VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

GENETICKÉ ALGORITMY

GENETICS ALGORITHM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MÁRIA MASÁROVÁ**
AUTHOR

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

BRNO 2018

Zadání bakalářské práce/3454/2017/xmasar13

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Masárová Mária**
Obor: Informační technologie
Téma: **Genetické algoritmy**
Genetic Algorithms
Kategorie: Umělá inteligence

Pokyny:

1. Prostudujte zadanou literaturu, zaměřte se přitom na různá použití genetického algoritmu.
2. Navrhněte program, který bude demonstrovat činnost genetického algoritmu a bude umožňovat práci s různými typy chromozómů i parametrů.
3. Zvažte možnost rozšíření návrhu programu o některý přístup swarm inteligence.
4. Navržený program implementujte.
5. Proveďte experimenty alespoň se třemi rozdílnými optimalizačními úlohami (TSP, multidimensionální optimalizace, interpretace logických formulí, ap.)

Literatura:

- Iba, H., Noma, N.: New Frontier in Evolutionary Algorithms, Imperial College Press, 2012
- Ashlock, D.: Evolutionary Computation for Modeling and Optimization, Springer, 2006
- Ajith, A.: Evolutionary computation, in Handbook of Measuring System Design, John Wiley & Sons, 2005

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zbořil František V., doc. Ing., CSc.**, UITS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Táto práca sa zaoberá genetickými algoritmami, ich terminológiou a využitím. Popisuje rôzne problémy, ktoré sa dajú pomocou genetických algoritmov riešiť. V práci sú taktiež predstavené rôzne algoritmy skupinovej inteligencie, pričom algoritmus svetlušiek slúži aj na porovnanie efektivity medzi ním a genetickým algoritmom. Hlavnou úlohou tejto práce je vykonať experimenty s tromi optimalizačnými úlohami, konkrétne sú to, problém obchodného cestujúceho, splniteľnosť logických formúl a hľadanie extrému funkcie.

Abstract

This thesis deals with genetic algorithm, their terminology and use. It describes various problems that can be solved by using genetic algorithms. Different algorithms of swarm intelligence are also presented in this thesis, while firefly algorithm also serves to compare the efficiency between it and genetic algorithm. The main task of this thesis is to perform experiments with three optimization tasks, namely, travelling salesman problem, boolean satisfiability problem and searching for extreme in function.

Kľúčové slová

genetický algoritmus, chromozóm, inteligencia skupiny, algoritmus svetlušiek, problém obchodného cestujúceho, problém splniteľnosti logických formúl, hľadanie extrému funkcie

Keywords

genetic algorithm, chromosome, swarm intelligence, firefly algorithm, travelling salesman problem, boolean satisfiability problem, searching for extreme in function

Citácia

MASÁROVÁ, Mária. *Genetické algoritmy.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. František V. Zbořil, CSc.

Genetické algoritmy

Prehlásenie

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

.....
Mária Masárová
13. mája 2018

Poďakovanie

V prvom rade, by som sa chcela poďakovať pánovi Doc. Ing. Fratiškovi V. Zbořilovi CSc. za jeho cenné rady a odbornú pomoc pri písaní tejto bakalárskej práce. V druhom rade, by som chcela poďakovať svojej rodine a priateľom. Ďakujem vám za vašu podporu a trpezlivosť, ktorou ste ma zahŕňali po celé tri roky a najviac pri písaní tejto práce.

Obsah

1 Úvod **3**

2 Genetické algoritmy **4**
2.1 Základné pojmy . . . 5
2.2 Priebeh genetického algoritmu . . . 8
2.2.1 Tvorba počiatočnej populácie . . . 9
2.2.2 Ohodnotenie . . . 9
2.2.3 Reprodukcia . . . 9
2.2.4 Rekombinácia . . . 13
2.2.5 Mutácia . . . 14
2.2.6 Tvorba novej populácie . . . 14

3 Riešené problémy **15**
3.1 Problém obchodného cestujúceho . . . 15
3.1.1 Problém obchodného cestujúceho v teórií grafov . . . 15
3.1.2 Riešenie TSP pomocou genetických algoritmov . . . 15
3.2 Splniteľnosť logických formúl . . . 18
3.2.1 Riešenie SAT pomocou genetických algoritmov . . . 18
3.3 Hľadanie minima alebo maxima funkcie . . . 20
3.3.1 Riešenie funkcií pomocou genetických algoritmov . . . 20
3.3.2 Riešenie funkcií pomocou algoritmu svetlušiek . . . 21

4 Inteligencia skupiny **22**
4.1 Optimalizácia mravčou kolóniou . . . 23
4.2 Optimalizácia rojom častíc . . . 24
4.3 Optimalizácia umelým včelím rojom . . . 25
4.4 Algoritmus svetlušiek . . . 27

5 Popis implementácie **29**
5.1 Použité technológie . . . 29
5.2 Štruktúra programu . . . 29
5.2.1 Hierarchia tried . . . 30
5.3 Užívateľské rozhranie . . . 32
5.3.1 Obecný genetický algoritmus . . . 32
5.3.2 Problém obchodného cestujúceho . . . 33
5.3.3 Logické formule . . . 34
5.3.4 Funkcie . . . 34

6 Experimentovanie **36**
6.1 Obecný genetický algoritmus 36
6.1.1 Jednotlivé typy parametrov 36
6.1.2 Zhrnutie 38
6.2 Problém obchodného cestujúceho 39
6.2.1 Testovanie 39
6.2.2 Zhrnutie 41
6.3 Logické formule 41
6.3.1 Testovanie 41
6.3.2 Zhrnutie 42
6.4 Funkcie 43
6.4.1 Bealeho funkcia 43
6.4.2 Boothova funkcia 45
6.4.3 Zhrnutie 46

7 Záver **47**

Literatúra **48**

A Výsledky testov **50**

Kapitola 1

Úvod

Cieľom tejto bakalárskej práce je naštudovať problematiku genetických algoritmov, navrhnúť a implementovať program, ktorý demonštruje činnosť genetického algoritmu a taktiež preštudovť algoritmy skupinovej inteligencie.

Prácu môžeme rozdeliť na dve časti, teoretickú a praktickú. Do teoretickej časti zaradíme kapitoly dva, tri a štyri, zatiaľ čo do praktickej, kapitoly päť a šesť.

Druhá kapitola sa zaoberá genetickými algoritmami. Obsahuje vysvetlenie, čo vlastne genetický algoritmus je, prehľad terminológie, ktorá je potrebná pre jeho pochopenie a popis fungovania tohto algoritmu. Dopodrobna vykladá jednotlivé kroky algoritmu tak, aby algoritmus pochopil aj človek, ktorý sa s ním ešte nestretol. Vysvetľuje pojmy ako kríženie, mutácia a dedičnosť, ktoré sú známe už z prírody.

Tretia kapitola popisuje jednotlivé problémy, ktoré sa pomocou genetických algoritmov dajú riešiť, ako napríklad problém obchodného cestujúceho (angl. travelling salesman problem), problémy splniteľnosti (angl. satisfiability) logických formúl, ktoré sa skrátene označujú SAT alebo hľadanie maxima, prípadne minima funkcií. Práve na hľadaní extrému funkcie je porovnávaný genetický algoritmus s algoritmom skupinovej inteligencie, konkrétne s algoritmom svetlušiek.

Štvrtá kapitola popisuje jednotlivé algoritmy inteligencie skupiny, ktorými sú napríklad optimalizácia mravčou kolóniou (angl. ant colony optimization), algoritmus svetlušiek (angl. firefly algorithm), roj častíc (angl. particle swarm intelligence) alebo optimalizácia umelým včelím rojom (angl. artificial bee colony optimization).

Piata kapitola dopodrobna opisuje vytvorený program, nástroje, ktoré boli použité a návod ako s ním pracovať. Taktiež obsahuje vzhľad grafického užívateľského rozhrania, návod na spustenie programu a vysvetlivky.

Šiesta kapitola popisuje jednotlivé experimenty, ktoré boli vykonané na všetkých riešených problémoch. Zhodnocuje získané výsledky a na ich základe porovnáva efektivitu jednotlivých parametrov genetického algoritmu, ako aj rozdiel genetického algoritmu a algoritmu svetlušiek.

Nasleduje záver, ktorý zhŕňa a vyhodnocuje zistené poznatky o genetických algoritmoch a ich použití, ako aj ich porovnanie s algoritmom svetlušiek.

Kapitola 2

Genetické algoritmy

Genetické algoritmy patria do triedy evolučných algoritmov, ktoré používajú techniky napodobujúce evolučné procesy z biológie (dedičnosť, prirodzený výber, kríženie a mutácia) pre nájdenie najlepšieho riešenia. Sú to vyhľadávacie algoritmy založené na prirodzenom výbere a princípoch genetiky. Genetický algoritmus je heuristický postup, ktorý sa snaží aplikáciou týchto princípov nájsť riešenie zložitých problémov, pre ktoré neexistuje použiteľný exaktný algoritmus. Základný genetický algoritmus je veľmi všeobecný a existuje mnoho aspektov, ktoré sa dajú implementovať odlišne podľa riešeného problému. Napríklad reprezentácia chromozómu, typ kódovania, stratégia výberu rodičov, typ kríženia a mutácie. Vychádzajú z Mendelových zákonov dedičnosti a Darwinovej teórie vývoja [1].

Mendelove zákony dedičnosti - Tieto zákony zhŕňajú pravidlá, ktoré sa uplatňujú pri dedičnosti znakov. Rozdeľujú sa na fenotypové a genotypové zákony.

Darwinova teória vývoja (evolúcie) - Jedinci, ktorí sa lepšie adaptujú na okolité prostredie majú väčšiu šancu na prežitie a tým na plodenie ďalších potomkov.

Výhody

- Výkonnosť algoritmu je nezávislá od reprezentácie v porovnaní s inými numerickými technikami, ktoré by mohli byť použiteľné len pre kontinuálne hodnoty alebo iné obmedzené množiny.
- Hodnotenie každého riešenia môže byť vypočítané paralelne a len výber rodičov, ktorý potrebuje aspoň súťaž dvojice, si vyžaduje určité sériové spracovanie.
- Tradičné metódy optimalizácie nedovoľujú dynamické zmeny v probléme s prostredím a často vyžadujú úplne opätovné spustenie s cieľom poskytnúť riešenie. Naproti tomu, genetické algoritmy môžu byť použité na prispôsobenie riešení meniacim sa okolnostiam.
- Sú jednoduché a použiteľné na veľkej škále rôznych úloh. V praxi sa používajú predovšetkým na optimalizáciu (riadenie výroby, riešenie problémov v priemysle, doprave a logistike, finančníctve, matematike).

2.1 Základné pojmy

Genetické algoritmy napodobňujú evolučné procesy z biológie, a preto je zrejmé, že majú podobnú terminológiu ako v genetike. V tejto časti sú vysvetlené jednotlivé pojmy z hľadiska biológie a z hľadiska informatiky, ako aj ich využitie v genetických algoritmoch. Tieto pojmy sú dôležité pre neskoršie pochopenie celého genetického algoritmu, pretože sú jeho stavebnými kameňmi a bez nich je zložité genetický algoritmus popísať.

Genotyp

Genotyp je súbor všetkých foriem génov v bunke jedinca, obsahujúci všetky informácie určujúce podobu a fungovanie všetkých jeho tkanív a orgánov. V genetických algoritmoch genotypom rozumieme genetický kód jedinca, ktorý je ekvivalentný chromozómom v bunkách.

Fenotyp

Súbor všetkých pozorovateľných vlastností a znakov, ktorý v danom prostredí odpovedá genotypu. Čiže platí, fenotyp = genotyp + prostredie. Fenotypy môžeme rozdeliť do dvoch kategórií, ak prostredie nemá vplyv, a ak prostredie má vplyv na fenotyp. Do prvej kategórie môžeme zaradiť napríklad farbu očí jedinca alebo jeho krvnú skupinu. Do druhej kategórie patrí napríklad správanie jedinca.

Jedinec

Jedinec je jedinečný predstaviteľ nejakého celku, napríklad populácie. Je definovaný genotypom, pričom každý jedinec má unikátny genotyp. V genetických algoritmoch býva jedinec definovaný jediným chromozómom, ktorý má pevnú, predom stanovenú dĺžku.

Alela

Je konkrétna forma génu. Každý gén môže mať jednu alebo niekoľko foriem. Ak je alel viac, jedná sa o genetický polymorfizmus. Rôzne alely definujú formy podstatne odlišné vo svojich prejavoch. Alela môže byť dominantná alebo recesívna. Dominantná alela pri stretnutí s recesívnou alelou v jednom organizme úplne potlačí jej prejav. V genetických algoritmoch alely rozumieme ako rôzne hodnoty, ktoré môže nadobúdať gén. Pri binárnej reprezentácii to je napríklad 0 a 1.

Gén

Je úsek, sekvenica, ktorá kóduje informáciu pre tvorbu nejakého produktu. Gén nadobúda rôzne hodnoty, ktorým sa hovorí alely. To znamená, že gén, ktorý ovláda farbu očí, môže mať rovnaké, ale aj odlišné alely. Preto budú mať dvaja jedinci rovnakú alebo rôznu farbu očí. V genetických algoritmoch gén reprezentuje parametre jedinca a je značne zjednodušený od génu z hľadiska biológie.

Chromozóm

Je nositeľ genetickej informácie. Chromozóm sa delí na jednotlivé gény, ktoré sú lineárne usporiadané. To znamená, že i-tý gén chromozómu rovnakého typu reprezentuje tú istú

vlastnosť. V genetických algoritmoch môže byť chromozóm zakódovaný rôzne a spôsob kódovania ovplyvňuje úspech alebo neúspech genetického algoritmu na danej úlohe. Typy kódovania sú[1]:

- **binárne kódovanie** - je najbežnejší typ kódovania. Každý chromozóm je reťazec bitov, 0 alebo 1. Binárne kódovanie poskytuje mnoho možných chromozómov, dokonca aj s malým množstvom alel. Na druhej strane však toto kódovanie nie je prirodzené pre mnohé problémy a niekedy je potrebné vykonať opravy po prekrížení alebo mutácií. Príklady chromozómov s binárnym kódovaním sú:

Chromozóm A	0111000101011001
Chromozóm B	1001010011011100

- **permutačné kódovanie** - môže byť použité pri problémoch usporiadania, ako je napríklad problém obchodného cestujúcého. Každý chromozóm je reťazec čísel alebo písmen, ktorý predstavuje číslo alebo písmeno v poradí. Jednotlivé čísla alebo písmená sa nachádzajú v chromozóme iba raz. Permutačné kódovanie je užitočné iba pre problémy usporiadania. Príklady chromozómov sú:

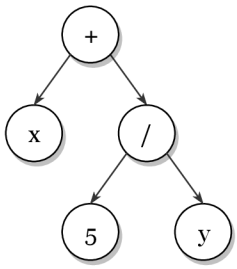
Chromozóm A	1 5 3 8 6 2 7 4 9
Chromozóm B	E A B G C D F

- **kódovanie hodnotou** - sa používa pri problémoch, v ktorých sa využíva nejaká zložitá hodnota, napríklad reálne čísla. Použitie binárného kódovania by bolo pre tento typ problémov veľmi zložité. V kódovaní hodnotou je každý chromozóm reťazec nejakých hodnôt. Hodnoty môžu byť čokoľvek spojené s problémom, reálne čísla alebo znaky. Toto kódovanie je veľmi dobré pre niektoré špeciálne problémy, avšak je často potrebné vyvinúť nejaký nový spôsob kríženia a mutácie špecifický pre daný problém. Príklady chromozómov s týmto kódovaním sú:

Chromozóm A	0.221 1.789 0.581 2.434 1.598 3.415
Chromozóm B	ABDEJGAIGJDFLDJEOFJVKDD
Chromozóm C	(hore), (vľavo), (dole), (vľavo), (hore)

- **stromové kódovanie** - sa využíva hlavne na genetické programovanie, vývoj programov alebo výrazov. Každý chromozóm je strom niektorých objektov, napríklad funkcií alebo príkazov v programovacom jazyku. Kríženie a mutácia môžu byť vykonané pomerne ľahko. Príklad chromozómu:

Chromozóm A



[1] http://www.obitko.com/tutorials/genetic-algorithms/encoding.php

Populácia

Populácia je súbor jedincov toho istého druhu, nachádzajúcich sa na jednom určitom mieste v danom čase. V genetickom algoritme populáciu tvorí množina jedincov, kde každý jedinec predstavuje riešenie zadaného problému, či už vhodné alebo nevhodné. Aby sa populácia mohla vyvíjať musí mať nasledujúce vlastnosti:

- Členovia musia byť schopný produkovať potomkov (vlastná reprodukcia).
- Potomstvo musí prenášať vlastnosti rodičov, s určitými zmenami (obmena).
- Potomstvo, ktoré sa prispôsobilo životnému prostrediu, musí prežiť s vyššou pravdepodobnosťou (prežitie najschopnejších).

Tieto vlastnosti nevyhnutne neznamenajú, že sa vývoj objaví, ale bez nich by bolo obtiažne, dokonca nemožné, aby sa populácia vyvinula a tým dospela k hľadanému riešeniu [7].

Generácia

Všetci jedinci, s ktorými sa pracuje v tom istom čase. Krížením jedincov jednej generácie (rodičia) vzniknú jedinci novej generácie (potomkovia). Nová generácia znova umožňuje kríženie jedincov, čím vznikne ďalšia generácia. Jedna populácia jedincov sa obvykle skladá z viacerých generácií. Pre každú novú generáciu platí:

- Počet chromozómov je rovnaký.
- Krížením a mutáciou vznikajú nové chromozómy.
- Chromozómy s nízkym ohodnotením sú nahradení chromozómami s vyšším ohodnotením. Prežijú tí najlepší.

Fitness

Fitness, tiež biologická zdatnosť, vyjadruje cenu jedinca z hľadiska evolúcie. Vyjadruje schopnosť konkrétného jedinca prispôsobiť sa prostrediu v ktorom sa nachádza, prežiť v ňom a predať svoje gény ďalším generáciam. Čím vyššia hodnota fitness, tým lepší jedinec je. Táto hodnota je stanovená z fenotypu v reakcii na životné prostredie. Každá fitness funkcia by mala spĺňať nasledujúce požiadavky [11]:

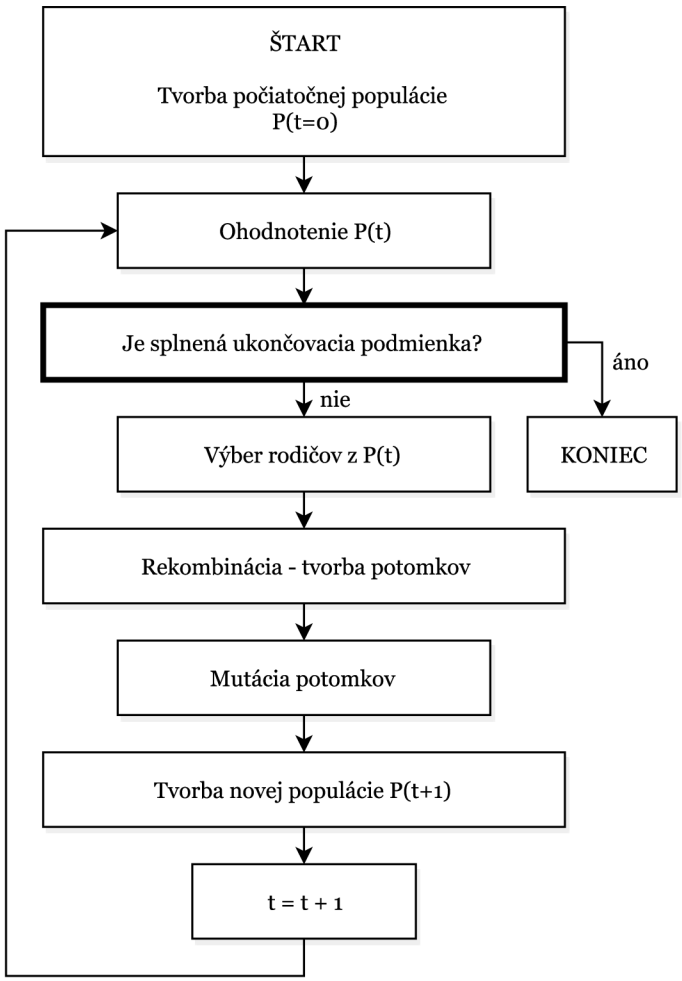
- Fitness funkcia by mala byť jasne definovaná. Užívateľ by mal byť schopný jasne pochopiť, ako sa vypočíta hodnota fitness.
- Fitness funkcia by sa mala vykonávať efektívne. Ak sa funkcia fitness stáva prekážkou algoritmu, celková účinnosť genetického algoritmu sa zníži.
- Fitness funkcia by mala kvantitatívne hodnotiť vhodnosť daného riešenia pri riešení zadaného problému.
- Fitness funkcia by mala generovať intuitívne výsledky. Najlepší/najhorší kandidáti by mali mať najlepšie/najhoršie ohodnotenie.

2.2 Priebeh genetického algoritmu

Genetický algoritmus sa skladá z viacerých krokov, kde každý krok plní úlohu, ktorá simuluje určitý evolučný proces z prírody. Týmito krokmi sú:

- tvorba počiatočnej populácie,
- ohodnotenie jedincov a populácie,
- výber rodičov/reprodukcia,
- kríženie/rekombinácia,
- mutácia,
- tvorba novej populácie.

Schému priebehu genetického algoritmu môžeme vidieť na obrázku 2.1. Opakovaním týchto krokov dostatočne dlho môžeme nájsť hľadané riešenie alebo najlepšie riešenie, ktoré sa hľadanému najviac približuje.



Obr. 2.1: Obecná schéma genetického algoritmu

2.2.1 Tvorba počiatočnej populácie

Počiatočná populácia je tvorená dopredu určeným počtom jedincov a jedinci sú obvykle generovaní náhodne. V prípade, že chceme generovať jedincov blízkych optimálnému riešeniu, môžeme využiť heuristiku, ktorá to umožní. Náhodne generovaní jedinci nie sú vhodní pri práci s obmedzeniami, kedy jedinci nesmú byť náhodní ale prípustní. Preto je nutné vytvoriť vlastný spôsob tvorby počiatočnej populácie pre konkrétny problém.

Počet jedincov v populácii závisí na riešenom probléme, typicky ale populácia obsahuje stovky až tisícky jedincov, pretože z biológie je známe, že malé populácie pravdepodobne zaniknú kvôli nedostatku dostatočnej genetickej rozmanitosti [2]. Všetci jedinci majú rovnakú, predom určenú, dĺžku.

2.2.2 Ohodnotenie

Hodnotíme jedincov a populáciu. Ohodnotenie jedinca spočíva v porovnaní, ako veľmi sa riešenie nájdené týmto jedincom líši od optimálneho, správného riešenia daného problému. Funkcia, ktorej hodnota udáva kvalitu riešenia sa volá fitness funkcia. Jej hodnota sa musí zvyšovať s kvalitou riešenia.

Ohodnotenie populácie udáva priemerná hodnota z hodnôt fitness funkcií všetkých jedincov populácie. V prípade, ak sa fitness funkcia v priebehu niekoľkých generácií nezmenila, ukončíme riešenie.

Riešenie taktiež ukončíme, ak bol dosiahnutý dopredu daný počet iterácií, ktorý určoval, koľko krát budeme vytvárať novú populáciu. Počet iterácií teda udáva počet generácií, ktoré algoritmus vytvorí v priebehu riešenia. V optimálnom prípade algoritmus skončí nájdením hľadaného riešenia, čiže jedinca, ktorý predstavuje toto riešenie.

Ohodnotenie jedincov slúži aj pri výbere rodičov. Zatiaľ čo v prírode jedinci medzi sebou súťažia v prirodzenom prostredí a rodičmi sa stávajú tí najlepší z nich, v genetických algoritmoch, čiže v umelom prostredí, musí byť táto súťaž napodobnená inými metódami. Na to slúži fitness, ktorá pri výbere rodičov nahradzuje prirodzený výber. Jedinci s vyššou hodnotou fitness bývajú pri výbere uprednostňovaní.

2.2.3 Reprodukcia

Výber rodičov (Selekcia)

Výber rodičov, selekcia, spočíva vo výbere dvojice jedincov z populácie, ktorí vzájomným krížením vytvoria nového jedinca, potomka. Môže sa jednať o jedného alebo viacerých potomkov. Aby sa kvalita populácie s každou novou generáciou zlepšovala a doviedla nás až k hľadanému riešeniu, výber rodičov musí simulovať prirodzený výber jedincov z prírody, teda výber tých najlepších.

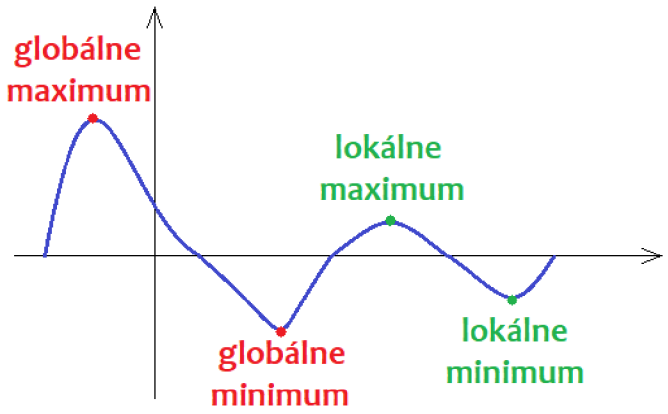
Prirodzený výber dosiahneme výberom najzdatnejších jedincov, k čomu napomáha selektívny tlak. Selektívny tlak potlačuje podpriemerných jedincov a naopak zvýhodňuje jedincov nadpriemerných. K hodnote fitness funkcie u nadpriemerných jedincov pripočíta kladnú hodnotu a naopak slabým jedincom z hodnoty fitness uberie. Môžeme si to vysvetliť na príklade, kde máme troch jedincov a ich fitness hodnoty sú `0.1`, `0.4` a `0.5`. Použitím selektívneho tlaku (napríklad s hodnotou 2) sa fitness hodnoty prepočítajú na `0`, `0.33` a `0.66`. Ako môžeme vidieť, selektívny tlak potlačil slabšieho jedinca (`0.1`→`0`) a zvýhodnil najlepšieho (`0.5`→`0.66`).

Pri správne nastavenom selektívnom tlaku algoritmus veľmi rýchlo konverguje k správnemu riešeniu. Nízky selektívny tlak môže spôsobiť, že sa kvalita populácie nebude dosta-

točne zlepšovať, naopak by ale vysoký selektívny tlak mohol spôsobiť, že budú vybraní iba jedinci, ktorých riešenie bude viesť do lokálneho optima. Je preto dôležité, zvoliť správnu hodnotu selektívneho tlaku, aby algoritmus dostatočne rýchlo konvergoval k hľadanému riešeniu, ale aby neskĺzol do lokálneho optima potlačením slabších jedincov, ktorí však môžu predať potomkom správne gény na nájdenie globálneho optima.

Globálne optimum je bod, ktorého fitness hodnota presahuje hodnotu akéhokoľvek iného hodnotenia (alebo je prekročená každou inou hodnotou, ak minimalizujeme) [2].

Lokálne optimum je bod, ktorý má vlastnosť, že v jeho okolí sú iba jedinci s nižším ohodnotením, ale nie je globálnym optimom. Rozdiel môžeme vidieť na obrázku 2.2.



Obr. 2.2: Globálne a lokálne optimum

Výber rodičov, selekciu, môžeme rozdeliť na dva kroky. Po prvé, prepočítame hodnoty fitness funkcie na pravdepodobnosti výberu jedincov ako rodičov, buď podľa proporcií fitness alebo podľa poradia. Po druhé, nasleduje skutočný výber rodičov pomocou niektorého z nasledujúcich algoritmov (ruleta, stochastické univerzálne vzorkovanie, výber elity, turnaj).

Prepočítavanie podľa proporcií

Prepočítavanie podľa proporcií je genetický operátor slúžiaci na výber jedincov, ktorí sú potencionálne užitoční na kríženie. Fitness funkcia priradí všetkým jedincom ich vhodnosť. Úroveň vhodnosti sa používa na spájanie pravdepodobnosti výberu s každým jednotlivým jedincom. Najznámejším predstaviteľom prepočítavania podľa proporcií je algoritmus Rulety.

Prepočítavanie podľa proporcií môžeme vyjadriť ako vzťah

$$P_{prop}(i) = \frac{f(i)}{\sum_{i=1}^{n} f(i)}, \text{ kde:}$$

i - index jedinca
$f(i)$ - hodnota fitness funkcie i-tého jedinca
n - počet jedincov v populácii

Prepočítavanie podľa poradia

Populácia je zoradená podľa objektívnych hodnôt. Fitness priradená každému jedincovi závisí len na jeho pozícií v kategórií jednotlivcov a nie od skutočnej objektívnej hodnoty. Zatiaľ čo medzi veľkosťou ohodnotenia jednotlivých jedincov môžu byť veľké rozdiely, použitím poradového čísla dôjde k zrovnomerneniu pravdepodobnosti výberu. Poradie zavádza jednotnú mierku v celej populácií a poskytuje jednoduchý a účinný spôsob kontroly selektívneho tlaku.

Prepočítavanie podľa poradia, ak sa obmedzíme iba na lineárne hodnoty, je dané vzťahmi

$$f_{lr}(i) = \frac{2 - SP + 2 * (SP - 1) * (i - 1)}{(n - 1)},$$

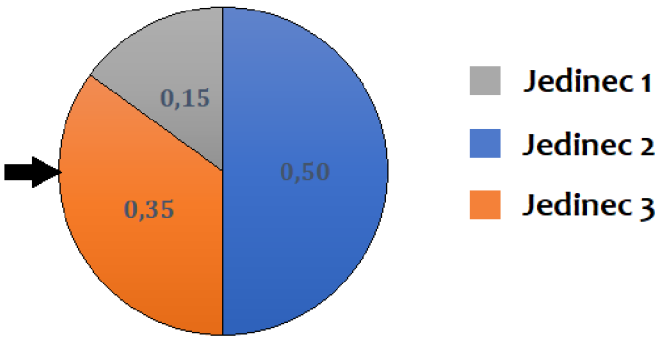
$$P_{lr}(i) = \frac{f_{lr}(i)}{\sum_{i=1}^{n} f_{lr}(i)}, \text{ kde:}$$

i - index jedinca (podľa pôvodnej hodnoty fitness funkcie)
$f(i)$ - hodnota prepočítanej fitness funkcie
SP - selektívny tlak
n - počet jedincov v populácii

Ruleta

Ruletu si môžeme predstaviť podobne ako ruletu v kasíne. Jedno koleso, ktoré je po obvode rovnomerne rozdelené na rovnaké časti, kde každá časť reprezentuje jedno číslo. V genetických algoritmoch sú však výseky kolesa rulety úmerné prepočítaným hodnotám fitness funkcií pre jednotlivých jedincov. Jedinci s vyššou hodnotou fitness funkcie, budú na rulete zaberať väčšiu časť, čím stúpne pravdepodobnosť ich výberu. Príklad rulety s tromi jedincami môžeme vidieť na obrázku 2.3.

Výber rodičov spočíva v roztočení rulety. Každým točením vyberáme jedného rodiča. Môžeme si predstaviť, že na jedno miesto na rulete ukazuje šípka. Po roztočení a následnom zastavení rulety ukazuje šípka na vybraného jedinca. Aby sme získali potrebný počet rodičov, musíme koleso rulety roztočiť toľko krát, koľko rodičov chceme.

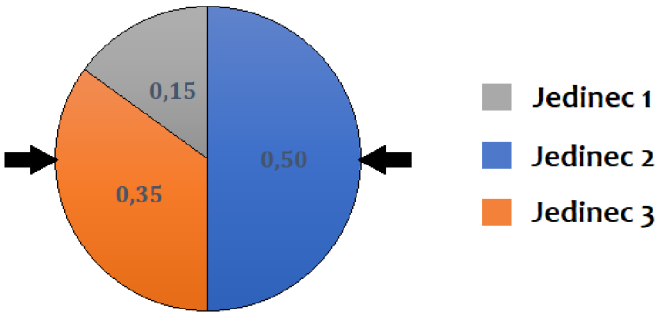


Obr. 2.3: Ruleta

Stochastické univerzálne vzorkovanie

Podobne ako v predchádzajúcom algoritme, aj tu slúži na výber rodičov ruleta. Výseky kolesa sú znova úmerné prepočítaným hodnotám fitness funkcií pre jednotlivých jedincov.

Výber rodičov je ale odlišný od predchádzajúceho. Na koleso rulety ukazuje toľko šípiek, koľko potrebujeme rodičov. Šípky sú rovnomerne rozmiestnené okolo rulety a po dotočení rulety, ukazujú šípky na vybraných rodičov, čo môžeme vidieť na obrázku 2.4. To znamená, že sa na jedno otočenie vyberú všetci potrební rodičia, narozdiel od predchádzajúceho algoritmu, kde bol počet rodičov rovný počtu točení.



Obr. 2.4: Stochastické univerzálne vzorkovanie

Výber elity

Jedinci sú zoradení vzostupne podľa hodnoty ich fitness. Vznikne poradie, ktoré sa využíva pri výbere rodičov. Rodičmi sa stanú jedinci, ktorý sa nachádzajú na vrchole tohto poradia, obvykle 10% až 50% z nich.

Výber elity potlačuje vplyv nadpriemerných jedincov. Zatiaľ čo v selekcii, ktorá využíva proporcionálne rozdelenie jedincov, by nadpriemerný jedinec zabral najväčšiu časť rulety a pravdepodobnosť jeho výberu by bola taká veľká, že by ostatní jedinci nemali šancu, v selekcii podľa poradia by nadpriemerný jedinec zabral prvú priečku a aj slabší jedinci by dostali šancu, pretože by boli v poradí hneď za ním.

Výber podľa poradia taktiež napomáha udržiavať selektívny tlak. Hlavne ku koncu výpočtu, keď sú zmeny medzi jedincami naozaj nepatrné pretože v populácií prevládajú zdatní jedinci.

Turnaj

Táto metóda je inšpirovaná prírodou, kedy jedinci musia súperiť o právo byť rodičmi. Populácia je rozdelená na niekoľko skupín, kde sú jedinci vyberaní náhodne. Počet skupín sa odvíja od celkového počtu jedincov v populácií a od veľkosti turnaja. Po rozdelení je z každej skupiny vybraný jedinec s najlepším ohodnotením, najvyššou hodnotou fitness, ktorý sa stane rodičom.

Turnaj zachováva rozmanitosť populácie, pretože do skupiny môžu byť náhodne vybraní aj horší jedinci a jeden z nich sa musí stať rodičom. Turnajový výber môžeme použiť v prípade, keď zadané riešenie rýchlo konverguje k lokálnemu optimu.

2.2.4 Rekombinácia

Rekombinácia, teda kríženie, značí produkovanie nových jedincov z informácií, ktoré sa nachádzajú v génoch rodičov. Spočíva vo vzájomnej výmene určitých častí chromozómov. Máme viacero druhov krížení:

- **jednobodové kríženie** - najjednoduchší spôsob kríženia. Náhodne je vybrané jedno miesto kríženia a potomok vznikne tak, že všetko pred týmto miestom je prekopírované z prvého rodiča a všetko za týmto miestom je prekopírované z druhého rodiča.

1.rodič:	a b c d e	f g h
2.rodič:	a b c d e	f g h
1.potomok:	a b c d e	f g h
2.potomok:	a b c d e	f g h

- **viacbodové kríženie** - obsahuje viacero miest kríženia, minimálne aspoň dve. Potomok vznikne tak, že všetko od začiatku po prvý bod kríženia je prekopírované od prvého rodiča, všetko od prvého miesta kríženia po druhé miesto kríženia je prekopírované od druhého rodiča, všetko od druhého miesta kríženia až po koniec alebo ďalšie miesto kríženia je znovu prekopírované od prvého rodiča, atď. Maximálny počet miest krížení pre n génov je $n - 1$.

1.rodič:	a b	c d e	f g h
2.rodič:	a b	c d e	f g h
1.potomok:	a b	c d e	f g h
2.potomok:	a b	c d e	f g h

- **uniformné kríženie** - každý binárny gén sa s určitou pravdepodobnosťou prenesie buď od prvého alebo druhého rodiča. Táto pravdepodobnosť je pevne stanovená a väčšinou sa jej hodnota rovná 0.5. To znamená, že je 50% pravdepodobnosť, že sa gén prenesie od prvého rodiča a 50% pravdepodobnosť, že sa prenesie od druhého.

1.rodič:	a b c d e f g h
2.rodič:	a b c d e f g h
1.výber:	1 2 1 1 2 1 1 2
2.výber:	2 2 1 2 1 2 1 1
1.potomok:	a b c d e f g h
2.potomok:	a b c d e f g h

- **aritmetické kríženie** - používa sa pri génoch s reálnymi hodnotami. Vyberá hodnoty jednotlivých génov náhodne z oboch rodičov s uniformným pravdepodobnostným rozložením koeficientu a_i:

$$g_i^{O_1} = g_i^{P_1} a_i + g_i^{P_2}(1 - a_i) \quad a_i \in [-0.25, 1.25]$$
$$g_i^{O_2} = g_i^{P_1}(1 - a_i) + g_i^{P_2} a_i$$

  g_i - hodnota génu i

  P_j - rodič (parent), $j = 1, 2$

  O_k - potomok (offspring), $k = 1, 2$

- **heuristické kríženie** - používa sa pri génoch s reálnymi hodnotami. Vyberá hodnoty jednotlivých génov z rodičov podľa náhodne zvoleného koeficientu $r \in < 0, 1 >$:

$$O_1 = P_{better} + r * (P_{better} - P_{worse})$$
$$O_2 = P_{better}$$

  P_j - rodič (parent), $j = better, worse (f_{better} > f_{worse})$

  O_k - potomok (offspring), $k = 1, 2$

- **kríženie v permutačných problémoch** - používa sa pri problémoch, kde nemôžeme jednoducho zobrať jednu časť génov od prvého rodiča a druhú časť od druhého rodiča, pretože by riešenie dané týmto jedincom nedávalo zmysel. Napríklad, pri probléme obchodného cestujúceho, by sa mohlo stať, že niektoré mestá nenavštívi ani raz a niektoré dokonca dvakrát. Tieto problémy sa preto rieša odlišne, napríklad použitím indexových tabuliek alebo PMX.

Kríženie aplikujeme na všetkých vybraných rodičov. V priebehu genetického algoritmu kríženie nemeníme, to znamená, že každá generácia sa kríži rovnakým spôsobom.

2.2.5 Mutácia

Mutácia je zmena genetického materiálu. V genetických algoritmoch ju chápeme napríklad, ako zmenu hodnoty náhodne vybraného génu u náhodne vybraného potomka. Mutácia má predom danú malú pravdepodobnosť. Väčšinou sa pohybuje v rozmedzí 0.001 - 0.05, pričom ju môžeme vyrátať ako $P = \frac{1}{n}$, kde n je počet jedincov v populácii.

V prípade, že sa jedinec skladá z binárnych hodnôt 0 a 1, mutácia znamená invertovanie hodnoty jedného bitu. Bit s hodnotou 0 sa invertuje na 1 a naopak bit s hodnotou 1 sa invertuje na 0. Pri jedincovi, ktorý obsahuje reálne hodnoty, mutácia znamená, že sa k hodnote génu pripočíta malá, náhodná reálna hodnota. Jej veľkosť je maximálne 10% a môže byť aj záporná.

2.2.6 Tvorba novej populácie

Nová populácia nahradí predchádzajúcu populáciu. Podľa počtu pôvodných jedincov, ktorí sú v novej generácií nahradení, hovoríme o troch typoch modelov:

- Inkrementačný model - v novej populácii je nahradený potomkom jediný jedinec pôvodnej populácie.
- Generačný model - všetci jedinci pôvodnej populácie sú v novej populácii nahradení potomkami.
- Modely s prekrytím generácií - v novej populácii je nahradená potomkami časť jedincov pôvodnej populácie. Inkrementačný model a generačný model sú extrémnými prípadmi tohoto modelu.

Kapitola 3

Riešené problémy

Táto kapitola popisuje jednotlivé problémy, ktoré boli použité pri demonštrácií genetického algorimu v programe `GA`. Jedná sa o problém obchodného cestujúceho, splniteľnosť logických formúl a hľadanie maxima/minima funkcie.

3.1 Problém obchodného cestujúceho

Problém obchodného cestujúceho, angl. traveling salesman problem - TSP, je obtiažny diskrétny optimalizačný problém. Matematicky vyjadruje a zobecňuje úlohu nájdenia najkratšej možnej cesty medzi všetkými zadanými bodmi.

Definícia - *Nech existuje n miest, medzi nimi cesty o známych dĺžkach. Úlohou je nájsť najkratšiu možnú trasu, ktorá bude prechádzať všetkými mestami a vráti sa naspäť do východzieho mesta.*

Problém obchodného cestujúceho je klasifikovaný ako NP-úplny, čiže so vzrastajúcim počtom miest rastie exponenciálne čas riešenia. Preto problém nespočíva v nájdení najkratšej cesty, ale v nájdení časovo efektívného algoritmu na nájdenie najkratšej cesty. Keďže sa jedná o NP-úplny problém, nepredpokladá sa nájdenie deterministického algoritmu, ktorý nájde optimálne riešenie v polynomiálne obmedzenom čase. Zatiaľ takýto algoritmus nie je známy, vziklo však veľké množstvo aproximačných algoritmov. Riešenie pomocou týchto algoritmov je síce horšie, ale nájdeme ho v prijateľnom čase.

TSP má využitie pri plánovaní, v logistike či výrobe mikročipov.

3.1.1 Problém obchodného cestujúceho v teórií grafov

V teórii grafov jednotlivé mestá predstavujú vrcholy. Prepojenia medzi týmito mestami sú hrany. Ide o hranovo ohodnotený súvislý graf $G = (V, H)$, v ktorom hľadáme hamiltonovskú kružnicu, ktorá má čo najmenší súčet ohodnotení hrán.

Hamiltonovská kružnica je taký podgraf, ktorý je kružnica a obsahuje všetky vrcholy pôvodného grafu.

Úlohou obchodného cestujúceho je teda nájsť čo najkratšiu hamiltonovskú kružnicu.

3.1.2 Riešenie TSP pomocou genetických algoritmov

Pri riešení problémov pomocou genetických algoritmov je dôležité zvoliť správnu reprezentáciu jedincov. Problém obchodného cestujúceho sa radí k problémom usporiadania, ktoré používajú permutačné kódovanie, čo znamená, že jedinec obsahuje reťazec čísel alebo

písmen, kde každé číslo/písmeno značí poradie. Tento reťazec nazývame chromozóm. Napríklad jedinec s chromozóm 5-2-1-4-3 predstavuje riešenie problému, kde obchodný cestujúci prechádza mestami v nasledujúcom poradí: vychádza z mesta číslo päť, pokračuje cez mesto číslo dva, potom cez mesto číslo jedna, pokračuje cez mesto číslo štyri do mesta číslo tri, odkiaľ prichádza do mesta číslo päť, z ktorého vychádzal.

Tvorba počiatočnej populácie

Pri probléme obchodného cestujúceho je počiatočná populácia vytvorená dopredu daným počtom jedincov. Títo jedinci sú generovaní náhodne. Ich dĺžka je rovná počtu miest, ktorými musí obchodný cestujúci prejsť. Dĺžka preto nemôže byť premenlivý parameter algoritmu. Jednotlivé gény obsahujú náhodne čísla od $1..n$, kde n je počet miest. Každé číslo sa musí v jedincovi nachádzať práve raz.

Ohodnotenie

Fitness funkcia konkrétneho jedinca je vypočítaná ako vzdialenosť, ktorú obchodný cestujúci prejde počas cesty, ktorá je reprezentovaná týmto jedincom. Čím kratšia je vzdilenosť, ktorú musí cestujúci prejsť, tým lepšie je riešenie reprezentované týmto jedincom a tým je teda vyššie aj jeho ohodnotenie.

Selekcia

Selekcia pri probléme obchodného cestujúceho prebieha rovnako ako pri základnom genetickom algoritme. Môžeme využiť rovnaké typy selekcie, napríklad ruletu, stochastické univerzálne vzorkovanie, výber elity alebo turnaj.

Rekombinácia

Pri rekombinácii, teda krížení, nie je možné použiť všetky typy kríženia. Kríženia, ktoré vezmú jednu časť génov (napríklad prvé tri gény) z jedného rodiča, druhú časť (napríklad zvyšné tri gény) z druhého rodiča nie sú vhodné. K týmto kríženiam patrí jednobodové, viacbodové, uniformné, aritmetické a heuristické kríženie. Nie je možné ich použiť, pretože riešenia, ktoré vzniknú týmito kríženiami nemusia dávať zmysel. Nemôžeme pomocou viacbodového kríženia pri probléme obchodného cestujúceho krížiť jedinca s chromozóm 5-2-1-4-9-6-7-3-8 a jedinca s chromozómom 1-6-3-5-2-7-8-9-5. Pretože pre akékoľvek miesta kríženia, by sme dostali nevalidné riešenie.

1.rodič:	5 2 1	4 9 6	7 3 8
2.rodič:	1 6 3	5 2 7	8 9 4
1.potomok:	5 2 1	5 2 7	7 3 8
2.potomok:	1 6 3	4 9 6	8 9 4

Môžeme vidieť, že obaja vzniknutí jedinci nie sú validní, pretože ich trasa neprechádza všetkými mestami. Niektorými neprechádza vôbec, niektorými prechádza viac krát.

Pri probléme obchodného cestujúceho sa preto na rekombináciu používa operátor kríženia s čiastočným zobrazením, angl. partially-mapped crossover - PMX. Tento operátor zachováva úseky miest a čiastočne aj ich poradie, čo je pri riešení TSP výhodou. Funguje tak, že sa časť jedného reťazca namapuje na časť druhého reťazca a zostávajúca informácia sa vymieňa.

Pre lepšie pochopenie kríženia s čiastočným zobrazením poslúži nasledujúci príklad s podrobne popísaným postupom [12].

1. V jedincoch, ktorí sa stanú rodičmi, PMX náhodne vyberie dve miesta kríženia.

1.rodič:	5 2 1	4 9 6	7 3 8
2.rodič:	1 6 3	5 2 7	8 9 4

2. Potomkovia budú obsahovať podreťazec, ktorý PMX zamení medzi rodičmi.

1.potomok:	x x x	5 2 7	x x x
2.potomok:	x x x	4 9 6	x x x

3. PMX určí vzťah mapovania. Vzťah mapovania je nasledujúci: $4 \leftrightarrow 5, 9 \leftrightarrow 2, 6 \leftrightarrow 7$. To znamená, že sa mesto s číslo štyri namapovalo na mesto s číslom päť, mesto s číslom deväť na mesto s číslom dva a mesto s číslom šesť na mesto s číslom sedem. Zvyšné mestá, označené v potomkoch ako x, sú vyplnené podľa pôvodného rodiča. Prvý potomok je vyplnený prvým rodičom, druhý potomok je vyplnený druhým rodičom. Ak sa však mesto v potomkovi už nachádza, je nahradené podľa vzťahu mapovania.

1.potomok:	4 9 1	5 2 7	6 3 8
2.potomok:	1 7 3	4 9 6	8 2 5

Môžeme vidieť, že obaja potomkovia obsahujú všetky mestá práve raz, čiže pri PMX nenastáva problém ako pri iných kríženiach, a preto je vhodné pre problém obchodného cestujúceho.

Mutácia

Existuje veľa druhov mutácie, niektoré sú známe a používané, iné nie [13].

Mutácia vloženia - náhodne vyberte dva gény. Druhý gén potom vložte za prvý gén a zvyšok chromozómu odsuňte.

Pred mutáciou: 4 9 1 5 2 7 6 3 8 → Po mutácii: 4 9 1 3 5 2 7 6 8

Inverzná mutácia - náhodne vyberte dva gény. Invertujte reťazec medzi nimi.

Pred mutáciou: 4 9 1 5 2 7 6 3 8 → Po mutácii: 4 9 1 6 7 2 5 3 8

Mutácia zamiešania - vyberie sa podreťazec a gény sa v ňom náhodne usporiadajú, premiešajú.

Pred mutáciou: 4 9 1 5 2 7 6 3 8 → Po mutácii: 4 9 3 7 2 5 1 6 8

Mutácia výmeny - vyberú sa náhodne dva gény a vymení sa ich pozícia.

Pred mutáciou: 4 9 1 5 2 7 6 3 8 → Po mutácii: 4 9 3 5 2 7 6 1 8

Tvorba novej populácie

Tvorba novej populácie sa, podobne ako selekcia, nelíši od tvorby populácie vo všeobecnom probléme riešenom genetickým algoritmom.

3.2 Splniteľnosť logických formúl

Pri logických formulách riešime problém, ktorý sa nazýva splniteľnosť, angl. satisfiability (SAT). Splniteľnosť v logike predstavuje odpoveď na otázku, či je zadaná formula pravdivá, teda splniteľná. Formula je zapísaná pomocou boolovských operácií, ktorými sú `AND` ($\wedge$), `OR` ($\vee$) a `NOT` ($\neg$) [10].

SAT je NP-úplny problém, čo znamená, že nepoznáme žiadny efektívny algoritmus, ktorý rieši všetky SAT problémy v polynomiálnom čase.

V teórií zložitosti predstavuje otázka splniteľnosti rozhodnutie, či pre všetky premenné v zadanej formule existujú hodnoty `TRUE` alebo `FALSE` v takej kombinácií, aby celá formula nadobudla hodnotu `TRUE`. Ak takáto kombinácia existuje, formulu označíme za splniteľnú, ak neexistuje, formula je nesplniteľná. Nesplniteľná formula môže byť napríklad

$$a \wedge \neg a,$$

kedy by premenná a musela súčasne nadobúdať hodnoty `TRUE` aj `FALSE`, aby bola formula splniteľná. Problém, či je zadaná formula splniteľná alebo nie, predstavuje kľúčovú úlohu v mnohých oblastiach informatiky (teoretická informatika, algoritmizácia, umelá inteligencia, a pod.).

Pri riešení problému splniteľnosti logických formúl sa stretneme s pojmami literál a klauzula. Literál je premenná alebo negácia premennej. Napríklad a je pozitívny literál a $\neg b$ predstavuje negatívny literál. Klauzula je disjunkcia literálov a môže vyzerať napríklad ako $a \vee \neg b$. Formula sa teda skladá z klauzúl a literálov. Napríklad formula

$$F = (a \vee \neg b) \wedge (\neg a \vee c \vee d) \wedge (b \vee d)$$

je formula so štyrmi premennými a tromi klauzulami. Táto formula sa taktiež nachádza v konjuktívnej normálovej forme (CNF), pretože všetky klauzule sú spolu v konjunkcii (`AND`, $\wedge$) a každá jedna klauzula je disjunkcia (`OR` , $\vee$) literálov. Plus platí, že je každý literál premenná alebo jej negácia (`NOT`, $\neg$).

3.2.1 Riešenie SAT pomocou genetických algoritmov

Ako bolo uvedené už pri probléme obchodného cestujúceho, dôležité pri riešení problémov pomocou genetických algoritmov je zvoliť správnu reprezentáciu jedincov. Problém splniteľnosti nevyžadoval žiadnu špeciálnu úpravu reprezentácie chromozómu. Chromozóm je jednoducho reprezentovaný bitmi 1 alebo 0 a jeho veľkosť závisí od počtu premenných vo formuly. To znamená, že chromzóm s bitmy 1011 predstavuje v poradí premenné a, b, c a d s hodnotami `TRUE`, `FALSE`, `TRUE` a `TRUE`. Tento jedinec by bol riešením napríklad formule

$$(a \vee \neg b \vee \neg c) \wedge d,$$

pretože celá formula vďaka hodnotám premenných, ktoré reprezentuje tento jedinec, nadobúda hodnotu `TRUE`. Jedna formula ale nemusí mať iba jedno správne riešenie. Jedincov, ktorí reprezentujú správne riešenie môže byť viac. Pre túto formulu to môžu byť okrem spomínaného jedinca aj jedinci 1101, 1001, 0001, 0101 alebo 0011, pretože všetci obsahujú takú kombináciu premenných, pri ktorých formula nadobúda hodnotu `TRUE`.

Jedinec taktiež nemusí byť riešením iba jednej formule. Jedinec 1011 je okrem vyššie spomenutej formule riešením aj pre formulu $a \wedge (\neg b \vee c) \wedge d$, poprípade iným.

Tvorba počiatočnej populácie

Pri probléme splniteľnosti logických formúl je počiatočná populácia vytvorená dopredu daným počtom jedincov. Títo jedinci sú generovaní náhodne. Ich dĺžka je rovná počtu premenných, ktoré sa nachádzajú vo formuly. Dĺžka preto nemôže byť premenlivý parameter algoritmu. Jednotlivé gény obsahujú náhodne čísla 1 alebo 0, kde 1 reprezentuje hodnotu `TRUE` a 0 hodnotu `FALSE`.

Ohodnotenie

Fitness funkcia konkrétneho jedinca je vypočítaná ako počet klauzúl, ktoré tento jedinec vyrieši s tým, že klauzula nadobudne hodnotu `TRUE`. V prípade, že všetky klauzule nadobudnú hodnotu `TRUE`, znamená to, že jedinec je riešením zadanej formule. Čím lepšie je riešenie reprezentované týmto jedincom, tým je vyššie jeho ohodnotenie. Najvyššia možná hodnota fitness je rovná počtu klauzúl vo formule. Pre formulu $(a \vee b) \wedge (\neg b \vee c) \wedge (d \vee \neg d)$ by bola najvyššia možná hodnota fitness funkcie tri, pretože sa daná formula skladá z troch klauzúl a jedinec musí vyriešiť všetky tri klauzule aby bol riešením celej tejto formule.

Selekcia

Selekcia pri SAT prebieha rovnako ako pri základnom genetickom algoritme. Neboli potrebné žiadne špeciálne úpravy, čiže môžeme využiť rovnaké typy selekcie, ktorými sú napríklad ruletu, výber elity alebo turnaj.

Rekombinácia

Pri rekombinácii, teda krížení, je možné opäť použiť všetky typy kríženia, ktoré môžeme použiť aj v obecnom genetickom algoritme. Keďže je jedinec reprezentovaný bitmi 1 a 0, nie je problém použiť jednobodové, viacbodové alebo uniformné kríženie, pretože jedinec, ktorý vznikne je aj naďalej validný.

Mutácia

Mutácia prebieha rovnako ako pri všeobecnom genetickom algoritme, neboli potrebné žiadne úpravy. Invertuje sa hodnota jedného génu, ktorý bol náhodne vybraný z jedinca, ktorý bol taktiež zvolený náhodne. Invertovaním génu je myslená zmena hodnoty z 0 na 1 alebo z 1 na 0.

Tvorba novej populácie

Tvorba novej populácie sa nelíší od tvorby novej populácie vo všeobecnom probléme riešenom genetickými algoritmami. Novú populáciu môžeme opäť vytvoriť tak, že zo starej populácie necháme všetkých jedincom okrem najhoršieho, ktorého nahradíme novým jedincom (inkrementačný model), alebo nahradíme celú populáciu (generačný model), alebo jednu časť jedincov zachováme a druhú časť jedincov nahradíme novými jedincami (model s prekrytím generácií).

3.3 Hľadanie minima alebo maxima funkcie

Pri funkciách hľadáme minimum alebo maximum danej funkcie. Ak máme zadanú napríklad funkciu $f = x^2 + y^2$ minimum alebo maximum funkcie zistíme tak, že dosádzame za premenné x a y rôzne hodnoty a snažíme sa nájsť tú najnižšiu, prípadne najvyššiu hodnotu. Pre túto konkrétnu funkciu je minimum v bode $[0, 0]$, pretože $0^2 + 0^2 = 0$ a pre všetky iné hodnoty, je vždy výsledok väčší ako 0. Pri maxime tejto funkcie, je to trochu komplikovanejšie, keďže vieme, že sa maximum blíži nekonečnu. Znamená to, že nikdy nenájdeme skutočné maximum tejto funkcie, ale môžeme sa k nemu čoraz viac približovať.

3.3.1 Riešenie funkcií pomocou genetických algoritmov

Pri hľadaní minima, či maxima funkcie je dôležité zvoliť správnu reprezentáciu jedincov. Keďže sa jedná o hodnoty s desatinnými miestami, musela byť zvolená iná ako binárna reprezentácia jedinca. Zadanému problému vyhovuje reprezentácia hodnotou, konkrétne desatinným číslom. Chromozóm je tvorený dvoma génmi, kde prvý reprezentuje x-ovú hodnotu a druhý y-ovú hodnotu. Chromozóm môže vyzerať napríklad ako dvojica čísel $3.16, -4.67$.

Tvorba počiatočnej populácie

Pri funkciách je počiatočná populácia vytvorená dopredu daným počtom jedincov. Títo jedinci sú generovaní náhodne a ich dĺžka je rovná počtu premenných, ktoré sa nachádzajú vo funkcii. Počet premenných však bol obmedzený na dva, pretože sa môžu používať iba funkcie s dvoma premennými. Dĺžka preto nemôže byť premenlivý parameter algoritmu. Jednotlivé gény obsahujú náhodne desatinné čísla v zadanom intervale.

Ohodnotenie

Fitness funkcia konkrétneho jedinca sa vypočíta tak, že sa vyhodnotí zadaná funkcia dosadením hodnôt za premenné x a y. Tieto hodnoty sú reprezentované génmi jedinca. V prípade, že hľadáme maximum funkcie, fitness hodnota je výsledok funkcie. Čím je vyššia hodnota fitness funkcie, tým je lepší jedinec.

V opačnom prípade, ak hľadáme minimum funkcie, fitness funkcia je prevrátená hodnota výsledku, čiže $\frac{1}{(v+k)}$, kde v je výsledok a k je malá číselná konštanta, aby sa hodnota fitness ľahšie vyčíslovala pre v blížiace sa nule. Je to z dôvodu, že chceme nájsť čo najmenšiu hodnotu funkcie(minimum), ktorá musí byť ohodnotená čo najvyššou fitness funkciou, a preto musí byť fitness obrátená. Čím je menšia hodnota výsledku funkcie, tým je vyššia hodnota fitness funkcie a tým je aj lepší konkrétny jedinec.

Selekcia

Selekcia pri funkciách prebieha rovnako ako pri ostatných riešených problémoch. Môžeme využiť rovnaké typy selekcie, napríklad ruletu, výber elity alebo turnaj.

Rekombinácia

Pri rekombinácii, teda krížení, by bolo možné použiť všetky typy kríženia, ktoré využívame v obecnom genetickom algoritme, ale obvykle sa nepoužívajú. Pri riešení problémov s reálnymi hodnotami sú preto využívané dva typy kríženia, aritmetické a heuristické. Navyše môžeme

použiť kríženie, ktoré je podobné aritmetickému kríženiu a spočíva v priemere hodnôt oboch rodičov. Príklad tvorenie jedinca pomocou priemeru:

1.rodič:	4.51	-3.25
2.rodič:	1.23	0.47
Potomok:	2.87	-1.39

Mutácia

Mutácia neprebieha rovnako ako pri všeobecnom genetickom algoritme. Keďže sa jedná o desatinné hodnoty, nie je možné iba jednoducho invertovať gén. Preto mutácia prebieha tak, že sa k náhodne vybranému génu buď pripočíta alebo odpočíta hodnota. Táto hodnota je 10% z celkovej hodnoty, ktorá bola vybraná na mutáciu.

Tvorba novej populácie

Tvorba novej populácie sa nelíší od tvorby novej populácie v ostatných problémoch. Opäť môžeme použiť inkrementačný alebo generačný model, prípadne model s prekrytím generácie.

3.3.2 Riešenie funkcií pomocou algoritmu svetlušiek

Druhý spôsob, ktorým je riešené hľadanie minima a maxima funkcie je algoritmus svetlušiek. Svetluška je reprezentovaná svojou pozíciou v priestore, jasom a intenzitou svetla, ktoré vyžaruje. V prípade, že hľadáme maximum funkcie, svetluška, ktorá má najvyššiu intenzitu je najbližšie k hľadanému riešeniu. V opačnom prípade, ak hľadáme minimum funkcie, svetluška musí mať čo najnižšiu intenzitu aby bola najlepšia a najbližšie k riešeniu.

Počiatočná populácia svetlušiek je vygenerovaná náhodne, v závislosti na zadanom intervale a jej veľkosť je dopredu určená. Celý algoritmus prebieha spôsobom, kedy porovnávame postupne všetky svetlušky a vždy horšiu svetlušku posunieme smerom k lepšej. Svetlušky sa tak postupne stále viac a viac približujú k hľadanému riešeniu.

Kapitola 4

Inteligencia skupiny

Inteligencia skupiny, angl. swarm intelligence (SI), je technika umelej inteligencie založená na kolektívnom chovaní decentralizovaných, samoorganizovaných systémov. Týka sa teda kolektívneho správania viacerých interaktívnych agentov, ktorí sa riadia jednoduchými pravidlami. Aj keď neexistuje centralizovaná štruktúra riadenia, ktorá určuje ako sa majú správať jednotlivci, lokálne interakcie medzi týmito agentami vedú ku vzniku zložitého globálneho správania [6]. Mnohé algoritmy sú inšpirované živočíšnymi organizmami alebo biologickými systémami. Hlavné vlastnosti algoritmu založeného na inteligencii skupiny možno zhrnúť nasledovne:

- Zdieľanie informácií medzi viacerými agentami.
- Agenti majú samoorganizáciu a kolaboratívnu evolúciu.
- Je vysoko efektívny pre svoje kolaboratívne učenie.
- Môže byť ľahko paralelizovaný pre praktické a real-time problémy.

Algoritmy inšpirované inteligenciou skupiny sú vhodné na riešenie optimalizačných problémov, kde nie je vhodné použiť analytické metódy kvôli množstvu parametrov alebo zložitosti problému. Sú vhodné aj v prípadoch, kedy by výpočet klasickými metódami trval príliš dlho alebo, ak do detailov nepoznáme optimalizovanú úlohu. Dokážu nájsť hľadané riešenie, ktoré je globálne, avšak niekedy môžu taktiež skĺznuť do lokálnych extrémov.

Algoritmy založené na SI patria k širšej skupine algoritmov nazývaných biológiou inšpirované algoritmy, zatiaľ čo biológiou inšpirované algoritmy sú podmnožinou algoritmov inšpirovaných prírodou. Mnohé biológiou inšpirované algoritmy nepoužívajú priamo skupinové správanie. Je lepšie ich nazývať biológiou inšpirované algoritmy, ale nie inšpirované inteligenciou skupiny. K týmto algoritmom patria napríklad genetické algoritmy, ktoré sú inšpirované biológiou ale nie sú založené na inteligencii skupiny.

Algoritmy inšpirované inteligenciou skupiny majú na začiatku náhodné rozmiestnenie agentov po ploche. Tí sa následne začnú správať podľa presne definovaných pravidiel, čím sa postupne dopracujú k požadovanému riešeniu. Tieto pravidlá závisia od typu živočíšnych organizmov alebo biologických systémov, ktorými je algoritmus inšpirovaný. Môže to byť napríklad roj častíc, mravčia kolónia, včelí roj, roj svetlušiek a iné.

4.1 Optimalizácia mravčou kolóniou

Optimalizácia mravčou kolóniou, angl. ant colony optimization (ACO), je algoritmus založený na mravčej kolónii inšpirovaný chovaním mravcov. Konkrétne postupom, ktorý používajú mravce pri hľadaní potravy. Ich snahou je nájsť najkratšiu cestu medzi ich mraveniskom a zdrojom potravy. Mravce najprv náhodne prehľadávajú blízke okolie mraveniska. Každý mravec počas svojho pohybu vypúšťa na zem feromón, chemickú látku, na ktorú dokážu reagovať ostatné mravce. Mravec túto látku cíti a pravdepodobnosť výberu jeho ďalšej cesty je úmerná aktuálnej koncentrácii feromónov na začiatkoch možných ďalších ciest.

Chovanie mravcom muselo byť prispôsobené umelému prostrediu, a preto sa umelé mravce líšia od tých skutočných. Hlavné rozdiely sú:

- Skutočné mravce sa pohybujú v prostredí asynchrónne, zatiaľ čo pohyb umelých mravcov je synchronizovaný.
- Skutočné mravce sa pri návrate riadia feromónovými stopami, umelé mravce sa v každom cykle vracajú po rovnakých cestách, po ktorých sa v tomto cykle pohybovali.
- Skutočné mravce vypúšťajú feromón neustále, umelé mravce značkujú cestu feromónom iba pri návrate.
- Chovanie skutočných mravcov je založené na implicitnom vyhodnocovaní ciest. To spočíva v tom, že pohyb po kratších cestách trvá mravcom kratšiu dobu, a preto ich môžu opakovať častejšie, a tým sa množstvo feromónu na týchto cestách zvyšuje. Umelé mravce vyhodnocujú cesty explicitne, a to pri návratoch podľa kvality a dĺžky cesty.
- Skutočné mravce sú prakticky slepé, umelé mravce zrak majú.
- Umelé mravce majú pamäť.
- Prostredie, v ktorom sa umelé mravce pohybujú, je diskrétne.

ACO, ako meta-heuristika, popisuje celú triedu algoritmov (napr. Ant colony system, Elitist ant system, Max-min ant system). Meta-heuristiku môžeme zjednodušene rozdeliť na tri postupy: vytváranie riešení, aktualizácia feromónu a vonkajšie zásahy [4].

Vytváranie riešení spravuje kolóniu mravcov, ktorá paralelne a asynchrónne prechádza susednými vrcholmi grafu a postupne buduje svoje riešenia. Pohybujú sa pomocou stochastického rozhodovacieho postupu, ktorý využíva feromónove stopy a heuristické informácie. Týmto spôsobom mravce postupne vytvárajú riešenia optimalizačného problému. Akonáhle mravec dokončí zostavenie svojho riešenia, vyhodnotí ho a výsledná hodnota bude použitá vo fázi aktualizácie feromónu.

Aktualizácia feromónu upravuje intenzitu feromónnových stôp na použitých cestách v grafe. Intenzita sa buď zvyšuje alebo znižuje. K zvýšeniu dochádza, ak je stopa používaná veľkým množstvom mravcov. Naopak intenzita feromónov klesá v priebehu času, ak mravce cestu používajú málo alebo vôbec. Feromónová stopa teda zanecháva ostatným mravcom informáciu o preferovanej voľbe.

Vonkajšie zásahy slúžia k vykonaniu úprav vo výpočte, ktoré sa nedaju vykonávať na úrovnii mravcov. Táto fáza nie je nutná, ale používa sa k vylepšeniam z globálneho hľadiska.

4.2 Optimalizácia rojom častíc

Optimalizácia rojom častíc, angl. particle swarm optimization (PSO), je optimalizačná technika inšpirovaná chovaním kŕdľa vtákov pri hľadaní potravy, ktorá optimalizuje problém iteračným skúšaním zlepšiť riešenie vzhľadom na danú mieru kvality. Rieši problém tak, že má niekoľko riešení, reprezentovaných časticami, a pohybuje nimi v prehľadávanom priestore pomocou matematických vzorcov na výpočet polohy a rýchlosti častice. Má niekoľko spoločných rysov s genetickými algoritmami:

- Pracuje s populáciami jedincov. Jedinci sa nazývajú častice. Častice predstavujú riešenie problému, podobne ako chromozómy v GA.
- Ohodnotenie kvality častíc sa vykonáva pomocou hodnotiacich (fitness) funkcií.
- Počiatočné rozmiestnenie častíc v prehľadávanom priestore je náhodné.

Každá častica je definovaná svojou polohou v n-rozmernom priestore, vektorom rýchlosti jej pohybu a pamäťou predchádzajúceho úspechu pri hľadaní. Častice sú ovplyvňované ostatnými úspešnejšími časticami v roji. Algoritmus počíta pohyb roja v diskrétnych časových krokoch a neustále upravuje hodnoty popisujúce častice. Častica je tvorená:

- pozíciou vo vyhľadávacom priestore,
- fitness hodnotou na tejto pozícií,
- rýchlosťou (v skutočnosti posunutím), ktorá sa použije pre výpočet ďalšej pozície,
- pamäťou, ktorá obsahuje najlepšiu pozíciu, ktorú doposiaľ našla častica,
- fitness hodnotou predchádzajúceho najlepšieho riešenia [3].

Vzorce pre výpočet polohy $\vec{x}^k$ a rýchlosti $\vec{v}^k$ častice sú:

$$\vec{x}^k(t + \Delta t) = \vec{x}^k(t) + \vec{v}^k(t) \cdot \Delta t,$$
$$\vec{v}^k(t + \Delta t) = \omega \cdot \vec{v}^k(t) + c_p \cdot r_p \cdot (\vec{x}^k_{best} - \vec{x}^k(t)) + c_g \cdot r_g \cdot (\vec{x}_{best} - \vec{x}^k(t)), \text{ kde}$$

$\vec{x}^k_{best}$ - doposiaľ najlepšia poloha tejto častice
$\vec{x}_{best}$ - doposiaľ najlepšia poloha nájdená všetkými časticami roja
ω - koeficient zotrvačnosti, $\omega \in [0.4, 0.9]$
c_p - kognitivný váhový koeficient
c_g - sociálny váhový koeficient
r_p a r_g - náhodné koeficienty
Koeficienty r_p a r_g sa nastavujú náhodne s rovnomerným rozložením pravdepodobnosti na intervale [0,1]. Koeficienty c_p a c_g udávajú mieru dôležitosti individuálnej pamäte a sociálného vplyvu.

Medzi hlavné výhody optimalizácie rojom častíc patrí ľahká implementácia a rýchla konvergencia k optimu pre širokú škálu účelových funkcií.

PSO je v podstate veľmi jednoduché. Ponúka sa však veľké množstvo rôznych modifikácií. Môžeme napríklad upravovať koeficienty ω a c, ktoré by mali nadobúdať doporučené hodnoty, mať rôzne varianty topológií alebo pridať mutáciu, či použiť nejakú hybridnú techniku, ktorá skombinuje PSO s iným algoritmom.

4.3 Optimalizácia umelým včelím rojom

Optimalizácia umelým včelím rojom, angl. artificial bee colony optimization (ABC), je numerický optimalizačný algoritmus založený na správaní medonosných včiel a bol navrhnutý Karabogom [8].

Z prírody vieme, že medonostné včely zbierajú nektár z obrovských oblastí okolo svojho úľu. Včelie kolónie posielajú včely zbierať med z kvetín v pomere množstvu potravy, ktorá je na danom mieste k dizpozícií. V úle včely navzájom komunikujú prostredníctvom tanca, ktorým ostatné včely informujú o smere, vzdialenosti a kvalite zdroju potravy.

Algoritmus obsahuje tri typy včiel: včely zamestnané spracovávaním zdrojov (employed bees), vyčkávajúce včely (onlookers) a skautské včely (scouts), ktoré lietajú v D-dimenzionálnom vyhľadávacom priestore náhodne a hľadajú zdroje potravy. Oba typy včiel, vyčkávajúce včely a skauti, sa taktiež nazývajú nezamestnané včely. Spočiatku sú všetky pozície zdrojov potravín objavené skautmi. Potom, nektár zo zdrojov potravy využívajú zamestnané včely a vyčkávajúce včely, a toto kontinuálne vykorisťovanie nakoniec spôsobí, že sa zdroj potravy vyčerpá. Potom sa zamestnaná včela, ktorá využívala tento vyčerpaný zdroj potravy, stane skautom a bude hľadať ďalší zdroj potravy. Inými slovami, zamestnaná včela, ktorej zdroj potravy je vyčerpaný, sa stáva skautskou včelou.

V ABC algoritme pozícia zdroja potravín predstavuje možné riešenie problému s optimalizáciou a množstvo nektáru tohto zdroja zodpovedá kvalite príslušného riešenia. Počet zamestnaných včiel sa rovná počtu počtu zdrojov potravy (riešenia), pretože každá zamestnaná včela je spojená s jedným jediným zdrojom potravy.

Inicializačná fáza

V prvom kroku, algoritmus ABC generuje náhodne rozmiestnenú populáciu n riešení (poloha zdroja potravy), kde n označuje veľkosť populácie. Každé riešenie $x_i (i = 1, 2, ..., n)$ je D-dimenzionálny vektor, kde D je počet optimalizačných parametrov.

Po inicializácií sa populácia podrobí opakovaným cyklom vyhľadávacích procesov zamestnaných včiel, vyčkávajúcich včiel a skautských včiel.

Fáza zamestnaných včiel

Zamestnaná včela alebo vyčkávajúca včela mení pozíciu (riešenie) vo svojej pamäti na vyhľadanie nového zdroja potravy a testuje množstvo nektáru (fitness) nového zdroja (nové riešenie). Náhodne si zvolia pozíciu zdroja potravy a vykonajú zmenu pozície vo svojej pamäti. Za predpokladu, že množstvo nektáru nového zdroja je vyššie ako množstvo predchádzajucého, včela si zapamätá novú pozíciu a zabudne starú. V opačnom prípade si zachová predchádzajúcu pozíciu.

Fáza vyčkávajúcich včiel

Keď všetky zamestnané včely dokončia vyhľadávanie, zdieľajú informácie o zdrojoch potravy a ich pozície s vyčkávajucími včelami. Vyčkávajúca včela vyhodnocuje informácie od všetkých zamestnaných včiel a vyberá zdroj s pravdepodobnosťou, ktorú určuje množstvo nektáru v tomto zdroji. Rovnako ako v prípade zamestnaných včiel, aj vyčkavajúca včela si v pamäti zmení pozíciu a kontroluje množstvo nektáru tohto zdroja. Ak je množstvo nektáru vyššie ako predchádzajúce, zapamätá si novú pozíciu a starú zabudne.

Fáza skautských včiel

Nezamestnané včely, ktoré náhodne vyberajú svoj zdroj potravy, sa nazývajú skauti. Zamestnané včely, ktorých riešenie nie je možné zlepšiť vopred stanoveným počtom skúšok špecifikovaných užívateľom, sa stanú skautmi a ich riešenia sú opustené. Potom, títo konvertovaní skauti začínajú hľadať nové riešenia náhodne.

Zatiaľ čo vyčkávajúce a zamestnané včely vykonávajú proces vykorisťovania, skautské včely kontrolujú proces prieskumu [9].

Priebeh algoritmu

ABC optimalizuje n-rozmernú (fitness) funkciu $f(x)$ a jeho pevnými parametrami sú:
neb - počet zamestnaných včiel (EBs)
nob - počet vyčkávajúcich včiel (OBs)
limit - počet cyklov do vyčerpania zdroja

1. Náhodne vygenerujte počiatočnú populáciu $EBs : eb_i = (eb_{i1}, eb_{i2}, ..., eb_{in})$ a vynulujte čítače cyklov $c_i, i = 1...neb$.
2. Pomocou fitness funkcie ohodnoťte jednotlivé $EB : f(eb_i)$.
3. Ulože najlepšie ohodnotenú EB do EB_{best}.
4. Pre každú $eb_i (i = 1...neb)$
   (a) určte je novú pozíciu eb_i^{new},
   (b) ak je hodnota fitness funkcie novej pozície $f(eb_i^{new})$ lepšia než hodnota fitness funkcie pôvodnej pozície $f(eb_i)$, tak starú pozíciu eb_i nahraďte jej novou pozíciou a vynulujte čítač cyklu c_i, inak čítač inkrementujte.
5. Každú $ob_l (l = 1...nob)$
   (a) priraďte algoritmom rulety k niektorej eb_m,
   (b) určte novú pozíciu eb_m,
   (c) ak je hodnota fitness funkcie novej pozície $f(eb_m^{new})$ lepšia než hodnota fitness funkcie pôvodnej pozície $f(eb_m)$, tak starú pozíciu eb_m nahraďte novou pozíciou a vynulujte čítač cyklu c_m, inak čítač inkrementujte.
6. Pre každú $eb_i (i = 1...neb)$

   Ak $c_i > limit$, tak dočasne zmeňte eb_i na sb_i, tj. náhodne nastavte polohu $eb_i = (eb_{i1}, eb_{i2}, ..., eb_{id})$ a vynulujte čítače cyklov c_i.
7. Ak má niektorá EB lepšie ohodnotenie, než uložená EB_{best}, tak aktualizujte EB_{best} práve touto EB.
8. Ak nie sú dosiahnuté ukončovacie kritéria, vráťte sa na bod 4.
9. Vráťte najlepšie riešenie EB_{best}.

4.4 Algoritmus svetlušiek

Algoritmus svetlušiek, angl. firefly algorithm (FA), je stochastická metóda globálnej optimalizácie inšpirovaná prírodou, konkrétne svetluškami. Svetlušky sú hmyz charakteristický blikajúcim svetlom, ktorý sa produkuje procesom bioluminiscencie [5]. Tento algoritmus imituje mechanizmus párenia svetlušiek a výmeny informácií používaním svetelných záblesков. Existuje veľa druhov svetlušiek ale väčšina z nich produkuje krátke a rytmické záblesky. Základné funkcie záblesков sú:

- vábenie partnerov na párenie (komunikácia),
- vábenie potencionálnej koristi,
- poskytuje mechanizmus varovania.

Záblesky môžu byť formulované takým spôsobom, že sú spájané s objektívnou funkciou, ktorá sa má optimalizovať, čo umožňuje formulovať nové optimalizačné algoritmy [16].

Existujú dva faktory, ktorých kombinácia spôsobuje, že sú svetlušky viditeľné iba na obmedzenú vzdialenosť. Po prvé, intenzita svetla v určitej vzdialenosti r od zdroja svetla spĺňa zákon prevrátených štvorcov, angl. inverse square law. Ten hovorí, že intenzita svetla I klesá s druhou mocninou vzdialenosti od zdroja. Po druhé, vzduch absorbuje svetlo, ktoré sa stáva slabším a slabším s narastajúcou vzdialenosťou [6].

FA boli vyvinuté Yangom [16], ktorý zformuloval tri idealizované pravidlá, ktoré opisujú správanie inteligentných svetlušiek, teda svetlušiek použitých v algoritme. Sú to:

1. Všetky svetlušky sú jednopohlavné, takže jedna svetluška bude priťahovaná inými svetluškami bez ohľadu na ich pohlavie.
2. Príťažlivosť je úmerná jasu, teda pre akékoľvek dve blikajúce svetlušky, menej jasnejšia svetluška sa bude pohybovať smerom k tej viac jasnejšej. Príťažlivosť je úmerná jasu a obe sa znižujú s narastajúcou vzdialenosťou. Ak nie je žiadna svetluška jasnejšia než konkrétna svetluška, svetluška sa bude pohybovať náhodne.
3. Jas svetlušky je ovplyvnený alebo určený tvarom objektívnej funkcie [16].

Intenzita svetla svetlušky i, I_i závisí na intenzite svetla I_0 vyžarovaného svetluškou i a na vzdialenosti r medzi svetluškou i a j. Platí vzorec

$$I_i = I_0 e^{-\gamma r_{ij}}, \text{ kde}$$

γ je koeficient absorbcie svetla. Keďže príťažlivosť β_{ij} svetlušky i závisí na intenzite svetla, ktorú vidí susedná svetluška j a jej vzdialenosti r_{ij}, tak príťažlivosť β_{ij} je daná vzťahom

$$\beta_{ij} = \beta_0 e^{-\gamma r_{ij}^2}, \text{ kde}$$

β_0 je príťažlivosť v $r_{ij} = 0$. Koeficient príťažlivosti sa používa ako aproximácia straty intenzity svetla podľa vzdialenosti. Môže byť stvárnený akoukoľvek monotónne klesajúcou funkciou.

Pohyb svetlušky i priťahovanej k inej atraktívnejšej (jasnejšej) svetluške j je určený vzťahom

$$x_i = x_i + \beta_{ij}(x_j - x_i) + \alpha \in_{ij}, \text{ kde}$$

druhý termín je kvôli príťažlivosti zatiaľ čo tretí termín je náhodnosť s tým, že α je parameter vplyvu náhodného riešenia. $\in_{ij}$ je náhodný parameter s rovnomerným rozložením pravdepodobnosti. α patrí do intervalu $[0, 1]$. Parameter α fyzicky reprezentuje šum existujúci v prostredí a ovplyvňuje prenos svetla, zatiaľ čo v umelom algoritme môže byť zvolený tak, aby umožňoval variáciu riešenia a tým zabezpečoval väčšiu rozmanitosť riešení. Parameter γ charakterizuje odchýlku príťažlivosti a jeho hodnota je rozhodujúca pre určenie rýchlosti konvergencie a spôsobu chovania algoritmu. V mnohých aplikáciach sa typicky pohybuje od 0.01 do 100. Vzdialenosť medzi svetluškou i a svetluškou j je označená ako r_{ij} a definovaná ako

$$r_{ij} = ||\vec{x}_i - \vec{x}_j||, \text{ kde}$$

$\vec{x}_i$ je vektor reprezentujúci pozíciu svetlušky i a $\vec{x}_j$ pozíciu svetlušky j.

Pseudo kód

Algoritmus svetlušiek, ktorý je inšpirovaný ich správaním, môžeme v pseudo kóde zapísať následovne [6]:

input : n počet svetlušiek
$NIter$ počet iterácií pre optimalizáciu
γ parameter príťažlivosti
α príspevok náhodnosti (šum v prostredí)
output: pozícia optimálnej svetlušky a jej ohodnotenie

Inicializujte náhodne populáciu n pozícií svetlušiek
Nájdite najlepšie riešenie na základe fitness
while *Nie je splnená ukončovacia podmienka* **do**
  for *Svetluška i* **do**
    for *Svetluška j* **do**
      if *Svetluška j je lepšia ako Svetluška i* **then**
        Presuňte Svetlušku i smerom k Svetluške j použitím vyššie spomenutého vzťahu
      end
    end
  end
  Vyhodnoťte pozície jednotlivých svetlušiek
end

Kapitola 5

Popis implementácie

Táto kapitola popisuje implementáciu programu, aplikácie `GA`, ktorá slúži na demonštráciu jednoduchého genetického algoritmu ako aj na riešenie konkrétnych problémov. Venuje sa štruktúre programu, vývojovému prostrediu, knižniciam a jazyku, v ktorom bola aplikácia napísaná. Opisuje jednotlivé časti kódu a vysvetluje ich funkčnosť. Taktiež ukazuje grafické rozhranie a manipuláciu s ním.

5.1 Použité technológie

Aplikácia `GA` je napísaná v programovacom jazyku `Java`. `Java` je objektovo orientovaný programovací jazyk vyvíjaný spoločnosťou Oracle. Syntax vychádza z jazykov `C` a `C++`.

Pre jednoduchšiu prácu s grafickým rozhraním a celkovou organizáciou programu bolo použité vývojové prostredie `Netbeans IDE`, ktoré je open-source a je vytvorené pod licenciou CDDL od spoločnosti Sun Microsystems. `Netbeans IDE` zjednodušuje vytváranie nových prvkov aplikácie, napr. pomocou Drag & Drop, má jednoduché pridávanie a mazanie prvkov, ich úpravu a jednoducho sa s ním pracuje. Grafické užívateľské rozhranie (GUI) z väčšej časti využíva komponenty z knižnice `Swing` avšak obsahuje aj prvky z knižnice `JavaFX`.

Medzi komponenty z knižnice `Swing` radíme napríklad JFrame, ktorý tvorí rámec celého programu, ďalej sa tu nachádzajú komponenty ako JPanel, JButton, ktorý reprezentuje tlačidlo na klikanie, JLabel, ktorý zobrazuje text, JTextBlock a podobne.

Z knižnice `JavaFX` je využitá komponenta JFXPanel, na ktorej je zobrazený grafický výstup hľadania maxima alebo minima funkcie, ktoré sú riešené pomocou genetického algoritmu alebo algoritmu svetlušiek.

5.2 Štruktúra programu

Celý projekt obsahuje niekoľko súborov, pričom každý z nich vykonáva iný, špecifický účel. Podľa ich funkcionality by sme ich mohli rozdeliť do troch kategórií: súbory na úpravu vzhľadu, súbory s implementáciou algoritmov a súbory s datovými typmi.

Súbory s označením `GUI` popisujú vzhľad aplikácie. Sú celkovo štyri a každý z nich slúži pre vzhľad inej časti programu.

Do kategórie implementácia algoritmu, spadá súbor `GeneticAlgorithm.java`, ktorý riadi poradie vykonávaných genetických operácií. Má v sebe taktiež uložené preddefinované hodnoty, ktoré slúžia na voľbu genetických operácií. K týmto hodnotám patrí napríklad veľkosť populácie, počet generácií, typ selekcie, kríženia, či tvorby novej populácie. Ďalší súbor,

ktorý patrí do tejto kategórie je `FireflyAlgorithm.java`, ktorý riadi algoritmus svetlušiek. Obsahuje preddefinované hodnoty algoritmu, ktoré užívateľ zadá na vstupe. Patrí k nim napríklad veľkosť populácie svetlušiek alebo počet iterácií, koľko krát sa má algoritmus zopakovať.

V kategórií datový typ sa nachádzajú všetky ostatné súbory, ktoré by sme mohli rozdeliť podľa riešenej úlohy, ktorou sa zaoberajú. Súbor `Individual.java` je jadrom genetických algoritmov, pretože definuje základný stavebný prvok algoritmu, jedinca. Obsahuje metódy, ktoré vytvárajú jedinca a pracujú s ním, ako napríklad jeho inicializáciu, kríženie, mutáciu. Súbor `Population.java` potom obsahuje metódy, ktoré z viacerých jedincov vytvoria populáciu a ohodnotia ju. Taktiež obsahuje metódu na výber rodičov, kde sa môže vyberať podľa elity, turnaja alebo rulety, metódu na kríženie a mutáciu.

Pre problém obchodného cestujúceho bolo potrebné vytvoriť objekty reprezentujúce jednotlivé mestá, cez ktoré musí obchodný cestujúci prejsť pri hľadaní najkratšej cesty. Mesto sa nachádza v súbore `City.java`, ktorý ukladá súradnice mesta a vykresluje ho. Na vykreslovanie ciest medzi jednotlivými mestami slúži súbor `Linker.java`, ktorý bol prevzatý zo stránky stackoverflow[1] a následne upravený pre potreby programu.

Splniteľnosť logických formúl taktiež potrebovala svoje vlastné datové typy. Jeden z nich sa nachádza v súbore `Clause.java`, ktorý reprezentuje jednotlivé klauzule. Súbor `Variable.java` reprezentuje premennú, ktorá sa v klauzule nachádza, a či sa jedná o negáciu tejto premennej alebo nie.

Aby mohol byť otestovaný rozdiel medzi genetickým algoritmom a algoritmom svetlušiek, musel byť zvolený problém, ktorý sa dá riešiť oboma algoritmami. Problém obchodného cestujúceho bol príliš zložitý pre algoritmus svetlušiek, preto bolo zvolené hľadanie maxima alebo minima funckie. Hľadanie minima alebo maxima funkcie pomocou genetických algoritmov využíva okrem vyššie uvedených tried aj ďalšie dve triedy, ktoré sú v súboroch `Test.java` a `Xform.java`. Tieto triedy sa starajú o inicializáciu scény, vykreslenie osí a vykreslovanie jedincov na JFXPanel. Vykreslovanie vychádza z voľne dostupnej aplikácie MoleculseSampleApp[2], ktorá sa nachádza v Java dokumentácii na stránke spoločnosti Oracle.

Riešenie funkcií pomocou algoritmu svetlušiek je implementované v súboroch `Firefly.java` a `FA_pop.java`, avšak tiež využíva triedy Test a Xform.

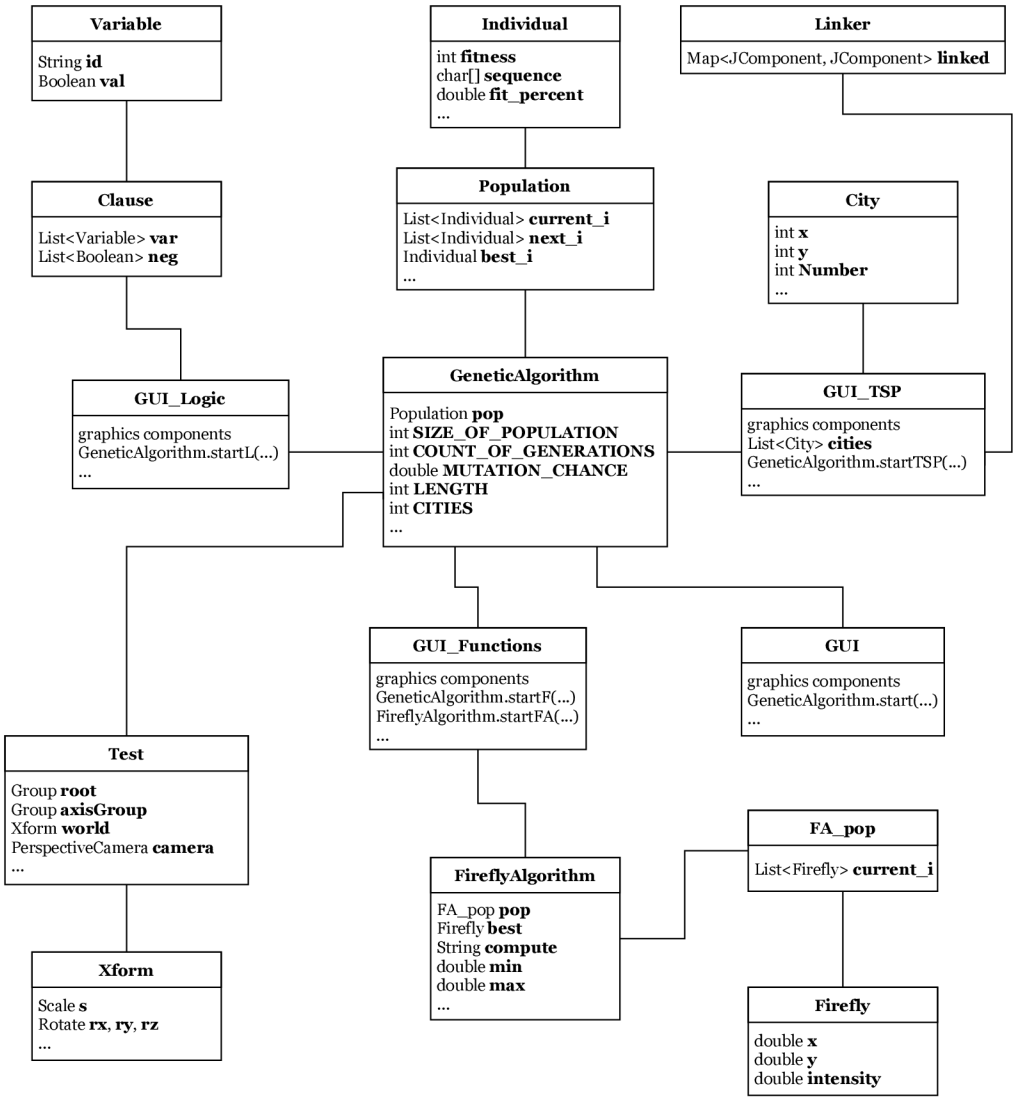
Pre počítanie funkcií bola použitá externá matematická knižnica mXparser[3], ktorá ulahčila počítanie funkčných hodnôt.

5.2.1 Hierarchia tried

Na obrázku 5.1 môžete vidieť jednotlivé triedy aplikácie a ich prepojenia. Obecný genetický algoritmus využíva triedy GUI, GeneticAlgorithm, Population a Individual. Problém obchodného cestujúceho taktiež používa triedy GeneticAlgorithm, Population a Individual, no namiesto GUI používa GUI_TSP. Navyše využíva triedy City a Linker. Splniteľnosť logických formúl taktiež využíva triedy GeneticAlgorithm, Population a Individual, na grafické rozhranie však používa triedu GUI_Logic a používa navyše dve triedy, Clause a Variable. Hľadanie minima alebo maxima funkcie využíva triedy GeneticAlgorithm, Population, Individual, GUI_Functions, Test a Xform v prípade, že počíta pomocou genetického algoritmu.

[1] https://stackoverflow.com/questions/12382184/how-can-we-draw-a-lines-between-2-panels-in-swing
[2] https://docs.oracle.com/javase/8/javafx/graphics-tutorial/sampleapp3d.htm
[3] http://mathparser.org/

V prípade, že počíta pomocou algoritmu svetlušiek využíva triedy FireflyAlgorithm, Firefly, FA_pop, Test, Xform a GUI_Functions.



Obr. 5.1: Hierarchia tried pre program GA

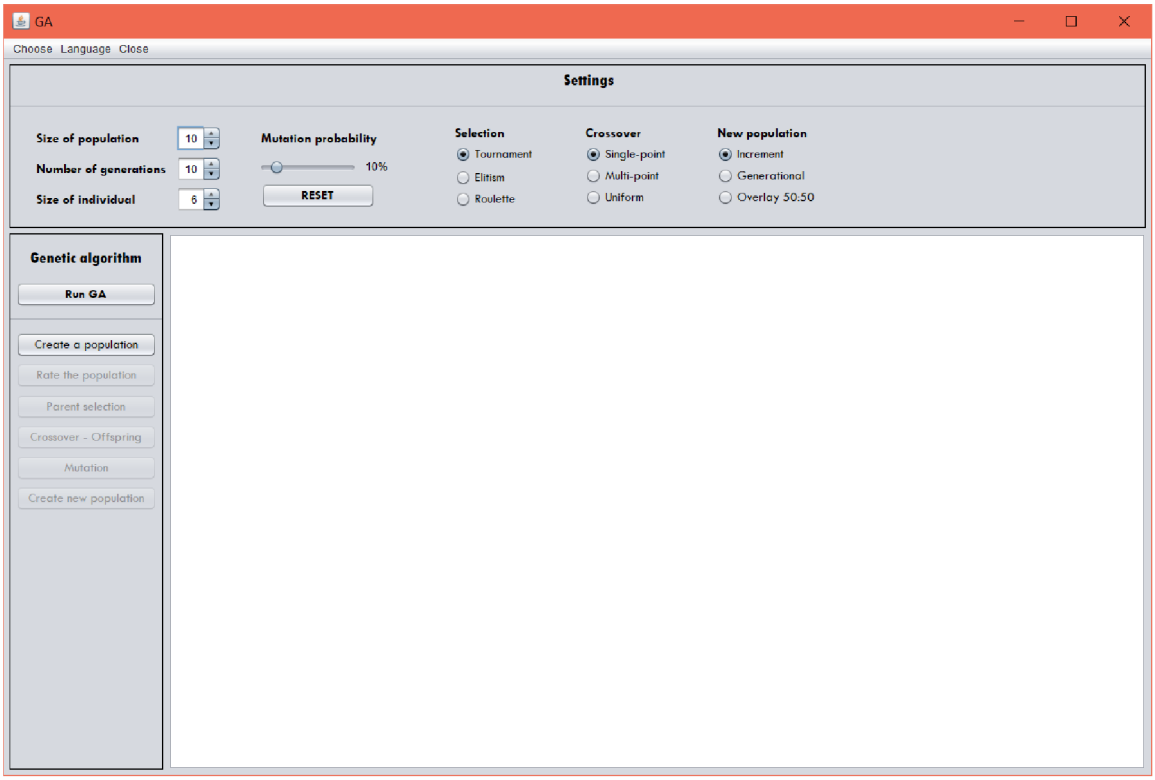
5.3 Užívateľské rozhranie

Užívateľské rozhranie je navrhnuté v `Netbeans IDE`. Využíva komponenty z knižnice `Swing` a `JavaFX`. Užívateľské rozhranie muselo byť prispôsobené konkrétnym problémom, a preto sa program skladá zo štyroch rôznych vzhľadov. Po spustení programu sa ako prvé zobrazí okno, ktoré demonštruje obecný genetický algoritmus. V hornej časti sa nachádza menu, v ktorom je možné prepnúť jednotlivé problémy (problém obchodného cestujúceho, splniteľnosť logických formúl a hľadanie minima alebo maxima funkcie). Menu taktiež obsahuje položku na zmenu jazyka. Program môžete mať v angličtine alebo slovenčine. Prednastavený je anglický jazyk.

5.3.1 Obecný genetický algoritmus

Grafické užívateľské rozhranie časti programu, ktorá demonštruje obecný genetický algoritmus sa skladá z troch častí. Vzhľad rozhrania je možné vidieť na obrázku 5.2. Ako už bolo spomenuté, horná časť obsahuje menu. Pod ním nasledujú nastavenia, kde si môže užívateľ zvoliť veľkosť jedinca, počet jedincov v populácií, počet generácií, o aký typ selekcie a kríženia sa jedná, ako aj pravdepodobnosť mutácie. Na ľavej strane je možné genetický algoritmus spustiť celý alebo ho postupne odkrokovať. Na pravej strane vidí užívateľ textový výstup genetického algoritmu, ktorý zobrazuje podrobné informácie o jeho priebehu.

Nastavenia genetického algoritmu sú už na začiatku prednastavené, aby užívateľ mohol spustiť program aj bez zbytočného zdržiavania.



Obr. 5.2: Grafické rozhranie pre obecný genetický algoritmus

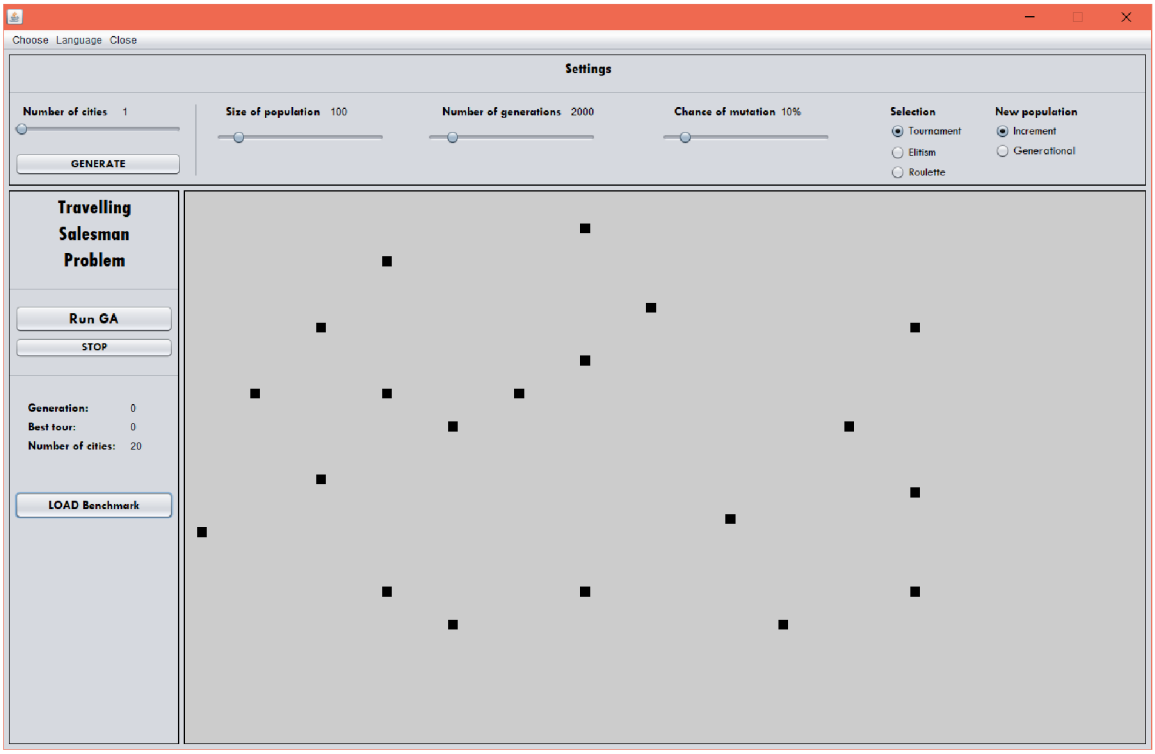
5.3.2 Problém obchodného cestujúceho

Grafické užívateľské rozhranie pre problém obchodného cestujúceho sa taktiež skladá z troch častí a jeho vzhľad je možné vidieť na obrázku 5.3.

Horná časť, rovnako ako pri obecnom genetickom algoritme, obsahuje nastavenia algoritmu. Tieto nastavenia sa ale delia na dve časti, ľavú a pravú. Ľavá časť obsahuje nastavenia, ktoré súvisia s tvorbou miest. Užívateľ môže posuvným koliečkom nastaviť, koľko miest má obchodný cestujúci prejsť. Po kliknutí na tlačidlo `GENERATE` sa daný počet miest náhodne vykreslí na ploche. Pravá časť nastavení obsahuje nastavenia, ktoré súvisia s genetickým algoritmom. Užívateľ môže opäť nastaviť veľkosť populácie, počet generácií, pravdepodobnosť mutácie, či typ selekcie alebo model pre tvorbu novej generácie. Parameter, ktorý sa nedá nastaviť, je typ kríženia, ktorý musel byť nastavený na PMX, pretože problém obchodného cestujúceho nemohol mať iný typ kríženia. Iné kríženie, napríklad jednobodové, viacbodové alebo uniformné, by spôsobilo, že by riešenie nebolo validné, pretože by cestujúci niektoré mestá navštívil viackrát a niektoré ani raz.

Ľavá časť rozhrania umožňuje užívateľovi spustiť algoritmus, a prípadne ho zastaviť. Taktiež obsahuje výpis, ktorý užívateľa informuje o generácii, ktorá je momentálne počítaná v algoritme, ďalej zobrazuje najkratšiu cestu, ktorú zatiaľ algoritmus našiel a počet miest, s ktorými algoritmus počíta. Ľavá časť obsahuje aj tlačidlo `LOAD Benchmark`, ktoré vykreslí 20 miest, ktoré boli použité pri testovaní.

Pravá časť rozhrania zobrazuje JPanel, na ktorom sa vykreslujú mestá a cesty medzi nimi. Mesto je možné vygenerovať po kliknutí na plochu.



Obr. 5.3: Grafické rozhranie pre problém obchodného cestujúceho

5.3.3 Logické formule

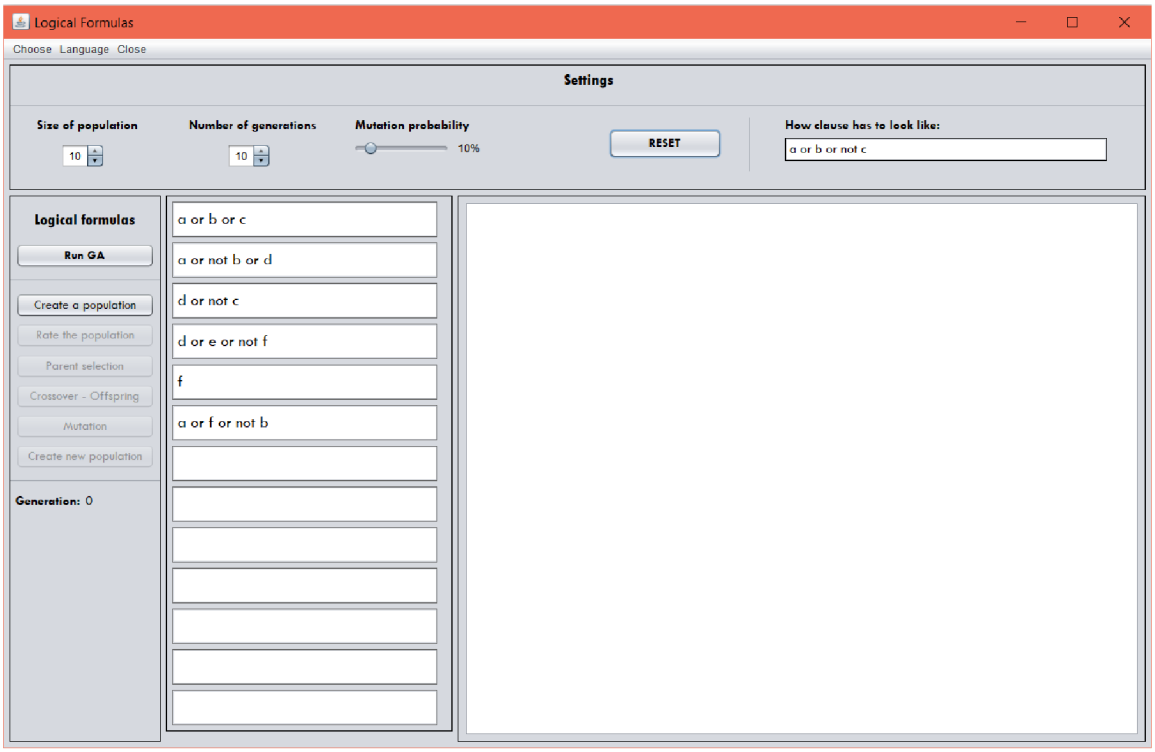
Grafické rozhranie pre splniteľnosť logických formúl je na obrázku 5.4 a môžeme ho rozdeliť na štyri časti.

Prvá časť, sa nachádza hore a obsahuje nastavenia genetického algoritmu podobne ako pri predchádzajúcich dvoch rozhraniach. Užívateľ môže zvoliť veľkosť populácie, počet generácií a pravdepodobnosť mutácie. V pravej časti sa nachádza nápoveda pre užívateľa, ako má vyzerať tvar formule. Nastavenia genetického algoritmu sú už na začiatku prednastavené, aby užívateľ mohol spustiť program bez zbytočného zdržiavania.

Ľavá časť rozhrania umožňuje užívateľovi spustiť algoritmus celý alebo ho postupne odkrokovať. Informuje užívateľa aj o generácií, ktorá je momentálne počítaná.

Nasledujúca časť sa nachádza napravo od ľavej časti a obsahuje trinásť textových polí, do ktorých môže užívateľ zadať klauzule. Každé pole môže obsahovať iba jednu klauzulu, takže maximálny počet klauzúl je trinásť. Tvar klauzule je ukázaný v hornej časti rozhrania. Niektoré klauzule sú už na začiatku vyplnené, aby užívateľ mohol spustiť program bez vymýšlania nových klauzúl.

Na pravej strane rozhrania vidí užívateľ textový výstup, ktorý zobrazuje podrobné informácie o priebehu genetického algorimu.



Obr. 5.4: Grafické rozhranie pre splniteľnosť logických formúl

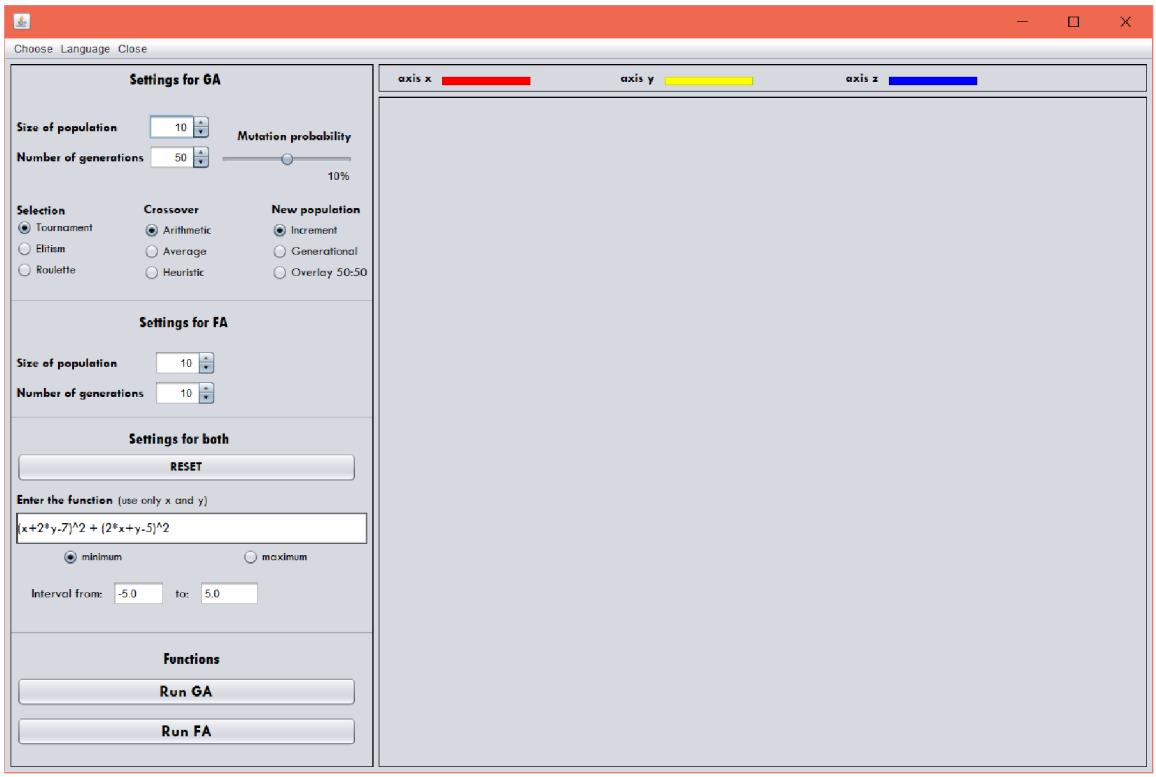
5.3.4 Funkcie

Užívateľské rozhranie pre hľadanie minima alebo maxima funkcie môžeme rozdeliť na dve časti. Vzhľad rozhrania je možné vidieť na obrázku 5.5.

Ľavá časť obsahuje nastavenia algoritmov. Túto časť, môžeme rozdeliť podľa typu algoritmu, ktorý nastavuje. Horná časť slúži pre genetický algoritmus. Užívateľ môže nastaviť

veľkosť populácie, počet generácií alebo pravdepodobnosť mutácie. Taktiež si môže zvoliť typ selekcie, kríženia alebo model pre tvorbu novej populácie. Ďalej následujú nastavenia pre algoritmus svetlušiek, kde môže užívateľ zvoliť veľkosť populácie alebo počet generácií. Pod nimi nasledujú nastavenia slúžiace pre oba algoritmy. Užívateľ volí funkciu, či má algoritmus hľadať jej minimum alebo maximum a interval, na ktorom sa počíta. Po nastavení všetkých parametrov môže užívateľ spusiť niektorý z algoritmov. Genetický algoritmus môže spustiť tlačidlom `Run GA` alebo algoritmus svetlušiek môže spustiť pomocou tlačidla `Run FA`. Nastavenia oboch algoritmov sú už na začiatku prednastavené, aby užívateľ mohol spustiť program bez zbytočného zdržiavania. Funkcia, s ktorou program počíta, je taktiež už prednastavená. Užívateľ ju však môže samozrejme zmeniť.

Pravú časť rozhrania tvorí JFXPanel, na ktorom môžeme vidieť grafický výstup. Pre lepšie pochopenie výstupu, obsahuje horná časť vysvetlivky, ktorá os je ktorá.



Obr. 5.5: Grafické užívateľské rozhranie pre funkcie

Kapitola 6

Experimentovanie

V tejto kapitole je popísané podrobné experimentovanie s programom `GA`. Každá časť programu, to znamená každý riešený problém, bol otestovaný na rôznej sade úloh. Bola skúmaná efektivita genetického algoritmu pri riešení daného problému, ako aj vplyv voľby parametrov na riešenie.

6.1 Obecný genetický algoritmus

Obecný genetický algoritmus bol otestovaný pri rôznych typoch parametrov. Každý typ bol otestovaný desať krát. Spoločnými parametrami bola veľkosť jedinca, nastavená vždy na šesť a model vytvárania novej generácie, ktorý bol inkrementačný. Jedinci sa teda skladali zo šiestich génov, kde každý gén mal hodnotu 0 alebo 1.

Riešením bolo nájsť jedinca, ktorý obsahuje gény iba s hodnotami 1. To znamená jedinca s chromozómom 111111.

6.1.1 Jednotlivé typy parametrov

Prvý typ

Pri tomto type parametrov bola veľkosť populácie nastavená na desať, rovnako ako aj počet generácií. Pravdepodobnosť mutácie bola 10%, selekcia prebiehala pomocou turnaja a kríženie jedincov bolo nastavené na jednobodové. Výsledky jednotlivých testov sú následovne:

Číslo testu	1	2	3	4	5	6	7	8	9	10
Fitness najlepšieho jedinca	4	5	3	3	4	5	5	3	4	4

Z vykonaných testov vidíme, že priemerná hodnota fitness funkcie najlepšieho jedinca je 4.0. Navyše sa algoritmu nepodarilo ani raz nájsť hľadané riešenie.

Druhý typ

Veľkosť populácie aj počet generácií boli nastavené na desať. Pravdepodobnosť mutácie sa taktiež nezmenila a zostala na 10%. Kríženie jedincov bolo znova jednobodové a model tvorby novej populácie bol inkrementačný. Selekcia však bola tentokrát nastavená na výber elity. Jednotlivé testy dopadli následovne:

Číslo testu	1	2	3	4	5	6	7	8	9	10
Fitness najlepšieho jedinca	5	5	5	5	4	5	4	5	5	5

Priemerná hodnota fitness funkcie najlepšieho riešenia je 4.8. Algoritmu sa opäť ani raz nepodarilo nájsť hľadané riešenie, ale priblížilo sa k nemu viac ako pri selekcii pomocou turnaja, čo môžeme vidieť z priemernej hodnoty fitness funkcií najlepších jedincov.

Tretí typ

Veľkosť populácie aj počet generácií boli opäť nastavené na desať, pravdepodobnosť mutácie taktiež na 10%. Kríženie naďalej zostalo jednobodové a model pre vytvorenie novej generácie zostal inkrementačný. Zmenila sa selekcia, ktorá bola nastavená na ruletu. Výsledky jednotlivých testov dopadli následovne:

Číslo testu	1	2	3	4	5	6	7	8	9	10
Fitness najlepšieho jedinca	6	3	5	5	5	5	5	4	3	5

Priemerná hodnota fitness funkcie najlepšieho riešenia je 4.6. Algoritmus našiel hľadané riešenie aspoň raz, vďaka čomu môžeme vyhlásiť, že genetický algoritmus dokáže riešiť takýto typ problému. Genetický algoritmus teda pri selekcii pomocou rulety našiel hľadané riešenie, ale v priemere sú riešenia nájdené s touto selekciou o trochu horšie ako so selekciou výberu elity.

Štvrtý typ

Keďže boli otestované všetky typy selekcie, v ďalšom type parametrov použijeme opäť selekciu pomocou turnaja a vyskúšame, či by iné kríženie nepomohlo k nájdeniu lepšieho riešenia. Počet jedincov, aj počet generácií zostal nastavený na desať, mutácia taktiež a nezmenil sa ani inkrementačný model. Použité bolo viacbodové kríženie. Jednotlivé testy dopadli následovne:

Číslo testu	1	2	3	4	5	6	7	8	9	10
Fitness najlepšieho jedinca	5	5	5	5	3	4	5	4	4	4

Priemerná hodnota fitness funkcie najlepšieho jedinca je 4.4. Môžeme usúdiť, že pri selekcii pomocou turnaja a pri tomto type problému viacbodové kríženie pomáha algoritmu hľadať lepšie výsledky ako jednobodové kríženie, keďže sa priemerná hodnota fitness funkcie zvýšila zo 4.0 na 4.4.

Piaty typ

V tomto type boli parametre rovnaké ako v predchádzajúcom, líšila sa len forma kríženia, pretože namiesto viacbodového bolo použité uniformné kríženie. Výsledky jednotlivých testov dopadli následovne:

Číslo testu	1	2	3	4	5	6	7	8	9	10
Fitness najlepšieho jedinca	5	4	4	4	4	5	5	4	3	4

Priemerná hodnota fitness funkcie najlepšieho jedinca je 4.2. Uniformné kríženie je teda pri tomto type parametrov efektívnejšie ako jednobodové, ale oproti viacbodovému neprinieslo zlepšenie.

Šiesty typ

Keďže algoritmus so selekciou rulety našiel aspoň raz hľadané riešenie, ďalší typ parametrov znova využíva selekciu rulety. Pre zistenie vplyvu veľkosti populácie bol počet jedincov navýšený na päťdesiat. Počet generácií zostal desať, rovnako ako aj pravdepodobnosť mutácie a inkrementačný model. Kríženie bolo nastavené na viacbodové. Jednotlivé testy dopadli následovne:

Číslo testu	1	2	3	4	5	6	7	8	9	10
Fitness najlepšieho jedinca	5	4	4	5	4	5	4	4	5	5

Priemerná hodnota fitness funkcie najlepšieho jedinca je 4.5. Viacbodové kríženie nenašlo hľadané riešenie ani raz, a ani priemer najlepších nájdených riešení nestúpol oproti jednobodovému kríženiu. Počet jedincov pri riešení tohto problému taktiež nepreukázal vplyv na zlepšenie výsledku.

Siedmy typ

Aby boli otestované všetky typy kríženia pri selekcii ruletou, v tomto type bolo zmenené viacboové kríženie na uniformné. Ostatné z parametrov zostali rovnaké ako v predchádzajúcom type. Výsledky jednotlivých testov dopadli následovne:

Číslo testu	1	2	3	4	5	6	7	8	9	10
Fitness najlepšieho jedinca	5	5	4	5	4	4	5	6	5	5

Priemerná hodnota fitness funkcie najlepšieho jedinca je 4.8. Ako môžeme vidieť, algoritmus našiel jeden krát hľadané riešenie a zvýšila sa aj priemerná hodnota fitness funkcie najlepšieho riešenia.

Ôsmy typ

Pri poslednom type parametrov bola veľkosť populácie znova nastavená na desať, avšak zvýšil sa počet generácií na päťdesiat. Prevdepodobnosť mutácie aj všetky ostatné parametre zostali rovnaké ako v siedmom type, čiže selekcia bola pomocou rulety, kríženie zostalo uniformné a model pre tvorbu novej populácie bol znova inkrementačný. Jednotlivé testy dopadli následovne:

Číslo testu	1	2	3	4	5	6	7	8	9	10
Fitness najlepšieho jedinca	5	6	5	5	6	5	6	6	6	5

Priemerná hodnota fitness funkcie najlepšieho jedinca je 5.5. Môžeme vidieť obrovské zlepšenie oproti predchádzajúcim typom parametrov. Algoritmus nielenže našiel oveľa viackrát hľadané riešenie, ale zlepšil sa aj priemer nájdených najlepších riešení.

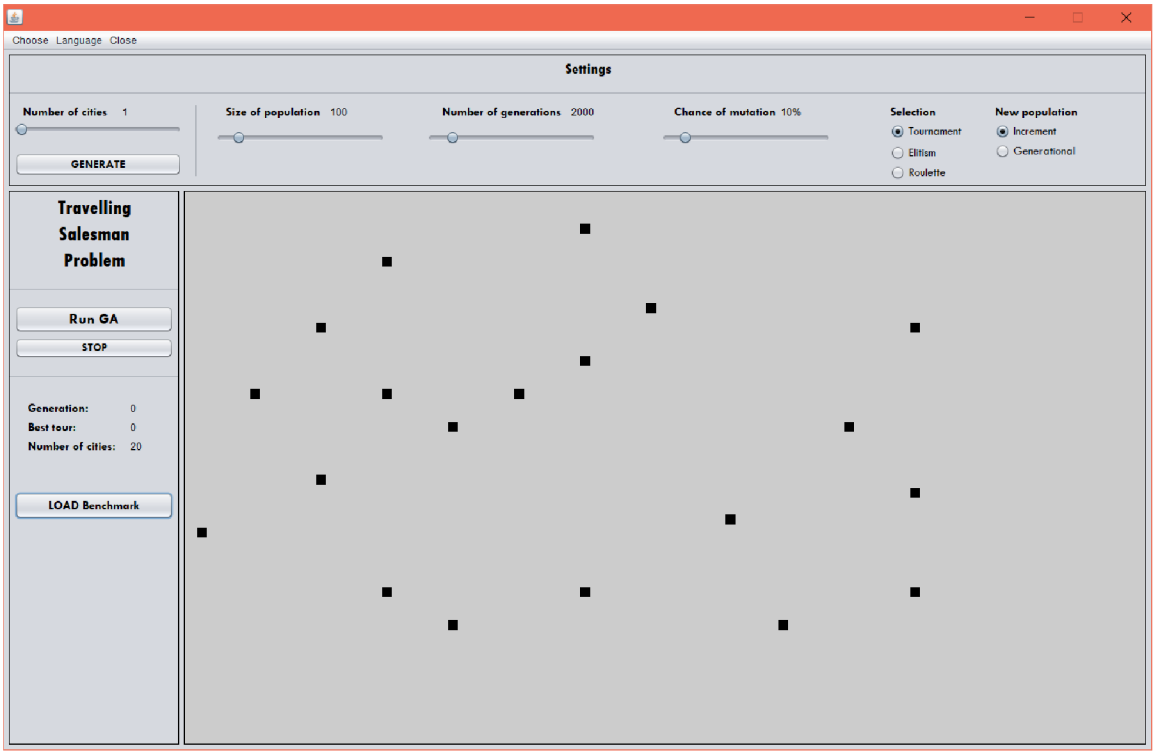
6.1.2 Zhrnutie

Ako môžeme vidieť z jednotlivých testov, najväčší vplyv na výsledok má počet generácií. Čím viac času má genetický algoritmus na nájdenie najlepšieho riešenia, tým lepší je výsledok. Zo selekčných operátorov sa ukázal ako najlepší výber elity, samozrejme to ale nemusí platiť pri iných typoch úloh. Z operátorov kríženia sa ako najlepšie preukázalo uniformné kríženie.

Z dosiahnutých výsledkov môžeme predpokladať, že algoritmus implementovaný v programe GA dokáže demonštrovať genetický algoritmus a vie riešiť typ problému, pri ktorom musí nájsť určitý počet niektorej z binárnych hodnôt.

6.2 Problém obchodného cestujúceho

Problém obchodného cestujúceho bol testovaný pre rôzne typy parametrov. Úlohou bolo zistiť či genetické algoritmy dokážu riešiť aj takýto typ problému. Aby mali všetky typy parametrov rovnaké podmienky a testovanie mohlo byť objektívne, bolo vytvorených 20 miest, na ktorých sa skúšali všetky typy parametrov. Rozloženie miest môžeme vidieť na obrázku 6.1.



Obr. 6.1: Rozloženie miest pre testovanie

Najkratšia cesta, ktorú musí obchodný cestujúci prejsť, aby navštívil všetky mestá na obrázku, má dĺžku 2883.

6.2.1 Testovanie

Prvý test

Pri prvom testovaní bola populácia nastavená na 100 jedincov, počet generácií na 2000 a pravdepodobnosť mutácie bola 10%. Pri tomto testovaní bol zvolený inkrementačný model pre tvorbu novej generácie. Selekcia bola postupne nastavená na turnaj, elitu aj ruleta, kde pre každý jeden typ bol program spustený desaťkrát. Následne bola zostavená tabuľka s najlepším a najhorším riešením, spolu s priemerom všetkých riešení. Jednotlivé testy dopadli následovne:

	Najlepšia	Priemerná	Najhoršia
Turnaj	5223	5403.3	5635
Elita	3820	4304.2	5305
Ruleta	3561	3843.5	4007

Z výsledkov môžeme vidieť, že najlepšie výsledky dosiahla selekcia pomocou rulety, no aj tak boli výsledky stále relatívne vzdialené od riešenia, ktorého veľkosť je 2883.

Druhý test

V druhom testovaní bola populácia opäť nastavená na 100 jedincov, počet generácií na 2000 a pravdepodobnosť mutácie bola 10%. Pri tomto testovaní bol však zvolený generačný model pre tvorbu novej generácie a nie inkrementačný ako v predchádzajúcom teste. Selekcia bola znova postupne nastavená na turnaj, elitu aj ruleta, kde pre každý jeden typ bol program spustený desaťkrát. Výsledky jednotlivých testov dopadli následovne:

	Najlepšia	Priemerná	Najhoršia
Turnaj	10000+	10000+	10000+
Elita	3481	4118.9	4712
Ruleta	10000+	10000+	10000+

Výsledky pre selekciu turnajom a ruletou boli oproti inkrementačnému modelu neuveriteľne zlé. Všetky testovania našli najlepšieho jedinca, ktorý mal najkratšiu cestu viac ako 10000, preto neboli jednotlivé výsledky ani spracované. Oproti inkrementačnému modelu sa teda generačný model oplatil iba v prípade výberu elity, kde môžeme vidieť zlepšenie výsledkov.

Tretí test

V treťom testovaní bola zvolená selekcia výberom elity, keďže mala v predchádzajúcich testoch najlepšie výsledky. Počet generácií zostal 2000 a bol zvolený inkrementačný model. Tento test spočíval v navýšení pravdepodobnosti mutácie z 10% na 100%. Test prebiehal pre veľkosť populácie 100 a 500 jedincov.

	Najlepšia	Priemerná	Najhoršia
100 jedincov	3097	3452.3	3950
500 jedincov	3049	3543.6	3994

Z výsledkov, ktoré boli získané vidíme, že veľkosť populácie nemala výrazný vplyv na zlepšenie riešenia, dokonca môžeme v priemere vidieť mierne zhoršenie.

Štvrtý test

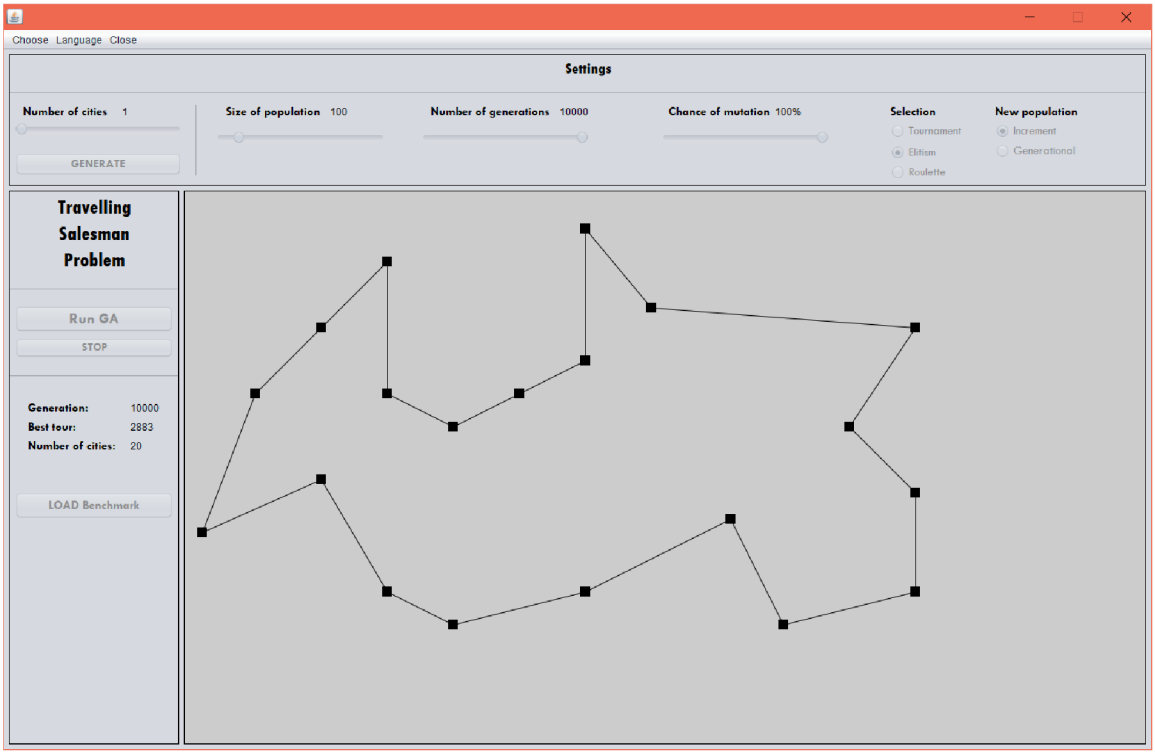
V poslednom teste bola opäť zvolená selekcia pomocou výberu elity a inkrementačný model. Veľkosť populácie bola nastavená na 100. Počet generácií bol zmenený z 2000 na 10000, aby mohol byť otestovaný vplyv počtu iterácií na hľadané riešenie. Test prebiehal pre pravdepodobnosť mutácie 10% a 100%.

	Najlepšia	Priemerná	Najhoršia
10% mutácia	3277	3634.8	3924
100% mutácia	2883	3404.6	4021

Výsledky ukazujú, že mutácia má veľmi priaznivý vplyv na nájdenie najkratšej cesty. Spolu so zvýšením počtu generácií výrazne ovplyvnila kvalitu riešenia. Nielenže mala cesta najnižšiu priemernú hodnotu, ale algoritmus dokonca našiel aj skutočnú najkratšiu cestu, ktorej hodnota je 2883.

6.2.2 Zhrnutie

Z dosiahnutých výsledkov vidíme, že problém obchodného cestujúceho môže byť riešený aj pomocou genetických algoritmov. Najlepšenie riešenie bolo dosiahnuté pri veľkosti populácie 100 a počte generácií 10000 s pravdepodobnosťou mutácie 100%, z čoho môžeme usúdiť, že počet generácií a vyššia pravdepodobnosť mutácie dopomáhajú k nájdeniu lepšieho riešenia. Selekcia bola nastavená na elitu a model pre tvorbu novej populácie na inkrementačný. Najkratšia nájdená cesta je ukázaná na obrázku 6.2.



Obr. 6.2: Najkratšia cesta pri dvadsiatich mestách

6.3 Logické formule

Splniteľnosť logických formúl bola testovaná pri piatich rôznych parametroch. Kvôli objektivite boli všetky typy parametrov testované na šiestich klauzulách, ktoré dohromady reprezentujú formulu:

$$F = (a \vee b \vee c) \wedge (a \vee \neg b \vee d) \wedge (d \vee \neg c) \wedge (d \vee e \vee \neg f) \wedge (f) \wedge (a \vee f \vee \neg b).$$

6.3.1 Testovanie

Pri všetkých testovaniach bola nastavená selekcia turnajom a kríženie bolo jednobodové. Tvorba novej populácie bola vykonaná pomocou inkrementačného modelu.

Prvý test

V prvom teste bola zvolená počiatočná populácia veľkosti 5. Počet generácií bol nastavený na 10 a pravdepodobnosť mutácie na 10%. Výsledky jednotlivých testov obsahujú generáciu, v ktorej bolo nájdené správne riešenie.

Číslo testu	1	2	3	4	5	6	7	8	9	10
Generácia	0	-	-	0	0	0	-	0	0	-

Druhý test

Pri druhom teste bola počiatočná populácia znovu nastavená na 5. Počet generácií bol ale zvýšený z 10 na 50. Pravdepodobnosť mutácie zostala na 10%. Výsledky obsahujú generáciu, v ktorej algoritmus našiel riešenie.

Číslo testu	1	2	3	4	5	6	7	8	9	10
Generácia	0	11	0	0	0	0	46	-	37	0

Tretí test

Pri treťom testovaní bola zmenená počiatočná populácia na 10. Počet generácií bol nastavený na 5, pravdepodobnosť mutácie zostala na 10%. Výsledky obsahujú generáciu, v ktorej bolo nájdené riešenie.

Číslo testu	1	2	3	4	5	6	7	8	9	10
Generácia	-	0	0	0	0	-	-	-	0	0

Štvrtý test

V štvrtom testovaní bola populácia opäť 10, zvýšil sa však počet generácií oproti predchádzajucému testovania, a to na 10. Pravdepodobnosť mutácie stále zostala na 10%. Výsledky jednotlivých testov obsahujú generáciu, v ktorej bolo nájdené riešenie.

Číslo testu	1	2	3	4	5	6	7	8	9	10
Generácia	0	5	0	0	0	0	0	0	-	7

Piaty test

Pri poslednom teste bol počet jedincov v populácií aj počet generácií nastavený na 10. Zvýšila sa však pravdepodobnosť mutácie z 10% na 50%. Výsledky obsahujú generáciu, v ktorej algoritmus našiel riešenie.

Číslo testu	1	2	3	4	5	6	7	8	9	10
Generácia	0	0	0	-	0	0	8	0	0	1

6.3.2 Zhrnutie

Z dosiahnutých výsledkov môžeme vidieť, že genetický algoritmus dokáže riešiť aj splniteľnsť logických formúl. Algoritmus už pri malom počte jedincov alebo generácií dokázal nájsť hľadané riešenie. Dokonca hľadané riešenie veľakrát našiel už pri vygenerovaní počiatočnej populácie, čo mohlo byť spôsobené ľahkou zadanou formulou.

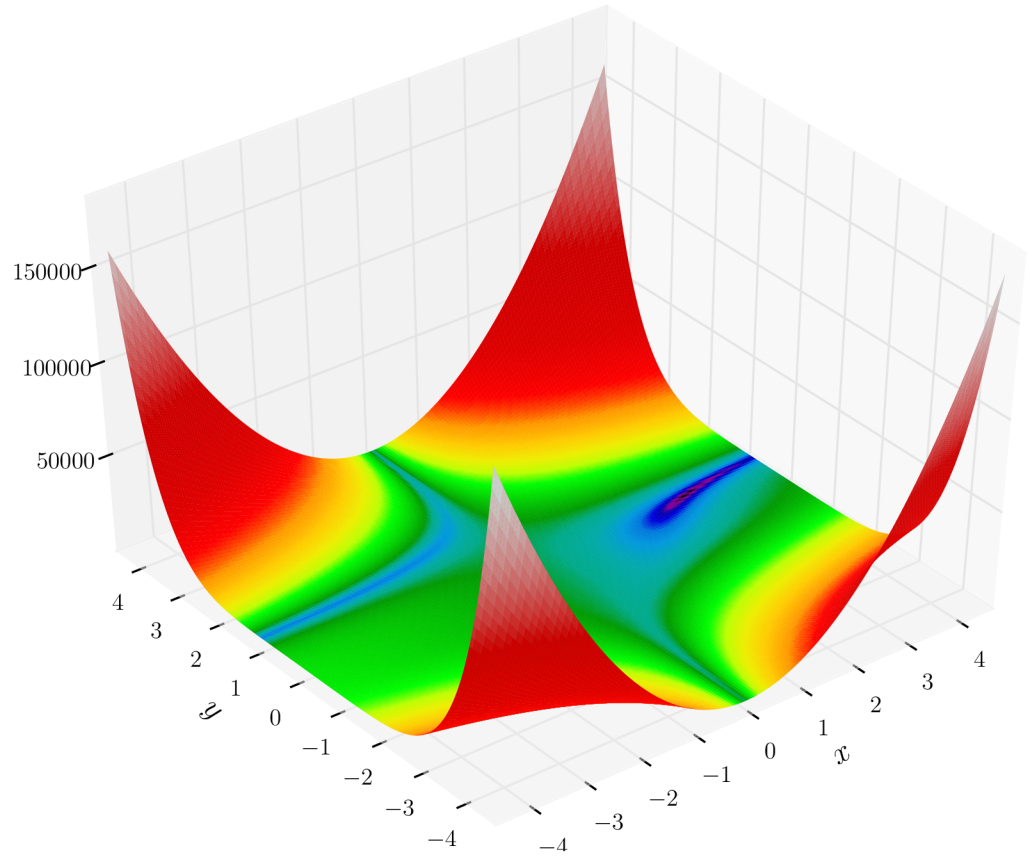
6.4 Funkcie

Hľadanie maxima alebo minima funkcie, bolo otestované pre genetické algoritmy, aj pre algoritmus svetlušiek. Testovanie prebiehalo s dvoma funkciami, konkrétne to boli Bealeho funkcia a Boothova funkcia. Program mal za úlohu nájsť minimum týchto funkcií, a keďže je minimum týchto funkcií známe, mohla byť zistená efektivita programu `GA`. Taktiež bola porovnaná efektivita genetického algoritmu s algoritmom svetlušiek.

6.4.1 Bealeho funkcia

Bealeho funkcia, má globálne minimum v bode $[3, 0.5]$, ktoré je rovné 0. Funkcia sa zvyčajne počíta na intervale od -4.5 do 4.5. Ukážka funkcie [14] je na obrázku 6.3 a má tvar:

$$f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$$



Obr. 6.3: Bealeho funkcia na intervale $-4.5 < x, y < 4.5$

Program bol testovaný vždy na upravenom intervale od -4.0 do 4.0. Boli vyskúšané štyri rôzne nastavenia parametrov, kde každý typ parametrov bol otestovaný desaťkrát. Následne bol zostavený priemer najlepšieho riešenia, najlepšie nájdené riešenie a najhoršie riešenie.

Pri genetických algoritmoch bola použitá selekcia turnajom, bolo zvolené aritmetické kríženie a inkrementačný model pre vytvorenie novej populácie. Pri algoritme svetlušiek bola β nastavená na.

Prvý test

Pri prvom testovaní bola veľkosť populácie nastavená na 10 jedincov a počet generácií, čiže počet iterácií, bol taktiež nastavený na 10. Výsledky jednotlivých testov dopadli následovne:

	Najlepšie	Priemer	Najhoršie
GA	0.1117	5.96063	17.3530
FA	0.0023	0.12586	0.5233

Druhý test

V druhom testovaní bola veľkosť populácie ponechaná na 10 jedincov, zmenil sa však počet generácií, ktorý stúpol na 50. Jednotlivé testy dopadli následovne:

	Najlepšie	Priemer	Najhoršie
GA	0.8931	6.53270	10.5104
FA	$4.86e^{-4}$	0.01182	0.0286

Tretí test

V treťom teste stúpla veľkosť populácie na 50 jedincov, počet generácií však opäť klesol na 10. Výsledky jednotlivých testov sú:

	Najlepšie	Priemer	Najhoršie
GA	0.0726	1.51288	3.7431
FA	$2.90e^{-4}$	0.00159	0.0061

Štvrtý test

V poslednom testovaní bol počet jedincov v populácií aj počet generácií nastavený na 50. Jedntlivé testy dopadli následovne:

	Najlepšie	Priemer	Najhoršie
GA	0.0186	1.35777	3.7143
FA	$0.175e^{-4}$	$8.47e^{-4}$	0.0028

Zhodnotenie

Zo získaných výsledkov môžeme vidieť, že sa riešenie aj pri genetickom algoritme, aj pri algoritme svetlušiek zlepšuje so zvyšujúcim počtom jedincov ako aj so zvyšujúcim počtom generácií. V poslednom testovaní oba algoritmy dosiahli najlepšie výsledky.

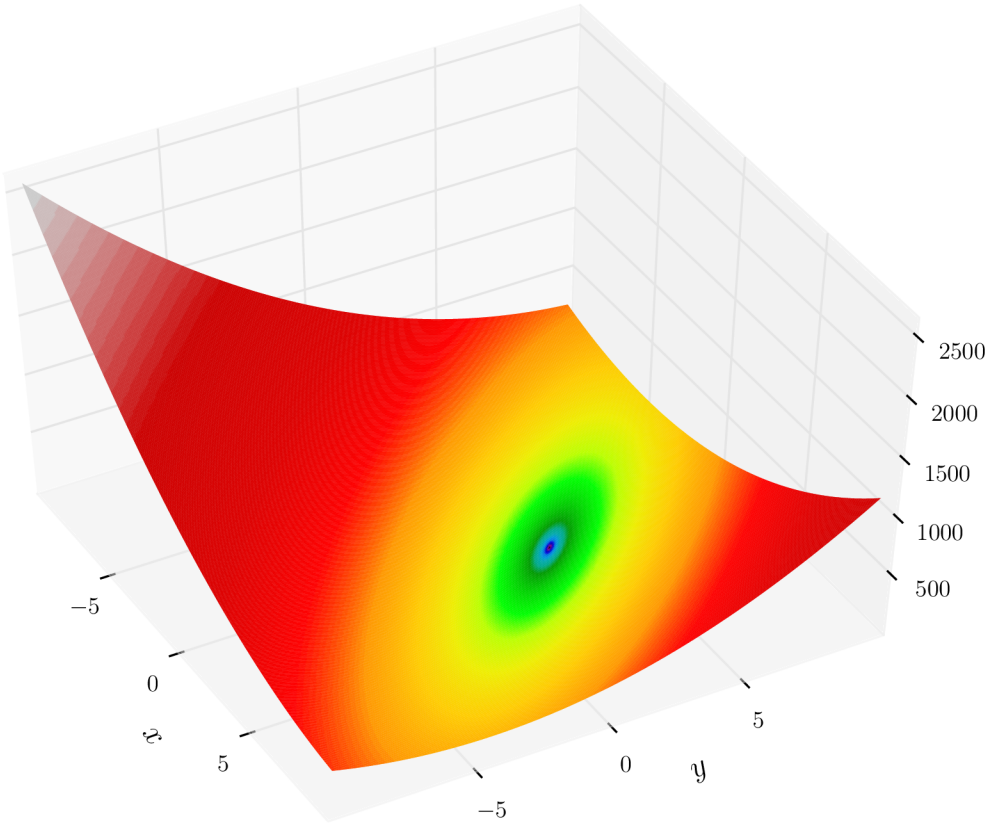
Môžeme však pozorovať aj rozdiel medzi genetickým algoritmom a algoritmom svetlušiek. Pri všetkých rôznych nastaveniach, algoritmus svetlušiek našiel oveľa lepšie riešenie ako genetický algoritmus, z čoho usúdime, že je algoritmus svetlušiek pri tomto type problému efektívnejší ako genetický algoritmus.

Keďže sa výsledky oboch algoritmov blížia hľadanej 0, môžeme usúdiť, že oba algoritmy dokážu riešiť daný typ problému. To znamená, že vedia hľadať minimum alebo maximum funkcie, pričom algoritmus svetlušiek daný problém rieši efektívnejšie.

6.4.2 Boothova funkcia

Boothova funkcia, má globálne minimum v bode $[1, 3]$, ktoré je rovné 0. Funkcia sa zvyčajne počíta na intervale od -10.0 do 10.0. Ukážka funkcie [15] je na obrázku 6.4 a má tvar:

$$f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2$$



Obr. 6.4: Boothova funkcia na intervale $-10 < x, y < 10$

Program bol testovaný vždy na upravenom intervale od -5.0 do 5.0, ale so štyrmi rôznymi nastaveniami parametrov. Každý typ parametrov bol otestovaný desaťkrát, a potom bol zostavený priemer najlepšieho riešenia, najlepšie nájdené riešenie a najhoršie riešenie.

Prvý test

Pri prvom teste bola veľkosť populácie nastavená na 10 a počet generácií bol taktiež nastavený na 10. Výsledky jednotlivých testov dopadli následovne:

	Najlepšie	Priemer	Najhoršie
GA	0.9252	6.06462	25.5656
FA	0.0185	0.59266	1.8224

Druhý test

V druhom teste veľkosť populácie zostala na 10 jedincoch, počet generácií však bol navýšený na 50. Jednotlivé testy dopadli následovne:

	Najlepšie	Priemer	Najhoršie
GA	0.6605	2.08226	6.9716
FA	$0.46e^{-4}$	0.07415	0.1867

Tretí test

V predposlednom testovaní bol počet jedincov v populácií navýšený z pôvodných 10 na 50. Počet generácií však opäť klesol na 10. Výsledky jednotlivých testov sú:

	Najlepšie	Priemer	Najhoršie
GA	0.0941	1.25542	3.0069
FA	0.0017	0.00572	0.0090

Štvrtý test

V poslednom teste sa počet jedincov nastavil na 50, rovnako ako aj počet generácií. Výsledky testov dopadli nsledovne:

	Najlepšie	Priemer	Najhoršie
GA	0.1010	0.97655	2.3839
FA	$0.696e^{-4}$	$6.237e^{-4}$	0.0012

Zhodnotenie

Rovnako ako aj pri predchádzajúcej funkcii môžeme zo získaných výsledkov vidieť, že sa riešenie aj pri genetickom algoritme, aj pri algoritme svetlušiek zlepšuje so zvyšujúcim počtom jedincov ako aj so zvyšujúcim počtom generácií. V poslednom testovaní opäť oba algoritmy dosiahli najlepšie výsledky.

Keďže sa výsledky oboch algoritmov aj pri tejto funkcii blížia k hľadanej 0, môžeme usúdiť, že oba algoritmy naozaj dokážu riešiť daný typ problému.

6.4.3 Zhrnutie

Pri týchto pokusoch môžeme pozorovať rozdiel medzi genetickým algoritmom a algoritmom svetlušiek. Zatiaľ čo algoritmus svetlušiek veľmi rýchlo nájde veľmi dobré riešenie, pri genetických algoritmoch hrá veľkú rolu veľkosť počiatočnej populácie. Čím väčšia populácia je, tým je väčšia pravdepodobnosť, že sa niektorý jedinec priblíži k hľadanému riešeniu. Počet generácií však v genetickom algoritme pri tomto type úlohy veľkú rolu nehral, ak niektorý jedinec už na začiatku nenašiel dobré riešenie, genetický algoritmus sa nemal ako dobracovať k lepšiemu riešeniu. Počet generácií slúžil skôr pre horších jedincov, aby mohli vďaka lepším jedincom plodiť potomkov, ktorí sa stále približovali k najlepšiemu jedincovi. Preto môžeme vidieť, že pri veľkom počte generácií je riešenie nájdené najhorším jedincom oveľa nižšie ako pri menšom počte generácií.

Ako si môžeme všimnúť, algoritmus svetlušiek je efektívnejší než genetický algoritmus. Už pri malom počte jedincov a malom počte generácií dokázal nájsť veľmi dobré riešenie, ktoré sa s narastajúcim počtom jedincov alebo generácií ešte zlepšovalo.

Kapitola 7

Záver

V teoretickej časti tejto práce sme sa zoznámili s genetickými algoritmami, popísali ich terminológiu a jednotlivé kroky, ktoré sa vykonávajú v genetickom algoritme. Ďalej boli uvedené jednotlivé problémy, ktoré môžu byť pomocou genetických algoritmov riešené. K týmto problémom patril problém obchodného cestujúceho, splniteľnosť logických formúl a hľadanie extrému funkcie. Následne boli vysvetlené pojmy zo skupinovej inteligencie, predovšetkým algoritmus svetlušiek, ktorý bol použitý pri hľadaní extrému funkcie.

V praktickej časti bol uvedený popis implementácie programu `GA`, ktorý slúži na demonštráciu jednoduchého genetického algoritmu a umožňuje prácu s rôznymi typmi parametrov. Program taktiež slúži na riešenie troch rôznych optimalizačných úloh. Každá úloha bola podrobená rôznym experimentom, pri ktorých boli menené jednotlivé parametre algoritmu. Problém obchodného cestujúceho ja splniteľnsť logických formúl bola riešená pomocou genetického algoritmu.

Hľadanie extrému funkcie bolo implementované dvoma spôsobmi, genetickým algoritmom aj algoritmom svetlušiek. Na tejto optimalizačnej úlohe bola porovnaná efektivita oboch algoritmov.

Na záver experimentovania s každou úlohou, boli zhodnotené získané výsledky. Najzaujímavejší výsledok priniesol algoritmus svetlušiek, ktorý sa ukázal ako efektívnejší oproti genetickým algoritmom.

Literatúra

[1] Ajith, A.: *Evolutionary computation, in Handbook of Measuring System Design.* John Wiley and Sons, 2005, ISBN 9780470021439.

[2] Ashlock, D.: *Evolutionary Computation for Modeling and Optimization.* Springer Science and Business Media, 2006, ISBN 9780387319094.

[3] Clerc, M.: *Standard Particle Swarm Optimisation.* 2012, [Online: navštívené 11.4.2018].
URL https://hal.archives-ouvertes.fr/file/index/docid/764996/filename/SPSO_descriptions.pdf

[4] Dorigo, M.; Stützle, T.: *Ant Colony Optimization.* 2004, [Online: navštívené 20.3.2018].
URL
http://sci2s.ugr.es/sites/default/files/files/Teaching/GraduatesCourses/Metaheuristicas/Bibliography/ABC-algorithm-numerical-function-2007.pdf

[5] Elkhechafi, M.; Hachimi, H.; Elkettani, Y.: *A New Hybrid Firefly with Genetic Algorithm for Global Optimization.* ročník 3, 06 2017: s. 47–51.

[6] Hassanien, A. E.; Emarz, E.: *Swarm Intelligence: Principles, Advances, and Applications.* CRC Press, 2015, ISBN 9781498741064.

[7] Iba, H.; Noman, N.: *New Frontier in Evolutionary Algorithms.* Imperial College Press, 2012, ISBN 9781848166813.

[8] Karaboga, D.: *An idea based on honey bee swarm for numerical optimization.* 2005, [Online: navštívené 11.4.2018].
URL https://abc.erciyes.edu.tr/pub/tr06_2005.pdf

[9] Karaboga, D.; Basturk, B.: *A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm.* Springer Science+Business Media, 2007, ISBN 9780262042192.

[10] Lengál, O.: *NP-completeness.* [Online: navštívené 25.4.2018].
URL https://www.fit.vutbr.cz/study/courses/SLOa/private/prednasky/slo-np-completeness.pdf

[11] Mallawaarachchi, V.: *How to define a Fitness Function in a Genetic Algorithm?* [Online; navštívené 10.04.2018].
URL https://towardsdatascience.com/how-to-define-a-fitness-function-in-a-genetic-algorithm-be572b9ea3b4

[12] Singh, V.; Choudhary, S.: *Genetic Algorithm for Traveling Salesman Problem: Using Partially-Mapped Crossover Operator*. March 2009, 20-23 s.

[13] Soni, N.; Kumar, T.: *Study of Various Mutation Operators in Genetic Algorithms*. ročník 5, 2014: s. 4519–4521, [Online: navštívené 16.3.2018].
URL http://ijcsit.com/docs/Volume%205/vol5issue03/ijcsit20140503404.pdf

[14] Wikipedia: 2012, [Online: navštívené 27.4.2018].
URL https://upload.wikimedia.org/wikipedia/commons/d/de/Beale%27s_function.pdf

[15] Wikipedia: 2012, [Online: navštívené 27.4.2018].
URL https://upload.wikimedia.org/wikipedia/commons/6/6e/Booth%27s_function.pdf

[16] Yang, X. S.: *Firefly Algorithm, Lévy Flights and Global Optimization, in: Research and Development in Intelligent Systems XXVI*. 2010, [Online: navštívené 15.3.2018].
URL https://arxiv.org/pdf/1003.1464.pdf

Príloha A

Výsledky testov

Bealeho funkcia

Genetické algoritmy

Populácia = 10, Počet generácií = 10

1.	$[0.869, 0.782]$	=	10.0918
2.	$[1.210, 0.160]$	=	3.3971
3.	$[-0.225, -0.203]$	=	17.3530
4.	$[3.350, 0.220]$	=	2.5935
5.	$[2.290, -0.600]$	=	5.3237
6.	$[1.013, 0.912]$	=	11.9822
7.	$[-0.540, 1.590]$	=	4.4146
8.	$[2.490, 0.400]$	=	0.1117
9.	$[-1.400, 1.490]$	=	1.3236
10.	$[1.510, -0.580]$	=	3.0151

Populácia = 10, Počet generácií = 50

1.	$[-1.760, 1.250]$	=	3.6090
2.	$[0.433, 0.394]$	=	10.0003
3.	$[0.722, 0.650]$	=	9.3295
4.	$[0.060, -3.640]$	=	10.5104
5.	$[0.577, 0.520]$	=	9.3693
6.	$[2.380, -0.010]$	=	0.8937
7.	$[1.960, 0.350]$	=	0.8931
8.	$[-2.660, 1.130]$	=	5.7162
9.	$[0.440, 0.396]$	=	9.9477
10.	$[1.740, -0.580]$	=	3.0578

Populácia = 50, Počet generácií = 10

1.	$[3.080, 0.730]$	=	1.6573
2.	$[2.630, 0.090]$	=	0.9266
3.	$[2.570, 0.330]$	=	0.0726
4.	$[2.810, 0.010]$	=	1.9908
5.	$[-2.740, 1.290]$	=	0.9500
6.	$[-0.200, 2.280]$	=	3.7431
7.	$[2.230, 0.540]$	=	1.2309
8.	$[2.570, 0.200]$	=	0.3620
9.	$[1.830, 0.420]$	=	1.6101
10.	$[-1.120, 1.420]$	=	2.5854

Populácia = 50, Počet generácií = 50

1.	$[3.970, 0.620]$	=	0.1968
2.	$[2.090, 0.200]$	=	0.3933
3.	$[-1.650, 1.380]$	=	1.3401
4.	$[3.420, 0.590]$	=	0.0186
5.	$[2.070, -0.560]$	=	3.7143
6.	$[-3.060, 1.220]$	=	1.2709
7.	$[-1.030, 1.680]$	=	2.2886
8.	$[1.670, 0.280]$	=	1.5774
9.	$[2.420, -0.090]$	=	1.3585
10.	$[2.250, -0.170]$	=	1.4192

Algoritmus svetlušiek

Populácia = 10, Počet generácií = 10

1.	$[2.645, 0.414]$	=	0.0339
2.	$[2.737, 0.457]$	=	0.0301
3.	$[2.574, 0.502]$	=	0.2967
4.	$[3.657, 0.649]$	=	0.0663
5.	$[2.080, 0.317]$	=	0.5233
6.	$[2.423, 0.336]$	=	0.1084
7.	$[2.971, 0.432]$	=	0.0747
8.	$[3.525, 0.615]$	=	0.0306
9.	$[3.011, 0.493]$	=	0.0023
10.	$[4.000, 0.689]$	=	0.0923

Populácia = 10, Počet generácií = 50

1.	$[2.996, 0.468]$	=	0.0212
2.	$[3.004, 0.465]$	=	0.0286
3.	$[3.056, 0.513]$	=	$4.8612e^{-4}$
4.	$[2.768, 0.409]$	=	0.0233
5.	$[3.195, 0.554]$	=	0.0075
6.	$[3.060, 0.521]$	=	0.0017
7.	$[3.026, 0.518]$	=	0.0037
8.	$[2.922, 0.471]$	=	0.0028
9.	$[3.368, 0.588]$	=	0.0179
10.	$[2.895, 0.451]$	=	0.0110

Populácia = 50, Počet generácií = 10

1.	$[3.028, 0.497]$	=	0.0026
2.	$[2.901, 0.489]$	=	0.0061
3.	$[2.978, 0.491]$	=	$3.3685e^{-4}$
4.	$[3.028, 0.504]$	=	$2.8957e^{-4}$
5.	$[3.128, 0.526]$	=	0.0027
6.	$[2.992, 0.504]$	=	$8.0539e^{-4}$
7.	$[2.972, 0.490]$	=	$3.8839e^{-4}$
8.	$[2.942, 0.487]$	=	$6.3662e^{-4}$
9.	$[2.942, 0.483]$	=	$6.5204e^{-4}$
10.	$[2.939, 0.478]$	=	0.0014

Populácia = 50, Počet generácií = 50

1.	$[3.086, 0.518]$	=	0.0012
2.	$[3.043, 0.514]$	=	$5.8600e^{-4}$
3.	$[2.996, 0.500]$	=	$1.7515e^{-5}$
4.	$[2.913, 0.473]$	=	0.0017
5.	$[3.011, 0.504]$	=	$9.2187e^{-5}$
6.	$[2.908, 0.468]$	=	0.0028
7.	$[3.016, 0.500]$	=	$3.3918e^{-4}$
8.	$[2.972, 0.486]$	=	0.0011
9.	$[3.055, 0.511]$	=	$5.8861e^{-4}$
10.	$[3.014, 0.503]$	=	$4.8047e^{-5}$

Boothova funkcia

Genetické algoritmy

Populácia = 10, Počet generácií = 10

1.	$[3.250, 1.600]$	=	9.9125
2.	$[1.883, 2.093]$	=	1.6060
3.	$[1.856, 2.063]$	=	1.6384
4.	$[1.660, 2.640]$	=	0.9252
5.	$[2.015, 2.015]$	=	2.0041
6.	$[2.490, 2.241]$	=	4.9336
7.	$[1.690, 1.960]$	=	2.0477
8.	$[3.260, 0.830]$	=	9.8489
9.	$[-0.980, 2.660]$	=	25.5656
10.	$[2.096, 2.096]$	=	2.1642

Populácia = 10, Počet generácií = 50

1.	$[0.760, 2.020]$	=	6.9716
2.	$[1.952, 2.169]$	=	1.6553
3.	$[1.962, 2.158]$	=	1.6918
4.	$[1.852, 2.058]$	=	1.6457
5.	$[0.570, 3.600]$	=	0.6605
6.	$[1.895, 2.085]$	=	1.6407
7.	$[1.899, 2.089]$	=	1.6388
8.	$[1.885, 2.074]$	=	1.6483
9.	$[1.958, 2.176]$	=	1.6690
10.	$[1.903, 2.115]$	=	1.6009

Populácia = 50, Počet generácií = 10

1.	$[1.020, 3.360]$	=	0.7076
2.	$[1.710, 3.080]$	=	3.0069
3.	$[1.670, 2.310]$	=	0.9266
4.	$[1.962, 1.962]$	=	2.0255
5.	$[1.510, 2.890]$	=	0.9122
6.	$[1.140, 3.280]$	=	0.8036
7.	$[0.550, 3.640]$	=	0.7565
8.	$[1.790, 1.989]$	=	1.8399
9.	$[0.840, 3.030]$	=	0.0941
10.	$[0.480, 2.970]$	=	1.4813

Populácia = 50, Počet generácií = 50

1.	$[1.903, 2.114]$	=	1.6008
2.	$[1.859, 2.065]$	=	1.6342
3.	$[1.860, 2.067]$	=	1.6324
4.	$[1.020, 2.720]$	=	0.3492
5.	$[0.820, 3.620]$	=	1.1912
6.	$[1.330, 2.960]$	=	0.4469
7.	$[0.750, 3.380]$	=	0.2745
8.	$[2.146, 2.146]$	=	2.3839
9.	$[0.850, 3.010]$	=	0.1010
10.	$[0.710, 3.230]$	=	0.1514

Algoritmus svetlušiek

Populácia = 10, Počet generácií = 10

1.	$[1.004, 2.936]$	=	0.0185
2.	$[1.150, 2.780]$	=	0.0916
3.	$[1.931, 2.484]$	=	1.8224
4.	$[1.536, 2.425]$	=	0.6235
5.	$[1.307, 3.057]$	=	0.6281
6.	$[1.311, 2.419]$	=	0.7251
7.	$[0.614, 2.997]$	=	0.7544
8.	$[0.347, 3.572]$	=	0.7791
9.	$[1.162, 2.722]$	=	0.1571
10.	$[0.587, 3.268]$	=	0.3268

Populácia = 10, Počet generácií = 50

1.	$[0.771, 3.047]$	=	0.1867
2.	$[1.015, 3.122]$	=	0.0912
3.	$[0.985, 3.011]$	=	$4.2983e^{-4}$
4.	$[0.890, 3.031]$	=	0.0382
5.	$[0.971, 3.123]$	=	0.0510
6.	$[0.996, 3.005]$	=	$4.634e^{-5}$
7.	$[0.757, 3.102]$	=	0.1497
8.	$[0.726, 3.283]$	=	0.1554
9.	$[1.086, 2.862]$	=	0.0369
10.	$[1.112, 2.954]$	=	0.0319

Populácia = 50, Počet generácií = 10

1.	$[0.971, 3.052]$	=	0.0057
2.	$[0.955, 3.020]$	=	0.0050
3.	$[0.942, 3.048]$	=	0.0060
4.	$[1.000, 3.042]$	=	0.0089
5.	$[1.029, 2.955]$	=	0.0039
6.	$[0.995, 2.970]$	=	0.0059
7.	$[1.024, 2.950]$	=	0.0057
8.	$[1.070, 2.950]$	=	0.0090
9.	$[0.946, 3.036]$	=	0.0054
10.	$[0.989, 3.026]$	=	0.0017

Populácia = 50, Počet generácií = 50

1.	$[1.004, 3.005]$	=	$3.0335e^{-4}$
2.	$[0.998, 2.986]$	=	0.0012
3.	$[1.000, 2.996]$	=	$6.9611e^{-5}$
4.	$[0.996, 2.999]$	=	$1.3732e^{-4}$
5.	$[1.012, 2.987]$	=	$3.2259e^{-4}$
6.	$[0.974, 3.020]$	=	0.0012
7.	$[1.020, 2.992]$	=	$9.5413e^{-4}$
8.	$[1.005, 3.003]$	=	$2.4404e^{-4}$
9.	$[1.001, 3.010]$	=	$6.0604e^{-4}$
10.	$[1.004, 2.982]$	=	0.0012