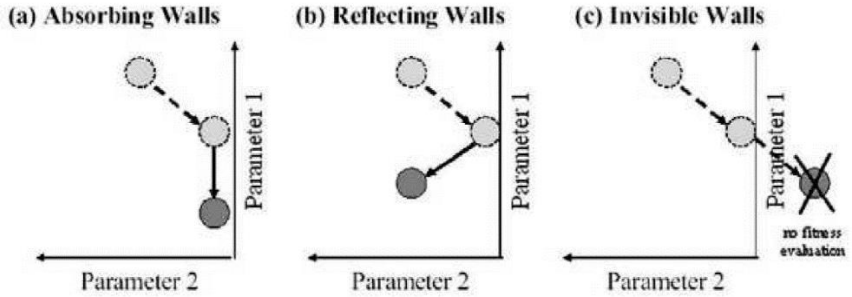


Obrázek 12 Typy topologie u hejn Zdroj: Volná (2012)

Toto „ořezané“ propojení částic zpomaluje konvergenci k nějakému řešení, ale umožňuje algoritmu prohledat větší oblast prostoru možných řešení. Vzájemné propojení sousedů umožňuje postupné šíření informací, jelikož částice se budou postupně pohybovat určitým směrem a přitahovat své sousedy, kteří budou zase přitahovat ty své. To je mechanizmus pomalého šíření informací skrz celé hejno, jelikož je čas šíření této informace podstatně nižší než u standardního PSO algoritmu, diferenciace částic se zvyšuje a lze tak prohledat větší rozsah možných řešení, protože se stále pohybují kolem svého PBest.

2.8.5 Hraniční podmínky

Při hledání optima nastavujeme oblast prostoru, kde toto optimum vyhledáváme. Stává se, že částice dosáhne hranice definovaného prohledávaného prostoru. Hodnoty jako konstanty snižující rychlost, maximální rychlost částice (V_{max}) a ani setrvačné váhy neomezí částici pouze ve vymezeném prostoru. V těchto případech se používají hraniční podmínky, viz obrázek 13.



Obrázek 13 Pohyb částic po dosažení hranic D(f) Zdroj: Robinson (2004)

Obrázek 13a popisuje absorpční stěnu, pokud se částice dostane na hranici definovaného prostoru možných řešení u jedné nebo více dimenzí, bude rychlost částice v těchto dimenzích vynulována. Následně se bude částice pohybovat podél této hranice nebo bude přitažena zpět. Částice na obrázku 13a dosáhla hranice u parametru 2 a rychlost v této dimenzi se vynulovala.

Jiným způsobem usměrnění částic je odrážející stěna. Vždy, když se částice dostane na hranici určité dimenze, znaménko rychlostí se u této dimenze prohodí. Výsledkem je odražená částice. Tento princip je zobrazený na obrázku 13b.

Posledním typem pro řešení hraničních podmínek je neviditelná stěna. Výhodou této metody je snížení časové náročnosti, protože se vypouštějí částice mimo dovolené hranice. Částice se pohybují i mimo vymezené hranice, což nechává algoritmu volný průběh. Na obrázku 13c vidíme částici, která se dostala mimo hranici dimenze parametr 2. Tato částice se poté nevyhodnotí účelovou funkcí.

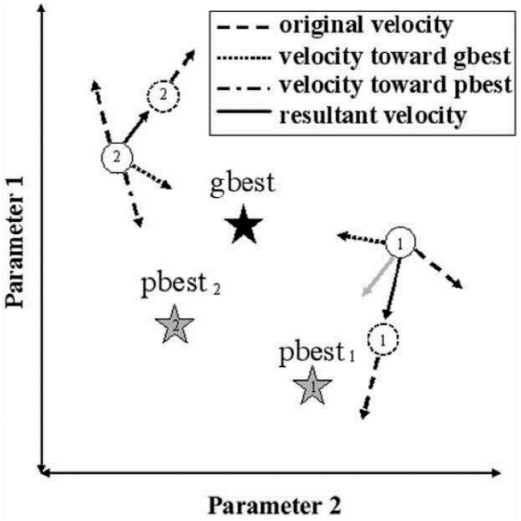
2.8.6 Stanovení parametrů PSO

Na zjištění ideálních parametrů bylo provedeno již mnoho experimentů. Výsledkem měl být algoritmus s optimálním poměrem mezi globálním hledáním a exploatací v místech maxim. Jak bylo naznačeno výše, byl problém s řízením hejna ve vymezeném prostoru, kde je částečné řešení nastavení V_{max}. Parametry $rand_1$, $rand_2$ vrací náhodné číslo v rozmezí 0.0 a 1.0. Tím vkládáme do algoritmu jistý stochastický prvek, který mění přitažlivost k X^{GBest} a X_i^{PBest}. Hejno se tak stává poměrně nepředvídatelné v dalších krocích algoritmu a tím pomáhá zabránit cyklickému pohybu částic.

Abychom se vyhnuli přílišné divergenci částic, nastavujeme maximální rychlosti částic, optimální nastavení konstant přitažlivosti a nastavení setrvačnosti. Částice se může pohybovat chaoticky. Abychom se tomu vyhnuli, nastavíme V_{max} a V_{min}. Tato omezení je vhodné přizpůsobovat řešenému prohledávanému prostoru, protože veliké V_{max} může přejít některá optima a malé V_{min} snižuje

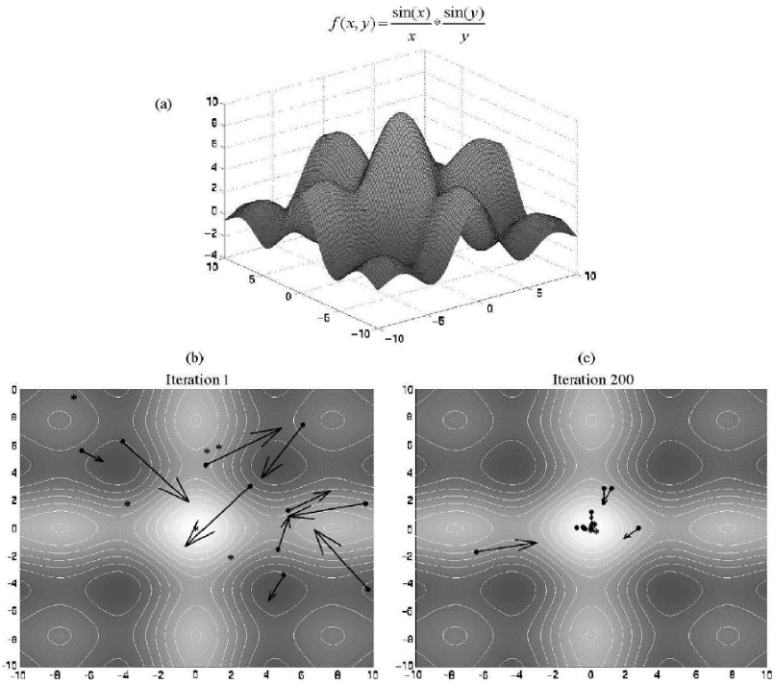
konvergenci k optimu a nemusíme optimum vůbec nalézt. Proměnné c_1 a c_2 usměrňují pohyb k nalezeným extrémům. Nízké hodnoty příliš limitují tento pohyb a vysoké hodnoty způsobují příliš velký rozptyl částic do prostoru. Bylo zjištěno, že při nízkých hodnotách c_x se trajektorie částice podobá sinusoidě a naopak při vysokých hodnotách (vyšších než 4) částice mířila do nekonečna. Optimální hodnotou pro proměnné c_2 a c_1 byla stanovena hodnota 2, kdy ale záleží na řešeném problému Tyto hodnoty lze v závislosti na řešeném problému upravovat.

2.8.7 Pohyb částic v prohledávaném prostoru



Obrázek 14 Princip pohybu části ve 2D prostoru Zdroj: Robinson (2004)

Každá částice se pohybuje skrz prostor možných řešení, viz obrázek 14. Pohyb částic probíhá postupně na další pozici. Spojení tří vektorů, původní rychlost (original velocity), přitažlivost k nejlepšímu řešení celého hejna X^{GBest} (velocity toward GBest) a přitažlivost k nejlepšímu řešení dané částice X_i^{PBest} (velocity toward PBest), jak je zobrazeno na obrázku 14, nám dává vektor výsledné rychlosti, který je označen jako $V_i(t)$ (resultant velocity).



Obrázek 15 Příklad hledání optima algoritmem PSO Zdroj: Robinson (2004)

Obrázek 15 ukazuje proces hledání optima u dvoudimenzionální funkce, viz obrázek 15a. Obrázek 15b představuje částice v procesu hledání. Každá šipka ukazuje směr a sílu rychlosti částice a hvězdička označuje X_i^{PBest} částic, přičemž jedna z hvězdiček označuje jejich X^{GBest}. Obrázek 15c zobrazuje již dvousté opakování algoritmu, kdy X^{GBest} i X_i^{PBest} jsou shromážděny v jedné lokalitě stejně jako většina částic.

3 Metodika práce

Cílem práce je efektivní implementace algoritmu PSO spolu s vhodným uživatelským rozhraním. Účelem uživatelského rozhraní je přehledné nastavení parametrů běhu algoritmu, přehledné zobrazení výsledného výstupu a zajištění přehledu, v jakém stavu se výpočet nachází. Následně bude možné vypsat a uložit podrobný popis průběhu výpočtu a výsledné řešení. Z důvodu maximální efektivnosti programu bude výsledná binární forma programu spustitelná pod 64bitovým OS MS Windows.

Pro vývoj této aplikace byla vybrána platforma .NET, která umožňuje efektivní vývoj aplikace spolu se zaměřením na její výkon. Programovacím jazykem, jenž tato platforma využívá, je objektově orientovaný programovací jazyk C #. Nejvýraznější nevýhodou je omezení běhu aplikací pouze na operačních systémech společnosti Microsoft.

Jako vývojové prostředí pro platformu .NET jsem zvolil Microsoft Visual Studio 2013 Ultimate. Toto prostředí jako většina jiných zajišťuje nástroje pro psaní a kompilaci programového kódu. Dále poskytuje možnosti generování UML diagramů a generování programového kódu na základě diagramu tříd.

Pro tvorbu grafického uživatelského prostředí byla použita knihovna Windows.Forms, která je podporována ve Visual Studiu a je možné pohodlně používat nástroje této knihovny.

Postup práce a jednotlivé kroky při tvorbě aplikace

1. Shrnutí uživatelských požadavků představujících vstupy a výstupy aplikace, parametry programu pro funkčnost algoritmu.
2. Návrh a schémata implementace zahrnujících návrh funkčních modelů aplikace a objektových tříd a ostatních programových elementů.
3. Kontrola navržené aplikace bude provedena tak, že poskytnuté výsledky budou ověřovány na lehce ověřitelných problémech. Ověření funkčnosti bude provedeno na nominálních, binárních i numerických typech dat.

4. Vyhodnocení schopností algoritmu PSO najít optimum a porovnání výsledků s jiným optimalizačních algoritmem. Shrnutí bude obsahovat zhodnocení algoritmu PSO z hlediska jeho parametrizace a efektivity vzhledem k objemu dat.

Při práci na praktické části byly využity znalosti z výše uvedené teoretické části.

4 Vlastní práce

4.1 Optimalizované úlohy

Řešenými úkoly u numerických umělých dat bude sada unimodálních a multimodálních funkcí, u nichž zjišťujeme optimální hodnoty ze stanoveného definičního oboru hodnot.

Pro numerická reálná data využíváme stejnou sadu unimodálních a multimodálních funkcí, ovšem pro tato data nemáme stanovený definiční obor hodnot, ale získáváme je z externího csv souboru. Každý řádek v souboru představuje jednu řešenou dimenzi, pokud je tedy řešená funkce dvojrozměrná, bude aplikace využívat pouze první dva řádky souboru a hledat jejich optimální kombinaci pro danou funkci.

Následující typy dat mají řešený úkol společný. U nominálních i binárních dat hledáme nejdelší společnou sekvenci hodnot, viz obrázek 16.

Sekvence 1	A	A	D	D	A	A	B	C	A	D	B	A	C	D	D	D	B	B	D	D	B	C	D	B	D	A	D	B	C	C
	A	A	C	C	D	C	C	A	B	B	A	D	A	A	D	C	B	C	B	D	A	B	D	C	A	C	B	B	A	D
.	B	D	D	B	D	B	C	D	D	B	A	B	D	C	D	C	C	C	C	C	A	D	B	A	C	C	C	A	D	C	B
.	A	A	A	C	A	A	A	D	A	B	A	B	C	A	D	D	B	B	D	C	A	D	D	D	D	D	D	A	D	A
.	B	C	B	C	A	D	B	B	C	C	B	D	B	A	D	C	C	A	D	D	B	D	D	A	B	B	A	A	C	A
	B	D	A	D	D	B	A	A	D	D	B	D	D	A	A	A	B	C	D	B	B	C	B	D	D	C	D	A	B	D
Sekvence 7	C	A	C	B	B	C	D	B	A	C	B	C	B	D	B	B	D	D	A	A	D	B	B	B	D	C	C	A	A	B

Obrázek 16 Stejná sekvence znaků u nominálních dat

Každý řádek je reprezentován sekvencí znaků. Umělá data jsou náhodně vygenerována v rozsahu stanoveném definičním oborem, jenž představuje délku prohledávaných sekvencí, přičemž počet sekvencí je stanoven navoleným počtem sekvencí. Každý řádek u nominálních dat představuje jeden genom složený ze 4 typů hodnot. Aplikace se snaží najít nejdelší společnou podmnožinu sekvence alespoň u dvou genomů.

Nominální i binární reálná data získáme importem z csv souboru. Stejně jako u umělých dat aplikace hledá nejdelší společnou sekvenci v jednotlivých genomech.

4.2 Analýza a specifikace požadavků

Jádro aplikace bude tvořeno standardním PSO algoritmem a bude obstarávat nalezení nejvhodnějšího řešení. Tento standardní algoritmus je popsán výše v teoretické části práce. Pohyb částice, pokud dosáhne hranice definičního oboru pro danou dimenzi, bude absorbován, viz obrázek 13a. Topologie hejna bude představovat úplný graf, tedy každá částice bude moci ovlivnit každou částici v hejnu, viz obrázek 12a.

Uživatel aplikace bude moci specifikovat typ optimalizovaných dat. Může si vybrat mezi nominálními, binárními a numerickými daty, kdy tato data mohou být reálná nebo uměle vytvořená. Pro pohodlné používání aplikace bude vytvořeno grafické uživatelské rozhraní, kde si uživatel může navolit typ prohledávaných dat a také si zvolit, budou-li data automaticky vygenerována v uživatelem zvoleném rozsahu nebo zda uživatel využije externí zdroj dat. Následně se aplikace pokusí najít v datech optimum ve výše uvedených řešených úlohách. Základní výsledky bude aplikace ihned zobrazovat a bude možné exportovat podrobnější výsledky.
Aplikace bude uživateli poskytovat možnost parametrizace algoritmu. Nastavení parametru ovlivňuje chování algoritmu a dosahování co možná nejlepších výsledků. V aplikaci je možné upravovat níže uvedené parametry algoritmu PSO:

- Váha
- Konstanta osobní
- Konstanta globální
- Počet částic
- Rozsah definičního oboru

Při hledání optima v datech je obtížné stanovit, jestli jsme už globální optimum nalezli nebo algoritmus uvíznul v lokálním optimu. Při řešení složitějších úloh, u kterých neznáme globální optimum, jsme po nějakém čase běhu algoritmu nalezli určité řešení, nemůžeme však ověřit, jestli je toto řešení nejoptimálnější. Proto je důležité definovat ukončovací podmínku algoritmu. Nevhodným řešením by bylo zvolit jako ukončovací podmínku časový interval.

V tomto případě by byl počet iterací závislý na výkonu stroje, což by znemožnilo porovnání dosažených výsledků. Mezi vhodné ukončovací podmínky patří například:

- Pevný počet iterací
- Počet iterací od posledního zlepšení globálního optima

V aplikaci je možné stanovit počet iterací běhu algoritmu. Dalším parametrem je při generování nominálních a binárních umělých dat počet sekvencí. Po dokončení výpočtu budou dosažené výsledky prezentovány v tabulce, kde uvidíme nejdelší nalezený řetězec, případně nejvyšší nalezenou hodnotu účelové funkce spolu s nalezenými pozicemi.

Důraz bude kladen i na přehlednost a jednoduchost grafického uživatelského rozhraní. Nepotřebné prvky při řešení různých úloh nebudou zobrazeny. Nalezené výsledky budou přehledně zpracovány v tabulce a podrobnější informace bude možné získat exportem. O stavu výpočtu bude aplikace uživatele informovat lištou s procentuálním vyjádřením dokončení výpočtu.

4.2.1 Funkční požadavky

Níže jsou uvedeny funkční požadavky na aplikaci:

1. Nalezení optima účelových funkcí.
2. Nalezení nejdelší společné sekvence.
3. Import reálných dat pro zpracování.
4. Export výsledků v textovém souboru.
5. GUI pro zobrazení výsledků a nastavení parametrů.

4.2.2 Nefunkční požadavky

Níže jsou uvedeny nefunkční požadavky na aplikaci:

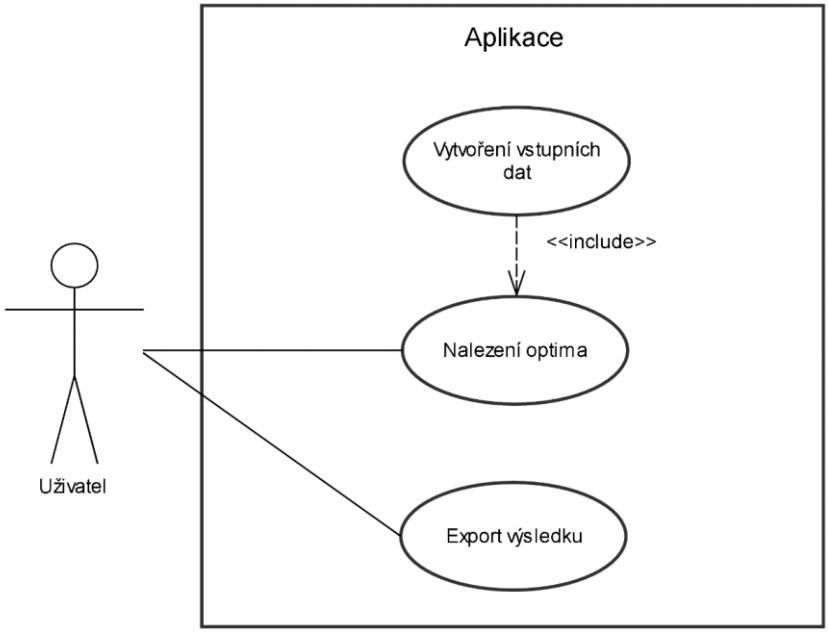
1. Běžné hardwarové nároky.
2. Aplikace poběží na PC s OS Windows.

4.2.3 Use case diagramy

Diagram případu užití popisuje chování systému z pohledu jeho uživatele. Zobrazuje funkcionalitu systému, ale nezobrazuje nic o její vnitřní implementaci.

Z tohoto důvodu je při návrhu nové aplikace většinou vytvářen tento diagram jako první. Jedná se o vyzkoušenou techniku sběru požadavků na funkcionalitu. Případ užití napojený vazbou <<include>> je spuštěný vždy, je-li spuštěn případ, na nějž se odkazuje (Čápka, 2013).

Následuje obrázek 17 popisující diagram případu užití se třemi scénáři, které bude uživatel pravděpodobně často používat.



Obrázek 17 Use case diagram

Use case Vytvoření vstupních dat

Případ užití popisující vytvoření vstupních dat pro běh aplikace, viz tabulka 1.

Název	Vytvoření vstupních dat
Aktéři	Uživatel, Aplikace
Podmínky zahájení	Aplikace se nachází ve výchozím stavu pro zadání nové úlohy
Základní tok	**1.** Uživatel definuje rozsah generovaných dat **2.** Aplikace data vygeneruje a převede do vlastní reprezentace
Alternativní krok	**1.1** Uživatel určí externí zdroj dat

Tabulka 1 Scénář případu užití – Vytvoření vstupních dat

Use case Nalezení optima

Případ užití popisující nalezení optima ve zvolených datech, viz tabulka 2.

Název	Nalezení optima
Aktéři	Uživatel, Aplikace
Podmínky zahájení	Aplikace má k dispozici data k prohledávání
Základní tok	**1.** Zahrnuje (Vytvoření vstupních dat) **2.** Uživatel nastaví parametry pro běh algoritmu **3.** Uživatel spustí algoritmus **4.** Aplikace nalezne řešení **5.** Aplikace zobrazí řešení
Výstupní podmínka	Nalezeno nějaké řešení

Tabulka 2 Scénář případu užití – Nalezení optima

Use case Export výsledku

Případ užití popisující export výsledku, který obsahuje průběh prohledávání dat a nalezený výsledek, viz tabulka 3.

Název	Export výsledku
Aktéři	Uživatel, Aplikace
Podmínky zahájení	Aplikace má k dispozici výsledky řešené úlohy
Základní tok	**1.** Zahrnuje (Nalezení optima) **2.** Uživatel určí místo pro export výsledku **3.** Aplikace shromáždí výsledky **4.** Aplikace uloží výsledky
Výstupní podmínka	Cesta pro uložení výsledků byla korektně zadána

Tabulka 3 Scénář případu užití – Export výsledku

4.2.4 Vstupy

Vstupní data můžou být generována aplikací na základě uživatelem definovaných parametrů nebo jsou importovány z csv souboru.

Každý řádek z externího souboru dat značí jednu prohledávanou dimenzi pro účelovou funkci. U nominálních a binárních dat, máme-li na vstupu deset sekvencí, je pozice každé částice z hejna reprezentována deseti čísly a aplikace zjišťuje, zda se na této pozici nevyskytuje optimum. Data z importovaného csv souboru jsou ukládána v binární, nominální nebo numerické formě podle typu zvoleného při importu dat, viz tabulka 4. Nominálními daty v importovaném souboru mohou být jakékoliv znaky, ovšem při generování umělých nominálních dat jsou generovány pouze znaky (A, B, C, D), které reprezentují znaky v genomu. Binární data v importovaném souboru mohou být 0 nebo 1 a jsou uložený jako datový typ boolean. Numerická data potřebuji na vstupu pouze tolik řádku, kolik má účelová funkce neznámých parametrů. Formát dat je ukázán v následující tabulce. Jednotlivé hodnoty jsou od sebe odděleny mezerou. Jednotlivé sekvence v importovaném souboru mohou mít různou délku.

Numerická	-19 -2 20 105 115 60 0 -12 -55 -150 -70 -21 68 97 20 -180 -220 25 190 500 0 114 3 -500 -700 -1 500
Nominální	A B A A C D D A D C C B A B B C D B C C D A C C A B A A B B B A D C C A A B C D A C C A
Binární	0 1 0 0 1 1 1 0 1 0 0 1 0 0 0 0 1 1 0 1 1 0 1 0 1 1 0 1 1 0 0 1 0 1 0 1 0 1 1 0 0 1 0 1 0 1 1 1 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 0 1 1 0 1 0 0 1 1 0

Tabulka 4 Možný formát vstupních dat

4.2.5 Výstupy

Výstupem je optimum nalezené ve zvoleném problému. V exportu výstupu bude definiční obor, vstupní data, výstupy z náhodně zvolené částice (např. její rychlost, pozice, aktuální fitness, nejlepší fitness dané částice), nalezené globální optimum a čas trvání výpočtu.

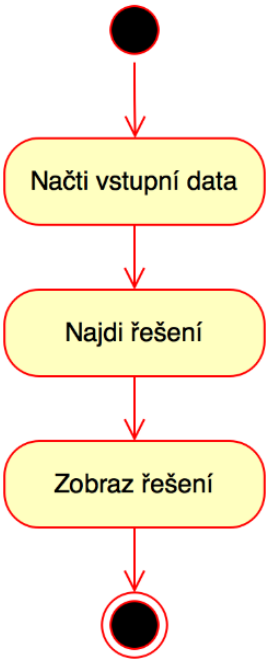
4.3 Diagramy aktivit

Patří mezi základní UML diagramy umožňující znázornit vnitřní chování systému. Modeluje procesy složené z množiny uzlů, které jsou vzájemně propojeny hranami. Převážně se tento diagram používá pro modelování procedurální logiky, procesů a zachycení workflow (Kanisová, 2004).

Další část kapitoly zobrazuje a popisuje chování implementovaného algoritmu.

4.3.1 Hlavní diagram aktivit

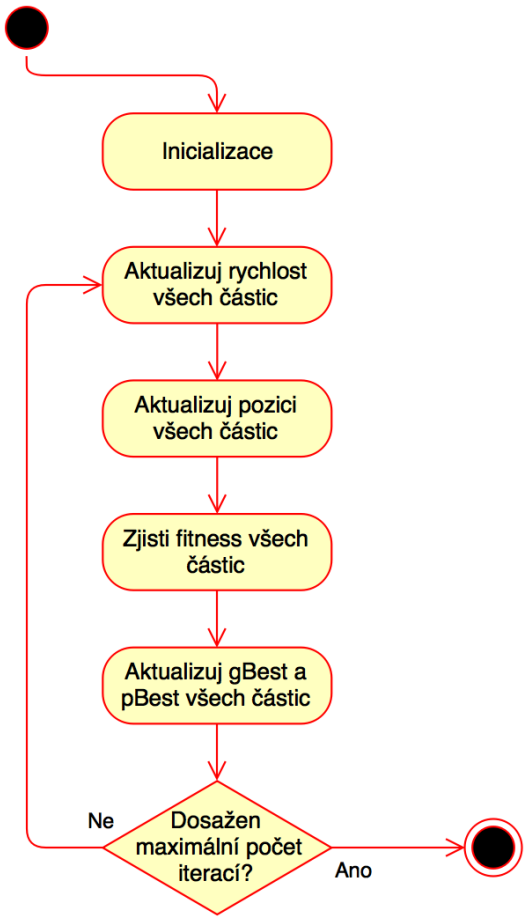
Tento diagram popisuje chod aplikace při spuštění hledání optima, skládá se z pěti částí definujících chod aplikace a jejich posloupnost. Startovacím bodem je výchozí stav aplikace, např. po spuštění. Následuje načtení vstupních dat (např. ze souboru, vygenerování nebo stanovení definičního oboru). Další aktivitou je hledání řešení v datech a poté zobrazení řešení uživateli. Poslední částí je koncový stav, kdy je aplikace připravena k dalšímu použití. Na obrázku 18 je rozebrána aktivita Najdi řešení.



Obrázek 18 Diagram aktivit - Hlavní diagram

4.3.2 Diagram aktivit Najdi řešení

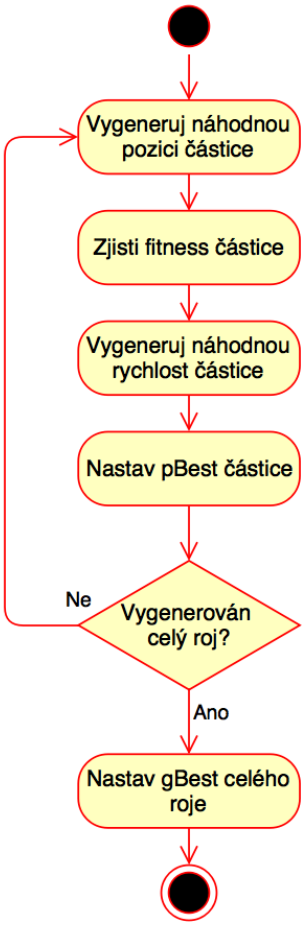
Výchozí bod nastává, má-li aplikace k dispozici data, ve kterých může algoritmus hledat řešení. Nezbytné je také správné nastavení parametrů algoritmu. Pokud je v této chvíli spuštěn algoritmus, začne inicializace roje, viz obrázek 20. Bližší informace o inicializaci roje jsou popsány níže. Další aktivitou je aktualizace rychlosti všech částic, kde jsou vypočteny nové rychlosti na základě původních rychlostí částic, globálního a osobního nalezeného optimálního řešení, viz kapitola 2.8.2. Aktualizace nové pozice částice započítává aktuální rychlost částice a její původní pozici. Následně dochází k výpočtu dané účelové funkce na základě aktuální pozice částice. Po zjištění rychlostí, pozic a ohodnocení všech částic celého roje dochází k aktualizaci osobního optima všech částic a také k aktualizaci globálního optima celého roje. Bylo-li dosaženo maximálního počtu iterací nastavených v parametrech, je algoritmus ukončen, viz obrázek 19.



Obrázek 19 Diagram aktivit - Hledání řešení

4.3.3 Inicializace roje diagram aktivit

Startovacím bodem je spuštění algoritmu PSO. První aktivitou je vygenerování náhodné pozice v prostoru možných řešení a přiřazení pozice částici. Tato pozice je ohodnocena účelovou funkcí a poté je vygenerována náhodná rychlost pro danou částici. Zjištěná hodnota se přiřadí do osobního optima nalezeného danou částicí. Postup je zopakován pro celý roj částic a nakonec je zjištěno globální optimum.



Obrázek 20 Diagram aktivit - Inicializace roje

4.4 Diagramy tříd

Tvorbu složitějších systémů nemůžeme vytvářet bez kvalitního návrhu. Diagram tříd zobrazuje statický pohled na modelovaný systém, což je umožněno pomocí tříd a vztahů mezi nimi. Tento diagram je možné plně přepsat do kódu, měl by být přeložitelný a vhodný pro další práci. Podobné objekty se stejnými vlastnostmi můžeme sdružovat do tříd, kdy pro každou instanci této třídy definujeme její společné vlastnosti a metody.

Vztahy mezi třídami jsou různé. Nejpoužívanější jsou asociace, agregace a kompozice. Dalším používaným vztahem je realizace, která se vyskytuje mezi třídou a interface, která poskytuje vnější pohled na objekt.

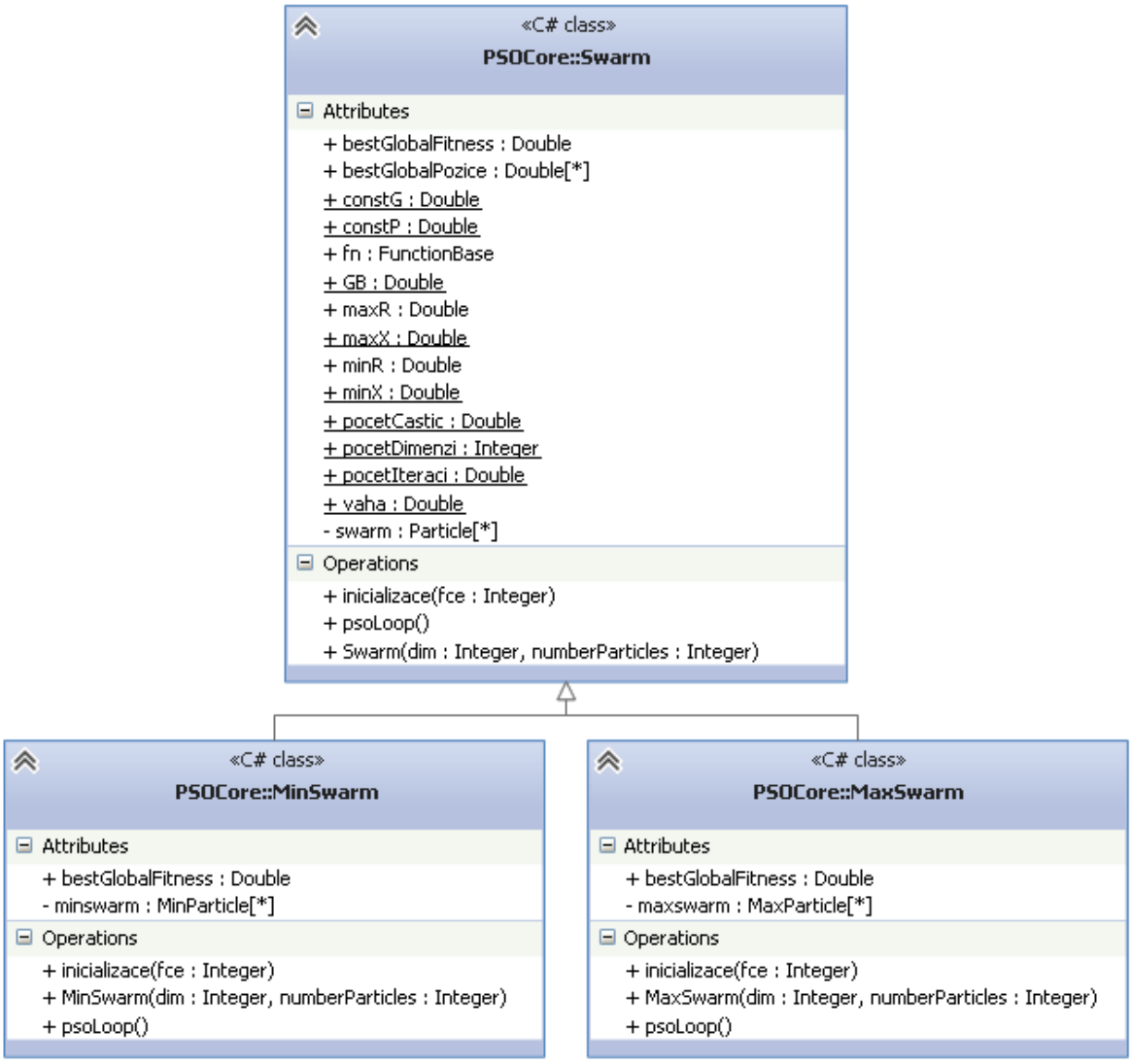
Generalizace

Z pohledu implementace se jedná o vztah dědičnosti. Hlavní výhody dědičnosti jsou lepší správa programového kódu a snížení redundance programového kódu. Potomek je obohacený o atributy nebo metody, které rodič nemá. Jedná se o vztah nadtřída/podtřída. Další výhodou je polymorfismus, kdy podtřída může mít různou vnitřní implementaci dané metody.

4.4.1 Swarm class

Třída Swarm je rodičem tříd MinSwarm a MaxSwarm, viz obrázek 21. Třídy MinSwarm a MaxSwarm se od sebe liší hlavně metodami psoLoop a inicializace, kde byl pomocí polymorfismu různým způsobem implementován vnitřek těchto metod. V třídě Swarm vidíme atributy a metody, které budou používat i potomci MaxSwarm i MinSwarm. Atributy Swarm třídy jsou nezbytné pro správný výpočet a výpis výpočtu. Z názvů těchto tříd je zřejmé, že instanci MinSwarm budeme

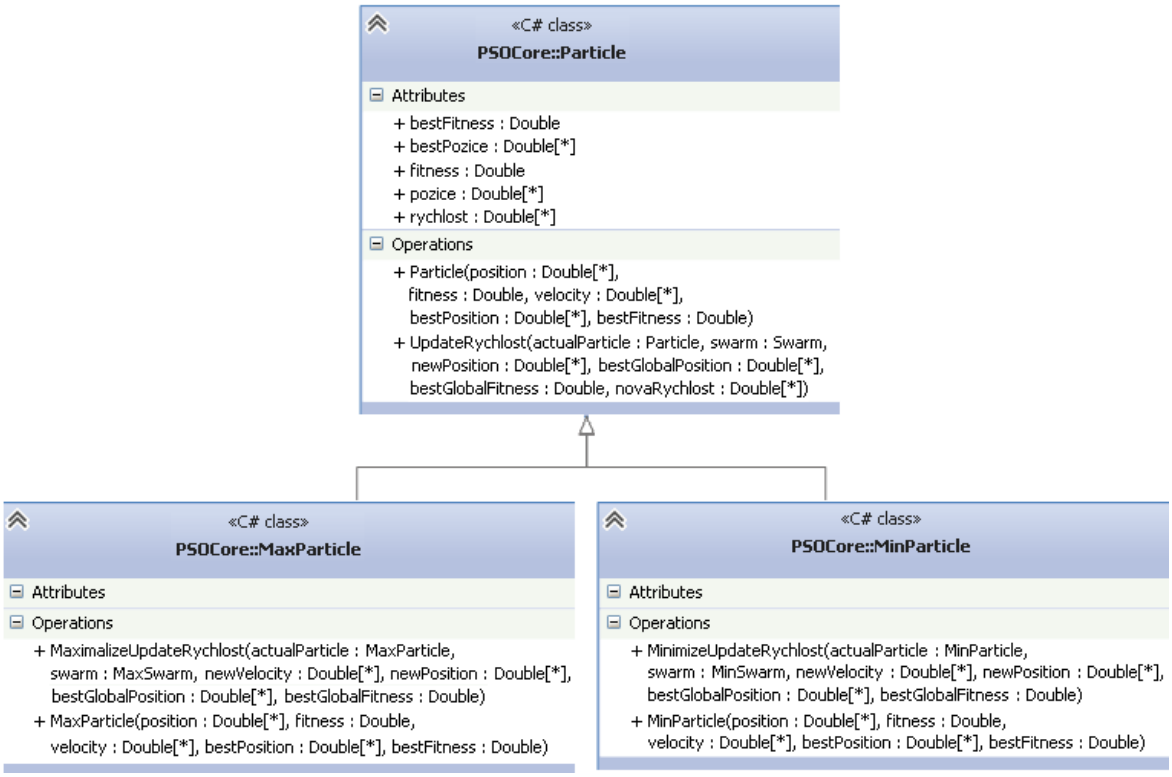
používat při hledání minima účelové funkce a naopak instanci MaxSwarm používat při hledání maxima účelové funkce.



Obrázek 21 Diagram tříd - Swarm

4.4.2 Particle class

Třídy MaxParticle a MinParticle jsou potomci třídy Particle, viz obrázek 22. Třída Particle obsahuje atributy nezbytné pro každou částici v roji (poloha částice v prostoru, rychlost částice, aktuální hodnota účelové funkce, nejlepší dosažená hodnota účelové funkce a její pozice). Potomci se liší metodou pro aktualizaci rychlosti částice. Implementace této metody záleží na tom, jestli hledáme maximum nebo minimum účelové funkce.



Obrázek 22 Diagram tříd - Particle

4.5 Návrh uživatelského rozhraní

Vhodné uživatelské rozhraní je velmi důležité pro každou aplikaci. Rozložení prvků by mělo dodržovat vizuální tok, tzn. pohyb očí uživatele z levého horního rohu okna aplikace do spodního pravého rohu okna aplikace. Uživatelské rozhraní propojuje funkční část aplikace s uživatelem.

Pro návrh efektivního uživatelského rozhraní je potřebná znalost typických uživatelů této aplikace, a proto budou níže charakterizováni. Dále bude popsána logika a návrh uživatelského rozhraní.

4.5.1 Charakteristika uživatelů

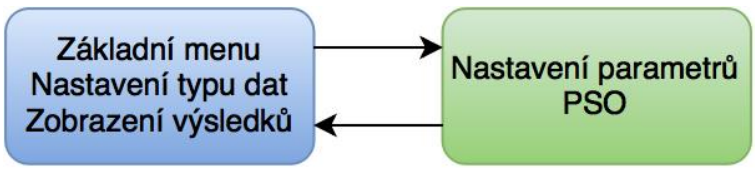
Důležité je před tvorbou jakéhokoliv softwaru zjistit cílovou skupinu uživatelů. V tomto případě se převážně bude jednat o uživatele zabývající se problematikou optimalizace a evolučních algoritmů. Aplikace bude sloužit pro vyzkoušení možností algoritmu PSO a porovnání výsledků s konkurenčními algoritmy.

Uživatelé provádějící tyto experimenty budou mít pravděpodobně znalosti o problematice algoritmů pracujících na principu inteligence roje. Předpokládá se, že uživatel zná možnosti aplikace a optimalizované úlohy, které řeší. Aplikace je demonstrací možností optimalizačního algoritmu PSO, pomocí něhož můžeme optimalizovat různorodé úlohy.

4.5.2 Logická struktura grafického rozhraní

Jedná se o paralelu ke stavovému diagramu, kde každý stav zobrazuje nové okno, příp. přepisuje grafické prvky v okně původního stavu a zobrazuje uživateli požadované informace pro daný stav. Počet stavů a rozložení prvků záleží hlavně na rozsahu aplikace a účelu aplikace. Existují aplikace s jedním stavem, kdy je to nejlepší řešení pro danou aplikaci, protože nepotřebuje pro splnění svého účelu mnoho grafických prvků a nemá další dodatečná nastavení. To zvyšuje přehlednost aplikace a snižuje zátěž na uživatele při jejím používání.

Aplikace vytvořená v rámci této práce obsahuje dva stavy. Startovacím stavem aplikace je přímo hlavní stav, ve kterém uživatel může vybrat typ optimalizovaných dat a řešenou úlohu. Na základě typu zvolených dat se aplikace interaktivně mění pomocí skrývání nepotřebných grafických prvků pro zvolená data a úlohu. Například pokud zvolíme reálná binární data, není nutné zadávat počet prohledávaných sekvencí a typ optimalizované úlohy. Ve startovacím stavu bude aplikace zobrazovat základní výsledky jednotlivých řešených úloh v přehledné tabulce. Dále můžeme z tohoto stavu aplikaci vypnout nebo exportovat podrobnější informace o řešené úloze, viz kapitola 4.2.5. Ze startovacího stavu můžeme kdykoliv přejít do stavu nastavení PSO algoritmu, které je zobrazeno v novém okně. Tato nastavení jsou podstatná nehledě na typ prohledávaných dat a vyčlenění tohoto nastavení mimo startovací stav je z důvodu zvýšení přehlednosti aplikace.

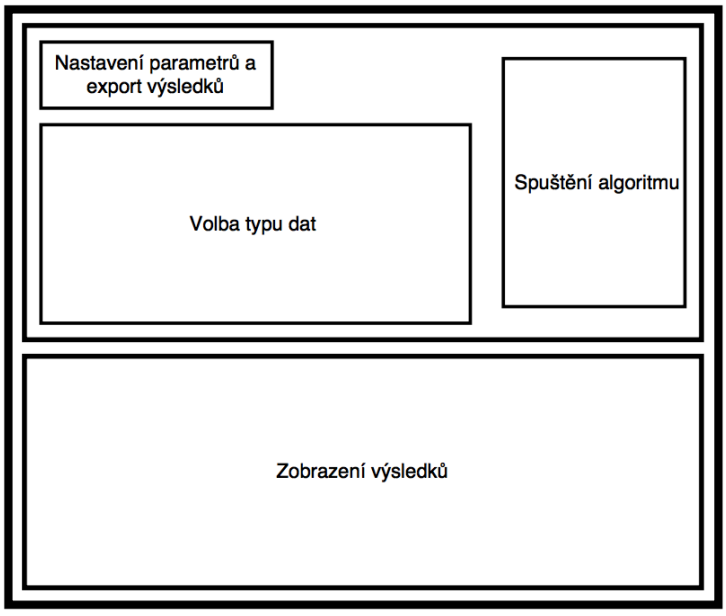


Obrázek 23 Logická struktura grafického rozhraní

4.5.3 Drátěný model

Úspěšně se používá při tvorbě návrhu rozmístění grafických prvků u jednotlivých stavů aplikace, pro každý stav tvoříme nový model. Účelem modelu není naprosto detailní a neměnné rozmístění všech grafických ovládacích prvků, ale jejich přibližné rozložení v daném stavu. V modelu také nejsou zachyceny obrázky nebo barevné grafické prvky. Detailnost návrhu se může měnit a velmi záleží na typu vyvíjené aplikace. Návrh by měl odpovídat požadované funkcionalitě v daném stavu.

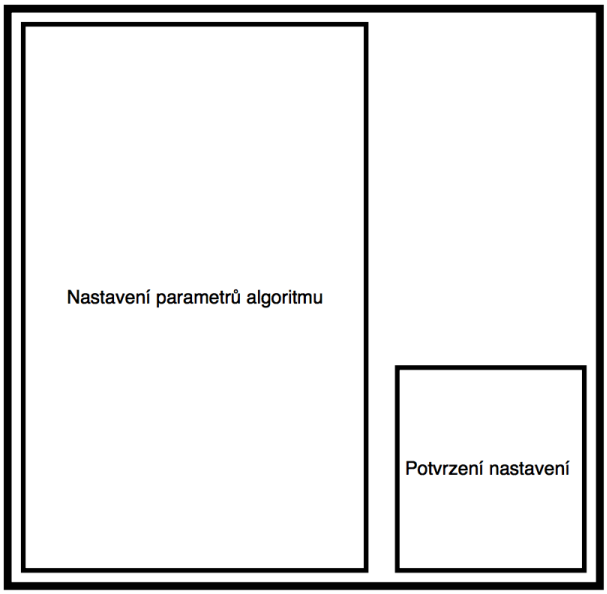
Základní menu by mělo být přehledné a jednoduché, protože se zobrazí po spuštění aplikace. Návrh základního menu v této chvíli zahrnuje možnost volby typu prohledávaných dat a další nastavení se bude měnit podle vybraného typu dat. Bude zahrnuta volba vstupních dat, řešené úlohy a také typ optimalizace (jestli budeme účelovou funkci maximalizovat nebo minimalizovat). Ze základního menu bude možné spustit okno zajišťující nastavení parametrů PSO. Základní menu bude dále obsahovat zobrazení výsledků a možnost spuštění algoritmu, viz obrázek 24.



Obrázek 24 Drátěný model - základní menu

Obrázek 25 zobrazuje model nastavení parametrů algoritmu PSO. Obrazovka je v tomto stavu rozdělena do dvou částí.

V levé části bude možné editovat jednotlivé obecné parametry PSO jako je například počet iterací, počet částic, osobní a globální konstanty upravující rychlost nových částic, konstantu váhy původní rychlosti částice a definiční obor zahrnující prostor možných řešení. V pravé spodní části bude potvrzení nastavení parametrů PSO.



Obrázek 25 Drátěný model - nastavení parametrů

4.6 Implementace

4.6.1 Parametry algoritmu

Uživateli je umožněno v GUI nastavovat parametry a může tak s algoritmem experimentovat a zjišťovat jeho možnosti a tím aplikace splňuje svůj účel. Po spuštění má aplikace nastaveny výchozí parametry algoritmu. Jak bylo zmíněno výše v teoretické části hodnota osobní a globální konstanty je stanovena na hodnotu 4. Nastavené hodnoty parametrů jsou univerzální a lze s nimi najít dobré řešení. Tyto hodnoty byly převzaty jako doporučené od jiných autorů. Podle vlastních pokusů jsem určil, že pro každou účelovou funkci je nutné parametry odladit zvlášť. Nejdůležitějším parametrem je váha původní rychlosti částice (můžeme si tuto hodnotu představit jako setrvačnost částice).

Další podstatné parametry jsou osobní a globální konstanta (určují, ke kterému bodu bude částice směřovat). Níže je uvedená implementace aktualizace rychlosti částice v jedné z dimenzí.

```
novaRychlost[j] = (Swarm.vaha * actualParticle.rychlost[j]) + (Swarm.constP *
                  r1 *(actualParticle.bestPozice[j] –actualParticle.pozice[j]))
                  + (Swarm.constG * r2 * (bestGlobalPosition[j] -
                  actualParticle.pozice[j]))
```

Program 3 : Implementace aktualizace stavu rychlosti částice

V kapitole srovnávající PSO s genetickým algoritmem jsou popsány přesné hodnoty parametrů, při kterých byl PSO nejefektivnější k jednotlivým testovaným účelovým funkcím. Obecně je pro numerická data vhodnější nastavení parametru osobní a globální konstanty na hodnotu 2, ale pro nominální a binární data je vhodnější osobní konstantu nastavit na hodnotu 3 a globální na hodnotu 1. Není to z důvodu různých typů dat, ale z důvodu jiné účelové funkce. U nominálních a binárních dat hledáme nejdelší společnou sekvenci, kdy se jako klíčové ukázalo nastavení osobní konstanty, protože při nastavení vyšší globální konstanty se hledání často zasekne v lokálním optimu.

Výchozí nastavení definičního oboru je od 0 do 100. Tyto hodnoty se při vložení reálných dat z csv souboru automaticky přenastaví a jejich pozdější případná úprava znemožní ideální prohledání dat daného souboru a nalezení jeho optima. Jako minimum je nastavena hodnota 0 a maximum je nastavena délka nejdelšího řádku. Definiční obor u umělých numerických dat znamená rozsah možných hodnot, kterých může daná účelová funkce nabývat a ve kterých PSO hledá optimum. Definiční obor u umělých nominálních a binárních dat znamená počet hodnot v jednotlivých generovaných sekvencích.

Parametry počet částic a počet opakování zvyšují čas prohledávání dat. Při testování bylo zjištěno, že tyto hodnoty je vhodné navyšovat při prohledávání rozsáhlejších dat (např. více dimenzí nebo vyšší rozsah definičního oboru).
Níže je uvedena tabulka výchozích hodnot parametru při spuštění aplikace.

Parametr	Hodnota
Počet opakování	10
Počet částic	10
Definiční obor (maximum)	100
Definiční obor (minimum)	0
Váha	0,75
Konstanta osobního maxima	2
Konstanta globálního maxima	2

Tabulka 5 Výchozí hodnoty parametrů algoritmu

4.6.2 Zobrazení výsledků

Výsledky výpočtů jsou zobrazovány postupně v přehledné tabulce, viz obrázek 26. V prvním sloupci je pořadové číslo výpočtu. Následuje typ prohledávání dat, bylo-li v datech hledáno maximum nebo minimum. Ve sloupci nalezené pozice záleží na zvolených datech, zda jsou numerická, nominální nebo binární. Optimální nalezené řešení u numerických umělých dat je zobrazeno jako množina *n* čísel podle počtu prohledávaných dimenzí, kde první hodnota v řádku patří do první dimenze až n-ta hodnota do n-té dimenze.

Výsledky numerických reálných dat jsou zobrazeny také postupně podle dimenzí, navíc je ale zobrazena pozice v daném řádku (dimenzi) a v závorce hodnota, která se na této pozici nachází. Ve sloupci nalezené pozice u nominálních i binárních dat (reálných i umělých) jsou zapsány řádky (sekvence), ve kterých bylo nalezeno optimální řešení, přičemž může být nalezeno ve dvou, ale i více řádcích. V následujícím sloupci je optimální řešení zapsáno. Může se jednat o řetězec, nebo hodnota účelové funkce záleží na typu zvolených dat. Při exportu podrobnějších výsledků můžeme nejdelší podřetězec najít ve zdrojových datech. Pro ověření výsledků účelové funkce stačí dosadit zjištěné hodnoty do funkce.

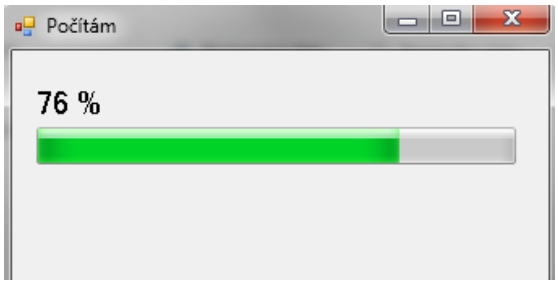
Poslední sloupec RunTime ukazuje čas trvání jednotlivých výpočtů. Mnohem podrobnější informace i o průběhu výpočtu, zdrojových datech atd. dostaneme exportem přes tlačítko Soubor.

P...	Typ	Nalezené pozice	Nalezené řešení	RunTime
1	Maximum	Na řádcích 8, 11,	B A D C C B B A	00:00:02.43
2	Minimum	0,00000 | 0,00000 | 0,00000 | 0,00000 |	0,00000	00:00:00.24
3	Maximum	100,00000 | 100,00000 | 100,00000 | 100,00000 |	40000,00000	00:00:00.34
4	Maximum	Na řádcích 7, 11,	1 1 1 0 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0	00:00:03.45

Obrázek 26 Ukázka zobrazení výsledků

4.6.3 Stav výpočtu

Uživatel je informován o stavu výpočtu lištou a procentem dokončení, viz obrázek 27.



Obrázek 27 Stav výpočtu

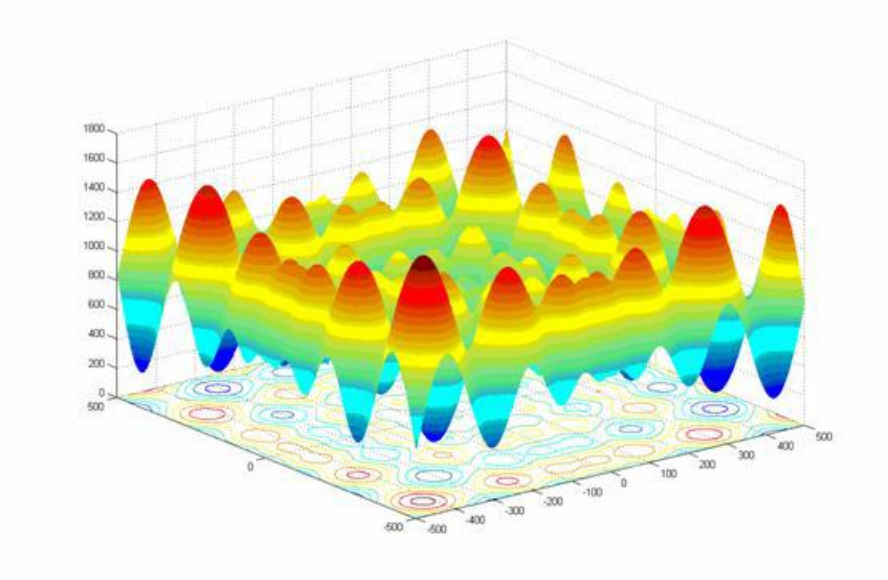
4.7 Ověření správnosti implementace

Zvolit optimální řešení je u některých úkolů obtížné. Například na první pohled nepoznáme v datech společnou nejdelší podsekvenci nebo optimální hodnoty pro účelovou funkci. Můžeme použít příklady, u kterých známe řešení pro ověření funkčnosti aplikace.

Schwefelova funkce je multimodální a proto je vhodná pro ověření funkčnosti algoritmu, neuvízne-li v nějakém lokálním optimu.

$$f(x) = 418.9829 - \sum_{i=1}^{n} (x_i \times \sin\sqrt{|x_i|}) \qquad (1.7)$$

Definiční obor $-500 \leq x_i \leq 500$. Funkce obsahuje několik lokálních minim, přičemž globální se nachází na $x = (1, \ldots, 1), f_{(x)} = 0$



Obrázek 28 Graf Schwefelovy funkce pro n = 2 Zdroj: Hedar (2012)

Po otestování byl algoritmus ve většině případů schopen najít optimum již po 100 opakováních při 100 vygenerovaných částicích při výchozím nastavení a počtu dimenzí $n = 2$. Ověření funkčnosti bylo úspěšné na numerických umělých datech. Stejným způsobem proběhlo i otestování s numerickými reálnými daty rovněž s úspěšným výsledkem. Otestování algoritmu proběhlo i na dalších funkcích jako např. Rastrigin funkce, Easom funkce a Eggholder funkce.

Otestování funkčnosti hledání optima v nominálních a binárních umělých datech je náročné z důvodu náhodného generování sekvencí, proto byla funkčnosti otestována s reálnými daty. Do prvního a posledního řádku zdrojového souboru, který obsahuje 100 sekvencí o délce každé sekvence přibližně 500 hodnot, byla uměle vložena společná podsekvence o délce 30 stejných hodnot. Algoritmus dokázal spolehlivě najít správné řešení ve výchozím nastavení při 100 opakováních a vygenerovaných 100 částicích.

4.8 Srovnání s genetickým algoritmem

Nakonec byla vytvořená implementace PSO algoritmu srovnávána s Genetickým algoritmem pro závěrečné shrnutí možností PSO.

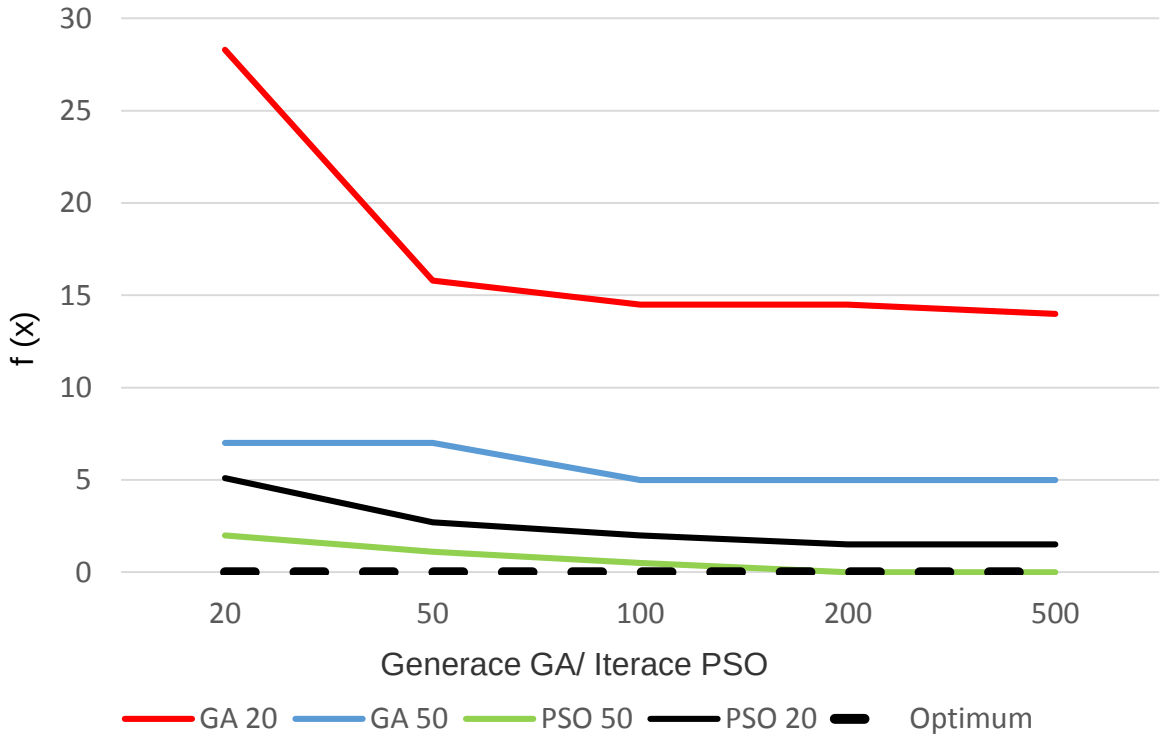
Po vyzkoušení mnoha volně dostupných implementací Genetického algoritmu bylo nakonec použito řešení od autora Jamese McCaffrey. Zdrojové kódy jsou napsány v jazyce C # a parametry algoritmu je možné snadno upravit stejně jako účelovou funkci, viz McCaffrey (2014). Po ověření funkčnosti implementace Genetického algoritmu bylo možné přejít na porovnávání dosažených výsledků. To probíhalo pomocí numerických umělých dat a dosažené výsledky byly následně přehledně srovnány v grafech. Tento typ dat pro porovnání byl zvolen z důvodu snadné porovnatelnosti výsledků a objektivnosti měření. Účelová funkce řešící nejdelší společnou podsekvenci v daném bodě, ve kterém se částice v daném prostoru možných řešení právě nachází, byla navržena pro algoritmus PSO a kdyby tato účelová funkce byla upravena pro Genetický algoritmus, testování by mohlo být neobjektivní. Nominální a binární data byla použita při testování schopnosti najít optimum vzhledem k velikosti dat a vlivu parametrizace. Dosažené výsledky uváděné níže jsou zprůměrovány z 20 běhů algoritmu. Tato práce se nezabývá Genetickým algoritmem, proto byly parametry ponechány ve výchozím nastavení, viz McCaffrey (2014). Ve výchozím nastavení je selekční tlak nastaven na hodnotu 0.4, pravděpodobnost křížení na 0.2 a pravděpodobnost mutací na hodnotu 0.01. Při testování byly pouze upraveny parametry počet generací, velikost populace, definiční obor, počet dimenzí a účelová funkce.

4.8.1 Rastriginova funkce

Obrázek 29 popisuje hledání minima Rastriginovy funkce. Definiční obor je $D\left(f\right) = < -5\,.\,12, 5\,.\,12 >$ pro všechna x_i. Počet dimenzí je $n = 4$. Vidíme, že optimum pro všechna x_i se nachází v bodě 0 a s počtem generací v případě GA, resp. iterací v případě PSO, se k optimu přibližujeme. Po mnoha pokusech s nastavením parametrů jsem optimální nastavení PSO pro tuto funkci zvolil $w\left(t\right) = 0\,,4\,; c_1 = 1\,,8\,;\, c_2 = 1\,,5$ první parametr je váha, dále osobní konstanta a globální konstanta. Zmíněné nastavení nemusí být optimální a je pravděpodobně možné najít lepší nastavení. Na vodorovné ose je vyznačen počet iterací/generací a na svislé ose je vyznačena hodnota účelové funkce.

V legendě máme červeně vyznačeno GA 20, jedná se o zkratku genetického algoritmu o velikosti populace 20 jedinců. Ve stejném pojetí je i legenda PSO 20, kde číslo 20 znamená počet částic. Výsledky vyznačené v grafu vyznívají ve prospěch PSO algoritmu, ovšem odladěním parametrů GA algoritmu můžeme nejspíše dosahovat lepších výsledků. Od 200. iterace už u PSO nedochází ke zlepšení, protože optimum bylo dosaženo.

$$f(x) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i)) \quad (1.8)$$

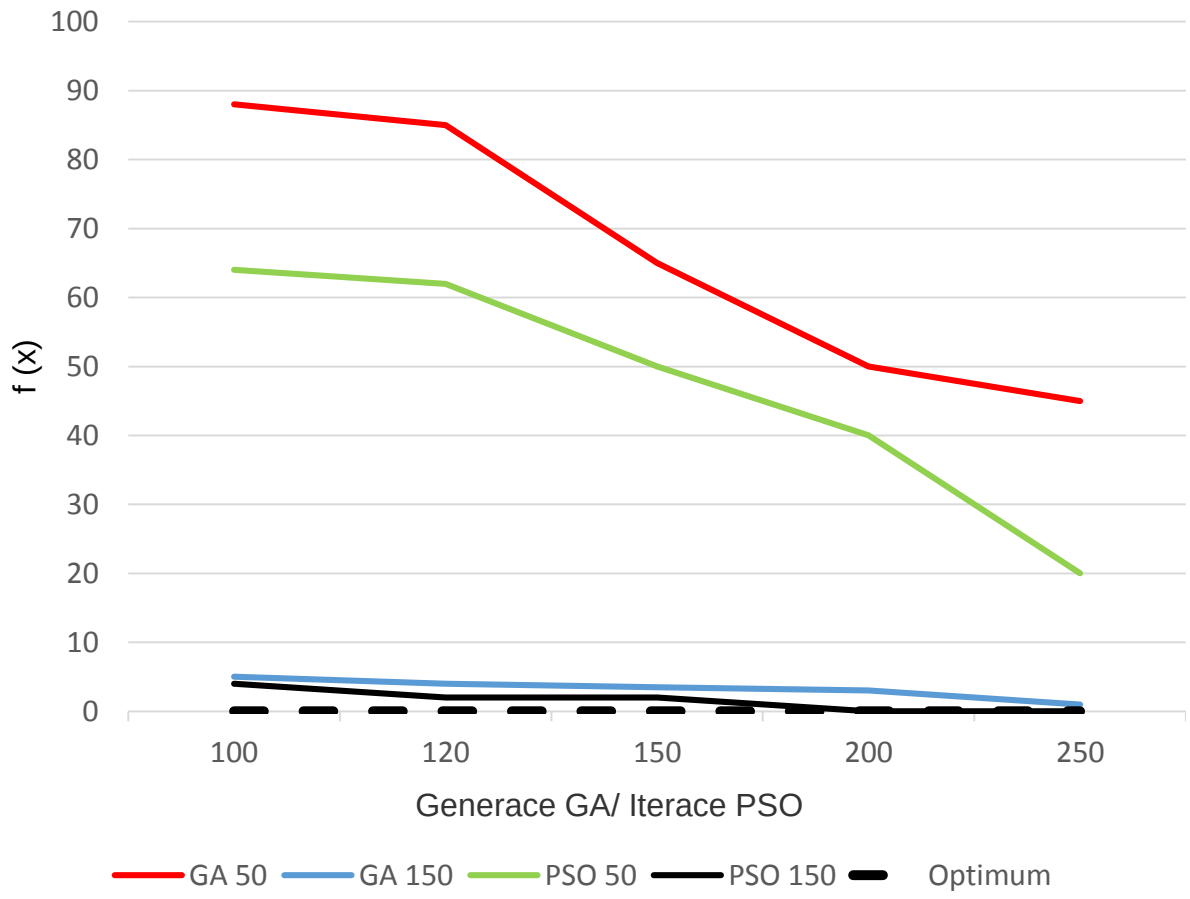


Obrázek 29 Porovnání – Rastriginova funkce

4.8.2 Schwefelova funkce

Obrázek 30 popisuje hledání minima Schwefelovy funkce a v grafu zobrazené dosažené výsledky porovnané s Genetickým algoritmem. Definiční obor je $D(f) = <-500, 500>$ pro všechna x_i při počtu dimenzí $n = 4$. Stejně jako v předchozím příkladu se optimum nachází pro všechna x_i na hodnotě 0.

Po otestování byly zvoleny parametry pro Schwefelovu funkci $w\left(t\right) = 0{,}6; c_{1} = 3;\ c_{2} = 1$. Osy a legendy mají stejný význam jako v předchozím příkladu, pouze byly upraveny počty částic/jedinců a iterací/generací. Výsledky má opět lepší PSO, můžeme to přičíst optimalizaci hodnot parametrů pro danou funkci.



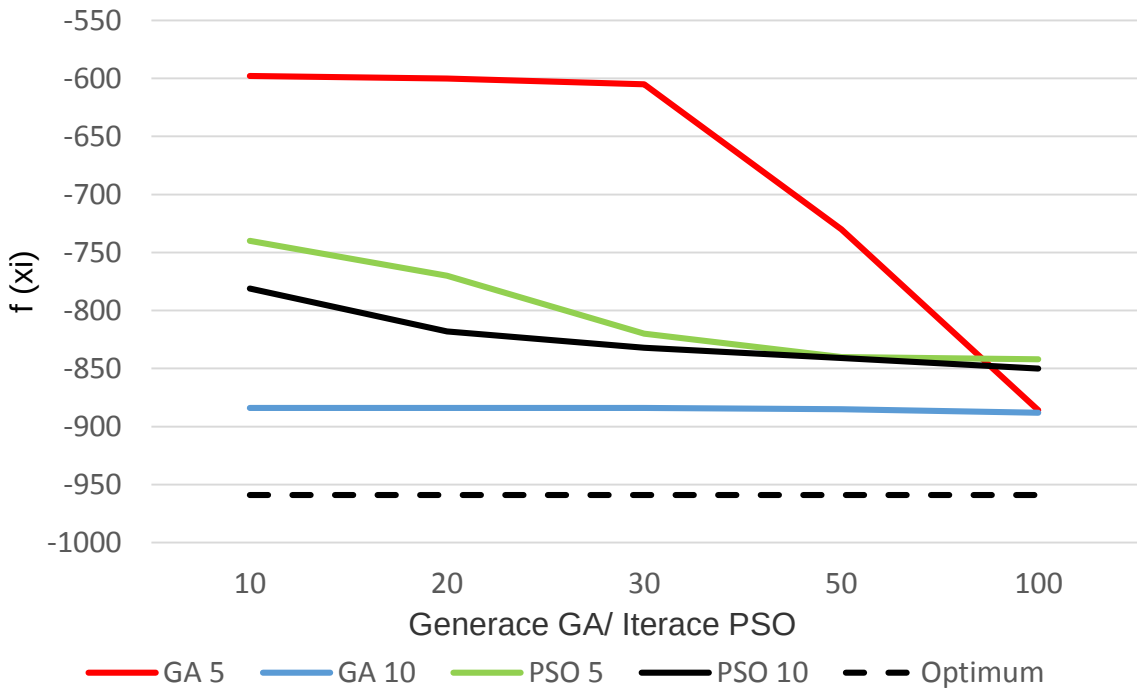
Obrázek 30 Porovnání – Schwefelova funkce

4.8.3 Eggholder funkce

Eggholder funkce je další z multimodálních funkcí. Obrázek 31 koresponduje s předchozím způsobem testování, liší se pouze počtem iterací/generací a částic/jedinců. Níže je uvedená účelová funkce, kde počet dimenzí je $n = 2$. Definiční obor byl stanoven na $D\left(f\right) = < -512, 512 >$ pro x, y. Globální minimum na tomto $D\left(f\right)$ je rovno $f\left(x, y\right) = -959.6407$ v bodě $x = 512$ a $y = 404.2319$. Optimum je na obrázku 31 vyznačeno přerušovanou čarou.

Nejlepší výsledky byly zaznamenány při nastavení PSO $w\left(t\right) = 0{,}5; c_{1} = 2;\ c_{2} = 1.8$. V tomto případě Genetický algoritmus dosahoval lepších výsledků. Pro demonstraci možnosti uvíznutí algoritmu bylo pro testování zvoleno pouze 5 resp. 10 částic a PSO často uvízl v lokálním optimu. Při zvýšení počtu částic PSO mnohem častěji nalezl globální optimum.

$$f\left(x,y\right) = -\left(y + 47\right) \times \sin\left(\sqrt{\left|y + \frac{x}{2} + 47\right|}\right) - x \times \sin\left(\sqrt{\left|x - y - 47\right|}\right) \quad (1.9)$$



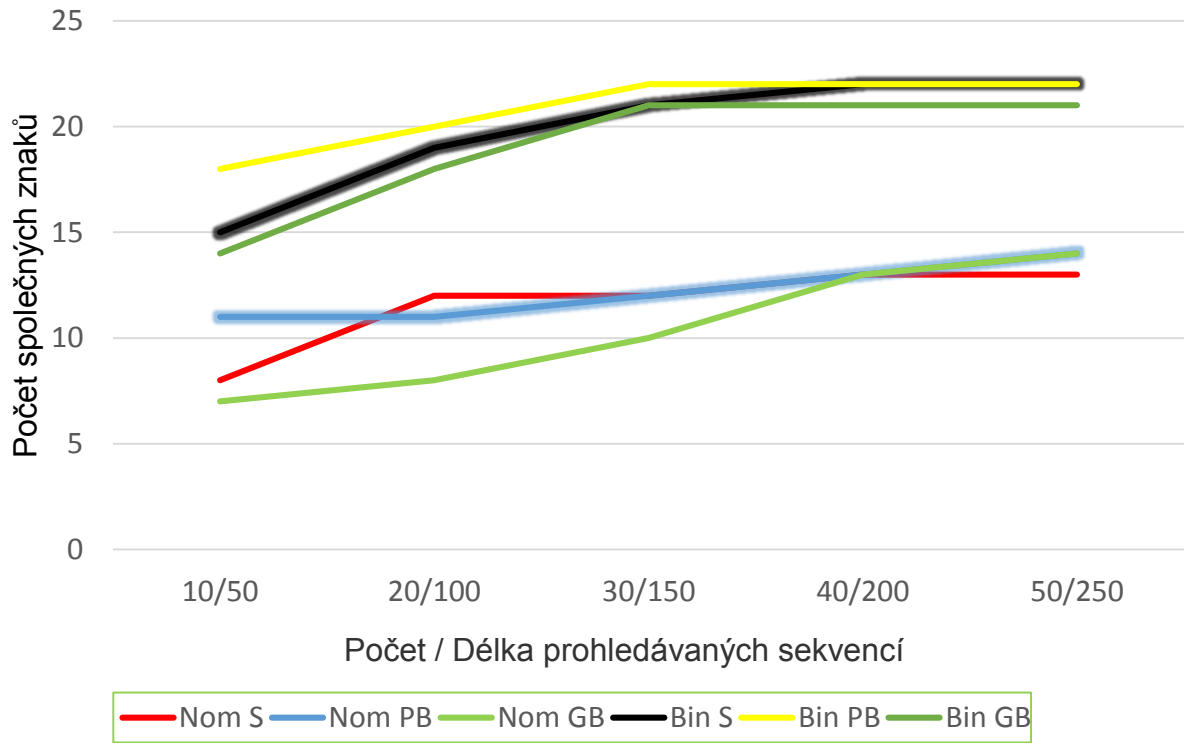
Obrázek 31 Porovnání – Eggholder funkce

4.9 Testování vlivu nastavení parametrů

Dalším testováním bylo hledání nejdelší společné sekvence v nominálních a binárních datech. Testován byl vliv nastavení parametrů na chování PSO při hledání optima a schopnost PSO rychle a úspěšně prohledávat velké množství dat. V tomto případě neznáme optimum v testovaných reálných datech, proto obrázek 32 vypovídá hlavně o vlivu parametrizace na chování PSO.

Na obrázku 32 můžeme vidět, že s vyšším počtem sekvencí a jejich zvyšující se délkou bylo dosaženo lepších výsledků. Dále je patrný zřejmý vliv nastavení parametrů. V legendě označení Nom S značí Nominální data při standardních, resp. výchozích parametrech. Označení Nom PB (PersonalBest) můžeme rozumět jako nominální data s následujícím nastavením parametrů $w(t) = 0.7; c_1 = 2.5; c_2 = 1.5$, kdy je zvýhodňována osobní (individuální) konstanta. Stejným způsobem je pojmenována i značka Nom GB (GlobalBest) s nastavením parametrů $w(t) = 0.7; c_1 = 1.5; c_2 = 2.5$. Zde zvýhodňujeme globální (sociální) konstantu. Obdobně jsou označena i binární data se stejným nastavením parametrů.

S vyšším počtem opakování algoritmu výsledek nakonec konverguje k určitému řešení a dosažené hodnoty jsou podobné. Lepších výsledků dosahujeme při nižším počtu opakování algoritmu při zvýhodnění osobní konstanty (PersonalBest).



Obrázek 32 Hledání optima v binárních a nominálních datech

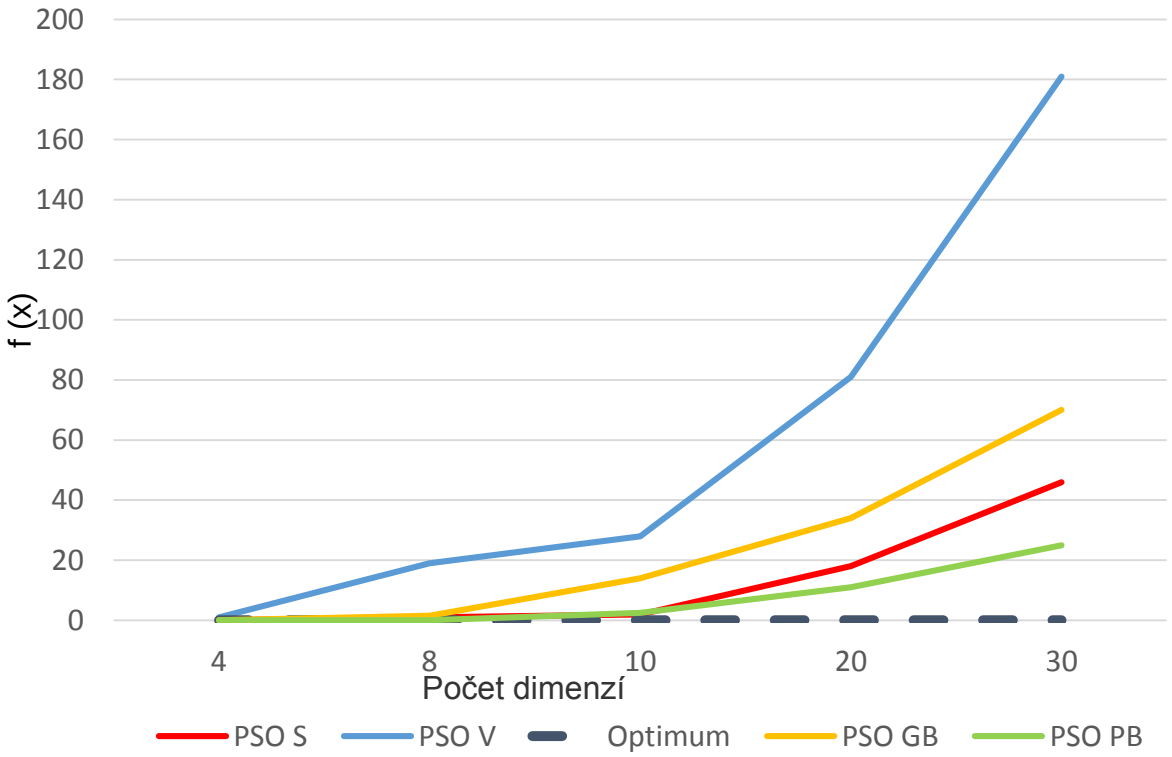
4.10 Schopnost najít řešení vzhledem k velikosti dat

Posledním testováním byla schopnost najít optimum vzhledem k velikosti dat. Při testování byl počet iterací a počet částic stejný. Oba parametry byly nastaveny na hodnotu 500. V legendě přibylo označení PSO V, kde jsou všechny parametry stejné jako u výchozího nastavení, pouze $w\left(t\right) = 1.0$ (váha). Nastavení PSO PB, PSO GB a PSO S je stejné jako v předchozím příkladu. Hledáme minimum Rastriginovy funkce, přerušovaná čára je na hodnotě 0 při $D\left(f\right) = < -5.12, 5\,.12 >$ pro všechna x_i.

Na vodorovné ose je vyznačen počet dimenzí a se zvyšujícím se počtem dimenzí ztrácí algoritmus schopnost najít globální optimum. Zvýšením hodnoty parametru váhy algoritmus velmi brzy se zvýšením počtu dimenzí ztratí schopnost najít dobré řešení. Naopak zvýhodněním osobní konstanty bylo dosahováno nejlepších výsledků.

Při vyšším množství dat je nutné zvýšit i počet částic a iterací, lehce se tím však zvýší čas vyhledání optima. Algoritmus je schopen v rozumném čase najít optimum i ve velkém množství dat, viz obrázek 33.

Vlastní práce



Obrázek 33 Porovnání výsledků vzhledem k velikosti dat - Rastriginova funkce

5 Závěr

Tato diplomová práce se zabývala problematikou algoritmu Particle Swarm Optimization, který řadíme do evolučních algoritmů, přesněji do algoritmů založených na inteligenci hejna. Cílem této práce bylo seznámení se s algoritmem a následně jeho efektivní implementace.

V úvodní části práce je čtenář seznámen s teoretickými základy problémů optimalizace a je zde nastíněno, o jak rozsáhlou problematiku se jedná a do kolika oborů zasahuje. Následuje seznámení s některými evolučními algoritmy.
Práce čtenáře dále podrobně seznamuje s PSO algoritmem a jeho historií. Původ vzniku je velmi zajímavý, zejména proto, že příroda nepřestane lidi fascinovat a inspirovat svojí dokonalostí.

Hledání optimálního řešení u některých typů problémů je natolik náročné, že pokud bychom chtěli vyzkoušet všechny možné kombinace způsobů, jak řešit daný problém, byla by časová náročnost neúnosná. I s využitím současné moderní výpočetní techniky by vyzkoušení všech možných kombinací řešení některých typů problémů nebylo záležitostí pouhých hodin nebo dnů, jednalo by se o řád tisíců a milionů let i déle. V této chvíli přicházejí na řadu promyšlenější způsoby hledání optima místo zkoušení všech možných kombinací. Mezi tyto způsoby patří přírodou inspirované algoritmy.

Mezi požadavky na aplikaci patřila i tvorba vhodného uživatelského rozhraní pro snadné používání a testování algoritmu. Rozhraní bylo navrženo, aby bylo co nejpřehlednější a zároveň plnilo ostatní požadavky jako možnost otestování různých typů dat. Pro podrobnější seznámení s chodem algoritmu zajišťuje aplikace možný výstup popisující postupný výpočet optima a další zajímavé informace. Uživatel aplikace se může sám přesvědčit o možnostech algoritmu řešit různé typy úloh.

Po vytvoření aplikace následovalo ověření funkčnosti algoritmu a schopnosti najít dobré řešení v rozumném čase. Po ověření funkčnosti bylo možné přejít k testování algoritmu a k jeho porovnání s konkurenčním optimalizačním algoritmem.

Cílem nebylo dokázat nadřazenost PSO nad jiným optimalizačním algoritmem, ale mít alespoň přibližné srovnání s podobně fungujícím algoritmem.

Po srovnání s konkurencí lze říci, že navržená implementace PSO je schopná porovnání s jinými optimalizačními algoritmy a dokáže poměrně spolehlivě najít optimum. Při testování byly používány, mimo jiné, multimodální funkce a kvůli nižšímu počtu vygenerovaných částic vzhledem k prohledávanému prostoru vyhledávání občas skončilo v lokálním extrému. Při správném nastavení parametrů a přiměřeném počtu vygenerovaných částic, kterým umožníme konvergovat ke správnému řešení, však algoritmus dokáže najít velmi dobrá řešení, čímž lze považovat cíl práce za splněný.

6 Literatura

Antošovský, J., 2013. Registrace obrazu (Bakalářská práce). Západočeská univerzita v Plzni, Fakulta aplikovaných věd.

Čápka, D., n.d. 2. díl - UML - Use Case Diagram [WWW Document]. URL http://www.itnetwork.cz/navrhove-vzory/uml/uml-use-case-diagram/ (accessed 12.29.15).

Clerc, M., 2012. Standard Particle Swarm Optimisation.

Clerc, M., n.d. random_topology.pdf [WWW Document]. URL http://clerc.maurice.free.fr/pso/random_topology.pdf (accessed 12.29.15).

Dorigo, M., Birattari, M., Stutzle, T., 2006. Ant colony optimization. IEEE Computational Intelligence Magazine 1, 28–39. doi:10.1109/MCI.2006.329691

Dostál, P., 2012. Pokročilé metody rozhodování v podnikatelství a věřejné správě. Akademické nakladatelství CERM.

Dostál, P., 2008. Pokročilé metody analýz a modelování v podnikatelství a veřejné správě. Akademické nakladatelství CERM.

Hedar, A.-R., 2013. Test functions for unconstrained global optimization.

Ježek, M., 2005. Algoritmus Particle Swarm v prostředí Mathematica (Bakalářská práce). Tomas Bata University in Zlín, Faculty of Technology.

Jones, M.T., 2005. GNU/Linux Application Programming. Charles River Media.

Jurčík, L., n.d. Evoluční algoritmy při řešení problému obchodního cestujícího. Brno University of Technology.

Kanisová, H., Müller, M., 2004. UML srozumitelně. Computer Press.

Kvasnička, V., Pospíchal, J., Tiňo, P., n.d. Evolučné algoritmy [WWW Document]. URL https://katalog.mendelu.cz/documents/29683?locale=cs (accessed 12.29.15).

McCaffrey, James, n.d. Artificial Intelligence - Particle Swarm Optimization [WWW Document]. URL https://msdn.microsoft.com/en-us/magazine/hh335067.aspx (accessed 12.29.15a).

McCaffrey, J., n.d. Learning to Use Genetic Algorithms and Evolutionary Optimization [WWW Document]. Visual Studio Magazine. URL https://visualstudiomagazine.com/articles/2014/02/01/evolutionary-optimization-using-c.aspx (accessed 12.29.15b).

Metelková, L., 2006. Analytic Programming applied on operators of evolutional algorithms [WWW Document]. URL http://hdl.handle.net/10563/759

Robinson, J., Rahmat-Samii, Y., 2004. Particle swarm optimization in electromagnetics. IEEE Transactions on Antennas and Propagation 52, 397–407. doi:10.1109/TAP.2004.823969

Truhlář, M., 2009. Modelování vysokofrekvenční diody v programu Comsol Multiphysics. Modeling HF diode in Comsol Multiphysics.

Volná, E., 2002. Neuronové sítě. Ostrava: Ostravská univerzita.

Zelinka, I., 2004. SOMA—self-organizing migrating algorithm, in: New Optimization Techniques in Engineering. Springer Berlin Heidelberg, pp. 167–217.

Zelinka, I., 2002. Umělá inteligence v problémech globální optimalizace. BEN-Technická literatura.

Žižka, J., Chalupová, N., n.d. Informační systémy pro podporu rozhodování [WWW Document]. URL http://docplayer.cz/7043767-Informacni-systemy-pro-podporu-rozhodovani.html (accessed 12.29.15).

Přílohy

Elektronické přílohy

1. Aplikace ve spustitelné podobě.
2. Vzorové vstupní soubory pro nominální, binární a numerická data.
3. Zdrojové kódy implementace.