VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

OPTIMALIZAČNÍ ALGORITMY INSPIROVANÉ PŘÍRODOU

OPTIMIZATION ALGORITHMS INSPIRED BY NATURE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **LENKA BABJARČIKOVÁ**
AUTHOR

VEDOUCÍ PRÁCE — **Doc. Ing. FRANTIŠEK ZBOŘIL, CSc.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

Studentka: **Babjarčiková Lenka**
Program: Informační technologie
Název: **Optimalizační algoritmy inspirované přírodou**
Optimization Algorithms Inspired by Nature
Kategorie: Umělá inteligence

Zadání:

1. Prostudujte zadanou literaturu.
2. Z několika desítek známých algoritmů si vyberte několik z nich (4 až 6) a navrhněte program pro demonstraci jejich činnosti.
3. Navržený program implementujte.
4. Proveďte experimenty s různými praktickými problémy.
5. Porovnejte a zhodnoťte výsledky dosažené jednotlivými algoritmy.

Literatura:

- Agarwal, P., Mehta, S.: Inspired Algorithms: State of Art, Problems and Prospects, International Journal of Computer Applications Volume 100-No.14, August 2014
- Hassanien, A.E., Emary, E.: Swarm Intelligence, Principles, Advances, and Applications, CRC Press, 2016, ISBN: 978-1-4987-4106-4

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Zbořil František V., doc. Ing., CSc.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Táto práca sa venuje štyrom optimalizačným algoritmom inšpirovaných prírodou. Popisuje algoritmus mravčej kolónie, algoritmus párenia včiel, algoritmus vĺčej svorky a algoritmus simulovaného žíhania. Súčasťou tejto práce bolo aplikovanie daných algoritmov pre tri optimalizačné úlohy. Jednou z úloh bol problém obchodného cestujúceho, ktorý je riešený pomocou algoritmu mravčej kolónie, ďalšou nájdenie extrému funkcie, ktoré je riešené algoritmom vĺčej svorky a algoritmom simulovaného žíhania a poslednou úlohou bol problém spĺniteľnosti logických formúl, ktorý bol v tejto práci riešený algoritmom párenia včiel. Práca obsahuje experimenty s danými algoritmami a vyhodnocuje získané výsledky.

Abstract

This thesis deals with four optimization algorithms inspired by nature. It describes ant colony optimization algorithm, marriage in honeybees optimization algorithm, grey wolf optimization algorithm and simulated annealing algorithm. The main part of this thesis is the application of these algorithms for solving three optimization problems. One of the problems is travelling salesman problem, which is solved by ant colony optimization, next problem is searching for extreme of function solved by grey wolf optimization and simulated annealing algorithms and the last is boolean satisfiability problem solved by marriage in honeybees optimization algorithm. Thesis contains experiments with these algorithms and reviews gained results.

Kľúčové slová

optimalizačné algoritmy, algoritmus mravčej kolónie, algoritmus párenia včiel, algoritmus vĺčej svorky, algoritmus simulovaného žíhania, problém obchodného cestujúceho, problém splniteľnosti logických formúl, hľadanie extrému funkcie

Keywords

optimization algorithms, ant colony optimization algorithm, marriage in honeybees algorithm, grey wolf optimization algorithm, simulated annealing algorithm, travelling salesman problem, boolean satisfiability problem, searching for extreme of a function

Citácia

BABJARČIKOVÁ, Lenka. *Optimalizační algoritmy inspirované přírodou.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. František Zbořil, CSc.

Optimalizační algoritmy inspirované přírodou

Prehlásenie

Prehlasujem, že som túto prácu vypracovala samostatne pod vedením pána Doc. Ing. Františka V. Zbořila CSc. Uviedla som všetky literárne pramene a publikácie, z ktorých som čerpala.

.....
Lenka Babjarčiková
13. mája 2019

Poďakovanie

Týmto by som sa chcela poďakovať pánovi Doc. Ing. Františkovi V. Zbořilovi CSc. za jeho čas a cenné odborné rady pri písaní tejto bakalárskej práce. Taktiež by som sa chcela poďakovať svojim rodičom a priateľom za ich podporu nie len počas písania tejto práce, ale aj počas celého doterajšieho štúdia.

Obsah

Zoznam použitých skratiek

ACO	Ant Colony Optimization
ACS	Ant Colony System
AS	Ant System
CSP	Constraint Safisfaction Problem
GWO	Grey Wolf Optimization
MBO	Marriage in Honey-Bee Optimization
MMAS	MAX-MIN Ant System
MVC	Model-View-Controller
SA	Simulated Annealing
SAT	Boolean satisfiability problem
TSP	Travelling Salesman Problem

Kapitola 1

Úvod

Cieľom tejto práce bolo vybrať si niekoľko optimalizačných algoritmov inšpirovaných prírodou a naštudovať si ich problematiku. Na základe tejto problematiky pre algoritmy navrhnúť a implementovať program, ktorý demonštruje ich činnosť, a pomocou neho vykonať experimenty na rôznych praktických problémoch a zhodnotiť ich výsledky.
Vybrané boli štyri algoritmy - algoritmus mravčej kolónie, algoritmus vlčej svorky, algoritmus párenia včiel a algoritmus simulovaného žíhania. Druhá kapitola obsahuje ich podrobnejší popis spolu s analógiou aká bola použitá pri vzniku ich umelého analogického modelu. Tento analogický model ďalej slúžil ako podklad pre ich samotnú implementáciu.
V tretej kapitole sú opísané riešené optimalizačné problémy. Menovite to sú - problém obchodného cestujúceho, v ktorom je hľadaná najkratšia cesta medzi zadanými mestami, problém hľadania extrému funkcie, kde má algoritmus nájsť globálne optimum na danom intervale pre funkciu o dvoch premenných a posledným problémom je nájdenie takej kombinácie hodnôt, pre premenné zadanej logickej formuly, aby spĺňala každú z jej klauzúl.
Nasledujúca štvrtá kapitola opisuje vytvorený program pre vykonanie daných algoritmov a problémov nimi riešených. V kapitole je opísaná štruktúra programu, aké technológie boli pre jeho tvorbu využité a stručne popisuje vytvorené užívateľské rozhranie programu.
Piata kapitola obsahuje zhrnutie vykonaných experimentov pre dané problémy a algoritmy, ktoré tieto problémy riešili. Problém TSP bol riešený algoritmom ACO, hľadanie extrému funkcie algoritmami GWO a SA, a splniteľnosť logických formúl bolo riešené pomocou algoritmu MBO. V tejto kapitole sú zhrnuté výsledky algoritmov. Konkrétne výsledky jednotlivých testov pre experimenty sú uvedené v prílohe práce.
V závere práce sú zhrnuté nadobudnuté poznatky o vybraných algoritmoch za základe získaných výsledkov.

Kapitola 2

Optimalizačné algoritmy inšpirované prírodou

Práve príroda sa stala veľkým zdrojom inšpirácie pri hľadaní nových optimalizačných algoritmov hlavne vďaka dlhému časovému obdobiu jej evolúcie a vylepšovania.
Algoritmy použité v tejto práci boli inšpirované biologickými javmi v prírode a sociálnym správaním živočíchov. Tieto algoritmy sa delia do nasledujúcich kategórii - algoritmy inšpirované správaním zvierat a algoritmy inšpirované fyzikálnymi a chemickými javmi.
Algoritmy inšpirované prírodou je možné rozdeliť do ďalších skupín na - algoritmy založené na skupinovej inteligencii, algoritmy inšpirované biológiou, algoritmy inšpirované fyzikálnymi a chemickými javmi a ostatné algoritmy. Táto kapitola opisuje štyri optimalizačné algoritmy, z ktorých dve patria do kategórie algoritmov založených na inteligencii skupiny (Algoritmus mravčej kolónie - ACO, Algoritmus vĺčej svorky - GWO), ďalší algoritmus (Algoritmus párenia včiel - MBO) sa radí k algoritmom inšpirovaných biológiou a štvrtý algoritmus (Algoritmus simulovaného žíhania - SA) patrí do kategórie algoritmov inšpirovaných fyzikálnymi a chemickými javmi. [12]
Opísané algoritmy sa často popisujú ako stochastické a metaheuristické. Čo znamená, že narozdiel od deterministických, pracujú s náhodnosťou. Pre deterministický algoritmus je charakteristické nájsť pri rovnakom vstupne stále rovnaký výsledok. Pri stochastickom algoritme to takto nie je, pretože pracujú s istou formou náhodnosti. Tieto algoritmy nedokážu stále garantovať nájdenie optimálneho riešenia. Avšak ich hlavným cieľom je nájdenie čo najkvalitnejšieho riešenia pre problémy, ktorých zložitosť nedovoľuje nájsť optimálne riešenie v reálnom čase deterministickým algoritmom, alebo by vyžadovala priveľké množstvo výpočetných zdrojov.
Pri metaheuristických algoritmoch predpona meta- znamená vyššie alebo na vyššej úrovni. Táto o niečo vyššia úroveň je dosiahnutá spojením použitej náhodnosti spolu s lokálnym vyhľadávaním čo daný algoritmus dokáže pri hľadaní riešenia vylepšiť. Náhodnosť vie tieto algoritmy často krát oslobodiť od uviaznutia v lokálnom optime a teda nájsť aj globálne optimálne riešenie. [20]

2.1 Algoritmy založené na inteligencii skupiny

Inteligencia skupiny (angl. Swarm Intelligence) opisuje kolektívne vyvíjajúce sa správanie viacerých jedincov, ktorí spolu interagujú na základe istých pravidiel. Každého jedinca je možné považovať ako neinteligentného, no celý systém viacerých jedincov ukazuje známky

samo-organizovaného správania a teda javí sa ako istý druh kolektívnej inteligencie. Hlavné vlastnosti algoritmu založeného na inteligencii skupiny sú:

- Zdieľanie informácii medzi viacerými agentami.
- Agenti sú samo-organizovaní a spoločne sa vyvíjajú.
- Je veľmi efektívny kvôli svoju kolaboratívnemu učeniu.
- Je možné ho jednoducho paralelizovať pre praktické a real-time problémy. [12]

2.1.1 Algoritmy mravčej kolónie

Model správania sa kolónie mravcov je jedným z najúspešnejších modelov založených na inteligencii skupiny pre optimalizáciu a tento vyhľadávací algoritmus bol inšpirovaný správaním skutočných mravcov v prírode. [2]
Na základe daného modelu vzniklo mnoho variácii algoritmov inšpirovaných správaním mravčích kolónii. Tieto algoritmy po prvý krát predstavil Marco Dorigo a jeho tým v 90. rokoch. Mravce sú sociálnym hmyzom a ich správanie je zamerané na cieľ prežitia celej kolónie a nie na jedinca. [4]
Algoritmy mravčej kolónie vznikli na základe pozorovania správania sa mravcov pri spôsobe akým si vedia zadovážiť potravu. Hlavným zameraním týchto algoritmov je postup akým si mravce vedia nájsť čo najkratšiu cestu medzi polohou ich mraveniska a miestom, kde sa nachádza zdroj potravy. Spôsob akým mravce zdroj potravy nájdu je často krát založený na náhodnom prehľadávaní okolia mraveniska. Pričom pri svojom pohybe mravce na zemi zanechávajú chemickú feromónovú stopu, ktorú vedia ucítiť nie len oni sami, ale aj všetky ostatné mravce. Koncentrácia feromónovej stopy zanechanej na ceste slúži ako vodítko pre mravcov akou cestou sa vydať k zdroju potravy.
Pri nájdení zdroja potravy mravce daný zdroj ohodnotia na základe jeho kvality a kvantity a prinesú jeho späť do mraveniska jeho časť. Po ceste do mraveniska mravce zanechávajú feromónovú stopu podľa kvality a kvantity zdroja potravy.
Toto zanechávanie feromónových stôp sa nazýva stigmergia a slúži mravcom ako spôsob nepriamej komunikácie medzi sebou a tiež slúži k tomu aby mravce našli stále najkratšiu cestu k zdroju potravy. [8]

Umelý analogický model

Je niekoľko rozdielov medzi chovaním skutočných mravcov a mravcov, ktorých správanie je použité v modele ACO, dané rozdiely sú nasledujúce:

- Skutočné mravce sa pohybujú asynchrónne, umelé synchrónne.
- Skutočné mravce sa pri návrate do mraveniska riadia feromónovými stopami, umelé sa v každom cykle vracajú do mraveniska po rovnakých cestách akými prišli.
- Skutočné mravce vypúšťajú feromón neustále, umelé len pri návrate do mraveniska.
- Chovanie mravcov je založené na implicitnom vyhodnocovaní ciest, t.j. že pohyb po kratších cestách skutočným mravcom kratšiu dobu, preto môžu tieto cesty opakovať častejšie a tým sa feromónu na týchto cestách zvyšuje. Umelé mravce vyhodnocujú cesty explicitne, t.j. pri návrate do mraveniska podľa kvality a dĺžky cesty.
- Skutočné mravce sú prakticky slepé a bez pamäte cesty, umelé mravce majú

Algoritmus

Nasledujúci algoritmus popisuje jeden zo základných postupov priebehu ACO algoritmu.

```
Inicializácia hodnôt parametrov - počet mravcov, miest, iterácii, hodnoty α a β.
Inicializácia pozícii miest a ciest tak aby bolo spojené každé mesto s každým.
Inicializácia hodnôt Najlepšieho mravca.
for Počet iterácii do
    foreach Mravec do
        Náhodné priradenie začiatočného mesta.
        Označenie začiatočného mesta ako navštíveného.
        while Existuje nenavštívené mesto do
            Výpočet pravdepodobností navštívenia miest, ktoré ešte neboli
             navštívené a sú spojené s aktuálnym mestom.
            Výber a navštívenie vybraného mesta.
        end
    end
    foreach Cesta do
        foreach Mravec do
            if Mravec prešiel danou cestou then
                Pripočítanie množstva feromónu pre danú cestu.
            end
        end
    end
    Výber najlepšieho mravca danej iterácie.
    if Najlepší mravec danej iterácie je lepší ako Najlepší mravec then
        Najlepší mravec = Najlepší mravec danej iterácie.
    end
    Vygenerovanie nových mravcov.
end
Najlepší mravec je výsledkom algoritmu.
```

Algoritmus 1: Všeobecný algortimus ACO

Medzi najrozšírenejšie a najpoužívanejšie variácie ACO algoritmov patria Ant System - AS, Ant Colony System - ACS a MAX-MIN Ant System - MMAS.

Ant System

Ant System bol prvým z ACO algoritmov popísaných a publikovaných v literatúre. [10] Pre AS je charakteristické, že hodnoty feromónu upravuje každý mravec, ktorý prešiel celým grafom. Celé riešenie AS sa skladá z hrán grafu c_{ij} a hodnoty feromónu danej hrany τ_{ij}. Zápis τ_{ij} predstavuje hodnotu feromónu danej cesty spájajúcej miesta i a j a jeho zmena sa počíta pomocou rovnice (2.1).

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{2.1}$$

kde $\rho \in (0, 1]$ je miera evaporizácie, m je počet mravcov a $\Delta\tau_{ij}^{k}$ je množstvo feromónu nechaného na hrane (i, j) k-tým mravcom, opísané rovnicou (2.2).

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{L_k} & \text{ak mravec k použil hranu (i,j) vo svojom riešení} \\ 0 & \text{inak} \end{cases} \tag{2.2}$$

kde L_k je dĺžka cesty, ktorú prešiel mravec k.
Pri hľadaní riešení mravce prechádzajú grafom a v každý uzol navštívia s istou pravdepodobnosťou, teda v každom uzle musia urobiť isté pravdepodobnostné rozhodnutie kam pôjdu ďalej. Pravdepodobnosť $p(c_{ij}|s_k^p)$, že sa mravec k presunie z miesta i do mesta j je počítaná rovnicou (2.3).

$$p(c_{ij}|s_k^p) = \begin{cases} \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{c_{ij} \in N(s_k^p)} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}} & \text{ak } j \in N(s_k^p) \\ 0 & \text{inak} \end{cases} \tag{2.3}$$

kde $N(s_k^p)$ je množinou hrán, ktorými ešte mravec neprešiel, teda hrán ktoré zatiaľ nepatria do čiastočného riešenia s_k^p mravca k (nepatria do množiny hrán, ktoré už mravec prešiel),α a β sú parametre, ktoré určujú relativitu dôležitosti feromónu voči heuristickej informácii $\eta_{ij} = \frac{1}{d_{ij}}$, kde d_{ij} je dĺžkou časti c_{ij}, teda hrany (i, j). [7]

Ant Colony System

Ant Colony System ACS je vylepšením algoritmu Ant System. [9] Hlavným rozdielom ACS od AS je spôsob akým sa mravce rozhodujú pri výbere nasledujúceho miesta pri formovaní celkového riešenia. ACS predstavuje tzv. pseudonáhodné pravidlo úmernosti. Toto pravidlo hovorí, že pravdepodobnosť s ktorou sa mravec presunie z miesta i do miesta j závisí na náhodnej premennej q, ktorá má uniformné rozdelenie na intervale $[0, 1]$ a parametru q_0. Ak je $q \leq q_0$ potom spomedzi prijateľných častí riešenia (hrán) je vybraná tá časť (hrana), ktorá maximalizuje hodnotu feromónu, inak je použitá rovnaká rovnica ako pri AS.
Toto viac chamtivé pravidlo, ktoré uprednostňuje využitie informácie o feromóne je vyvážené ďalšou zmenou oproti AS a to pridaním lokálnej zmeny feromónu. Lokálnu zmenu feromónu robí každý mravec po každom kroku. Každý mravec ju aplikuje len na poslednú prejdenú hranu podľa rovnice (2.4).

$$\tau_{ij} = (1 - \varphi) \cdot \tau_{ij} + \varphi \cdot \tau_0, \tag{2.4}$$

kde $\varphi \in (0, 1]$ je koeficient feromónového úpadku, a τ_0 je začiatočná hodnota feromónu.
Hlavným cieľom lokálnej zmeny je obmena hľadania vykonávaná nasledujúcimi mravcami v jednej iterácii. Znižovanie koncentrácie feromónu na hranách ich prechádzaním počas jednej iterácie vyzýva nasledujúcich mravcov aby si zvolili iné hrany, a preto daný prístup môže poskytovať rozličné riešenia. Toto spôsobuje nižšiu šancu získania rovnakých riešení niekoľkými mravcami počas jednej iterácie. Kvôli zavedeniu lokálnej zmeny v ACS, musia byť limitované minimálne hodnoty feromónu.
Tak ako v AS, aj v ACS sa na konci hľadania riešenia urobí posledná zmena feromónu. V ACS urobí túto poslednú feromónovú zmenu len najlepší mravec. Len hrany, ktoré boli prejdené najlepším mravcom budú mať zmenený feromón podľa rovnice (2.5).

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij}^{best} \tag{2.5}$$

kde $\Delta\pi_{ij}^{best} = \frac{1}{L_{best}}$ ak najlepší mravec prešiel hranou (i, j) v jeho ceste, $\Delta\tau_{ij}^{best} = 0$ inak (L_{best} je možné nastaviť buď ako dĺžku najlepšej cesty nájdenej v aktuálnej iterácii, alebo ako dĺžku cesty riešenia, ktoré bolo nájdené nájdené od začiatku algoritmu).
Je nutné spomenúť, že väčšina vylepšení zavedená ACS bolo spomenutých prvý krát v algoritme Ant-Q, ktorý bol predbežnou verziou od rovnakých autorov. [7]

MAX-MIN ant system

MAX-MIN ant system (MMAS) je ďalším vylepšením, ktoré bolo navrhnuté pre pôvodnú myšlienku AS. [18]
MMAS sa od AS rozlišuje v:

- len najlepší mravec zanecháva feromónové stopy
- minimálne a maximálne hodnoty feromónu sú explicitne limitované

Zmena feromónu je popísaná rovnicou (2.6),

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \Delta\tau_{ij}^{best} \tag{2.6}$$

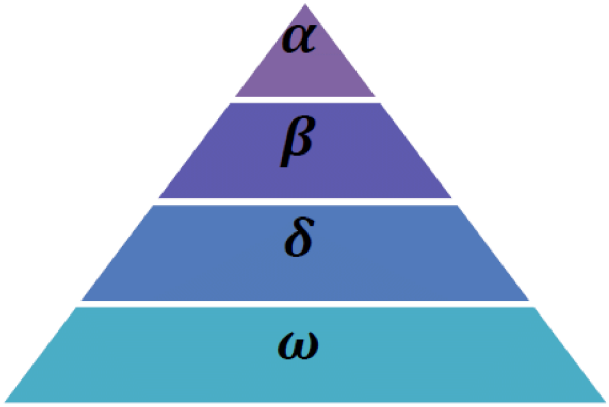
kde $\Delta\tau_{ij}^{best} = \frac{1}{L_{best}}$ ak najlepší mravec prešiel hranou (i, j) v jeho ceste, inak $\Delta\tau_{ij}^{best} = 0$, kde L_{best} je dĺžka cesty najlepšieho mravca. Tak ako v ACS, L_{best} môže byť nastavená buď ako najlepšie riešenie pre danú iteráciu alebo najlepšie riešenie od začiatku algoritmu, ale aj ako kombinácia oboch.
Hodnoty feromónu sú obmedzené medzi τ_{min} a τ_{max}.
τ_{ij} je nastavená na τ_{max} ak $\tau_{ij} > \tau_{max}$ a na τ_{min} ak $\tau_{ij} < \tau_{min}$. Je dôležité poznamenať, že rovnica pre zmenu feromónu MMAS je aplikovaná, tak ako pre AS na všetky hrany, kým v ACS je aplikovaná len na hrany, ktorými prešli najlepší mravci.
Minimálna hodnota τ_{min} je najčastejšie zvolená experimentálne (existuje teória navrhnutá v práci [18] ako túto hodnotu definovať analyticky). Maximálna hodnota τ_{max} môže byť vypočítaná analyticky ak je známa optimálna dlžka cesty mravca. V prípade TSP, $\tau_{max} = \frac{1}{\rho \cdot L^*}$, kde L^* je dĺžkou optimálnej cesty. Ak nie je L^* známa, môže byť zadaná približne podľa najlepšej cesty, ktorá sa našla od začiatku algoritmu. taktiež je nutné poznamenať, že začiatočná hodnota stopy je nastavená na τ_{max} a algoritmus je reštartovaný ak nie je pozorované žiadne zlepšenie po istom počte iterácii. [7]

2.1.2 Algoritmus vlčej svorky

Tento algoritmus je založený na skupinovej inteligencii vlčej svorky. Bol inšpirovaný spôsobom aký vlci používajú pri hľadaní a love svojej obete. Algoritmus GWO bol predstavený v článku [15] v roku 2014.
Vlk dravý (lat. Canis Lupus) je druh cicavca z čeľade psovitých. Vlci sú považovaní za špičkových dracov,a zväčša preferujú lov vo svorke. Počet vlkov vo svorke sa pohybuje priemerne v rozmedzí 5-12. Najzaujímavejšou vlastnosťou na každej svorke je jej veľmi striktná sociálna dominantná hierarchia, ktorej symbolika je znázornená na Obrázku 2.1.
Táto hierarchická pyramída zobrazuje štyri stupne postavenia, aké vlk vo svorke môže zastávať. Každé z týchto postavení má svoje práve a povinnosti, ktoré musí dodržiavať.

Vlk v postavení alfa (ďalej spomínaný ako alfa) je považovaný za vodcu celej svorky. Môže byť ním samec, ale aj samica. Alfa nemusí byť nutne fyzický najsilnejším vlkom vo svorke pretože pri voľbe vodcu zohráva veľkú rolu aj organizačná schopnosť vlka a jeho disciplína. Alfa udáva všetky hlavné rozhodnutia ohľadom lovu, miesta na prespanie, času prebúdzania atď. Tieto rozhodnutia sú dané pre celú svorku a každý vlk ich musí dodržiavať.

Vlk v postavení beta (ďalej spomínaný ako beta) je podriadení alfe a pomáha pri rozhodovaní. tiež slúži ako prostredník medzi svorkou a alfou, presnejšie povedané, ak potrebuje nejaký podriadení vlk zdeliť nejakú informáciu alfe, zdelí ju bete, tá ju zváži ako prvá



Obr. 2.1: Hierarchická pyramída postavenia vlkov. [15]

a prípadne posunie ďalej alfe. Úlohou bety je tiež udržanie disciplíny vo svorke a uistenie sa, že každý vlk dodržiava dané príkazy. Beta dominuje všetkým vlkom okrem alfy a v prípade úmrtia alfy je beta najlepším kandidátom na pozíciu alfy.

Tretím stupňom postavenia je stupeň delta, do ktorého spadá väčšia časť svorky. V tomto stupni je možné definovať ďalšie delenia na:

- Skautov - strážia hranice teritória a varujú svorku v prípade ohrozenia.
- Strážcov - ochraňujú svorku a garantujú jej bezpečnosť.
- Starších - sú to starší, skúsení vlci zvyčajne bývalí alfa a beta vlci.
- Lovcov - pomáhajú alfe a bete pri love a pri obstaraní jedla pre svorku.
- Opatrovateľov - starajú sa o slabých, chorých a zranených vlkov vo svorke.

Delta vlci sú nazývaní tiež aj ako podriadení pretože dominujú jedine vlku, ktorý je omegou.

Posledným, najnižším stupňom sociálnej hierarchie vlkov je omega. Vlk v tomto postavení (ďalej spomínaný len ako omega) je považovaný za obetného baránka svorky. Nedominuje žiadnemu inému vlku a má dovolené jesť ako posledný. Aj keď by sa mohlo zdať, že táto pozícia je zbytočná, po mnohých pozorovaniach sa zistilo, že prítomnosť omegy vo svorke je veľmi dôležité. Je to preto aby mali vlci možnosť si na niekom vybiť frustráciu a násilie. Ak svorka nemala omegu, boli v nej pozorované vnútorné boje a konflikty medzi všetkými vlkmi a bolo náročné pre alfu udržať vo svorke disciplínu. [15]

Ďalšou dôležitou vlastnosťou vlčej svorky pre algoritmus GWO, okrem jej sociálnej hierarchie, je skupinový lov. Podľa [16], sú hlavnými fázami lovu nasledujúce body:

1. Hľadanie, naháňanie a priblíženie sa k obeti.
2. Sledovanie, obkľúčenie a napádanie obete kým sa neprestane pohybovať.
3. Útok na obeť.

Tieto kroky sa dajú vidieť na Obrázku 2.2.

Obr. 2.2: Spôsob lovu vlkov - (A) hľadanie, prenasledovanie a priblíženie sa k obeti (B-D) sledovanie, obkľúčenie a napádanie obete kým sa neprestane pohybovať (E) stabilná pozícia a útok. [15]

Umelý analogický model

Sociálna hierarchia

Kedže umelý matematický model pracuje s troma najlepšími výsledkami, je počet vlkov v každej pozícii upravený.
Model pracuje s troma najlepšími výsledkami, ktoré budú predstavovať vĺci alfa, beta a delta. Ostatné výsledky budú predstavovať omega vlkov, ktorých pozície sa budú meniť na základe pozícii alfy, bety a delty.
Výsledkom, najlepším nájdeným riešením, celého algoritmu bude riešenie reprezentované vlkom alfa.

Obkľúčenie obete

Ako bolo vyššie spomenuté, počas lovu vlci svoju obeť obkľúčia. K tomu aby sa toto správanie dalo matematicky namodelovať sú použité rovnice (2.7) a (2.8).

$$\vec{D} = C \cdot \vec{X_p}(t) - \vec{X}(t) \tag{2.7}$$

$$\vec{X}(t+1) = \vec{X_p}(t) - A \cdot \vec{D} \tag{2.8}$$

kde t značí súčasnú iteráciu, A a C sú koeficienty, $\vec{X_p}$ je vektor pozície obete a $\vec{X}$ je vektor pozície vlka. Koeficienty A a C sú počítané podľa rovníc (2.9) a (2.10).

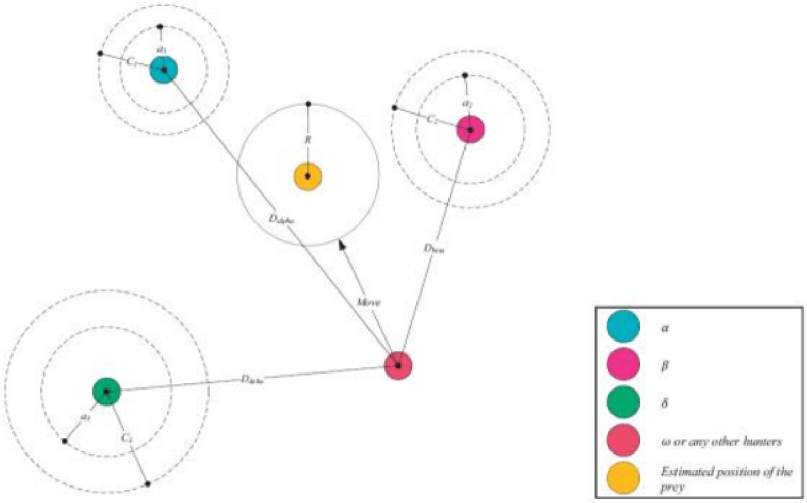
$$A = 2a \cdot r_1 - a \tag{2.9}$$

$$C = 2 \cdot r_2 \tag{2.10}$$

kde a je lineárne znižované od 2 po 0 cez sériu iterácii a r_1, r_2 sú náhodné hodnoty z intervalu $[0, 1]$.

Lov

Samotný lov je zvyčajne vedený alfou pričom beta a delta vĺci sa môžu k lovu pridať. V reálom svete vĺci vedia rozpoznať pozíciu obete a obkľúčiť ju. V umelom modely, ale nie je známa pozícia optima (obete). K simulácii obkľúčenia a lovu obete sa bude predpokladať, že riešenie alfa je k potenciálnej obeti (optimu) najbližšie a beta a delta sú v jeho blízkosti. Preto sa



Obr. 2.3: Zmena pozície agenta podľa alfy, bety a delty. [15]

tieto tri doposiaľ najlepšie riešenia uložia a podľa nich si ostatní vlci upravia svoje pozície. Toto správanie je matematicky opísané rovnicami (2.11), (2.12), (2.13) a (2.14).

$$\vec{D_\alpha} = C_1 \cdot \vec{X_\alpha} - \vec{X} \qquad \vec{X_1} = \vec{X_\alpha} - A_1 \cdot (\vec{D_\alpha}), \tag{2.11}$$

$$\vec{D_\beta} = C_2 \cdot \vec{X_\beta} - \vec{X} \qquad \vec{X_2} = \vec{X_\beta} - A_2 \cdot (\vec{D_\beta}), \tag{2.12}$$

$$\vec{D_\delta} = C_3 \cdot \vec{X_\delta} - \vec{X} \qquad \vec{X_3} = \vec{X_\delta} - A_3 \cdot (\vec{D_\delta}) \tag{2.13}$$

$$\vec{X}(t+1) = \frac{\vec{X_1} + \vec{X_2} + \vec{X_3}}{3} \tag{2.14}$$

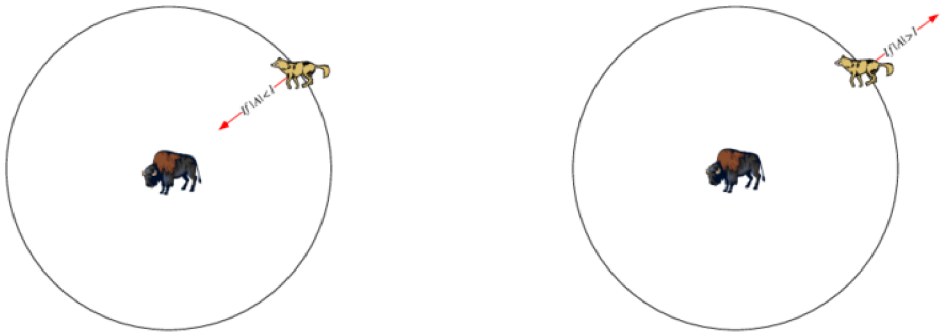
Obrázok 2.3 zobrazuje spôsob ako sa zmení pozícia omega vlka podľa alfy, bety a delty v 2D priestore. Finálna zmenená pozícia bude vybraná náhodne v kruhu, ktorý je naznačený pozíciami alfy, bety a delty. Teda alfa, beta a delta odhadujú pozíciu obete (optima) a ostatní vlci menia svoje pozície náhodne okolo obete.

Útok na obeť

Koniec lovu nastáva v momente kedy sa obeť nemá kam pohnúť. Približovanie sa k obeti (optimu) je v matematickom modely opísané znižovaním hodnoty parametru a.
Avšak so znižovaním parametru a sa taktiež mení parameter A. Inak povedané, A je náhodná hodnota z intervalu $[-a, a]$ kde a je znižované od hodnoty 2 po 0 počas celej série iterácii. Ak sú náhodné hodnoty A v intervale $[-1, 1]$, ďalšia pozícia agenta môže byť hociakej pozícii medzi jeho súčasnou pozíciou a pozíciou obete. Obrázok 2.4 ukazuje, že $A < 1$ urýchľuje vlkov k útoku na obeť.

Hľadanie obete

Vlci zväčša hľadajú obeť (optimum) podľa pozície alfy, bety a delty. Pri hľadaní obete sa snažia rozbehnúť od seba a približujú sa k sebe ak na obeť útočia. K tomu aby sa takéto rozbiehanie dalo matematicky modelovať, je upravený parameter A tak, že sú mu priradené



Obr. 2.4: Útok na obeť vs. hľadanie obete. [15]

náhodné hodnoty väčšie ako 1 alebo aj menšie ako -1. Toto zdôrazňuje skúmanie a dovoľuje algoritmu vyhľadávať globálne. Obrázok 2.4 taktiež ukazuje, že $A > 1$ urýchľuje vlkov k divergencii od obete s úmyslom nájsť inú, lepšiu obeť.
Ďalšou zložkou algoritmu, ktorá podporuje skúmanie priestoru je parameter C, nadobúda náhodnú hodnotu z intervalu $[0, 2]$. Tento parameter náhodne ovplyvňuje najlepšie pozície a týmto sa stále snaží podporiť hľadanie lepšieho riešenia aj v okolí najlepšieho bodu. To pomáha algoritmu ukázať viac náhodné správanie v jeho priebehu, a toto náhodné správanie pomáha k vyhnutiu sa lokálnemu optimu.
Parameter C nie je znižovaný lineárne narozdiel od A. C úmyselne poskytuje náhodné hodnoty stále preto aby sa zdôraznilo skúmanie nie len počas začiatočných iterácii, ale tiež aj v iteráciách finálnych. [15]

Algoritmus

Algoritmus 2 v krátkosti popisuje spôsob akým GWO hľadá optimálne riešenie.

Inicializácia hodnôt vstupných parametrov - počet vlkov, iterácii.
Inicializácia parametrov a, A a C .
Inicializácia súradníc vlkov na náhodné pozície.
Výpočet fitness hodnoty každého vlka. (Výpočet skutočnej hodnoty vybranej funkcie pre dané súradnice.)
while *Číslo iterácie nie je rovné maximálnemu počtu iterácii* **do**
  Výber najlepších vlkov - X_α, X_β a X_δ .
  foreach *Vlk* **do**
    Výpočet novej pozície podľa najlepších vlkov.
    Výpočet fitness daného vlka. (Výpočet skutočnej hodnoty vybranej funkcie pre dané súradnice.)
  end
  Pripočítanie iterácie. Výpočet parametrov a, A a C.
end
Výsledkom algoritmu je hodnota vlka X_α.

Algoritmus 2: Algoritmus GWO

Proces vyhľadávania začína s vytvorením náhodnej populácie vlkov (kandidátov na riešenie). Počas série iterácii, vlci alfa, beta a delta odhadujú pravdepodobnú pozíciu obete (optima). Každý vlk mení svoju pozíciu na základe najlepších vlkov - alfy, bety a delty. Pa-

rameter a je lineárne znižovaný od hodnoty 2 k 0 preto aby na začiatku algoritmu zdôraznil skúmanie a neskôr útok na obeť. A parameter C, ktorého hodnoty sú náhodné sa snaží podporovať skúmanie priestoru počas celej sérii iterácii. Algoritmus končí po uplynutí všetkých iterácii a najlepšie riešenie je reprezentované vlkom alfa.

2.2 Algoritmy inšpirované biológiou

Algoritmy z tejto skupiny nevyužívajú priamo skupinové správanie. Do tejto kategórie patrí napríklad aj genetický algoritmus, ktorý je inšpirovaný prírodou ale nevyužíva skupinové správanie.

2.2.1 Algoritmus párenia včiel

Včely sú jedným z najlepšie preštudovaných druhov hmyzu, ktorý si dokáže vytvárať sociálne vzťahy. Disponujú mnohými vlastnosťami, ktoré ich predurčujú k tomu, aby sa stali modelovým vzorom sociálneho správania. Týmito vlastnosťami sú napríklad rozdelenie práce, individuálna komunikácia či komunikácia s celou kolóniou a ich vzájomná spolupráca. Tento algoritmus je inšpirovaný filogenetikou[1] spoločenského života živočíchov z radu blanokrídlovcov, ako sú včely, mravce a osy, a procesom párenia včiel medonosných.
Včely sú eusociálnym[2] hmyzom, akého charakteristickými vlastnosťami sú:

- Spolupráca medzi dospelými jedincami pri staraní sa o potomkov a pri budovaní hniezda.
- V jeho kolónii sa prekrývajú aspoň dve generácie.
- Medzi jedincami funguje reproduktívna deľba práce.

Hmyz, ktorý tieto vlastnosti nespĺňa sa označuje ako osamelý a ten, ktorému chýba jedna či dve z týchto vlastností sa nazýva pre-sociálnym hmyzom. K tomu aby sa druh stal eusociálnym vedie mnoho sérii hniezdenia a budovania kolónie.
Tento algoritmus je opísaný modelom, kedy séria hniezdení kolónie začína s jedinou kráľovnou, bez potomkov. Kolónia sa ďalej postupne rozširuje o trúdov, ktorí vzniknú z neoplodnených vajíčok a ďalšie potomstvo vzniká práve za pomoci procesu párenia až kým kolónia nedosiahne pre-sociálnu úroveň. Vlastnosti, ktoré potvrdzujú pre-sociálnu úroveň namodelovanej kolónie sú - prekrývanie sa aspoň dvoch generácii, tým že algoritmus pracuje v každej iterácii s matkou a jej potomkami a funguje v ňom reproduktívna deľba práce tým, že kráľovná sa špecializuje len na rozmnožovanie a pracovnice sa starajú o potomkov.
Sociálne správanie včiel medonosných je výsledkom ich genetického potenciálu, ekologického a fyziologického prostredia, sociálnych podmienok v kolónii a rôznych predošlých a súčasných interakcii medzi týmito tromi faktormi. Každá včela má na starosti celú sériu úloh, ktoré sa odvíjajú podľa genetiky, prostredia a sociálneho predpisu kolónie. Výsledok každej úlohy sa stáva súčasťou prostredia a výrazne vplýva aj na ďalšie akcie samotnej včely ale aj na život celého úľa. [1]

[1] Filogenetika - študium historického vývoja a vzťahov medzi jedincami alebo skupinami organizmov.

[2] Eusociálny - živočíšny druh - zvyčajne hmyz, je taký druh, ktorý preukazuje znaky vyššieho stupňa sociálnej organizácie. Kde jeden jedinec, prípadne jedna kasta druhu, produkuje potomstvo a jedinci, ktorí sa nepodieľajú na reprodukcii sa vo vzájomnej spolupráci starajú o dané potomstvo.

Kolónia typicky pozostáva z kráľovnej, trúdov, robotníc a potomkov.

- Kráľovná je hlavným reprodukčným jedincom a zameriava sa na kladenie vajíčok.
- Trúdy sa liahnu z neoplodnených vajíčok a sú teda haploidní[3], keďže ich bunky majú len polovičný počet chromozómov. Ich úlohou je oplodniť kráľovnú.
- Robotnice sa liahnu z oplodnených vajíčok a ich úlohou je starať sa o potomstvo a kráľovnú. [2]

Páriaci let

Proces párenia včiel sa odohráva mimo ich hniezda. Miesto párenia sa stáva zhromaždiskom trúdov, kde sa môžu vyskytovať aj trúdy z iných úľov. Páriaci let začína s tancom kráľovnej, počas ktorého trúdy kráľovnú nasledujú a pária sa s ňou vo vzduchu. Typicky, sa počas páriaceho letu kráľovná pári so siedmimi až dvadsiatimi trúdmi. Po párení trúd umiera. [11]
Pri každom párení sa spermie ukladajú do spermotéky[4] kráľovnej, kde sa hromadia. Týmto sa sfromuje genetický fond kolónie[5].
Pri kladení vajíčka vyberá kráľovná náhodnú zmes spermii zo svojej spermotéky a pomocou nej vajíčko oplodní.

Umelý analogický model

Páriaci let predstavuje množinu párení kráľovnej s trúdmi, pri ktorých došlo k pridaniu spermii trúda do spermotéky kráľovnej. Kráľovná má na začiatku každého letu istú energiu, rýchlosť a jej spermotéka má limitovanú veľkosť. Počas párení sa energia a rýchlosť kráľovnej znižuje a spermotéka sa napĺňa. To, s akým trúdom sa bude kráľovná páriť a akého spermie budú v spermotéke je matematicky dané istou pravdepodobnosťou, ktorá je definovaná rovnicou (2.15).

$$prob(Q, D) = e^{\frac{-difference}{speed}} \tag{2.15}$$

kde, $prob(Q, D)$ je pravdepodobnosť pridania spermii trúda D do spermotéky kráľovnej Q, teda je to pravdepodobnosť úspešného párenia, $difference$ je absolútny rozdiel medzi fitness D a Q, a $speed$ je momentálna rýchlosť kráľovnej Q. Táto funkcia sa správa ako žíhacia (pomaly chladne), kde pravdepodobnosť párenia je vysoká buď vtedy ak je kráľovná na začiatku jej páriaceho letu, teda jej rýchlosť je vysoká, alebo ak je fitness trúda tak dobrá ako je fitness kráľovny. Znižovanie energie a rýchlosti po každom párení je matematicky opísané rovnicami (2.16) a (2.17).

$$speed(t + 1) = \alpha \cdot speed(t) \tag{2.16}$$

$$energy(t + 1) = energy(t) - step \tag{2.17}$$

kde α je faktor $\in [0, 1]$ a $step$ je množstvo energie odobrané po každom prechode. [1]

[3] Haploidný - väčšina buniek je diploidná tzn. má 2 sady chromozómov, od otca a od matky, haploidný jedinec má len jednu sadu chromozómov.
[4] Spermotéka - miesto uloženia spermii samca.
[5] Geneticky fond - súbor genotypov jedincov v kolónii.

Algoritmus

Celý algoritmus je možné zhrnúť v nasledujúcich krokoch.
Každý jedinec v kolónii reprezentuje možné riešenie problému, ktorý algoritmus rieši.

```
Inicializácia hodnôt vstupných parametrov - počet páriacich letov, trúdov,
  potomkov a veľkosť spermotéky pre kráľovnú.
Inicializácia riešenia reprezentovaného kráľovnou na náhodnú hodnotu.
Vylepšenie riešenia reprezentovaného kráľovnej heuristikou.
for Počet páriacich letov do
    Inicializácia hodnôt energie a rýchlosti kráľovnej.
    Inicializácia kroku akým sa má znižovať energia kráľovnej.
    Vygenerovanie náhodných hodnôt ako riešenia, ktoré budú reprezentované
      trúdmi.
    Označenie náhodných génov každého trúda pre haploidné kríženie.
    while Je energia kráľovnej väčšia ako nula do
        Vypočítanie pravdepodobnosti párenia sa pre každého trúda.
        if Nie je spermotéka kráľovnej plná then
            Výber trúda s najväčšou pravdepodobnosťou párenia a pridaj jeho
              spermie do spermotéky kráľovnej.
        end
        Upravenie energie a rýchlosti kráľovnej.
    end
    Vytvorenie potomstva pomocou haploidného kríženia.
    Použitie heuristiky pre vylepšenie potomkov.
    if Najlepší potomok lepší ako kráľovná then
        Nahraď chromozóm kráľovnej chromozómom najlepšieho potomka.
    end
    Zahodenie všetkých potomkov a trúdov.
end
```

Algoritmus 3: Algoritmus MBO

Na začiatku celého algoritmu je kráľovná, ktorej je pridelené náhodné riešenie. Keďže kráľovná by mala byť stále o niečo lepším jedincom, teda mala by niesť stále to najlepšie riešenie, je možné už na začiatku algoritmu použiť nejakú heuristickú funkciu, ktorá riešenie reprezentujúce kráľovnou vylepší.
Každý páriaci let začína kráľovná s energiu a rýchlosťou, ktoré postupne klesajú podľa počtu párení v danom lete. V každom lete je vygenerovaná nová množina trúdov, ktorých riešenia sú vygenerované náhodne a bez použitia heuristiky. Pre každého z týchto trúdov je vypočítaná pravdepodobnosť s akou budú ich spermie pridané do spermotéky kráľovnej. Ak ešte nemá kráľovná spermotéku plnú, vyberie sa trúd s najvyššou pravdepodobnosťou pridania spermii a jeho spermie sa pridajú do spermotéky kráľovej. Pridané nemusia byť stále len spermie najlepšieho trúda, je možno definovať istú prahovú hodnotu, podľa ktorej bude vybraný náhodný trúd z množiny trúdov, ktorých pravdepodobnosť párenia je vyššia ako daná prahová hodnota. Táto voľba závisí na implementácii algoritmu.
Ak je hodnota energie kráľovnej nulová, alebo má kráľovná plnú spermotéku, dochádza k tvorbe potomstva. Nový potomok vzniká haploidným krížením kráľovnej a náhodne vybraných spermii trúda z jej spermotéky. Toto haploidné kríženie je znázornené na Obrázku 3.2 a bližšie popísané v kapitole 3.2. Pre novovytvorených jedincov je možné použiť heuris-

tickú funkciu, ktorá riešenia nimi reprezentované ešte vylepší. Algoritmus MBO považuje dané heuristiky za robotnice, ktoré sa majú starať o potomkov. Týchto heuristík môže byť v algoritme použitých aj viacero druhov kedy je možné ich náhodne striedať.
Ak sa nájde vo vytvorenom potomstve lepšie riešenie ako to, aké reprezentuje kráľovná, je toto riešené pridelené kráľovnej a v ďalšom páriacom lete reprezentuje doteraz najlepšie nájdené riešenie. Ostatní jedinci z potomstva a trúdy umierajú, teda ich riešenia sú zahodené a ak nebolo nájdené riešenie problému pokračuje ďalší páriaci let.
Po skončení všetkých páriacich letov je najlepšie nájdené riešenie reprezentované kráľovnou.

2.3 Algoritmy inšpirované fyzikálnymi a chemickými javmi

Algoritmy v tejto kategórii vznikli napodobnením fyzikálnych či chemických javov. Sú to algoritmy inšpirované rôznymi zákonmi zo sveta fyziky či chémie, ale taktiež aj algoritmy opisujúce javy v prírode ako napríklad systém vytvárania riečnych tokov a podobne. [12]

2.3.1 Algoritmus simulovaného žíhania

Algoritmus simulovaného žíhania je metaheuristckým algoritmom, ktorý je inšpirovaný žíhacím procesom v hutníctve. Žíhanie je fyzikálny proces kde je pevná látka pomaly ochladzovaná kým jej štruktúra nedosiahne bodu s minimálnou energetickou konfiguráciou.
Tento fyzikálno-chemický proces vytvára vysokokvalitné a pevné materiály. Analogicky sa dá toto ochladzovanie opísať ako druh optimalizácie a práve tak vznikla myšlienka algoritmu simulovaného žíhania, kde fyzikálny stav látky zodpovedá riešeniam problému, energia cene, za akú sa dané riešenie našlo, a hodnota teploty slúži ako parameter regulácie. [3]

Umelý analogický model

Ochladzovanie je v SA interpretované ako pomalé klesanie pravdepodobnosti prijatia aj horšieho výsledku pri prehľadávaní priestoru riešení. Algoritmus pracuje stále s jedným výsledkom, ktorý sa mení na základe istej pravdepodobnosti. Táto pravdepodobnosť závisí na hodnote momentálnej teploty. Teda SA nemusí akceptovať len zmeny, aké vylepšujú výsledok, ale môže prijať aj tie, ktoré sa nezdajú ako lepšie voči riešeniu s ktorým pracuje.
Pravdepodobnosť prijatia horšieho výsledku je hlavne na začiatku algoritmu, kedy je hodnota teploty vysoká. SA sa týmto spôsobom snaží podporiť proces prehľadávania okolitého priestoru aby hneď neupadol do lokálneho optima. [3]
Napríklad v prípade hľadania minimálnej hodnoty riešenia, by neboli prijateľné všetky zmeny, ktoré znižujú hodnotu riešenia, tie ktoré, zvyšujú hodnotu budú s pravdepodobnosťou p akceptované. Táto pravdepodobnosť je popísaná rovnicou (2.18).

$$p = e^{-\frac{\Delta E}{k_B T}} \tag{2.18}$$

kde k_b je Bolzmanovou konštantou. Hodnota tejto konštanty sa pre jednoduchosť uvádza ako $k_B = 1$. T je hodnota teploty pre kontrolovanie procesu. ΔE je zmenou energie, ktorá nastala. Táto pravdepodobnosť používa Boltzmanovo rozloženie.
Zmena energie je definovaná pomocou rovnice (2.19).

$$\Delta E = \gamma \Delta f \tag{2.19}$$

kde γ je reálnou konštantou. Ako bolo spomenuté v rovnici pre pravdepodobnosť prijatia zmeny (2.18), je použité zjednodušenie pre konštantu $k_B = 1$ a aj pre konštantu $\gamma = 1$.

Teda rovnica pre výpočet pravdepodobnosti zmeny sa dá prepísať na tvar uvedený v rovnici (2.20).

$$p(\Delta f, T) = e^{-\frac{\Delta f}{T}}. \tag{2.20}$$

To, či sa bude akceptovať daná zmena sa taktiež rozhodne pomocou náhodného čísla r, ktoré bude slúžiť ako prahová hodnota. Ak $p > r$ bude daná zmena akceptovaná inak algoritmus pracuje ďalej s aktuálnym riešením. [19]

Je veľmi dôležité zvoliť si správnu začiatočnú teplotu.
Ak má T priveľmi vysokú hodnotu ($T \to \infty$) pre danú zmenu Δf, bude sa hodnota p približovať k číslu 1, čo znamená, že skoro každá zmena bude akceptovaná.
Pri veľmi nízkej hodnote T, ($T \to 0$) bude každá zmena s hodnotou $\Delta f > 0$ (teda potenciálne horšie riešenie) len zriedka akceptovaná, keďže hodnota p sa bude približovať k nule a týmto bude limitovaná rozmanitosť riešenia a každé vylepšenie Δf bude skoro stále akceptované, ale za cenu neprehľadávania okolitého priestoru.
Špeciálny prípad kedy sa T približuje k nule zodpovedá metóde Hill climbing[6], pretože prijaté budú len lepšie riešenia a celý algoritmus sa bude pohybovať len jediným smerom, bude podľa zadania hľadať buď iba vyššiu alebo len nižšiu hodnotu, bez možnosti prechádzania širšieho okolia a teda nemusí nájsť globálne optimum. Ak je hodnota T vysoká, a teda aj energia daného stavu riešenia je vysoká, bude systém pokračovať v ochladzovaní a preto sa hneď nechytí do lokálneho optima, kde by bol nedostatok energie k preskúmaniu prípadných ďalších optím, spoločne s globálnym optimom.
Je veľmi dôležité vedieť ako regulovať proces ochladzovania, tak aby bol systém ochladzovaný postupne z vyššej teploty k teplote, kde bude v stave globálneho optima. Existuje viacero spôsobov ako regulovať ochladzovanie, teda znižovanie hodnoty teploty.
Dvoma najčastejšími spôsobmi kontroly ochladzovania sú lineárny a geometrický. Lineárny spôsob regulácie ochladzovania je definovaný rovnicou (2.21).

$$T = T_0 - \beta t, \tag{2.21}$$

či $T \to T - \Delta T$, kde T_0 je začiatočnou teplotou a t reprezentuje pseudo počítadlo iterácii. β je krok ochladenia. Tento krok by mal byť zvolený tak, že ak sa T bude približovať k 0 potom by sa malo t blížiť k t_f, ktoré reprezentuje maximálny počet iterácii. Z toho teda vyplýva rovnica pre výpočet kroku ochladzovania - $\beta = \frac{(T_0 - T_f)}{t_f}$.
Na druhú stranu, geometrický spôsob ochladzovania znižuje hodnotu teploty pomocou ochladzovacieho faktoru α, ktorý sa volí v intervale $0 < \alpha < 1$ a tak T je nahradené αT. Toto geometrické ochladzovanie je popísané rovnicou (2.22).

$$T(t) = T_0 \alpha^t, \quad t = 1, 2, \ldots, t_f, \tag{2.22}$$

Výhodou tohto spôsobu je, že situácia kedy $T \to 0$ nastane ak sa $t \to \infty$ a teda nie je nutné špecifikovať maximálny počet iterácii. Keďže by mal byť proces ochladzovania pomalý, aby sa systém ľahšie stabilizoval volí sa zvyčajne $\alpha = 0.7 \sim 0.99$. [19]
To akým spôsobom sa systém ochladzuje rozhoduje o počte hľadaní a rozhodovaní o prijatí nových výsledkov. Implementovaný algoritmus využíva ochladzovanie podobné lineárnemu kedy je zadaný krok ochladzovania ako parameter algoritmu a je možné hneď určiť počet iterácii.

[6] Hill Climbing - algoritmus lokálneho prehľadávania stavového priestoru, ktorý nahradzuje aktuálne riešenie len zlepšujúcim riešením.

Algoritmus

Stručný popis fungovania algoritmu simulovaného žíhanie je zhrnutý v algoritme 4.

Inicializácia hodnôt parametrov - začiatočná teplota, koncová teplota, krok ochladzovania.
Inicializácia prvého bodu na náhodne vybrané súradnice z intervalu.
Priradenie hodnoty prvého bodu ako najlepšie riešenie.
while *aktuálna teplota > ako koncová teplota* **do**
  Výber náhodného bodu z okolia aktuálneho bodu.
  Vygenerovanie náhodného čísla r z intervalu [0,1].
  if *pravdepodobnosť prijatia vybraného bodu > r* **then**
    Novým aktuálnym bodom sa stáva novo vygenerovaný bod.
  end
  if *aktuálny bod lepší ako najlepší bod* **then**
    Najlepším bodom je aktuálny bod.
  end
  Podľa zadaného kroku zmeň hodnotu aktuálnej teploty.
end
Ukáž najlepší bod.

Algoritmus 4: Algoritmus simulovaného žíhania

Algoritmus stále pracuje s aktuálnym bodom od ktorého sa vyvíja či sa posunie niekam ďalej v jeho okolí alebo ostane pri ňom. Zmena nastáva na základe výpočtu pravdepodobnosti podľa hodnoty riešenia daným bodom a aktuálnej teploty.
Algoritmus má svoju pamäť a pamätá si dosiaľ najlepšie nájdené riešenie, no aktuálny bod sa stále môže posúvať a meniť hodnotu najlepšieho bodu a tým neupadnúť do lokálneho optima.

Kapitola 3

Riešené optimalizačné problémy

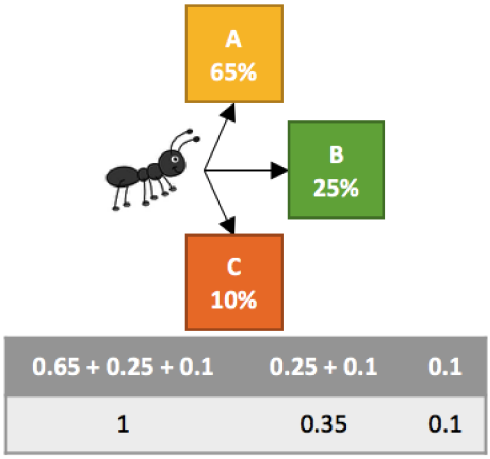
3.1 Problém obchodného cestujúceho

Problém obchodného cestujúceho (angl. Travelling Salesman problem TSP) je jedným z najviac študovaných problémov v oblasti kombinatorickej (diskrétnej) optimalizácie. Jeho hlavým cieľom je nájsť pre obchodného cestujúceho čo najkratšiu cestu medzi mestami tak, aby navštívil každé miesto práve raz a ukončil svoju cestu v mieste, kde ju začal.
Aj napriek zdanlivo jednoduchému cieľu sa tento problém radí do skupiny NP - náročných (angl. NP-hard)[1], keďže s počtom pribúdajúcich miest rastie náročnosť nájdenia cesty v najhoršom prípadne exponenciálne, ak je potrebné nájsť garantované optimálne riešenie. Tento problém sa dá matematicky opísať pomocou teórie grafov.
Nech $G = (V, A)$ je grafom kde V je množina n vrcholov (miest), A je množinou hrán (ciest) a nech $C = (c_{ij})$ je maticou vzdialeností priradenou k A.
Problém obchodného cestujúceho pozostáva z určenia okruhu s minimálnou vzdialenosťou, ktorý však prechádza každým uzlom v grafe práve raz. Takýto okruh je tiež známy ako Hamiltonovský. [13]

Aplikovanie ACO na TSP

Aplikovanie ACO na problém obchodného cestujúceho je jednoduché. Na začiatku je vopred definovaný počet miest, kedy každé mesto má svoje súradnice a vzdialenosť medzi nimi je počítaná pomocou nich.
Ďalej je definovaný počet mravcov, ktorí budú grafom prechádzať za účelom nájdenia najkratšej cesty medzi všetkými mestami. Miesto, od ktorého mravec začne graf prehľadávať mu bude na začiatku algoritmu pridelené náhodne.
Mravce majú svoju istú formu pamäte, teda pamätajú si miesta aké už navštívili. Výber miesta kde ešte neboli je vypočítaný na základe vzdialenosti a feromónu, ktorý na ceste zanechali buď oni sami alebo iné mravce.
Definovaný je tiež aj počet opakovaní hľadania cesty každého mravca pričom na konci každej iterácie sa prepočítava množstvo feromónu pre každú cestu. Cesta, ktorá bola častejšie prechádzaná bude mať v ďalšej iterácii o niečo vyššiu pravdepodobnosť výberu podľa feromónu, ktorý na nej pribudol ako tá, po ktorej šlo menej mravcov.
Rozhodnutie mravca akou cestou sa vyberie je namodelované pomocou rulety. [14]
Čo znamená, ak má mravec na výber viacero miest ktoré ešte nenavštívil, vypočíta sa pravdepodobnosť navštívenia pre každé z nich, a z týchto pravdepodobností sa vypočíta

[1] NP-hard problémy - problémy, ktoré nie je možné vyriešiť v polynomiálnom čase.



Obr. 3.1: Príklad výpočtu kumulatívnej sumy pri výbere miest.

kumulatívna početnosť. Ďalej sa vygeneruje náhodné číslo z intervalu $[0, 1]$ a na základne hodnoty tohto náhodného čísla sa podľa kumulatívnej početnosti vyberie mesto aké mravec navštívi ako ďalšie.

Pre lepšie pochopenie opisuje výber ruletou príklad znázornený na Obrázku 3.1.

Mravec má na výber z troch miest, ktoré ešte nenavštívil. Na základe vzdialenosti a feromónu sú pravdepodobnosti výberu miest nasledovné - mesto A si vyberie s pravdepodobnosťou 65%, mesto B s pravdepodobnosťou 25% a mesto C s pravedpodobnosťou 10%. Zápis pomeru (3.1) opisuje výslednú kumulatívnu početnosť na základe daných pravdepodobností pre tieto tri mestá. Spôsob akým bola získaná je ukázaný na Obrázku 3.1.

$$1 : 0,35 : 0,1 \tag{3.1}$$

Podľa náhodne vygenerovaného čísla n sa rozhodne aké mesto mravec navštívi. Výber mesta závisí na intervale, v ktorom sa náhodné číslo nachádza podľa rovnice (3.1).

$$vyberMesta(n) = \begin{cases} 0.35 < n \leq 1 & \text{vyber miesto A} \\ 0,1 < n \leq 0.35 & \text{vyber miesto B} \\ 0 \quad \leq n \leq 0.1 & \text{vyber miesto C} \end{cases}$$

Po každej iterácii je uložený najlepší výsledok, ktorý sa s každou novou iteráciou môže zmeniť, ak v nej bola nájdená kratšia cesta. Na začiatku novej iterácie sú stále vygenerované nové mravce. V algoritme je tiež možné nastaviť váhu, aká bude pridelená pri vypočítavaní pravdepodobnosti výberu miest pomocou premenných α, β, kde α zvyšuje váhu feromónu a β vzdialenosti.

3.2 Splniteľnosť booleovských formúl

Problém splniteľnosti booleovských formúl SAT, patrí do skupiny problémov, ktoré majú za úlohu nájsť takú skupinu hodnôt pre premenné, aby vyhovovali všetkým zadaným obmedzeniam, a hodnoty priradené premenným boli v definičných oboroch daných pre ich premenné (angl. Constraint Satisfaction Problem CSP).

Formálne je CSP zadefinovaný ako trojica $\langle X, D, C \rangle$, kde [17]

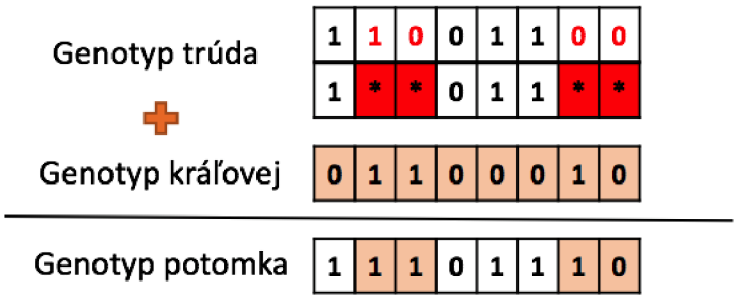
- $X = X_1, \ldots, X_n$ je množinou premenných,
- $D = D_1, \ldots, D_n$ je množina príslušných definičných oborov,
- a $C = C_1, \ldots, C_m$ je množina obmedzení.

Každá premenná X_i môže naberať hodnoty z príslušného, neprázdneho definičného oboru D_i. Každé obmedzenie $C_j \in C$ je zase dvojicou $\langle t_j, R_j \rangle$, kde $t_j \subset X$ je podmnožinou k premenných a R_j je k-árna relácia, ktorá priradí k premenným príslušnú podmnožinu definičných oborov D_j.
Ohodnotenie premenných je funkcia, ktorá z priradí premenným hodnoty z príslušného definičného oboru a ohodnotenie v spĺňa obmedzenie $\langle t_j, R_j \rangle$ ak hodnoty pridelené premenným t_j spĺňajú reláciu R_j.
SAT je špecifickým prípadom CSP, kde definičným oborom každej premennej je buď pravda alebo lož, ekvivalentne 1 alebo 0. Každý CSP môže byť interpretovaný ako SAT a existuje mnoho problémov v oblasti plánovania a rozhodovania, ktoré je možné prezentovať práve ako SAT. [2]
Pre SAT je vstupným parametrom logická formula v konjunktívnej normálnej forme[2] a riešením problému je také ohodnotenie premenných, pre ktoré daná formula nadobúda pravdivú hodnotu.

Aplikovanie MBO na SAT

Pre aplikovanie algoritmu MBO pre riešenie SAT je potrebné si definovať spôsob, akým bude daný jedinec reprezentovať potenciálne riešenie. V MBO je preto každý jedinec popísaný svojim genotypom[3]. Genotyp sa skladá z jednotlivých génov, ktoré v MBO predstavujú premenné použité v definovanej formule. Každý gén dokáže niesť informáciu, ktorá v tomto prípade bude ukazovať či je hodnota premennej pre riešenie popisované genotypom jedincom pravda alebo lož.
Napríklad formula $(A \vee B) \wedge (\neg B \vee C \vee \neg D)$ obsahuje štyri premenné. Teda genotyp každého jedinca v kolónii bude obsahovať 4 gény. A jedinec s genotypom 0110 by reprezentoval riešenie pre ktoré premenná $A = 0$, $B = 1$, $C = 1$ a $D = 0$.
Fitness ohodnotenie genotypu je pomer medzi počtom klauzúl, ktoré sú daným genotypom splniteľné a počtom všetkých klauzúl formuly. Hodnota fitness jedinca opísaného v predošlom príklade by bola rovná 1, keďže jeho genotyp spĺňa obe z dvoch definovaných klauzúl.
Keďže je trúd haploidným jedincom, gény v jeho genotype nesú aj isté označenie. Pri vytváraní genotypu trúda označí označovacia funkcia náhodne polovicu génov. Čo znamená, že pri tvorbe nového potomka budú práve tieto označené gény nahradené génmi z genotypu matky. Na Obrázku 3.2 sú označené gény znázornené červenou farbou.
Kráľovná je reprezentovaná svojim genotypom, rýchlosťou, energiou a spermotékou. Pred každým páriacim letom je inicializovaná rýchlosť a energia kráľovnej na náhodnú hodnotu z intervalu $[0.5, 1]$ aby bolo garantované, že prebehne aspoň jedno párenie s trúdom. Po inicializácii parametrov kráľovnej sa vypočíta fitness pre každého trúda podľa počtu formúl ktoré jeho genotyp spĺňa a s ňou aj pravdepodobnosť párenia daného trúda s kráľovnou. Ak ešte nie je spermotéka kráľovnej plná, vloží sa do dej sperma najlepšieho trúda. Trúd, s ktorým prebehlo párenie následne umiera. Po každom párení sa upraví rýchlosť a energia

[2]Konjunktívna normálna forma je konjunkciou klauzúl formuly, a každá klauzula formuly je disjunkciou literálov (premenných) alebo ich negáciou.
[3]Genotyp - súbor génov jedinca



Obr. 3.2: Príklad haploidného kríženia trúda s kráľovnou.

kráľovnej a v prípade, že energia kráľovnej nie je nulová a ani jej spermotéka nie je plná, bude nasledovať ďalšie párenie.

Po dokončení páriaceho letu prichádza na rad oplodňovanie vajíčok a tým vznik nového potomstva. Kráľovná si náhodne vyberá sperma trúdov, s ktorými sa v danom lete párila zo svojej spermotéky a haploidným krížením vzniká nový potomok. [2]

Haploidné kríženie je znázornené na Obrázku 3.2. Gény, ktoré boli v genotype trúda označené sú znázornené červenou farbou a nebudú predávané do génu potomka. A namiesto týchto génov prídu gény z genotypu kráľovnej, ktoré sú na obrázku znázornené oranžovým pozadím.

Po vytvorení určeného počtu potomkov nastáva ich ohodnotenie. Algoritmus v tomto čase môže skončiť ak sa v kolónii nachádza jedinec, ktorého hodnota fitness je rovná 1, teda riešenie, ktoré reprezentuje jeho genotyp spĺňa všetky kauzuly danej formule. Ak sa takýto jedinec v kolónii nenachádza a ešte nebol dosiahnutý požadovaný počet páriacich letov, algoritmus bude pokračovať ďalším páriacim letom. Avšak ešte pred začiatkom ďalšieho letu, sa vyberie najlepší potomok. Ak je fitness najlepšieho potomka lepšia, ako fitness kráľovnej, zmení sa genotyp kráľovnej, pre ďalší páriaci let, podľa genotypu tohto potomka. Na konci páriaceho letu všetci trúdi a potomkovia umierajú a na začiatku ďalšieho letu je vygenerovaná nová množina trúdov s náhodnými genotypmi.

3.3 Hľadanie extrému funkcie

Hľadaním extrému funkcie sa rozumie hľadanie bodu, z intervalu funkcie na ktorom je definovaná, a v ktorom nadobúda svoje maximum či minimum. Úlohou algoritmov je nájsť globálne maximum či minimum pre danú funkciu.

Definícia 1. Nech množina M je ľubovoľná podmnožina definičného oboru funkcie f.
Hovoríme, že funkcia f má v bode $a \in M$ *maximum* na množine M práve vtedy, keď pre všetky $x \in M$ platí $f(x) \leq f(a)$.
Hovoríme, že funkcia f má v bode $b \in M$ *minimum* na množine M práve vtedy, keď pre všetky $x \in M$ platí $f(x) \geq f(b)$.

Algoritmy implementované v tejto práci pracujú s funkciou o dvoch premenných x a y. Riešením algoritmu je bod $[x_i, y_i]$, v ktorom daná funkcia nadobúda želané optimum (minimum alebo maximum).

Existuje mnoho delení optimalizačných algoritmov, no v prípadne hľadania extrémov funkcie je najvhodnejšie spomenúť delenie podľa spôsobu využitia derivácie funkcie. Toto delenie delí algoritmy na tie, ktoré pri hľadaní používajú deriváciu funkcie (angl. gradient-based) a na algoritmy, ktoré pracujú priamo so samotnou hodnotou funkcie pre daný bod $f([x_i, y_i])$

(angl. gradient/derivative-free).
Algoritmy GWO a SA patria do kategórie algoritmov, ktoré používajú samotnú hodnotu funkcie, sú to gradient-free stochastické algoritmy, teda používajú určitú formu náhodnosti pri hľadaní extrémov funkcie. Rozdiel medzi týmito algoritmami je ten, že GWO pracuje s viacerými agentami, kde každý z agentov reprezentuje nejaký bod z intervalu, na ktorom je funkcia definovaná a SA pracuje s jedným bodom, ktorý je považovaný za optimum pre danú iteráciu a podľa teploty rozhoduje o zmene tohto bodu.
Ako je spomenuté v kapitole 2, jedným z hlavných problémov pri hľadaní optima funkcie je uviaznutie v lokálnom optime. Obe tieto algoritmy by sa mali vedieť tomuto uviaznutiu pomocou istého použitia náhodnosti vyhnúť. To, či je výsledok určite globálnym optimom funkcie však nedokážu garantovať.
Náhodnosť v implementovaných algoritmoch je použitá na začiatku, pri inicializácii začiatočných bodov algoritmu, ale taktiež aj v ďalších krokoch algoritmu pri prehľadávaní priestoru, aby algoritmus neupadol do lokálneho extrému v prípade, že náhodne vybraný inicializačný bod bude v jeho blízkosti.

Aplikovanie GWO

Princíp aplikovania GWO pre hľadanie optima funkcie je podobný algoritmom ACO keďže taktiež pracuje s agentmi, ktorý prehľadávajú priestor možných riešení. V GWO sú agentami vlci a obeťou je výsledné optimum. Počet vlkov je upresnený ešte pred začiatkom algoritmu a tento počet sa ani počas behu algoritmu nemení.
Každý vlk reprezentuje bod z intervalu, v akom sa má hľadať optimum danej funkcie. Na začiatku algoritmu je každému vlkovi pridelený bod náhodne, no GWO pracuje s náhodnosťou aj v ďalších krokoch. Zo všetkých vlkov sa vyberú traja najlepší, teda tri body, pre ktoré je hodnota zadanej funkcie buď najnižšia, alebo najvyššia, a podľa týchto troch bodov si ostatní agenti upravujú svoje polohy. Kedže vlci nikdy nevedia pozíciu obete (optima), stále odhadujú, že títo traja najlepší vlci (tri najlepšie výsledky) k nej budú najbližšie a preto sa k nim aj ostatní vlci chcú priblížiť. Po úprave polôh sa zase vyberú tri najlepšie hodnoty a takto úpravy polôh pokračujú až do poslednej iterácie.
Druhá forma náhodnosti, ktorá chráni algoritmus pred upadnutím do lokálneho optima sa skrýva práve pri úprave polôh vlkov. Úprava sa síce sústredí na tri najlepšie výsledky, no stále počíta aj s poradím iterácie, ktorý prebieha a pracuje s náhodne generovaním číslom, ktoré môže dovoliť vlkovi nie len sa približovať k optimu, ale aj prehľadať širšie okolie. Tento spôsob dáva možnosť algoritmu nájsť aj lepšie výsledky hlavne počas prvých iterácii.
Veľkou výhodou GWO je jeho práca s viacerými agentami. Kedy už aj ich prvotné náhodné rozdelenie vie často krát odhaliť lokálne optimum od globálneho.

Aplikovanie SA

SA narozdiel od GWO a ACO nepracuje so žiadnou podobou agentov.
Algoritmus SA sa počas celého svojho priebehu zameriava na jeden bod, ktorý si uchováva a aký podľa istej pravdepodobnosti zmení. Tento bod je považovaný za dosiaľ najlepší nájdený. Na začiatku sú hodnoty pre súradnice tohto bodu vybrané náhodne z intervalu na ktorom je pre funkciu určené nájsť optimum.
Ďalej SA pracuje so zadefinovanou začiatočnou teplotou, krokom o aký sa má daná teplota ochladzovať a hodnotou konečnej teploty pri akej sa má algoritmus zastaviť.
V každom kroku sa vyberie kandidát na nový najlepší výsledok z okolia aktuálne najlepšieho bodu. Pre vybraného kandidáta sa vypočíta pravdepodobnosť s akou by mal byť zvolený za

najlepší. Táto pravdepodobnosť nezávisí len od hodnoty pre funkciu, ktorú kandidát má ale tiež aj od momentálnej teploty. Vypočítaná pravdepodobnosť sa ďalej porovná s náhodne vygenerovaným číslom z intervalu $[0, 1]$ a ak je pravdepodobnosť väčšia, kandidát sa stáva novým najlepším bodom. Práve teplota zohráva v algoritme SA kľúčovú rolu kvôli tomu, že dokáže presadiť ako najlepší výsledok aj kandidáta, ktorý je horší ako momentálne najlepší výsledok. Pravdepodobnosť tejto situácie je vyššia v začiatočných fázach algoritmu, kedy je teplota vysoká. A toto presadenie pomáha algoritmu dostať sa z lokálneho optima a hľadať globálne.

Kapitola 4

Opis aplikácie

V tejto kapitole je opísaná aplikácia, ktorá implementuje preštudované algoritmy a demonštruje ich činnosť na vybraných optimalizačných problémoch. Prvá časť kapitoly popisuje vnútornú implementáciu aplikácie, teda v akom jazyku je napísaná, aké pomocné knižnice boli pri jej tvorbe použité a aké je členenie kódu, ktorý ju popisuje. A v druhej časti je bližšie ukázané a popísané vytvorené užívateľské prostredie aplikácie s pomocou ktorého by mal užívateľ pracovať.

4.1 Aplikácia

Programovacím jazykom, aký bol využitý pre implementáciu aplikácie je objektovo orientovaný jazyk Java. Jazyk Java je vyvíjaný spoločnosťou Oracle a jeho syntax vychádza z jazykov C a C++. Jednou z hlavných čŕt tohto jazyka je, že zdrojové programy ním napísané sa nekompilujú do strojového kódu, ale do medzistupňa, tzv. "byte-code", ktorý nie je závislý od konkrétnej platformy. Tento kód neskôr vykonáva a spracováva interpreter - Java Virtual Machine.
Pre tvorbu užívateľského prostredia bol použitý framework[1] JavaFX, ktorý pomáha vytvoriť MVC štruktúru aplikácie. Ďalšími externými knižnicami, ktoré sú použité v aplikácii sú commons-lang3, exp4j a jazyk MVEL.

Knižnica exp4j slúži k vyhodnocovaniu matematických výrazov. V aplikácii je použitá k tomu, aby aplikácia dokázala používať funkciu napísanú užívateľom nie len ako reťazec znakov, ale ako skutočnú funkciu, ktorá bude vyhodnocovať výsledky pre dané vstupy.

MVEL je jazyk, ktorý umožňuje zápis a interpretáciu výrazov. V aplikácii sa tento jazyk používa pre pomoc s vyhodnocovaním booleovských klauzúl zadaných užívateľom.

Knižnica commons-lang3 obsahuje väčšie množstvo pomocných funkcii pre prácu s číslami, reťazcami, vytváraním a serializáciou objektov apod. Aplikácia túto knižnicu používa pre prácu s intervalmi na ktorých je hľadaný extrém pre funkciu.

Štruktúra aplikácie je založená na modely MVC. Model MVC Model-View-Controller rozlišuje v aplikácii 3 základné časti - model, pohľad (angl. view) a riadič(angl. controller).

[1] Framework - softwarová štruktúra, ktorá slúži ako podpora pri programovaní a organizácii iných softwarových projektov.

Model pracuje s dátami a uchováva funkčnú logiku aplikácie pred užívateľom.

View (pohľad) poskytuje užívateľovi zobrazenie dát z vnútra aplikácie formou vhodnou k interakcii.

Controller (riadič) reaguje na udalosti vykonané užívateľom a zaisťuje zmeny v modele či pohľade.

V implementovanej aplikácii model predstavuje fungovanie algoritmov a funkcie, ktoré ich popisujú. Pohľad je užívateľské okno s ktorým užívateľ pracuje a riadičom sú funkcie priradené jednotlivým komponentám užívateľského prostredia. Pomocný framework JavaFX pomáha s implementáciou tohto modelu k tomu, aby bolo možné jednoduchšie prepojiť akcie užívateľa so želanou funkcionalitou aplikácie. Napríklad, na Obrázku B.2 je zobrazená úvodná obrazovka aplikácie. Pre toto okno je vytvorený samostatný fxml súbor, ktorý popisuje aké komponenty majú v danom okne byť a ako majú byť umiestnené. Súbor `.fxml` predstavuje pohľad. Ďalej je pre daný fxml súbor vytvorená trieda, ktorá predstavuje jeho kontrolér. Kontrolér, po akcii označenia voľby Functions zariadi, aby v možnostiach pre následný výber algoritmu bolo možné vybrať iba algoritmy, ktoré boli napísané pre riešenie vybraného problému. Model v tomto prípade ostáva nedotknutý a zatiaľ ostáva voči užívateľovi skrytý. Zvyčajne užívateľ ani priamo z modelovou časťou pracovať nebude, pretože všetky jeho akcie povedú cez kontrolér daného okna.

4.2 Užívateľské rozhranie

Celé užívateľské rozhranie bolo navrhnuté so zameraním na jednoduchosť a na prípadné rozšírenie ďalšími algoritmami.
Každá z obrazoviek problémov sa delí na tri časti.

- Časť pre parametre problému.
- Časť pre parametre špecifické pre algoritmus.
- Časť pre zobrazenie výsledku, prípadne priebehu algoritmu.

Cieľom takéhoto návrhu obrazoviek bolo prípadné pridanie ďalších algoritmov. Teda v prípadne, že sa niekto rozhodne začleniť ďalší algoritmus pre riešenie daného problému nemusí preň vytvárať samostatnú obrazovku. Obrázok B.6 znázoruňuje možnosti, aké sa objavia pri rozkliknutí dalej voľby. Teda pri každom z problémov je možné sa vrátiť na začiatočnú obrazovku, prípadne aplikáciu ukončiť. Na Obrázku B.1 je zobrazená úvodná obrazovka aplikácie podľa ktorej sa ďalej rozhoduje o probléme, aký bude riešený a akým algoritmom. V ľavej časti úvodnej obrazovky si môže užívateľ vybrať typ problému, pre ktorý bude chcieť užívateľ nájsť riešenie a v pravej časti si bude následne môcť zvoliť algoritmus pre daný problém. To, aké algoritmy za užívateľovi zobrazia záleží na zvolenom probléme, teda nemôže nastať situácia, že si užívateľ bude môcť zvoliť algoritmus, ktorý nie je naimplementovaný pre riešenie daného problému.

4.2.1 Užívateľské prostredie pre riešené problémy

Podľa výberu problému z úvodu sa vyberie obrazovka špecifická pre daný problém.
Pre problém obchodného cestujúceho je znázornené okno na Obrázku B.3. Výstupom tohto

algoritmu je zobrazenie grafu v pravej časti programu. Užívateľ má možnosť spustiť algoritmus ešte raz pre znázornené body pomocou tlačidla `Run again` alebo vygenerovať nové pozície miest pomocou tlačidla `Start new`.
Obrázok B.4 znázorňuje okno pre problém splniteľnosti booleovských formúl, ktoré užívateľ zadá na v ľavej časti programu. Ak bolo nájdené nejaké riešenie, bude v pravej časti vypísané číslo iterácie v ktorom bolo nájdené a pod ním riešenia, aké boli nájdené a aký jedinec ich reprezentoval. Nie je nutné zadávať klauzuly pre formulu v konjunkívnej normálnej forme, ale je to odporúčané. Medzi jednotlivými klauzulami je implicitne použitá logická spojka `AND` a negovaná hodnota je definovaná ako `!a`. Premenná taktiež musí byť definovaná len ako jediné písmeno a logické spojky musia byť znázornené ako `&&` pre `AND` a `||` pre `OR`.
Pre hľadanie extrému funkcie je okno aplikácie znázornené na Obrázku B.5. Výstupom algoritmu je písomný zápis súradníc bodu, ktorý bol vybraný ako výsledný a jeho funkčná hodnota pre vybranú funkciu. Užívateľ má na výber z preddefinovaných štyroch funkcii prípadne si môže zvoliť vlastnú funkciu.
Veľkým vylepšením tejto aplikácie by bola možnosť krokovania týchto algoritmov, no pre často veľký počet iterácii by musel byť vhodne zvolený krok prípadne aj to, čo je pre užívateľa potrebné vidieť v každej iterácii. V takejto podobe programu, sú iterácie vypísané ako konzolový výstup, ktorý si užívateľ môže následne pozrieť a zhodnotiť.

Kapitola 5

Experimenty a ich výsledky

V tejto kapitole sú opísané všetky experimenty a ich výsledky aké boli prevádzané na implementovaných algoritmoch pomocou aplikácie vytvorenej v tejto práci. Každý z algoritmov pracuje s vlastnými, špeciálnymi parametrami, podľa ktorých dokáže nadobúdať rozličné výsledky. Pri každom z experimentov sú zhrnuté jeho výsledky aké dosiahol pre danú kombináciu parametrov. Účelom experimentov je ukázať silu týchto parametrov, ktorú môžu mať na celkový výsledok algoritmu.
Každý z optimalizačných problémov bol riešený iným algoritmom vzhľadom k čomu boli upravené aj parametre algoritmu pre daný problém.

5.1 Problém obchodného cestujúceho - ACO

V prípade problému obchodného cestujúceho riešeného pomocou optimalizácie mravčou kolónie je možné nastaviť hodnoty pre päť parametrov

- počet miest, ktoré má cestujúci navštíviť,
- počet agentov, ktorí budú hľadať cestu v každej iterácii,
- počet iterácii, počas akých majú agenti hľadať najoptimálnejšie riešenie,
- parameter α, ktorý udáva akú váhu pri rozhodovaní agenta má mať hodnota feromónu,
- a parameter β, ktorý udáva akú váhu má mať pri rozhodovaní agenta o ďalšej ceste dĺžka cesty.

Spoločným parametrom pre každý z experimentov bol počet miest a taktiež ich umiestnenie, kedže hlavným cieľom je nájsť čo najkratšiu cestu medzi mestami.
Experimenty rozdelené do troch podkategórii podľa hodnôt paramterov α a β. V prvej podkategórii je hodnota parametrov α a β rovnaká, čiže váha hodnôt feromónu a dĺžky cesty pri výbere nasledujúceho mesta. V druhej podkategórii je váha parametru α o polovicu nižšia ako váha parametru β a teda pri rozhodovaní má väčšiu váhu dĺžka danej cesty ako hodnota feromónu na nej a v poslednej podkategórii má väčšiu silu parameter α kedy má pri rozhodovaní väčšiu silu hodnota feromónu, aký na ceste mravce zanechali.
Každá podkategória ďalej obsahuje tri druhy experimentov s parametrami pre počet agentov a počtom iterácii. V prvom experimente je počet agentov a iterácii rovnaký, v druhom je zvýšený počet agentov a v treťom je zvýšený počet iterácii.

Pre každý z experimentov bolo prevedených desať testov. Výsledok testu pozostáva z hodnoty dĺžky celkovej cesty, ktorá bola v teste získaná ako najkratšia a čísla, ktoré určuje poradie iterácie v akom sa táto najkratšia cesta našla.
Každý test pozostáva z pätnástich miest, ktorých súradnice sú uvedené v prílohe A.1.1. A každé mesto je prepojené s každým. Najkratšia možná cesta má dĺžku $\doteq 36.3$.

Podkategória 1

V tejto podkategórii mali parametre α a β rovnakú hodnotu. Presnejšie $\alpha = 1$ a aj $\beta = 1$. Konkrétne výsledky prevedených experimentov sú opísané v prílohe A.1.2. V jednotlivých experimentoch tejto sekcie sú uvedené najlepšie, najlepšie nájdené výsledky podľa dĺžky nájdenej cesty a priemerný výsledok všetkých testov.

Experiment 1

V tomto experiemnte bol počet mravcov a počet iterácii rovnaký. Obe parametre mali hodnotu 100.

Najlepší výsledok	37.3	Priemer výsledkov	39.82	Najhorší výsledok	43.0
Iterácia	56	Priemer iterácii	43.5	Iterácia	93

Experiment 2

Tento experiment pracuje s vyšším počtom agentov ako iterácii. Využitých bolo 300 agentov a 100 iterácii.

Najlepší výsledok	37.0	Priemer výsledkov	39.28	Najhorší výsledok	41.1
Iterácia	28	Priemer iterácii	37.7	Iterácia	85

Experiment 3

V poslednom experimente v tejto kategórii bol zvýšený počet iterácii. Počet agentov bol 100 a vykonaných bolo 300 iterácii.

Najlepší výsledok	37.9	Priemer výsledkov	39.22	Najhorší výsledok	42.2
Iterácia	140	Priemer iterácii	100.5	Iterácia	7

Zhrnutie

Z prevedených experimentov bol najlepší výsledok výsledok získaný v experimente 2 kedy bolo pre hľadanie najkratšej cesty väčšie množstvo agentov. Najlepší priemer výsledkov podľa dĺžky nájdenej cesty mal experiment 3, no priemer počtu iterácii bol najvyšší zo všetkých experimentov. Bohužiaľ ani jeden z týchto experimentov nenašiel najkratšiu cestu, ktorej dĺžka bola $\doteq 36.3$.

Pokategória 2

V tejto podkategórii mal parameter α nižšiu hodnotu ako β. Presnejšie $\alpha = 0,5$ a aj $\beta = 1$. Získané výsledky z jednotlivých testov sú uvedené v prílohe A.1.3. V tejto časti sú uvedené najlepší, priemerný a najhorší získaný výsledok pre každý experiment.

Experiment 4

V tomto experimente bol počet mravcov a počet iterácii rovnaký. Obe parametere mali hodnotu 100.

Najlepší výsledok	36.6	Priemer výsledkov	37.35	Najhorší výsledok	38.6
Iterácia	29	Priemer iterácii	43.1	Iterácia	35

Experiment 5

V druhom experimente tejto podkategórie bol zvýšený počet mravcov na 300 a počet iterácii ostal rovnaký, teda 100.

Najlepší výsledok	36.6	Priemer výsledkov	36.88	Najhorší výsledok	37.3
Iterácia	19	Priemer iterácii	48.9	Iterácia	64

Experiment 6

V poslednom experimente tejto podkategórie bol zvýšený počet iterácii na 300 a počet mravcov bol 100.

Najlepší výsledok	36.6	Priemer výsledkov	36.9	Najhorší výsledok	37.2
Iterácia	116	Priemer iterácii	192.2	Iterácia	42

Zhrnutie

Aj v tejto podkategórii bol najlepší výsledok z prevedených testov získaný v experimente 2 kedy bol zvýšený počet agentov. V tomto experimente bol výsledok 36.6 nájdený tri krát, no v jednom z nich už pri devätnástej iterácii. Tento experiment má aj najlepší priemer zo všetkých získaných výsledkov, no veľmi podobný priemer má aj experiment 6, kde bol zvýšený počet iterácii. Tak ako v podkategórii 1, ani v týchto experimentoch nebol nájdený úplne najlepší výsedok $\doteq 36.3$

Podkategória 3

V tejto podkategórii mal parameter β nižšiu hodnotu ako α. Presnejšie $\alpha = 1$ a aj $\beta = 0,5$. Výsledky jednotlivých testov pre experimenty sú uvedené v prílohe A.1.4. V uvedených experimentoch sú zosumarizované - najlepší výsledok, najhorší výsledok a priemer všetkých získaných výsledkov.

Experiment 7

V prvom experimente tejto podkategórie bol počet mravcov 100 a počet iterácii taktiež 100.

Najlepší výsledok	36.7	Priemer výsledkov	37.79	Najhorší výsledok	38.3
Iterácia	71	Priemer iterácii	66.5	Iterácia	89

Experiment 8

V ďalšom experimente bol počet mravcov zvýšený na 300 a počet iterácii bol 100.

Najlepší výsledok	36.3	Priemer výsledkov	36.95	Najhorší výsledok	37.8
Iterácia	38	Priemer iterácii	71.1	Iterácia	44

Experiment 9

V poslednom z experimentov bol zvýšený počet iterácii na 300 a počet mravcom bol 100.

Najlepší výsledok	36.6	Priemer výsledkov	36.89	Najhorší výsledok	37.5
Iterácia	237	Priemer iterácii	225.9	Iterácia	285

Zhrnutie

Z vykonaných experimentov bol najlepší výsledok získaný v experimente 8 kedy bolo pre hľadanie najkratšej cesty väčšie množstvo agentov, tento výsledok je naj najlepším možným aký sa dá pri danom rozložení bodov nájsť. Najlepší priemer v tejto kategórii experimentov mal experiment 9, no mal taktiež aj najvyšší priemer iterácii.

5.2 Spĺniteľnosť logických formúl - MBO

Pri probléme spĺniteľnosti logických formúl je možné pre algoritmus MBO nastaviť štyri parametre

- počet páriacich letov kráľovnej
- celkový počet trúdov, z ktorých si kráľovná bude v páriacom lete vyberať
- počet potomkov, ktorý po páriacom lete vznikne
- veľkosť spermotéky kráľovnej, teda maximálny počet párení počas páriaceho letu

Prevedené boli tri experimenty. Prvý z experimentov hľadal riešenie pre formulu 5.1, ktorá pozostáva zo šiestich klauzúl s ôsmimi premennými.

$$F = (a \vee e) \wedge (g \vee \neg h \vee e) \wedge (\neg c \vee d) \wedge (b) \wedge (\neg f \vee e \vee c) \wedge (\neg h) \wedge (a \vee g) \tag{5.1}$$

Ďalšie dva experimenty pracovali s formulou definovanou (5.2). Táto formula pozostáva z ôsmich klauzúl, kde každá z nich predpisuje každej premennej jedinú možnú hodnotu. Čiže existuje len jediné riešenie tejto formuly ak sa môže premenným priradiť iba hodnota 1 alebo 0.
V prvom z experimentov s formulou (5.2) sú počiatočné hodnoty voliteľných parametrov rovnaké ako v experimente s formulou (5.1) a v druhom bol zvýšený počet trúdov a potomkov.

$$F = (a) \wedge (\neg b) \wedge (c) \wedge (\neg d) \wedge (e) \wedge (\neg f) \wedge (g) \wedge (\neg h) \tag{5.2}$$

V každom z experimentov bolo prevedených 10 testov. Pri každom teste je uvedené číslo páriaceho letu v akom bolo nájdené riešenie a aký jedinec dané riešenie reprezentoval. Označenie P hovorí, že riešenie našiel genotyp potomka, T trúda a K kráľovnej.

Experiment 1

V tomto experimente bolo hľadné riešenie pre formulu (5.1) a hodnoty parametrov boli nasledovné:

Počet páriacích letov - 15

Počet trúdov - 30

Počet potomkov - 10

Veľkosť spermotéky kráľovnej - 20

Test č.	1	2	3	4	5	6	7	8	9	10
Let č.	1	1	1	1	1	1	1	1	1	1
Jedinec	P, T, K	P, T	P, T, K	P, T	P, T	T	P, T	P, T	T	P, T

Experiment 2

V tomto experimente bolo hľadané riešenie pre formulu (5.2) a hodnoty parametrov boli nasledovné:

Počet páriacích letov - 15

Počet trúdov - 30

Počet potomkov - 10

Veľkosť spermotéky kráľovnej - 20

Test č.	1	2	3	4	5	6	7	8	9	10
Let č.	2	4	3	7	3	6	3	8	2	2
Jedinec	P	T	T	P	T	T	P, T, K	T	T	P,T

Experiment 3

V tomto experimente bolo hľadané riešenie pre formulu (5.2) a oproti predošlému experimentu bol zvýšený počet trúdov a potomkov.

Počet páriacích letov - 15

Počet trúdov - 50

Počet potomkov - 20

Veľkosť spermotéky kráľovnej - 20

Test č.	1	2	3	4	5	6	7	8	9	10
Let č.	2	1	1	1	4	1	2	3	2	2
Jedinec	T	P	T	T	T	T	T	P, T, K	T	P, T

Zhrnutie

Každý z experimentov pracoval s formulou, ktorá obsahovala osem premenných. To znamená, že všetkých možností pre priradenie hodnôt 0 a 1 pre premenné bolo 2^8.
Experiment 1 pracoval s formulou, pre ktorú bolo možné nájsť viacero možných riešení. Pri daných hodnotách parametrov sa algoritmu podarilo nájsť riešenie v každom teste hneď v prvom páriacom lete. Z tohto dôvodu nebol prevedený žiaden ďalší experiment, ktorý by pracoval s vyššími hodnotami parametrov. Celkovo počas prevedených desiatich testov algoritmus našiel desať rôznych riešení, ktoré vyhovujú danej formule a jedincami, ktorí najčastejšie našli riešenie boli potomok a trúd.
Experimenty 2 a 3 hľadali riešenie pre formulu, pre ktorú zo všetkých možných vyhovuje iba jedno riešenie.

Premenná	a	b	c	d	e	f	g	h
Hodnota	1	0	1	0	1	0	1	0

Z výsledkov týchto experimentov je vidieť, že dané riešenie sa našlo v každom teste. Avšak pri zvýšenom počte potomkov a trúdov bolo riešenie nájdené, v najhoršom prípade v treťom páriacom lete, kde pri nižšom počte, v najhoršom prípade až v ôsmom lete. Jedincom, ktorý našiel riešenie najčastejšie bol trúd, no často krát aj potomok. Aj napriek tomu, že riešenie kráľovnej je na začiatku algoritmu vylepšované je tento záver očakávaný, keďže potomkov a trúdom je počet vyšší a kráľovná je len jedna. Prípadným pridaním heuristiky aj pre potomkov, by sa mohlo očakávať aj častejšie nájdenie riešenia potomkom.

5.3 Hľadanie extrému funkcie - GWO a SA

Hľadanie extrému funkcie bolo riešené pomocou dvoch algoritmov - GWO a SA. Pre experimenty boli vybrané dve funkcie a obe algoritmy hľadali ich minimum. Hľadanie extrému každej funkcie je rozdelené na časť pre algoritmus GWO pri ktorom je možné zvoliť hodnoty pre počet vlkov a počet iterácii, a pre SA začiatočnú teplotu, koncovú teplotu a krok ochladzovania.
Pri algoritme GWO boli vykonané tri experimenty. V prvom z nich bol počet vlkov a počet iterácii rovnaký. V druhom bol počet agentov vyšší a v poslednom bol vyšší počet iterácii. V algoritme SA boli prevedené dva druhy experimentov, jeden s nižšími hodnotami pre každý parameter a ďalší s vyššími.
Pre každý experiment bolo prevedených desať testov. Jednotlivé výsledky pre každý z testov sú uvedené v prílohe A.2 kde sú je uvedený bod, ktorý bol v danom teste výsledným a aká je jeho funkčná hodnota.
V kapitolách 5.3.1 a 5.3.2 sú zhrnuté výsledky každého z experimentov. Uvedené sú - najlepší výsledok, najhorší výsledok a priemer všetkých získaných výsledkov.

5.3.1 Bealeho funkcia

Bealeho funkcia je popísaná rovnicou (5.3). Jej minimum bolo hľadané na intervale $-4.5 < x, y < 4.5$. V tomto intervale má funkcia globálne minimum v bode $[3, 0.5]$ s funkčnou hodnotou 0. Graf funkcie je znázornený na Obrázku 5.1.

$$f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2 \tag{5.3}$$

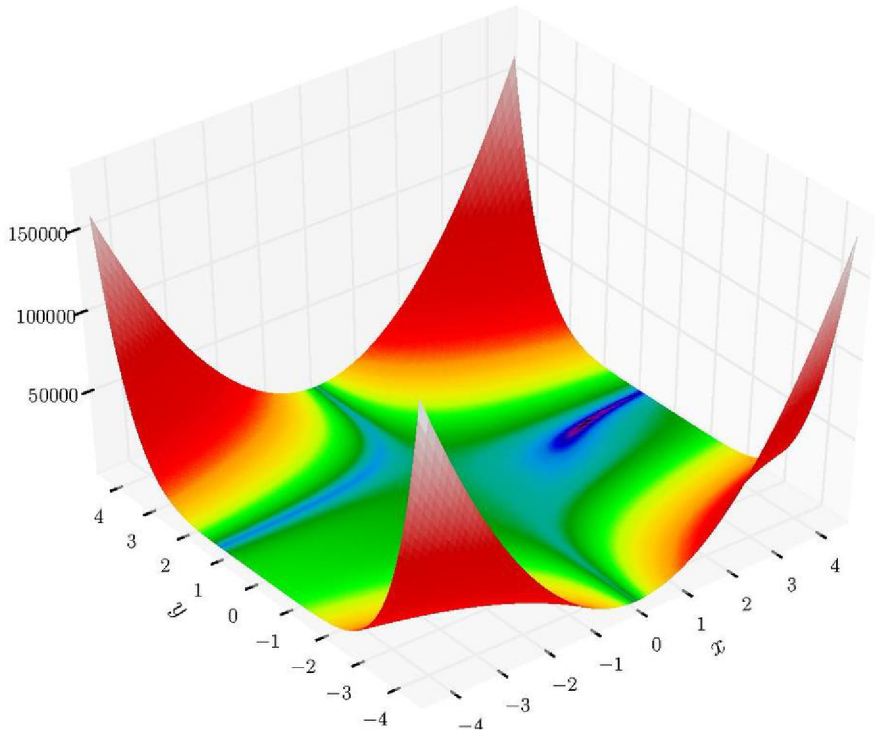
Algoritmus vĺčej svorky

V tejto sekcii sú popísané výsledky algoritmu GWO pre hľadanie extrému Bealeho funkcie (5.3). Výsledky všetkých prevedených testov sú uvedené v prílohe A.2.1.

Experiment 1

V tomto experimente pre algoritmus GWO bol počet vlkov a počet iterácii zvolený na hodnotu 50.

Najlepší výsledok	0.04	Priemer výsledkov	0.578	Najhorší výsledok	1.23
Bod	[3.46, 0.57]	Priemer bodov	[3.03, 0.43]	Bod	[1.8, 0.31]



Obr. 5.1: Graf Bealeho funkcie.
[5]

Experiment 2

Tento experiment pracoval so zvýšeným počtom vlkov na 200 počas päťdesiatich iterácii.

Najlepší výsledok	0.01	Priemer výsledkov	0.072	Najhorší výsledok	0.21
Bod	[3.29, 0.57]	Priemer bodov	[3.06, 0.49]	Bod	[2.74, 0.31]

Experiment 3

V poslednom experimente pre hľadanie minima Bealeho funkcie pomocou GWO bol počet vlkov 50 a zvýšený bol počet iterácii na 200.

Najlepší výsledok	0.14	Priemer výsledkov	0.81	Najhorší výsledok	1.66
Bod	[3.74, 0.68]	Priemer bodov	[2.67, 0.35]	Bod	[1.62, -0.20]

Zhrnutie

Najlepší výsledok získal algoritmus GWO pri hľadaní minima Bealeho funkcie (5.3) v experimente 2, kedy sa najviac priblížil bodu $[3, 0.5]$ v ktorom má funkcia na danom intervale svoje minimum. V tomto algoritme bolo použitých vyššie množstvo vlkov, čo mohlo pomôcť k rýchlejšej detekcii daného globálneho minima. Je tiež potrebné dodať, že aj napriek tomu, že sa priemer bodov priblížil v dvoch z týchto experimentov k bodu $[3, 0.5]$ nie je nutné, že aj priemer týchto bodov sa bude z daným priemerom zhodovať. Rozhodujúca je stále funkčná hodnota a aký z experimentov sa k nej priblížil najviac.

Algoritmus simulovaného žíhania

V tejto sekcii sú popísané výsledky algoritmu SA pre hľadanie extrému Bealeho funkcie.

Experiment 4

V tomto experimente je pre algoritmus SA hodnota začiatočnej teploty 500, koncovej teploty 1 a ochladzovacieho kroku 10.

Najlepší výsledok	0.02	Priemer výsledkov	0.49	Najhorší výsledok	1.51
Bod	[3.16, 0.56]	Priemer bodov	[2.45, 0.54]	Bod	[1.88, 0.43]

Experiment 5

Pre tento experiment sú zvýšené hodnoty všetkých experimentov. Začiatočná teplota má hodnotu 1000, koncová 10 a ochladzovací krok 50.

Najlepší výsledok	0.01	Priemer výsledkov	0.54	Najhorší výsledok	1.53
Bod	[2.91, 0.5]	Priemer bodov	[2.47, 0.51]	Bod	[-1.16, 1.49]

Zhrnutie

Najlepší výsledok našiel algoritmus SA pre Bealeho funkciu (5.3) v experimente 5, kedy boli použité vyššie hodnoty parametrov avšak počet iterácii bol nižší keďže ochladzovací krok mal vyššiu hodnotu.

Porovnanie GWO a SA pre Bealeho funkciu

Obe z týchto algoritmov dokázali detegovať a priblížiť sa ku globálnemu minimu. Ich výsledky sa môžu líšiť akurát v jednotlivých testoch a rozsahu bodov, ktoré kontrolovali, preto priemer výsledkov nemusí odpovedať funkčnej hodnote priemeru bodov.

5.3.2 Funkcia Three-hump camel

Funkcia Three-hump camel je popísaná rovnicou (5.4). Jej minimum bolo hľadané na intervale $-5 < x, y < 5$. V tomto intervale má funkcia globálne minimum v bode $[0, 0]$ s funkčnou hodnotou 0. Graf funkcie je znázornený na Obrázku 5.2.

$$f(x, y) = 2x^2 - 1.05x^4 + \frac{x^6}{6} + xy + y^2 \tag{5.4}$$

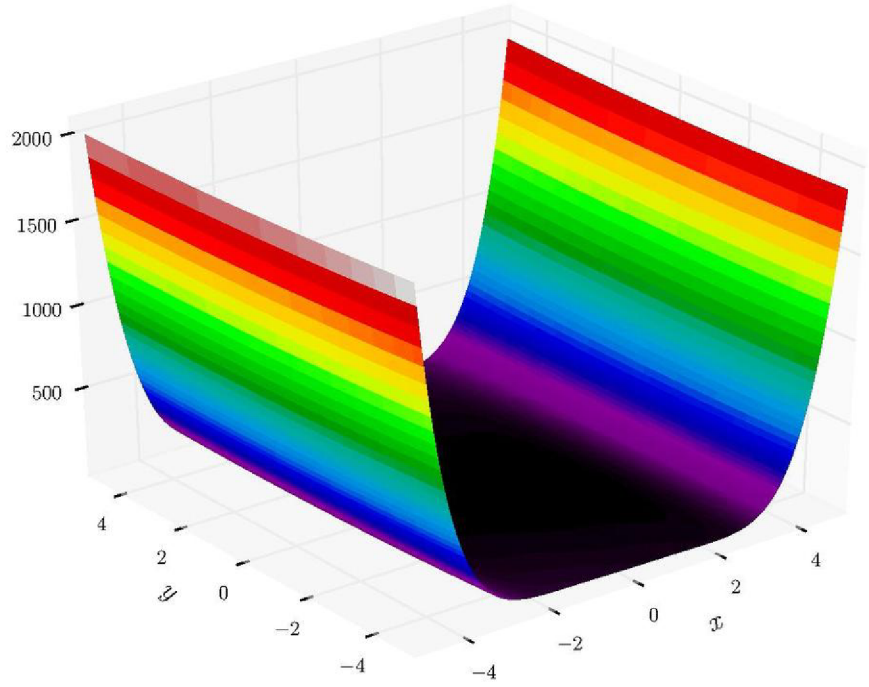
Algoritmus vĺčej svorky

V tejto sekcii sú popísané výsledky algoritmu GWO pre hľadanie extrému funkcie Three-hump camel.

Experiment 1

V prvom experimente je pre tento algoritmus počet vlkov a iterácii rovnaký a to 50.

Najlepší výsledok	0.03	Priemer výsledkov	0.21	Najhorší výsledok	0.53
Bod	[-0.13, 0.05]	Priemer bodov	[0.11, -0.008]	Bod	[0.3, 0.47]



Obr. 5.2: Graf funkcie Three-hump camel. [6]

Experiment 2

Ďalší experiment pre algoritmus pracuje s vyšším počtom vlkov - 200, no počet iterácii je 50.

Najlepší výsledok	0.003	Priemer výsledkov	0.04	Najhorší výsledok	0.10
Bod	[0.04, 0.004]	Priemer bodov	[0.02, -0.02]	Bod	[0.22, -0.24]

Experiment 3

V tomto algoritme je pre algoritmus nastavení vyšší počet iterácii - 200 a počet vlkov je 50.

Najlepší výsledok	0.04	Priemer výsledkov	0.24	Najhorší výsledok	0.53
Bod	[0.04, 0.18]	Priemer bodov	[0.02, -0.14]	Bod	[-0.45, -0.24]

Zhrnutie

Z experimentov pre hľadanie minima funkcie Three-hump camel s algoritmom GWO bol nájdený najlepší výsledok v experimente 2, kde bol použitý vyšší počet vlkov.

Algoritmus simulovaného žíhania

V tejto sekcii sú popísané výsledky algoritmu SA pre hľadanie extrému funkcie Three-hump camel.

Experiment 4

V prvom experimente pre algoritmus SA pre funkciu Three-hump camel je začiatočná teplota nastavená na hodnotu 500, koncová teplota je 1 a ochladzovací krok má hodnotu 10.

Najlepší výsledok	0.04	Priemer výsledkov	0.67	Najhorší výsledok	2.81
Bod	[0.08, 0.13]	Priemer bodov	[0.25, 0.05]	Bod	[1.80, 0.68]

Experiment 5

V druhom experimente pre funkciu Three-hump camel s algoritmom SA je začiatočná teplota nastavená na hodnotu 1000, koncová teplota je 10 a ochladzovací krok má hodnotu 50.

Najlepší výsledok	0.07	Priemer výsledkov	0.75	Najhorší výsledok	1.99
Bod	[0.009, 0.26]	Priemer bodov	[0.29, -0.34]	Bod	[-0.61, -0.90]

Zhrnutie

V prípade riešenie hľadania minima funkcie Three-hump camel algoritmom SA bol lepší výsledok nájdený v experimente 4, kde sú síce hodnoty parametrov nižšie, no vďaka menšiemu ochladzovacieho kroku nastalo viac iterácii.

Porovnanie GWO a SA pre funkciu Three-hump camel

Pri hľadaní minima funkcie Three-hump camel sa obe algoritmy priblížili ku globálnemu minimu funkciu avšak algoritmus GWO v experimente 2 našiel o niečo bližší výsledok ako algoritmus SA. V tomto prípade sa líšia aj priemery hodnôt výsledkov experimentov pre dané algoritmy. Kde priemery z experimentov s GWO majú o niečo nižšiu hodnotu ako v experimentoch pri algoritme SA.

Kapitola 6

Záver

Úvodné kapitoly tejto práce popisujú vznik, fungovanie a umelý analogický model vybraných optimalizačných algoritmov inšpirovaných prírodou. Tieto, ale aj mnohé iné algoritmy majú uplatnenie v rôznych oblastiach techniky, ale aj vedy a výskume, keďže optimalizácia procesov môže výrazne ušetriť čas a použité prostriedky.
V tejto práci boli ďalej pomocou vytvoreného programu riešené známe optimalizačné problémy, ktoré vedia ukázať silu týchto algoritmov. Vybrané popísané algoritmy sú metaheuristické, teda nedokážu garantovať, že ich riešenie je najlepším možným riešením problému. Sila týchto algoritmov spočíva v tom, že dokážu nájsť veľmi dobré, ak nie to najlepšie, riešenie komplexného problému za dobu, v ktorej ho deterministické algoritmy neboli schopné nájsť.
Prvým riešeným problémom v tejto práci bol problém obchodného cestujúceho. Riešením tohto problému je nájdenie cesty medzi všetkými zadanými mestami tak, aby cestujúci prešiel každým mestom práve jeden krát a ukončil svoju cestu v meste, v akom ju začal. Tento problém bol riešený algoritmom mravčej kolónie. Z výsledkov vykonaných experimentov sa ukázalo ako najvhodnejšie nastaviť pre tento algoritmus vyššiu hodnotu pre parameter počtu použitých mravcov, ktorí prehľadávali graf zadaný súradnicami miest.
Druhým riešeným problémom bolo hľadanie extrému funkcie. Tento problém bol riešený algoritmom vlčej svorky a algoritmom simulovaného žíhania, ktoré hľadali minimum pre Bealeho funkciu a pre funkciu Three-hump camel. V prípade prvej funkcie obe algoritmy dosiahli podobné výsledky, no pri funkcii Three-hump camel našiel algoritmus vlčej svorky lepšie riešenia o niečo častejšie. Obe algoritmy boli spúšťané pri rovnakých podmienkach a aj napriek tomu, že na čas trvania nebol kladený dôraz je vhodné spomenúť, že testy pre algoritmus simulovaného žíhania mali kratšie trvanie ako pre algoritmus vĺčej svorky. To mohlo byť spôsobené práve tým, že algoritmus simulovaného žíhania pracuje v iterácii s jedným bodom, kdežto algoritmus vlčej svorky potrebuje čas pre prepočet polohy každého vlka v iterácii.
Pri samotnom algoritme vlčej svorky, odhliadnúc od porovnania s algoritmom simulovaného žíhania, mal algoritmus najlepšie výsledky pre obe funkcie v prípade, kedy bol pre algoritmus nastavený väčší počet vlkov. Algoritmus simulovaného žíhania dosahoval lepšie výsledky v prípade, keď boli jeho parametre zvolené tak, aby mal jeho priebeh viac iterácii. Teda výraznejším faktorom bola veľkosť ochladzovacieho kroku.
Posledným riešeným problémom bolo nájdenie vhodnej kombinácii hodnôt pre premenné použité v booleovskej formule, tak aby každá z jej klauzúl nadobúdala pravdivú hodnotu a tým aj bola splnená aj celá formula. Tento problém bol riešený pomocou algoritmu párenia včiel. Algoritmus bol testovaný na dvoch formulách. Pre prvú formulu je možné nájsť

viacero rôznych riešení. Pri tejto formule dokázal nájsť algoritmus riešenie hneď vo svojej prvej iterácii a teda nebolo nutné zvyšovať hodnoty parametrov. Pri druhej formule, kedy existovalo len jedno riešenie, našiel algoritmus riešenie v každom teste, no pri vyššom počte potomkov a trúdov ho našiel o niečo rýchlejšie. Zaujímavým zistením je, že riešenie našiel častejšie trúd, ktorého reprezentácia riešenia, je generovaná náhodne v každej iterácii ako riešenie reprezentované potomkom, ktorého riešenie pozostáva z kríženia kráľovnej vylepšenej heuristikou a najlepšieho trúda v danej iterácii.

Literatúra

[1] Abbass, H. A.: MBO: Marriage in honey bees optimization-A haplometrosis polygynous swarming approach. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, ročník 1, IEEE, 2001, s. 207–214.

[2] Abbass, H. A.: A single queen single worker honey–bees approach to 3-SAT. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, Morgan Kaufmann Publishers Inc., 2001, s. 807–814.

[3] Brownlee, J.: *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee, 2011.

[4] Colorni, A.; Dorigo, M.; Maniezzo, V.; aj.: Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life*, ročník 142, Cambridge, MA, 1992, s. 134–142.

[5] Commons, W.: Graph of Beale's function. 2012, file:Beale's function.pdf.
URL https://commons.wikimedia.org/wiki/File:Beale%27s_function.pdf

[6] Commons, W.: Graph of Three Hump Camel function. 2012, file:Three Hump Camel function.pdf.
URL https://en.wikipedia.org/wiki/File:Three_Hump_Camel_function.pdf

[7] Dorigo, M.: Ant colony optimization. *Scholarpedia*, ročník 2, č. 3, 2007: str. 1461.

[8] Dorigo, M.; Birattari, M.; Stützle, T.: Ant Colony Optimization. *Computational Intelligence Magazine, IEEE*, ročník 1, 12 2006: s. 28–39, doi:10.1109/MCI.2006.329691.

[9] Dorigo, M.; Gambardella, L. M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, ročník 1, č. 1, 1997: s. 53–66.

[10] Dorigo, M.; Maniezzo, V.; Colorni, A.; aj.: Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, man, and cybernetics, Part B: Cybernetics*, ročník 26, č. 1, 1996: s. 29–41.

[11] Drštková, D.: 25. Rozmnožování včel a včelstev.
URL http://www.sci.muni.cz/ptacek/AFH-vypracovane-otazky/25-Rozmnoz-vcel-a-vcelstev-Drstkova.htm

[12] Hassanien, A. E.; Emary, E.: *Swarm intelligence: principles, advances, and applications*. CRC Press, 2018.

[13] Laporte, G.: The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, ročník 59, č. 2, 1992: s. 231–247.

[14] Lipowski, A.; Lipowska, D.: Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, ročník 391, č. 6, 2012: s. 2193–2196.

[15] Mirjalili, S.; Mirjalili, S. M.; Lewis, A.: Grey wolf optimizer. *Advances in engineering software*, ročník 69, 2014: s. 46–61.

[16] Muro, C.; Escobedo, R.; Spector, L.; aj.: Wolf-pack (Canis lupus) hunting strategies emerge from simple rules in computational simulations. *Behavioural processes*, ročník 88, č. 3, 2011: s. 192–197.

[17] Russell, S. J.; Norvig, P.: *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[18] Stützle, T.; Hoos, H. H.: MAX–MIN ant system. *Future generation computer systems*, ročník 16, č. 8, 2000: s. 889–914.

[19] Yang, X.-S.: *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[20] Yang, X.-S.: Metaheuristic optimization. *Scholarpedia*, ročník 6, č. 8, 2011: str. 11472.

Príloha A

Výsledky experimentov

A.1 TSP riešený algoritmom ACO

A.1.1 Súradnice miest použitých pre experimenty TSP.

Mesto	[x,y]
A	[3, 2]
B	[2, 7]
C	[8, 5]
D	[5, 9]
E	[4, 10]
F	[3, 6]
G	[3, 8]
H	[8, 2]
I	[2, 1]
J	[0, 0]
K	[2, 2]
L	[3, 3]
M	[8, 8]
N	[9, 9]
O	[10, 10]

A.1.2 Výsledky testov podkategórie 1 pre TSP.

- $\alpha = 1$
- $\beta = 1$

Rovnaký počet mravcov a iterácii.

- Počet mravcov = 100
- Počet iterácii = 100

Test č.	1	2	3	4	5	6	7	8	9	10
Výsledok	39.5	38.3	40.5	37.3	43.0	39.1	39.2	41.2	39.7	42.4
Iterácia	30	52	78	56	93	0	6	47	5	68

Vyšší počet mravcov ako iterácii.

- Počet mravcov = 300
- Počet iterácii = 100

Test č.	1	2	3	4	5	6	7	8	9	10
Výsledok	41.1	38.5	40.2	37.0	39.5	40.3	39.8	39.4	37.7	39.3
Iterácia	85	60	85	28	4	19	30	15	18	33

Vyšší počet iterácii ako mravcov.

- Počet mravcov = 100
- Počet iterácii = 300

Test č.	1	2	3	4	5	6	7	8	9	10
Výsledok	40.7	37.9	40.2	38.3	38.1	42.2	38.4	38.0	38.9	39.5
Iterácia	91	140	59	237	218	7	37	44	90	82

A.1.3 Výsledky testov podkategórie 2 pre TSP.

- $\alpha = 0.5$
- $\beta = 1$

Rovnaký počet mravcov a iterácii.

- Počet mravcov = 100
- Počet iterácii = 100

Test č.	1	2	3	4	5	6	7	8	9	10
Výsledok	37.4	37.8	36.9	37.2	37.3	38.6	37.4	37.6	36.6	36.7
Iterácia	10	32	57	30	88	35	86	35	29	29

Vyšší počet mravcov ako iterácii.

- Počet mravcov = 300
- Počet iterácii = 100

Test č.	1	2	3	4	5	6	7	8	9	10
Výsledok	36.9	36.7	36.6	36.6	37.2	37.1	36.9	37.3	36.9	36.6
Iterácia	59	35	58	52	15	21	93	64	73	19

Vyšší počet iterácii ako mravcov.

- Počet mravcov = 100
- Počet iterácii = 300

Test č.	1	2	3	4	5	6	7	8	9	10
Výsledok	36.9	36.9	36.9	37.1	36.9	36.6	36.9	36.9	36.7	37.2
Iterácia	264	234	63	260	139	116	288	219	297	42

A.1.4 Výsledky testov podkategórie 3 pre TSP.

- $\alpha = 1$
- $\beta = 0.5$

Rovnaký počet mravcov a iterácii.

- Počet mravcov = 100
- Počet iterácii = 100

Test č.	1	2	3	4	5	6	7	8	9	10
Výsledok	37.2	37.1	37.6	38.3	37.8	39.9	37.3	38.2	37.8	36.7
Iterácia	85	48	75	89	86	23	32	71	85	71

Vyšší počet mravcov ako iterácii.

- Počet mravcov = 300
- Počet iterácii = 100

Test č.	1	2	3	4	5	6	7	8	9	10
Výsledok	36.3	36.9	36.7	36.9	37.2	36.3	37.3	36.9	37.2	37.8
Iterácia	81	59	96	93	50	38	79	79	92	44

Vyšší počet iterácii ako mravcov.

- Počet mravcov = 100
- Počet iterácii = 300

Test č.	1	2	3	4	5	6	7	8	9	10
Výsledok	36.9	36.7	37.0	37.5	36.9	36.9	36.6	36.9	36.6	36.9
Iterácia	267	185	218	285	282	115	201	282	237	187

A.2 Hľadanie extrému funkcie

Všetky výsledky sú zaokrúhlené na dve desatinné miesta.

A.2.1 Bealeho funkcia - algoritmus GWO

Rovnaký počet vlkov a iterácii

- Počet vlkov = 50
- Počet iterácii = 50

Test č.	Bod [x,y]	Hodnota	Test č.	Bod [x,y]	Hodnota
1	[3.04, 0.37]	0.40	6	[4.58, 0.63]	1.02
2	[1.87, 0.24]	0.84	7	[1.7, -0.007]	1.21
3	[3.82, 0.73]	0.47	8	[4.03, 0.64]	0.16
4	[3.46, 0.57]	0.04	9	[1.8, 0.31]	1.23
5	[2.65, 0.37]	0.04	10	[3.35, 0.4]	0.37

Vyšší počet vlkov ako iterácii

- Počet vlkov = 200
- Počet iterácii = 50

Test č.	Bod [x,y]	Hodnota	Test č.	Bod [x,y]	Hodnota
1	[3.71, 0.62]	0.06	6	[2.54, 0.40]	0.08
2	[4.18, 0.69]	0.08	7	[3.0, 0.52]	0.01
3	[2.77, 0.49]	0.05	8	[2.74, 0.31]	0.21
4	[2.51, 0.42]	0.13	9	[3.29, 0.57]	0.01
5	[2.53, 0.34]	0.07	10	[3.34, 0.55]	0.02

Vyšší počet iterácii ako vlkov

- Počet vlkov = 50
- Počet iterácii = 200

Test č.	Bod [x,y]	Hodnota	Test č.	Bod [x,y]	Hodnota
1	[1.83, 0.31]	1.13	6	[3.74, 0.68]	0.14
2	[2.20, 0.15]	0.33	7	[3.94, 0.55]	0.79
3	[1.62, -0.20]	1.66	8	[1.97, 0.25]	0.63
4	[1.77, 0.06]	1.0	9	[2.97, 0.60]	0.28
5	[4.67, 0.83]	1.54	10	[1.99, 0.25]	0.58

A.2.2 Bealeho funkcia - algoritmus SA

Nižšie hodnoty parametrov

- Začiatočná teplota - 500
- Koncová teplota - 1
- Ochladzovací krok - 10

Test č.	Bod [x,y]	Hodnota	Test č.	Bod [x,y]	Hodnota
1	[3.03, 0.42]	0.15	6	[2.42, 0.15]	0.37
2	[2.94, 0.62]	0.45	7	[4.18, 0.73]	0.27
3	[-1.51, 1.45]	1.24	8	[3.16, 0.56]	0.02
4	[2.74, 0.26]	0.38	9	[1.88, 0.43]	1.5
5	[2.35, 0.28]	0.15	10	[3.32, 0.46]	0.36

Vyššie hodnoty parametrov

- Začiatočná teplota - 1000
- Koncová teplota - 10
- Ochladzovací krok - 50

Test č.	Bod [x,y]	Hodnota	Test č.	Bod [x,y]	Hodnota
1	[3.49, 0.42]	1.07	6	[2.47, 0.003]	1.00
2	[-1.16, 1.49]	1.52	7	[3.23, 0.58]	0.04
3	[2.09, 0.28]	0.44	8	[2.55, 0.42]	0.09
4	[3.30, 0.60]	0.06	9	[3.36, 0.39]	0,91
5	[2.96, 0.50]	0.01	10	[2.41, 0.44]	0.29

A.2.3 Funkcia Thee-camel hump - algoritmus GWO

Rovnaký počet vlkov a iterácii

- Počet vlkov = 50
- Počet iterácii = 50

Test č.	Bod [x,y]	Hodnota	Test č.	Bod [x,y]	Hodnota
1	[-0.36, 0.36]	0.24	6	[-0.31, 0.55]	0.31
2	[1.72, -0.76]	0.31	7	[-0.10, -0.26]	0.11
3	[-0.2, -0.14]	0.12	8	[-0.28, -0.17]	0.22
4	[0.14, -0.16]	0.04	9	[0.28, -0.03]	0.14
5	[0.30, 0.47]	0.52	10	[-0.13, -0.05]	0.03

Vyšší počet vlkov ako iterácii

- Počet vlkov = 200
- Počet iterácii = 50

Test č.	Bod [x,y]	Hodnota	Test č.	Bod [x,y]	Hodnota
1	[0.01, 0.21]	0.05	6	[0.04, 0.08]	0.04
2	[0.22, -0.24]	0.10	7	[0.04, 0.01]	0.004
3	[0.04, 0.004]	0.003	8	[-0.09, -0.21]	0.05
4	[0.05, -0.27]	0.06	9	[-0.05, 0.10]	0.009
5	[-0.06, 0.16]	0.01	10	[-0.10, -0.02]	0.02

Vyšší počet iterácii ako vlkov

- Počet vlkov = 50
- Počet iterácii = 200

Test č.	Bod [x,y]	Hodnota	Test č.	Bod [x,y]	Hodnota
1	[0.08, -0.41]	0.15	6	[-0.45, -0.24]	0.53
2	[0.04, 0.18]	0.04	7	[-0.36, -0.14]	0.31
3	[-1.83, 0.73]	0.4	8	[1.80, -0.73]	0.35
4	[0.18, 0.01]	0.06	9	[0.17, -0.51]	0.23
5	[0.26, -0.03]	0.12	10	[0.31, -0.24]	0.17

A.2.4 Funkcia Thee-camel hump - algoritmus SA

Nižšie hodnoty parametrov

- Začiatočná teplota - 500

- Koncová teplota - 1
- Ochladzovací krok - 10

Test č.	Bod [x,y]	Hodnota	Test č.	Bod [x,y]	Hodnota
1	[-0.25,-0.09]	0.16	6	[0.23, -0.67]	0.40
2	[0.26,-0.6]	0.36	7	[1.35, 0.19]	1.47
3	[1.80,0.68]	2.81	8	[0.26, -0.04]	0.13
4	[-0.71,0.80]	0.83	9	[-0.06, 0.22]	0.04
5	[-0.43,-0.12]	0.41	10	[0.08, 0.13]	0.04

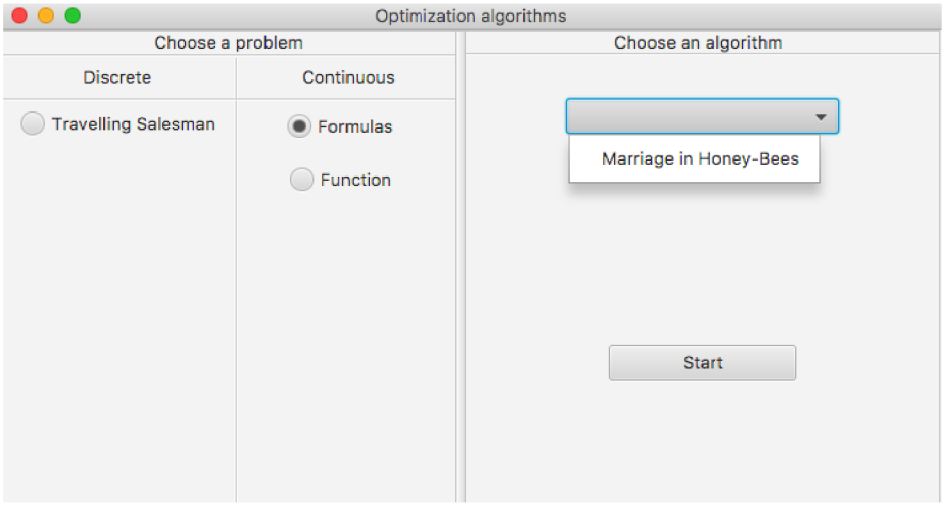
Vyššie hodnoty parametrov

- Začiatočná teplota - 1000
- Koncová teplota - 10
- Ochladzovací krok - 50

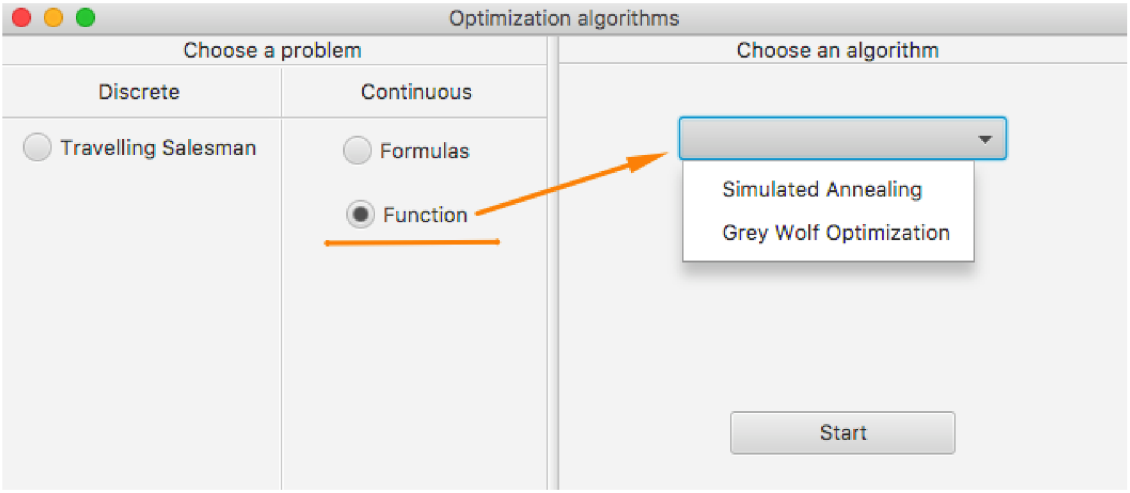
Test č.	Bod [x,y]	Hodnota	Test č.	Bod [x,y]	Hodnota
1	[0.19, 0.06]	0.08	6	[1.40, -0.01]	1.12
2	[1.50, -0.83]	0.52	7	[0.52, -0.10]	0.43
3	[-0.61, -0.90]	1.99	8	[-1.47, -0.12]	1.29
4	[0.009, 0.26]	0.07	9	[1.35, -1.12]	0.91
5	[1.79, -0.77]	0.32	10	[-1.78, 0.18]	0.80

Príloha B

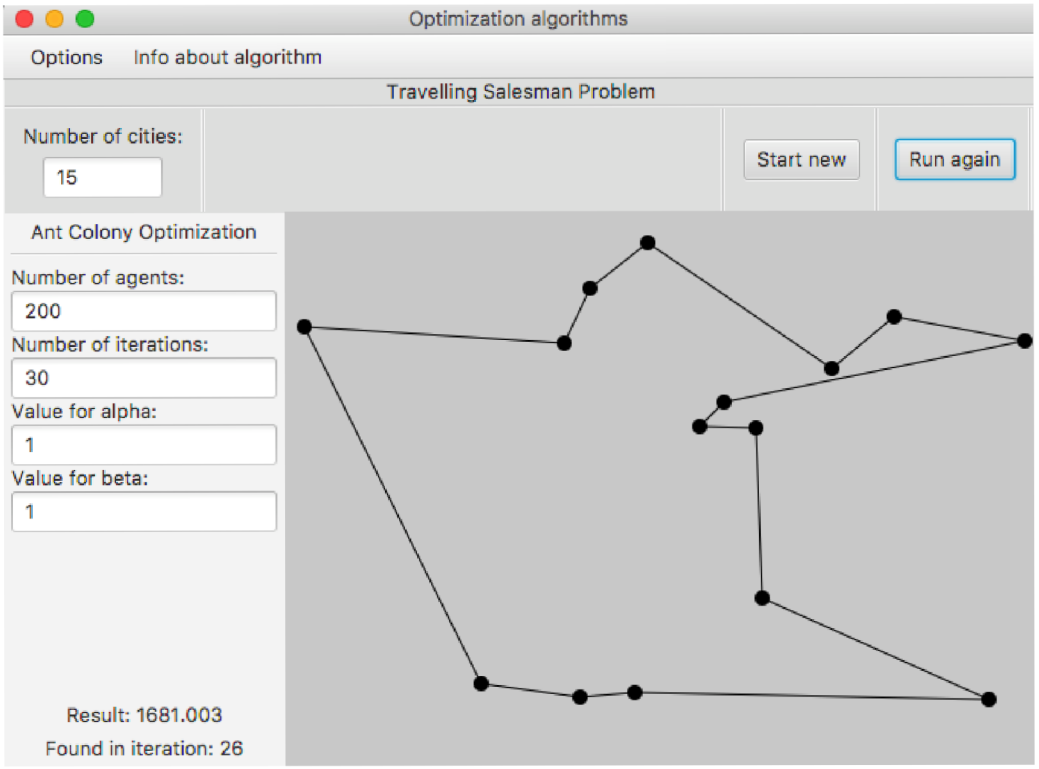
Obrázky uživateľského prostredia



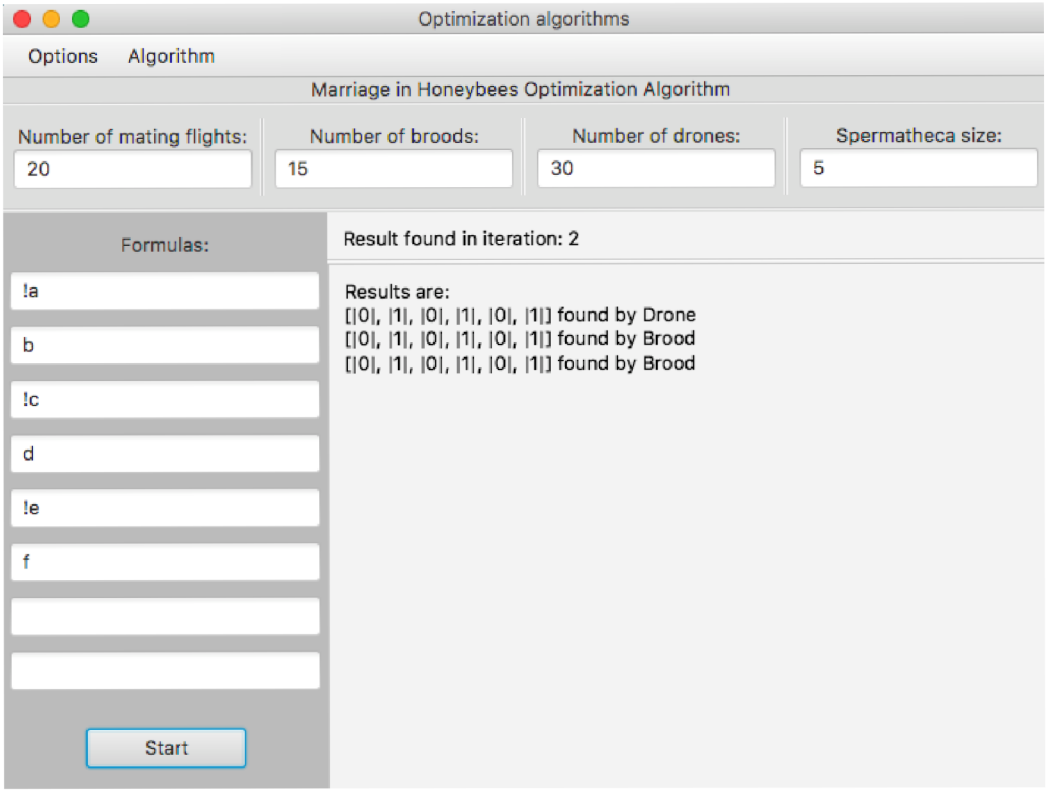
Obr. B.1: Úvodná obrazovka aplikácie s výberom problému a algoritmu.



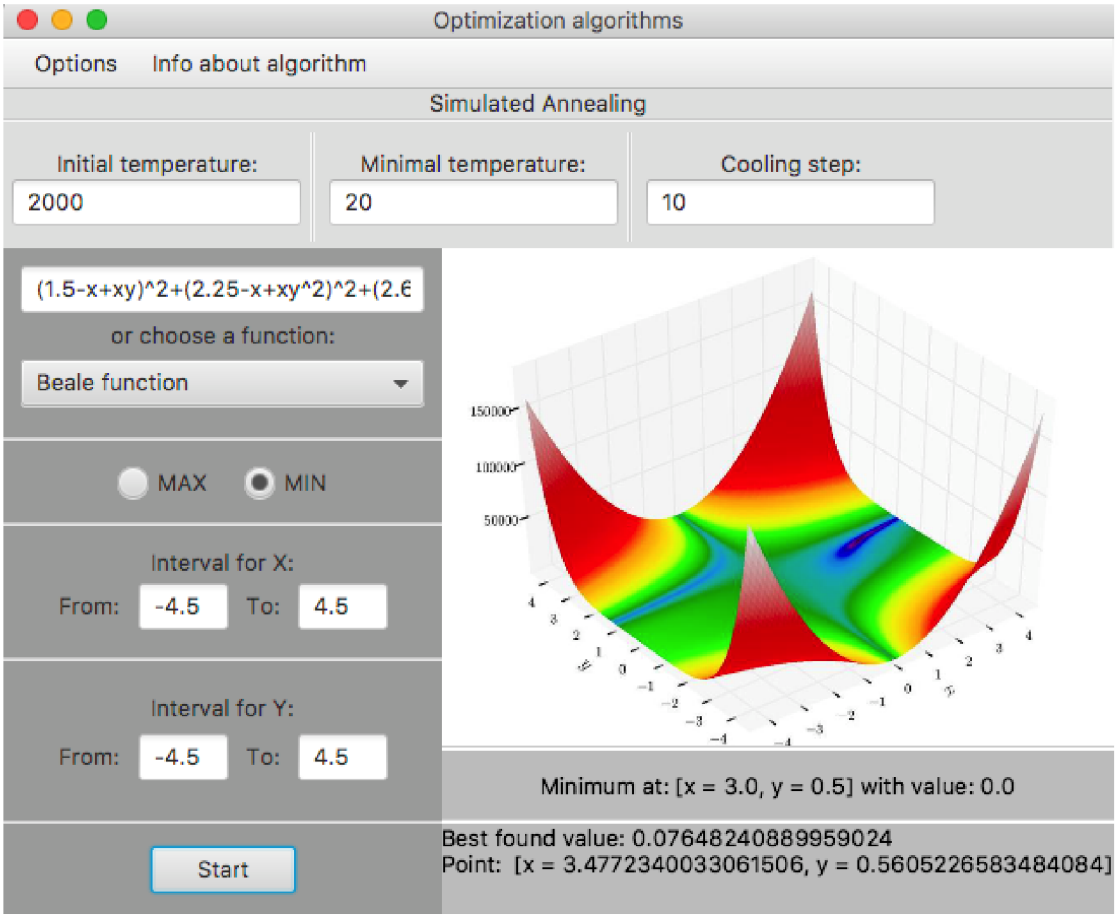
Obr. B.2: Ukážka úvodného okna aplikácie pre popis MVC modelu.



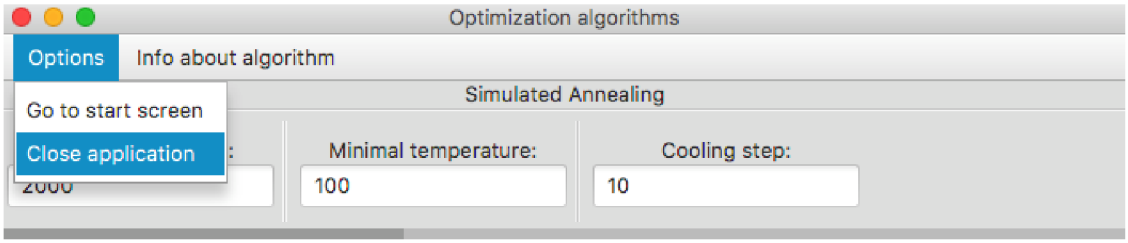
Obr. B.3: Okno aplikácie pre problém obchodného cestujúceho.



Obr. B.4: Okno aplikácie pre problém splniteľnosti formúl.



Obr. B.5: Okno aplikácie pre hľadanie extrému funkcie.



Obr. B.6: Vedľajší ovládací prvok pre vrátenie sa na začiatočné okno.