EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA MEDZINÁRODNÝCH VZŤAHOV
Evidenčné číslo: 103003/B/2023/36124048427518980

Možnosti využitia evolučných algoritmov pri riešení optimalizačných úloh

Bakalárska práca

2023 **Juraj Pajdlhauser**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA MEDZINÁRODNÝCH VZŤAHOV

Možnosti využitia evolučných algoritmov pri riešení optimalizačných úloh

Bakalárska práca

Študijný program: Hospodárska informatika
Študijný odbor: Hospodárska informatika
Školiace pracovisko: Katedra operačného výskumu a ekonometrie
Vedúci záverečnej práce: doc. Ing. Zuzana Čičková, PhD.

Bratislava 2023 **Juraj Pajdlhauser**

Poďakovanie

Týmto by som sa chcel veľmi poďakovať vedúcej bakalárskej práce, pani doc. Ing. Zuzana Čičková, PhD., za jej ústretovosť, trpezlivosť pri kozultáciach ako aj za jej odborný prístup a užitočné rady, ktoré mi pomohli pri písaní bakalárskej práce.

Abstrakt

PAJDLHAUSER, Juraj: *Možnosti využitia evolučných algoritmov pri riešení optimalizačných úloh* – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra operačného výskumu a ekonometrie. – Vedúci záverečnej práce: doc. Ing. Zuzana Čičková, PhD. Bratislava: FHI, 2023, 44s.

Cieľom bakalárskej práce je úspešne vyriešiť dané úlohy pomocou evolučného algoritmu. Práca je rozdelená do piatich kapitol. Obsahuje 16 obrázkov. Prvá kapitola je venovaná stručnému vysvetleniu základných pojmov ako algoritmus alebo optimalizácia a tiež programovaciemu jazyku Python či Darvinovej evolučnej teorií. V ďalšej časti sú charakterizované ciele a metodika práce. Záverečná kapitola sa zaoberá riešením daných funkcií pomocou nástrojov knižníc programovacieho jazyku Python. Výsledkom riešenia danej problematiky je minimalizácia Schweffelovej a Easomovej funkcie a tiež nájdenie optimálnej trajektórie cez nami navrhnuté body.

Kľúčové slová: Genetické algoritmy, Evolučné algoritmy, optimalizácia, minimalizácia

Abstract

PAJDLHAUSER, Juraj: Possibilities of using evolutionary algorithms in solving optimization tasks - University of Economics in Bratislava. Faculty of Economic Informatics; Department of operational research and econometrics. – Leader of thesis: doc. Ing. Zuzana Čičková, PhD. Bratislava: FHI, 2023, 44p.

The aim of the bachelor's thesis is to successfully solve the given tasks using an evolutionary algorithm. The work is divided into five chapters. Contains 16 images. The first chapter is devoted to a brief explanation of basic terms such as algorithm or optimization, as well as the Python programming language or Darwin's evolutionary theory. In the next part, the goals and methodology of the work are characterized. The final chapter deals with the solution of the given functions using the tools of the Python programming language libraries. The result of solving the given problem is the minimization of the Schweffel and Easom functions and also the finding of the optimal trajectory through the points proposed by us.

Key words: Genetic algorithms, Evolutionary algorithms, optimization, minimization

Obsah

Úvod

V dnešnej dobe je optimalizácia jedným z kľúčových faktorov úspešnosti vo väčšine odvetví. Ľudia vymysleli sofistikované teórie a dokázali úplne zmeniť svet okolo seba, ale stále nachádzajú inšpiráciu v prírode. Evolučná teória ukázala, že evolúcia organizmov môže byť mocnou metódou na riešenie rôznych problémov. Pretvorením evolučnej teórie do formálneho sveta matematiky a informatiky vzniklo zovšeobecnenie nazvané evolučné algoritmy. Evolučné algoritmy vďaka momentálnej rýchlej technologickej evolúcií dokážu nachádzať sub-optimálne riešenia pre rôzne typy problémov. Evolučné algoritmy sú používateľsky obľúbené hlavne vďaka ich dvom vlastnostiam:

- Sú schopné nachádzať cieľové sub-optimálne riešenia takmer univerzálne. Preto sa hodia na riešenia problémov, pre ktoré iná lepšia metóda nie je, užívateľ o nej nevie alebo trvanie klasického výpočtu rastie so zložitosťou problému exponenciálne.
- Evolučné algoritmy sú schopné simulovať podobné systémy v prírode alebo spoločnosti, pretože sa sami vyvíjajú a adaptujú.

Táto bakalárska práca sa zaoberá možnosťami využitia evolučných algoritmov pri riešení optimalizačných úloh. Cieľom práce je predstaviť evolučné algoritmy ako nástroj pre optimalizáciu, ukázať ich silné a slabé stránky a možnosti ich aplikácie. Evolučné algoritmy nedokážu nájsť optimálne riešenie pre daný problém, ale iba sub-optimálne, čiže najlepšie možné. Každý z algoritmov má svoje vlastné silné a slabé stránky, preto je dôležité zvážiť, ktorý z nich je najvhodnejší pre konkrétnu optimalizačnú úlohu. Celkovo teda táto bakalárska práca poskytne prehľad o možnostiach využitia evolučných algoritmov pri riešení optimalizačných úloh a ukáže ich potenciál a výhody v porovnaní s inými metódami optimalizácie.

V prvej časti si popíšeme a vysvetlíme základné pojmy akými sú optimalizácia alebo algoritmus. Taktiež si vysvetlíme niečo viac k programovaciemu jazyku Python, v ktorom budú riešené všetky praktické úlohy. A na záver si popíšeme niečo viac k Darwinovej teórii, ktorá sa neodmysliteľne spája s evolučnými algoritmami.

V ďalšej časti, *Cieľ práce,* si podrobnejšie rozoberieme, akým úlohám sa budeme v praktickej časti venovať.

V praktickej časti práce budeme pomocou evolučného algoritmu minimalizovať tri funkcie. Začneme minimalizáciou Schweffelovej funkcie, kde budeme skúmať schopnosť evolučného algoritmu vymaniť sa z lokálneho extrému funkcie. Pretože definičný obor funkcie obsahuje veľké množstvo extrémov, čo má za následok, že algoritmus má veľký nábeh v nich uviaznuť. Najprv sa pokúsime prísť k úspešnému riešeniu pomocou knižnice PyGad (genetický algoritmus) jej nástrojov pre túto funkciu a neskôr si výsledok porovnáme s riešením cez knižnicu Scipy (diferenciálna evolúcia). Ďalej budeme minimalizovať Easomovu funkciu, ktorá je pre algoritmus tým špecifická, že na veľkej časti definičného oboru je konštantná. Algoritmus bude musieť zvládnuť prehľadávať priestor efektívne aj bez spätnej väzby o úspechu v riešení. V poslednej tretej úlohe si na niekoľkých behoch vyskúšame algoritmus na nájdenie najkratšej alebo minimálnej trajektórie prechodu cez niekoľko daných bodov.

Posledná časť práce je venovaná záveru, kde sme zhodnotili či a prečo sú nami navrhnuté riešenia sú správne.

1 Súčasný stav problematiky doma a v zahraničí

V tejto časti budú vysvetlené základy teórie evolučných výpočtov. Oboznámime sa aj s terminológiou problematiky, ktorú budeme v ďalších kapitolách ďalej rozoberať.

1.1 Charakteristika základných pojmov

Viaceré algoritmy umelej inteligencie zahŕňajú vytváranie náhodných riešení a ich následnú úpravu, resp. optimalizáciu. Za takéto riešenie môžeme považovať napríklad cestu z Bratislavy do Prešova cez Brno, alebo pohyb robotického manipulátora po zvolenej trajektórii. Optimalizačné algoritmy sa používajú aj pre učenie neurónových sietí. Pri takýchto problémoch vieme my, ľudia, nájsť pomerne rýchlo riešenie, ktoré sa blíži k ideálnemu. Ideál je ale veľmi ťažké dosiahnuť, najmä ak na naše riešenie pôsobia vonkajšie vplyvy.[1]

1.1.1 Optimalizácia

Väčšinu činností vykonávame tak, aby sme nimi dosiahli čo najlepší možný výsledok. Nejde pritom len o ľudské či ekonomické činnosti, ale napríklad o priemyselné zariadenia a stroje. Technologické procesy a programy sú nastavované tak, aby sa nimi po zohľadnení nákladov a rôznych podmienok dosahoval čo najlepší a najvyšší výkon. Optimalizáciou rozumieme všetko, čím získame najlepšie, najvyhovujúcejšie a najvhodnejšie podmienky a riešenia situácií, plánov, programov a podobne. Optimalizáciu môžeme využiť nielen na riešenie súčasných problémov, ale zohľadňuje sa aj pri tvorbe plánov, projektov či stratégií. Z ekonomického hľadiska optimalizáciou rozumieme dosahovanie najvyššieho zisku pri čo najnižších nákladoch. Z matematického hľadiska sa optimalizáciou rozumie aktivita, ktorej cieľom je nájsť najvhodnejšie, teda optimálne riešenie danej matematickej úlohy.

Problematiku optimalizácie radíme do oblasti matematiky a techniky, kde sa optimalizačné úlohy dajú rôzne aplikovať. Tieto úlohy sú známe už z obdobia antiky. V tejto dobe totiž vznikali úlohy o nájdení maxima alebo minima. V súčasnosti existuje veľa špedičných a logistických firiem po celom svete, ktoré v rámci svojho podnikania na dennom poriadku riešia problém optimálnej trasy rozvozu prepravovaného tovaru. Optimálna trajektória je veľmi rozsiahla a známa optimalizačná úloha vo svete biznisu. Preto aj v tejto práci sa v poslednej praktickej

úlohe zameráme aj na túto problematiku a na niekoľkých behoch programu si ukážeme ako pomocou evolučných algoritmov si vieme veľmi ľahko pomôcť a prísť k požadovanému sub-optimálnemu výsledku.[2]

1.1.2 Algoritmus

Algoritmus môžeme definovať ako postup, vďaka ktorému zo zadaných vstupných údajov získame po konečnom počte činností správne výsledky alebo vyriešime problém. Na to, aby sme mohli postup považovať za algoritmus, musí mať nasledujúce vlastnosti:

1. *Elementárnosť* – algoritmus je zložený z činností, resp. krokov, ktoré sú pre realizátora elementárne.
2. *Determinovanosť* – algoritmus musí byť zostavený tak, aby bolo po každom kroku jasné, ktorá činnosť nasleduje, alebo či sa postup už skončil.
3. *Rezultatívnosť* – pre rovnaké vstupné údaje dáva algoritmus vždy rovnaké výsledky.
4. *Konečnosť* – algoritmus skončí vždy po vykonaní konečného počtu krokov a v reálnom čase.
5. *Hromadnosť* – algoritmus je určený na riešenie veľkého počtu úloh rovnakého alebo podobného typu. Dobrý algoritmus by mal pripúšťať premenlivé vstupné údaje, ale existujú aj algoritmy na riešenie jedinej úlohy.
6. *Efektívnosť* – táto vlastnosť je často považovaná za doplnkovú. Efektívnosť algoritmu znamená, že algoritmus získa výsledok v čo najkratšom čase a s využitím čo najmenšieho počtu prostriedkov. Požiadavka získania výsledku v čo najkratšom čase je často v rozpore s požiadavkou minimalizovať použité prostriedky.

Je vhodné navrhovať algoritmus, ktorý je rozdelený na menšie, logicky na seba nadväzujúce celky. Štruktúrovaný algoritmus je ľahšie pochopiteľný a modifikovateľný. Medzi formy zápisu algoritmu patrí zápis v jazyku vývojových diagramov, slovný zápis a zápis štruktúrogramom. *Zápis v jazyku vývojových diagramov* (JVD) používa značky stanovené štátnou normou, graficky vyjadruje tok riadenia a dát. Ide o staršiu formu zápisu algoritmu. *Slovný zápis* používa tzv.

vyhradené slová, napríklad ak-tak-inak, čítaj, začiatok a pod. *Zápis štruktúrogramom* je najnovšia grafická forma zápisu algoritmu.[2]

1.2 Programovací jazyk Python

Python je jeden z najpopulárnejších programovacích jazykov na svete. Okrem neho poznáme aj iné programovacie jazyky, napríklad Java, PHP, C alebo C++. Python je moderný univerzálny open source programovací jazyk, ktorý je chránený autorským právom v rámci licencie kompatibilnej s GPL certifikovanou Open Source Initiative. Programovanie v jazyku Python má jednoduchú a ľahko použiteľnú syntax, vďaka čomu sa radí medzi zaujímavé jazyky vhodné pre začiatočníkov. Svoje uplatnenie však našiel aj vo veľkých spoločnostiach, ako napríklad Wikimedia Foundation, Yahoo! či NASA.

Python je multi-paradigmový programovací jazyk, ktorý je vhodný na numerické výpočty a spracovanie veľkého množstva dát. Zároveň sa nejedná o novú technológiu, nakoľko Python vznikol v roku 1989. Jeho kľúčovou vlastnosťou je schopnosť ľahko ho rozširovať. Samotný jazyk je možné využiť na rozšírenie už existujúcej aplikácie.

Medzi výhody programovania v jazyku Python patrí jeho jednoduchosť, bezplatnosť či produktivita, nakoľko programovanie je veľmi produktívne a pochopenie jeho princípov si nevyžaduje veľa času. V neposlednom rade patrí medzi jeho výhody aj veľká štandardná knižnica Python. S používaním jazyka Python sú spojené aj nevýhody, napríklad pomalší výkon oproti iným jazykom, využíva veľké množstvo pamäte a pri využívaní v oblasti mobilných zariadení nie je pamäťovo efektívny.[3]

Obrázok 1 - Logo programovacieho jazyka Python

1.2.1 História programovacieho jazyka Python

Python je programovací jazyk, ktorého popularita stále rastie. Bol vytvorený v decembri 1989 holandským programátorom Guido van Rossumom, keď začal ako vedľajší osobný projekt tvoriť nový jazyk pod názvom Python, pretože bol fanúšikom Montyho Pythona. Tento názov mu zostal dodnes. Svoj kód publikoval vo februári 1991 na alt.sources a táto verzia bola označená ako verzia 0.9.0. Verzia 1.0 bola vydaná až v januári 1994. Od 1. januára 2020 nie je druhá verzia programovacieho jazyka Python oficiálne podporovaná. Python 2 je v podobe 2.7.18 zmrazený a neplánuje sa vyvíjať ďalej. 3. decembra 2008 bola vydaná verzia s označením 3.0 ako nová verzia jazyka, ktorá nie je kompatibilná s verziami radu 2.x. Oproti verziám radu 2.x je vo veľa aspektoch rovnaký, ale viacero detailov, napríklad slovníky a reťazce alebo to, ako fungujú vstavané objekty sa vo verzií 3.0 zmenilo a veľa zastaraných funkcií bolo zmenených.

Na rozdiel od iných programovacích jazykov, napríklad Pascal, C alebo C++, sa zdrojový kód Pythonu s príponou .py nekompiluje do strojového kódu, ale spúšťa sa interpreterom. Interpreter nevytvára spustiteľný kód a umožňuje interaktívnu spoluprácu s prostredím. Na rozdiel od staticky typovaných jazykov, pri ktorých je potrebné dopredu deklarovať typy všetkých dát, je jazyk Python dynamicky typovaný, teda neexistujú žiadne deklarácie. Python obsahuje črty moderných programovacích jazykov, ako napríklad podporu práce s dátovými štruktúrami či objektovo-orientovanú tvorbu softvéru.[4]

1.3 Darwinova evolučná teória

Pokiaľ ide o evolúciu života, rôzni filozofi a vedci, vrátane anglického lekára z 18. storočia menom Erasmus Darwin, navrhli rôzne aspekty toho, čo sa neskôr stane evolučnou teóriou. Evolúcia však nedosiahla status vedeckej teórie, kým Darwinov vnuk, slávnejší Charles Darwin, nevydal svoju slávnu knihu O pôvode druhov. Darwin a jeho vedecký súčasník Alfred Russel Wallace navrhli, že k evolúcii dochádza v dôsledku javu nazývaného prírodný výber. Táto teória hovorí, že v rámci každej populácie existujú prirodzené variácie v genetickom materiáli jednotlivých jedincov. Niektoré z týchto variácií sa náhodou stanú výhodnejšími pre prežitie a reprodukciu v danom prostredí. Organizmy s týmito výhodnými vlastnosťami teda prežijú a tieto vlastnosti predávajú svojim potomkom. Týmto spôsobom sa postupne zvyšuje zastúpenie výhodných vlastností v populácii. To vedie k tomu, že populácia sa postupne mení, čím vznikajú nové druhy. Táto teória hovorí, že v rámci každej populácie existujú prirodzené variácie v genetickom materiáli jednotlivých jedincov. Niektoré z týchto variácií sa náhodou stanú výhodnejšími pre prežitie a reprodukciu v danom prostredí. Organizmy s týmito výhodnými vlastnosťami teda prežijú a predajú ich ďalej svojim potomkom.

Týmto spôsobom sa postupne zvyšuje zastúpenie výhodných vlastností v populácii. To vedie k tomu, že populácia sa postupne mení, čím vznikajú nové druhy. Biológovia ale odvtedy pozorovali množstvo ďalších príkladov prirodzeného výberu ovplyvňujúceho evolúciu. Dnes vedci vedia, že je to len jeden z niekoľkých viacerých mechanizmov, ktorými sa život rozvíja. Napríklad fenomén taktiež známy ako genetický drift môže spôsobiť aj vývoj jednotlivých druhov. Pri genetickom posune niektoré organizmy – čisto náhodou – produkujú viac potomkov, ako by sa očakávalo. Tieto organizmy nie sú nevyhnutne najvhodnejšie zo svojho druhu, ale sú to ich gény, ktoré sa prenášajú na ďalšiu generáciu.

V konečnom dôsledku Darwinova teória evolúcie ponúka vysvetlenie pre vznik a rozvoj rôznych druhov živých organizmov na Zemi a ako sa prispôsobujú prostrediu. Je to jedna z najvýznamnejších teórií v biológii a má významný vplyv na mnoho ďalších vedných odborov, ako napríklad na medicínu, ekológiu a antropológiu.

Evolučná teória je podľa Charlesa Darwina jediná teória o existencií života. Podľa nej sa vyvinuli život a druhy náhodou postupnými nepatrnými zmenami, ktoré trvali veľmi dlhý čas. Medzi hlavné princípy tejto teórie patrí prirodzený

výber a mutácia. Mutácia v tejto teórií predstavuje rôzne charakteristické znaky jednotlivcov nachádzajúcich sa v populácií. Prirodzený výber predstavuje kríženie len tých jedincov, ktorí sú životaschopnejší, priebojnejší a silnejší ako jedinci s nízkou mierou kvality. Zároveň sú v neustálom boji o prežitie, v ktorom prežijú len niektoré jedince. Do ďalšej generácie sa teda dostávajú informácie len od najsilnejších jedincov.[5; 6; 7]

2 Cieľ práce

Cieľ práce je verifikácia nami navrhnutých testovaní optimalizačných algoritmov založených na prístupe genetickej evolúcie. V počiatočných úlohách tejto práce zvolíme na testovanie nami algoritmov matematické funkcie, ktoré zvolíme podľa cieľa danej úlohy. Ako prvou verifikačnou metódou bude hľadanie minima Schweffelovej funkcie, ktorá sa svojimi mnohopočetnými extrémami hodí na otestovanie tohto algoritmu. V tejto úlohe budeme najmä vyšetrovať schopnosť evolučného algoritmu vyviaznuť z lokálneho extrému.

Túto úlohu budeme ďalej riešiť aj pomocou metódy diferenciálnej evolúcie a porovnáme výsledky s výsledkami riešenia podľa prvej metódy. Ako druhou úlohou bude hľadanie extrému Easomovej funkcie, ktorá verifikuje diverzitu nami navrhnutého genetického algoritmu, nakoľko je táto funkcia na väčšinovej časti konštantná a algoritmus s vysokým selektívnym tlakom a nízkou diverzitou môže mať problém s hľadaním extrému takejto funkcie. V tretej praktickej časti využijeme poznatky získané pri riešení prvej a druhej praktickej úlohy, kde sa zameriame na pomerne komplexný problém hľadania optimálnej trajektórie. Úlohy daného charakteru majú pomerne vysokú citlivosť na optimalizačné parametre a aj napriek vhodne zvoleným parametrom optimalizácie môže genetický algoritmus častokrát uviaznuť v lokálnom extréme.

Všetky úlohy budeme riešiť v programovacom jazyku Python a jeho známej knižnici PyGad, ale v prvej úlohe minimalizácie Schweffelovej funkcie si otestujeme aj knižnicu Scipy a jej metódu diferenciálnej evolúcie.

Všetky výsledky jednotlivých behov programu si na konci zhodnotíme porovnáme a podrobnejšie popíšeme.

3 Metodika práce a metódy skúmania

3.1 Dátové typy v jazyku Python

Python je dynamicky typovaný jazyk, premenné teda nemusíme deklarovať s ich dátovým typom. Vďaka tomu je práca v ňom jednoduchšia. Python podporuje 3 základné dátové typy, a to celé čísla (int), desatinné čísla (float) a textový reťazec (str). Na rozdiel od textových reťazcov, ktoré sa vyskytujú vo väčšine programovacích jazykov, sú textové reťazce v jazyku Python nemenným typom. To znamená, že operácie, ktoré by inak menili reťazec (napríklad zámena znakov), vracajú namiesto toho nový reťazec.

Jednou zo základných čŕt jazyka Python je koncept kolekčných, resp. kontajnerových typov. Kolekcia je vo všeobecnosti objekt, ktorý obsahuje iné objekty, ku ktorým je možné pristupovať pomocou indexov alebo kľúčov. Medzi dve základné formy kolekcií patria mapované typy a sekvenčné typy.

Mapované typy charakterizujeme ako nezoradené implementované typy v podobe asociatívneho poľa, ktoré mapujú, podobne ako matematické funkcie, množinu objektov alebo kľúčov na elementy v množine hodnôt. Iným typom kolekcií sú zoradené postupnosti, sekvenčné typy. Patria medzi ne zoznamy, tuple a textové reťazce. Všetky sekvenčné typy sú pozične indexované, teda od indexu 0 po dĺžku postupnosti -1, a taktiež všetky okrem textových reťazcov môžu obsahovať objekty ľubovoľného typu. Textové reťazce môžu obsahovať iba znaky, ktoré sú v Pythone reprezentované ako jednoznakové reťazce. Tuple a reťazce sú nemenné. Hodnotu zoznamov je ale možné meniť, teda je možné pridávať, meniť alebo odoberať elementy jednotlivých zoznamov. [8]

3.2 Knižnice jazyka Python

V obyčajnom svete, knižnica je miesto, kde sú uložené knihy pre budúce použitie. Rovnako v svete programovania, knižnica je zbierka kódu, ktorý môže byť neskôr použitý v programe na určité, presne definované operácie. Jazyk Python má rozsiahly ekosystém knižníc, ktoré umožňujú vývojárom rýchlo a efektívne riešiť rôzne úlohy bez nutnosti od základov vyvíjať vlastný kód. Knižnice sú súbory kódu, ktoré môžu byť importované a použité vo vašom projekte, aby ste zjednodušili alebo rozšírili funkcionalitu jazyka Python.

Knižnice sú dostupné pre rôzne úlohy, ako je spracovanie dát, strojové učenie, webový vývoj, vizuálne a interaktívne aplikácie, atď. Môžu obsahovať funkcie, triedy a objekty, ktoré môžu byť použité vo vašom projekte, ako aj dokumentáciu a príklady použitia.

Výber správnej knižnice pre konkrétny projekt závisí od požiadaviek a cieľov projektu. Je dôležité vyhľadať knižnice s aktívnou komunitou a dobrým zázemím pre podporu a aktualizácie. Zdroje, ako sú PyPI (Python Package Index) a Anaconda, umožňujú ľahko vyhľadávať a nainštalovať knižnice pre jazyk Python.

3.2.1 Ako operovať s knižnicami

Štandardná knižnica Pythonu obsahuje presnú syntax, semantiku a symboly. Obsahuje taktiež vstavané moduly, ktoré poskytujú prístup k základnej funkcionalite systému ako vstup alebo výstup a niektoré iné hlavné moduly. Veľa knižníc v Pythone je napísaných v programovacom jazyku C, nakoľko Python ponúka široké možnosti vzájomnej kompatibility. Všetky tieto moduly spolu pracujú na tom, aby Python bol vysokoúrovňovým programovacím jazykom. Štandardná knižnica je veľmi dôležitá a bez nej programátori nemôžu mať prístup k funkcionalitám Pythona. Avšak okrem toho existuje niekoľko ďalších knižníc v Pythone, ktoré uľahčujú život programátora. Priblížme si niekoľko kližníc spolu s tou ktorú budeme používať v tejto práci:

1. ***Numpy:*** Názov „Numpy“ (Numeric Python). Je to bežne používaná knižnica. Hovoríme o populárnej knižnici ktorá podporuje veľké matice a viacrozmerné údaje. Je vyskladaná zo vstavaných matematických funkcií pre numerické výpočty. Dokonca aj knižnice ako TensorFlow (používa sa v algoritmoch strojového učenia) sú vzájomne kompatibilné s touto knižnicou. Rozhranie poľa je jednou z primárnych funkcií tejto knižnice.
2. ***SciPy:*** Názov „SciPy“ znamená „Scientific Python“. Táto knižnica má otvorený zdrojový kód a používa sa na vedecké výpočty a operácie. Kooperuje s Numpy na zvládnutie zložitých výpočtov. Zatiaľ čo Numpy umožňuje triedenie a indexovanie dát poľa, SciPy obsahuje konkrétne implementácie matematických algoritmov, vrátane preddefinovaných funkcií na riešenie optimalizačných problémov, spracovania digitálnych signálov a podobne. Takisto je veľmi obľúbená medzi vývojármi aplikácií a inžiniermi.

Obrázok 2 - Logo knižnice Scipy

3. ***PyGame:*** Vytvára jednoduché rozhranie pre grafické, zvukové a vstupné knižnice nezávislé od platformy Standard Directmedia Library (SDL). Využíva sa na vývoj počítačových videohier pomocou grafiky a zvukových knižníc spolu s programovacím jazykom Python.
4. ***PyTorch:*** Je to jedna z najrozsiahlejších knižníc strojového učenia, ktorá pomocou špecifických výpočtov optimalizuje výkon umelých neurónových sietí. Má bohaté rozhrania API na vykonávanie výpočtov s orientáciou na tenzorové dátové štruktúry a taktiež vo vysokej miere podporuje GPU akceleráciu . Pomáha tiež riešiť aplikačné problémy súvisiace s neurónovými sieťami.
5. ***Pandas:*** Pandas je knižnica ktorá uľahčuje prácu s veľkými dátovými súbormi. Dátové štruktúry (polia, matice, tabuľky) predstavujú počas behu programu istú formu databázy (často dataframe), na ktorej je možné vykonávať rôzne príkazy na zoraďovanie, čistenie a manipulovanie datasetu.
6. ***PyGAD:*** Názov „PyGad“ znamená Python Genetic Algorithm. Je to ďalšia z open-source knižníc a používa sa na zostavenie genetických algoritmov a optimalizácie strojového učenia algoritmov. Spolupracuje s knižnicami Keras a PyTorch. PyGAD podporuje rôzne typy zmien génov medz reťazcami, tzv. kríženia, alebo mutácie a rodičovského výberu. PyGAD umožňuje optimalizovať rôzne typy problémov pomocou genetického algoritmu prispôsobením funkcie fitness. Okrem vytvárania genetického algoritmu, vytvára a optimalizuje algoritmy strojového učenia. V súčasnosti PyGAD masívne podporuje vytváranie a trénovanie (pomocou genetického algoritmu) umelých neurónových sietí pre klasifikačné problémy. Knižnica sa aktívne vyvíja a pravidelne sa pridávajú ďalšie funkcie.[9]

Obrázok 3 - Logo knižnice PyGad vytvorené Asmaaom Kabilom

3.3 Evolučné algoritmy

3.3.1 História evolučných algoritmov

Matematici, biológovia a neskôr aj informatici inšpirovaní evolučným procesom si začínali pokladať čoraz častejšie otázku, či by bolo tento mechanizmus možné použiť na riešenie nebiologických problémov. Výsledky ich pokusov sa dostavili až s príchodom výkonnej výpočtovej techniky. Bez nej by totiž nebolo možné simulovať tisíce alebo milióny evolučných cyklov, ktoré prebiehali v prírode. V druhej polovici dvadsiateho storočia boli na rôznych pracoviskách vo svete podľa vzoru prirodzenej evolúcie vytvorené viaceré prístupy.

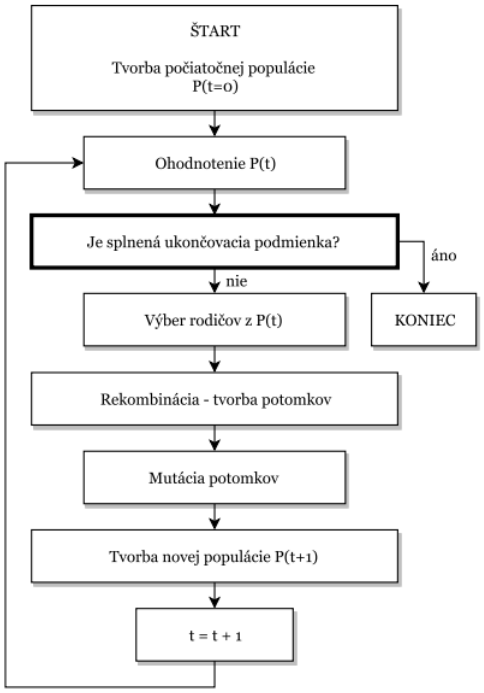
Pri optimalizácií konštrukčných úloh boli v Nemecku, v polovici šesťdesiatych rokov, vyvíjané tzv. evolučné stratégie I. Rechenbergom a H.P.Schwefelom. V tom období vyvíjal v USA pri modelovaní a návrhu automatov Lawrence Fogel techniku s názvom „evolučné programovanie“. Za vznik genetických algoritmov používaných v dnešnej dobe najmä pri optimalizácií sa považujú práce skupiny vedenej Johnom Hollandom. Ten pôsobil v sedemdesiatych rokoch dvadsiateho storočia ako profesor na Michiganskej Univerzite v USA. Autorom historicky mladšieho „genetického programovania“ je John Koza. Genetické programovanie je evolučný prístup, ktorý je určený najmä na optimalizáciu štruktúr, symbolickú regresiu, optimalizáciu parametrov a podobne. V súčasnej dobe všetky takéto smery nazývame evolučné algoritmy.[10]

3.3.2 Súčasná charakteristika evolučných výpočtov

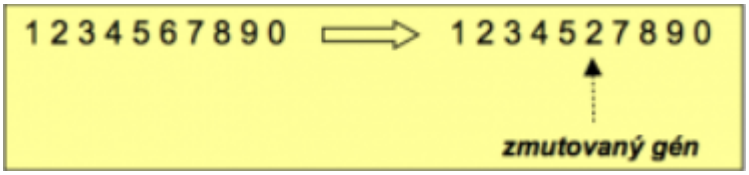
V súčasnosti patria evolučné algoritmy medzi heuristické prehľadávacie algoritmy, ktoré sa používajú na riešenie rôznych typov optimalizačných úloh, napríklad na riešenie rôznych zložitých problémov v oblasti informačných systémov. Môžeme ich zaradiť nie len medzi základné prostriedky numerickej matematiky, ale aj biológie a informatiky, kde majú svoj pôvod. Medzi ich hlavnú charakteristiku patrí práca s populáciou a využívanie evolučných operátorov – selekcie, náhrady, mutácie či kríženia. Napriek tomu, že sa tento typ algoritmu používa čoraz častejšie na riešenie bežných praktických úloh aj zložitejších optimalizačných úloh, je neustálym predmetom skúmania.

Evolučné výpočty chápeme ako podoblasť umelej inteligencie. Vo veľkej miere sa používajú pri optimalizačných úlohách alebo na riešenie problémov, ktoré majú príliš veľa premenných pre použitie tradičných algoritmov. Výpočtové modely, ktoré využívajú evolučné algoritmy, aplikujú evolučné procesy na riešenie zložitých problémov. Tieto evolučné procesy sú inšpirované teóriou biologickej evolúcie. Evolučné algoritmy využívajú princípy, akými sú napríklad dedenie z predchádzajúcich úspešných generácií a prirodzený výber, kde najlepšie riešenia odovzdávajú svoje vlastnosti nasledujúcim generáciám. O evolučné algoritmy multikriteriálnej optimalizácie rastie v poslednej dobe záujem, nakoľko kombinujú dve hlavné disciplíny: teoretické rámce rozhodovania na základe viacerých kritérií a evolučné výpočty.

Definovanie viacerých cieľov pomáha získať lepšiu predstavu o úlohe. Na rozdiel od množstva dostupných techník na optimalizáciu s jedným cieľom bolo vyvinutých pomerne málo techník na optimalizáciu s viacerými cieľmi. Pri optimalizácií s jedným cieľom je priestor vyhľadávania veľmi často dobre definovaný. Ak existuje niekoľko možných protichodných cieľov, ktoré je potreba optimalizovať súčasne, už neexistuje jediné optimálne riešenie, ale celý súbor možných riešení rovnakej kvality. Keď chceme optimalizovať niekoľko cieľov súčasne, priestor vyhľadávania sa taktiež čiastočne usporiada. Na dosiahnutie optimálneho riešenia teda bude existovať súbor optimálnych kompromisov medzi protichodnými cieľmi.[11]



Obrázok 4 - Všeobecná schéma genetického algoritmu



Obrázok 5 - Príklad mutácie celočíselného reťazca

3.4 Technika genetických algoritmov

Ako sme už spomínali v predchádzajúcej podkapitole, genetické algoritmy patria do triedy evolučných algoritmov. Používajú techniky, ktoré napodobňujú evolučné procesy z biológie (dedičnosť, kríženie, prirodzený výber a mutáciu) na nájdenie najlepšieho riešenia. Genetický algoritmus chápeme ako heuristický postup, ktorý sa aplikáciou spomínaných princípov snaží nájsť riešenie zložitých problémov, pre ktoré nevieme použiť exaktný algoritmus. Existuje mnoho aspektov, ktoré sa dajú odlišne implementovať v závislosti od riešeného problému, napríklad typ kódovania, stratégia výberu rodičov, reprezentácia chromozómu či typ kríženia a mutácie. Vychádzajú z Darwinovej teórie vývoja a Mendelových zákonov dedičnosti.

Genetické alebo evolučné algoritmy možno využiť na riešenie širokého spektra úloh. Podmienkou je schopnosť sformulovať kriteriálnu, resp. účelovú funkciu, ktorá sa má maximalizovať alebo minimalizovať. Pri maximalizácií sa jedná napríklad o maximalizáciu účinnosti, výkonu či zisku. Pri minimalizácií ide napríklad o minimalizáciu spotreby energie, paliva alebo nákladov. Medzi ďalšiu podmienku použitia evolučných algoritmov patrí počítačová reprezentácia problému, ktorý chceme optimalizovať. Znamená to, že musí existovať matematický model alebo počítačová simulácia deja, pre ktorý hľadáme optimálne riešenie. Vďaka týmto kritériám dokážeme vyčísliť hodnotu účelovej funkcie pre ľubovoľný bod v prehľadávanom priestore a z hľadiska dosiahnutia požadovaného cieľa a z hľadiska úspešnosti ho vieme ohodnotiť. Pomocou evolučných algoritmov teda vieme nájsť určitú množinu, ktorá najlepšie spĺňa vopred stanovené požiadavky.

Existuje veľa príkladov využitia evolučných algoritmov, no pre lepšie pochopenie si môžeme spomenúť napríklad inžinierske aplikácie. Evolučné algoritmy môžu byť silným nástrojom pri optimalizácií elektrických obvodov alebo optimalizácií prevádzky elektrizačnej sústavy. Môžeme si uviesť aj príklad z praxe, kedy boli pri optimalizácií motorov Boeingu 777 použité evolučné algoritmy. Pomocou malej konštrukčnej úpravy sa vtedy podarilo usporiť až 2,5% paliva. V tej dobe to bol významný posun vpred, keďže po prepočte tejto úspory na jeden rok to predstavovalo sumu 2 milióny dolárov. Evolučné algoritmy sa používajú aj v stavebníctve, najmä na optimalizáciu konštrukcie budov alebo cestných komunikácií. V neposlednej rade majú veľký prínos aj v oblasti ekonomických problémov pri riešení zložitých úloh s mnohými obmedzeniami.[10; 12]

3.4.1 Diferenciálna evolúcia

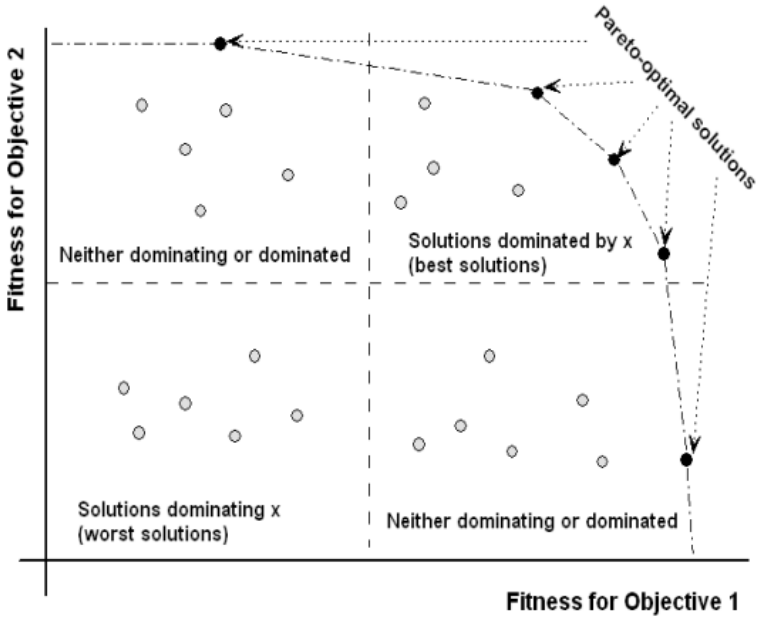
Diferenciálna evolúcia (DE) je metóda paralelného priameho vyhľadávania, ktorá využíva vektory veľkosti populácie D-rozmerných parametrov ako populáciu pre každú generáciu G. Veľkosť populácie sa počas procesu minimalizácie nemení. Počiatočný vektor populácie je vybraný náhodne a mal by pokrývať celú množinu prípustných riešení. Stochastická zložka DE sa prejavuje v generovaní náhodných čísel a náhodných zmenách parametrov, ktoré ovplyvňujú výsledné riešenia. Tieto náhodné činitele sú dôležité pre dosiahnutie rozmanitosti v populácii kandidátov a pre prekonanie lokálnych extrémov. V prípade, že je k dispozícii predbežné

riešenie, počiatočný súbor možno vygenerovať pridaním normálne rozdelených náhodných odchýlok k nominálnemu riešeniu xnom;0. DE generuje nové vektory parametrov pridaním váženého rozdielu medzi dvoma vektormi populácie k tretiemu vektoru. Nech sa táto operácia nazýva mutácia. Parametre mutovaného vektora sa potom zmiešajú s parametrami iného vopred určeného vektora, cieľového vektora, čím sa získa takzvaný skúšobný vektor. Miešanie parametrov sa v komunite evolučných stratégii často označuje ako „crossover“ alebo tiež kríženie. Ak skúšobný vektor poskytne nižšiu nákladovú funkčnú hodnotu ako cieľový vektor, skúšobný vektor nahradí cieľový vektor v nasledujúcej generácii. Táto posledná operácia sa nazýva výber. Každý populačný vektor musí slúžiť raz ako cieľový vektor, aby sa súťaže NP uskutočnili v jednej generácii.[13]

3.4.2 Paretovo optimum

Optimalizácia s viacerými cieľmi by sa dala napísať vo forme minimalizovať $[f_1(x), f_2(x),..., f_k(x)]$, pre k účelových funkcií $f_i: R^n \rightarrow R$ podliehajúce niekoľkým obmedzeniam rovnosti a nerovnosti. Pre $x = [x_1, x_2, \ldots, x_n]^T$, vektor rozhodovacích premenných, určíme množinu F všetkých vektorov, ktoré spĺňajú všetky obmedzenia, konkrétne množinu hodnôt $[x_{1*}, x_{2*}..., x_{n*}]$ a taktiež zistíme optimálne hodnoty pre všetky účelové funkcie.

Ako môžeme vidieť na *obrázku 5*, riešenie môže byť najlepšie, najhoršie a tiež neutrálne k iným riešeniam vzhľadom na cieľové hodnoty. Najlepšie riešenie je teda riešenie, ktoré nie je najhoršie v žiadnom z cieľov a zároveň je aspoň lepšie v jednom cieli ako v druhom. Optimálne riešenie je riešenie, ktorému v priestore vyhľadávania nedominuje žiadne iné riešenie. Takéto optimálne riešenie sa nazýva Paretovo optimum.[14]



Obrázok 6 - Paretovo optimum

3.4.3 Tvorba počiatočnej populácie

Tvorbu počiatočnej populácie možno realizovať dvomi známymi spôsobmi:

- Náhodne - Naplníme počiatočnú populáciu úplne náhodnými riešeniami.
- Heuristicky - Naplníme počiatočnú populáciu pomocou známej heuristiky problému.

Vo väčšine prípadov sa používa náhodne generovaná populácia, čo zaisťuje veľa možných riešení. [15]

3.4.4 Fitness funkcia

Fitness funkcia, alebo tiež známa ako hodnotiaca funkcia vyhodnocuje, ako blízko je dané riešenie optimálnemu riešeniu požadovaného problému. Určuje, nakoľko je riešenie vhodné. V genetických algoritmoch je každé riešenie vo všeobecnosti reprezentované ako reťazec binárnych čísel, známy ako chromozóm. Musíme tieto riešenia otestovať a prísť s najlepším súborom riešení na vyriešenie daného problému. Každé riešenie preto musí byť ocenené bodovým ohodnotením, ktoré naznačuje, ako blízko sme k splneniu celkovej špecifikácie požadovaného riešenia. Toto skóre sa generuje aplikáciou fitness funkcie na test alebo výsledkov

získaných z testovaného riešenia. Napríklad funkcia fitness, ktorá je nulová, pokiaľ odpoveď nie je správna, nie je dobrá, pretože nám nepomôže získať predstavu o tom, ako blízko je riešenie k správnej odpovedi. Tiež funkcia fitness, ktorá sa zvyšuje, keď sa riešenia zlepšujú, ale neidentifikuje najlepšie riešenie, tiež nie je taká dobrá, pretože naša populácia sa do určitého bodu zlepší a potom sa zasekne.

Každá fitness funkcia by mala spĺňať tieto parametre:

- Funkcia fitness by mala byť jasne definovaná. Čitateľ by mal byť schopný jasne pochopiť, ako sa vypočítava skóre fitness.
- Fitnes funkcia by mala byť navrhnutá efektívne a exaktne. Zle navrhnutá fitnes funkcia môže znížiť schopnosť algoritmu konvergovať k optimu funkcie, pripadne môže tento proces znemožniť.
- Funkcia fitness by mala kvantitatívne merať, ako je dané riešenie vhodné pri riešení problému.
- Fitness funkcia by mala generovať intuitívne výsledky. Najlepší/najhorší kandidáti by mali mať najlepšie/najhoršie hodnoty skóre.

3.4.5 Ako vytvoriť fitness funkciu

Každý problém má svoju vlastnú fitness funkciu. Vymyslieť fitness funkciu pre daný problém je najťažšia časť pri formulovaní problému pomocou genetických algoritmov. Neexistuje žiadne pevné pravidlo, že v konkrétnom probléme by sa mala použiť konkrétna funkcia. Vedci údajov však prijali určité funkcie týkajúce sa určitých typov problémov. Typicky sa pre klasifikačné úlohy, kde sa používa učenie pod dohľadom, vo veľkej miere používajú chybové merania, ako je euklidovská vzdialenosť a vzdialenosť na Manhattane. Pri optimalizačných problémoch možno ako fitness funkciu použiť základné funkcie, ako je súčet množiny vypočítaných parametrov súvisiacich s doménou problému.[16]

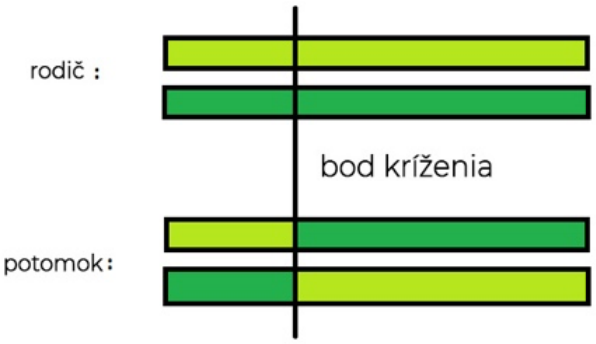
3.4.6 Kríženie

Je genetický operátor, tiež nazývaný rekombinácia, používaný na zmenu programovania chromozómu alebo chromozómov z jednej generácie na druhú. Kríženie je sexuálna reprodukcia. Z párenia sa náhodne vyberú dve struny, aby sa

prekrížili, aby sa splodili vynikajúce potomstvo. Zvolená metóda závisí od metódy kódovania.

Poznáme tri typy kríženia:

- ***Jednobodové kríženie***: Vyberie sa bod kríženia na reťazci rodičovského organizmu. Všetky údaje za týmto bodom v reťazci organizmov sú vymenené medzi dvoma rodičovskými organizmami.

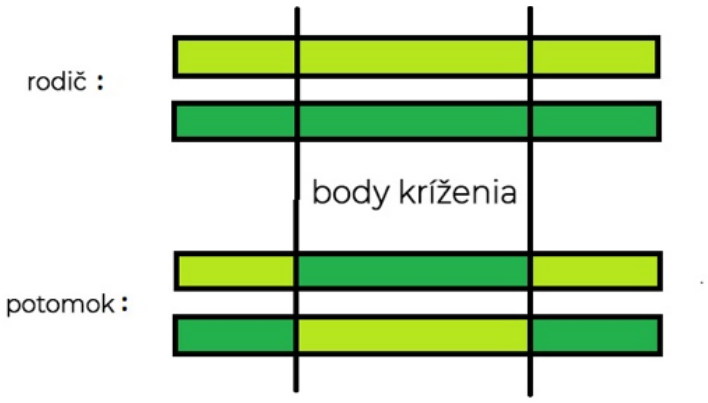


Chromozóm 1	11011|00100110110
Chromozóm 2	11011|11000011110
Potomok 1	11011|11000011110
Potomok 2	11011|00100110110

Jednobodové kríženie

Obrázok 7 - Jednobodové kríženie

- ***Viacbodové kríženie:*** Vyznačuje sa viacerými bodmi kríženia, najmenej však dvoma. Potomok sa vytvorí tak, že všetko od začiatku po prvý bod kríženia je skopírované od prvého rodiča, ďalej všetko od prvého miesta kríženia po druhé miesto je skopírované od druhého rodiča a všetko od druhého miesta kríženia až po koniec alebo ďalšie miesto kríženia je znovu skopírované od prvého rodiča a tak ďalej. Najviac krížení môže byť pre *n* génov *n -1*

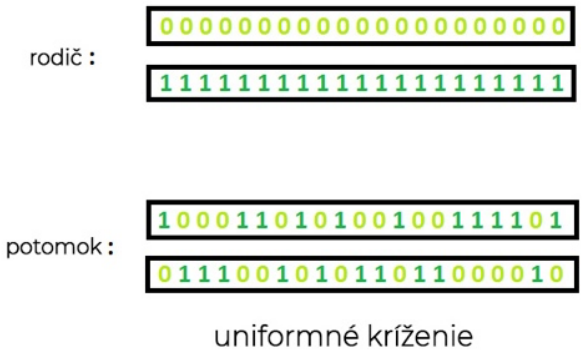


Chromozóm 1	11011|00100|110110
Chromozóm 2	10101|11000|011110
Potomok 1	11011|11000|110110
Potomok 2	10101|00100|011110

Dvojbodové kríženie

Obrázok 8 - Dvojbodové kríženie

- ***Uniformné kríženie*** - Každý gén je vybraný náhodne z jedného zo zodpovedajúcich génov rodičovských chromozómov. Výber sa uskutočňuje s určitou pevne stanovenou pravdepobnosťou. Jej hodnota sa väčšinou rovná 0.5. To znamená, že existuje 50% pravdepodobnosťou že gén sa prenesie z prvého rodiča a taktiež 50% že sa prenesie z druhého. [17]



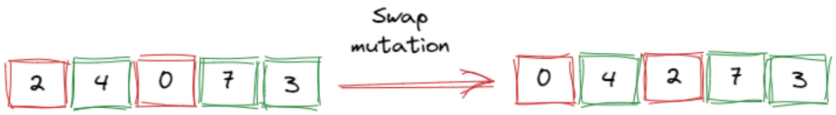
Obrázok 9 - Uniformné kríženie

3.4.7 Mutácia

Využíva sa na to aby riešenie neuviazlo v lokálnom optime daného problému. Zväčšuje prehľadávaný priestor o riešenia, ktoré nemožno dosiahnuť krížením. Mutácia sa myslí to, že niektoré bity v bitovom reťazci možno prevrátiť na opačnú hodnotu daného bitu. Jedná sa o náhodnú genetickú zmenu práve vytvoreného potomka. Každý nový jedinec však nemusí mutovať, je daná hodnota, ktorá určuje koľko percent potomkov je podrobených mutácií.

3.4.8 Zmiešavacia mutácia

Takáto mutácia sa vykonáva, keď kódujeme chromozómy ako permutácie daného súboru prvkov. Pri zmiešavacej mutácii sú časti dvoch náhodne vybraných génov zamenené, ako je znázornené na nasledujúcom diagrame. Tento typ mutácie budeme ďalej využívať aj v praktickej časti tejto bakalárskej práce v rámci tretej praktickej úlohy optimálnej trajektórie.[18]



Obrázok 10 - Vizualizácia zmiešavacej mutácie

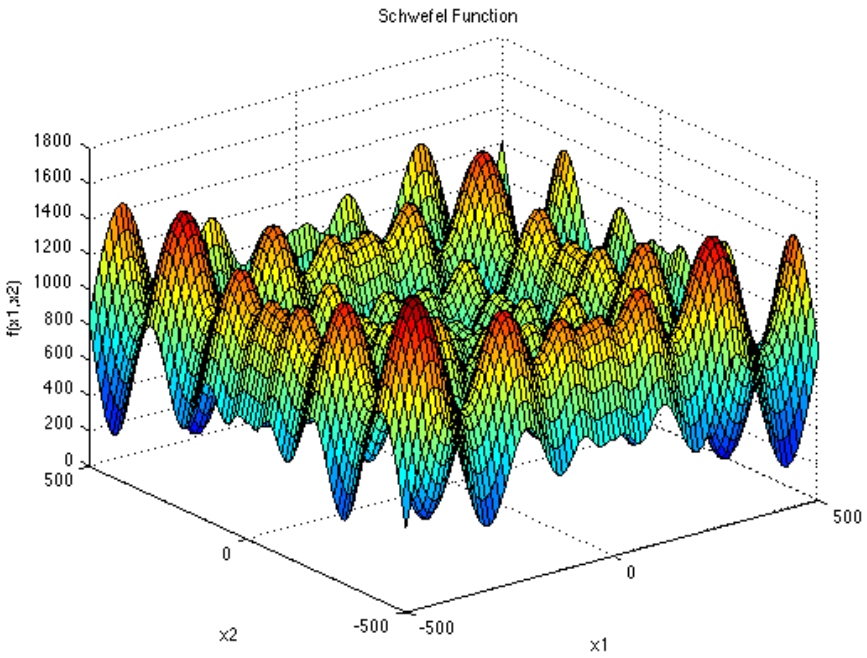
3.4.9 Selekcia

Alebo tiež proces výberu rodičov, ktorí sa pária a rekombinujú, aby vytvorili potomstvo pre ďalšiu generáciu. Výber rodičov je veľmi dôležitý pre mieru konvergencie genetického algoritmu, pretože dobrí rodičia vedú jednotlivcov k lepším a vhodnejším riešeniam. Malo by sa však zabrániť tomu, aby jedno mimoriadne vhodné riešenie prevzalo celú populáciu v priebehu niekoľkých generácií, pretože to vedie k tomu, že riešenia sú v priestore riešení blízko seba, čo vedie k strate rozmanitosti. Udržanie dobrej diverzity v populácii je mimoriadne dôležité pre úspech genetického algoritmu. Toto zaberanie celej populácie jedným extrémne vhodným riešením je známe ako predčasná konvergencia a je nežiaducim stavom v genetickom algoritme.

4 Výsledky práce

Pri každej zvolenej úlohe opíšeme parametre a ostatné faktory, ktoré majú markantný vplyv na proces optimalizácie. Taktiež pri každej optimalizačnej úlohe spustíme proces optimalizácie päťkrát, kvôli overeniu konzistencie nami navrhnutého genetického algoritmu. S týmito parametrami budeme taktiež manipulovať, za cieľom demonštrácie vplyvu jednotlivých parametrov na samotný optimalizačný proces.

4.1 Minimalizácia Schwefellovej funkcie



Obrázok 11 - Vizualizácia Schweffelovej funkcie dvoch premenných

Ako prvou praktickou úlohou je minimalizácia Schwefellovej funkcie, ktorej priebeh v intervale <-500;500> pre dve premenné môžeme vidieť na *obrázku 11*. Túto funkciu budeme minimalizovať pre päť premenných, kde minimálna funkčná hodnota 0 je pre každú premennú v bode 420.97.

Schweffelovu funkciu opisuje nasledovná rovnica:

$$f(x) = 418.9828d - \sum_{i=1}^{d} x_i \sin\left(\sqrt{(|(x_i)|)}\right)$$

4.1.1 Optimalizačné parametre

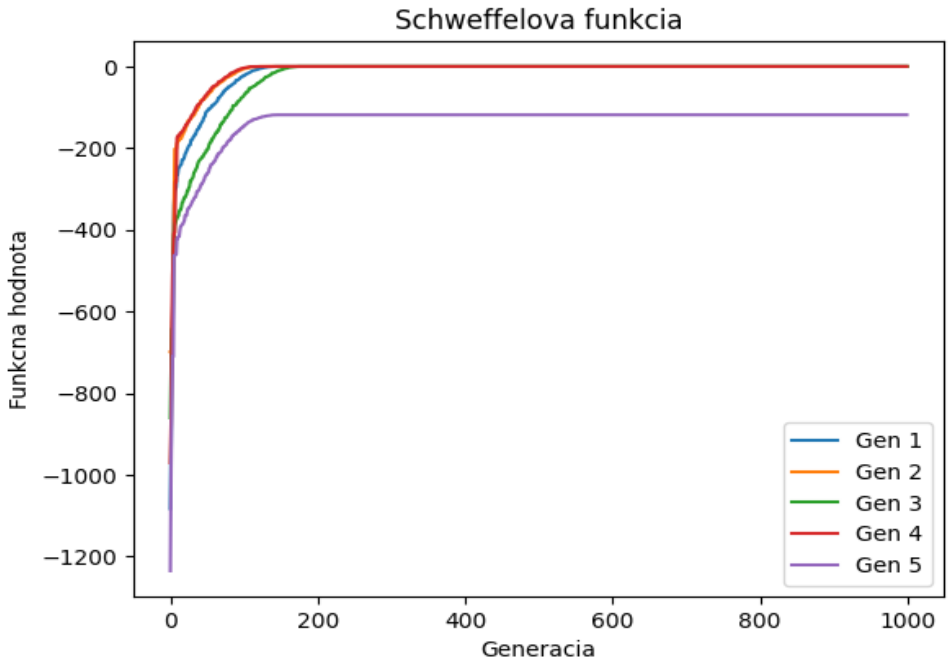
V tejto praktickej úlohe bolo cieľom nájsť vhodné parametre mutácie, nakoľko evolučný algoritmus má pri minimalizácii Schweffelovej funkcie tendenciu uviaznuť v lokálnom extréme a to najmä z dôvodu vysokého počtu lokálnych extrémov. Spôsob mutácie sme volili ako náhodný z ohraničenia definičného oboru v intervale <-500;500> s pravdepodobnosťou mutácie génu o hodnote 15 %.

Populáciu tvorilo 50 chromozómov, z čoho päť chromozómov nepodliehalo žiadnym modifikačným genetickým operáciam. Týchto päť chromozómov bolo presunutých do novej populácie za účelom zachovania genetickej informácie. Spôsob výberu jedincov bol turnajový výber, ktorý z populácie náhodne vyberie N chromozómov, kde N je parameter (v našom prípade rovný 5). Vybraným jedincom sa následne porovnávajú funkčné hodnoty a najlepší z tohto výberu putuje do novej populácie. Tento cyklus sa opakuje dokým nie je naplnený predpísaný počet jedincov v populácii.

Ako krížiacu metódu sme volili jednobodové kríženie, kde pravdepodobnosť kríženia dvoch susedných reťazcov nastávala s pravdepodobnosťou 15%.

4.1.2 Priebeh optimalizácie

Celý optimalizačný proces trval po dobu tisíc generácii, kde počet generácií je zároveň ukončovacou podmienkou nášho algoritmu.



Obrázok 12 - Priebeh minimalizácie Schweffelovej funkcie

Na *obrázku 12* môžeme vidieť priebeh minimalizácie Schweffelovej funkcie. Globálne minimum sa nám podarilo dosiahnúť vo všetkých generáciách, okrem piatej generácie, kedy sme odstránili operáciu mutácie, aby sme demonštrovali dôležitosť tejto genetickej operácie. V generácii č. 4 sme vypli operáciu kríženia, no nami navrhnutý genetický algoritmus sa dokázal s touto modifikáciou vysporiadať, no táto modifikácia mala vplyv na dobu ustálenia hodnoty fitness na globálne minimum.

Toto je však riešenie pomocou knižnice PyGad a jej fukncií, Schweffelova funkcia sa však dá riešiť aj pomocou evolučného algoritmu s názvom diferenciálna evolúcia, ktorý sa nachádza v knižnici Scipy. Scipy sa zväčša využíva pre vedecké výpočty a zložitejšie matematické operácie. Funkcia diferenciálnej evolúcie je globálna funkcia získaná balíkom optimalizácie SciPy, ktorá sa používa na nájdenie globálneho minima funkcií s viacerými premennými. Táto funkcia funguje na skutočných hodnotách. Naše skutočné hodnoty pre túto funkciu je veľkosť riešenej populácie 50 a počte generácii 1000.

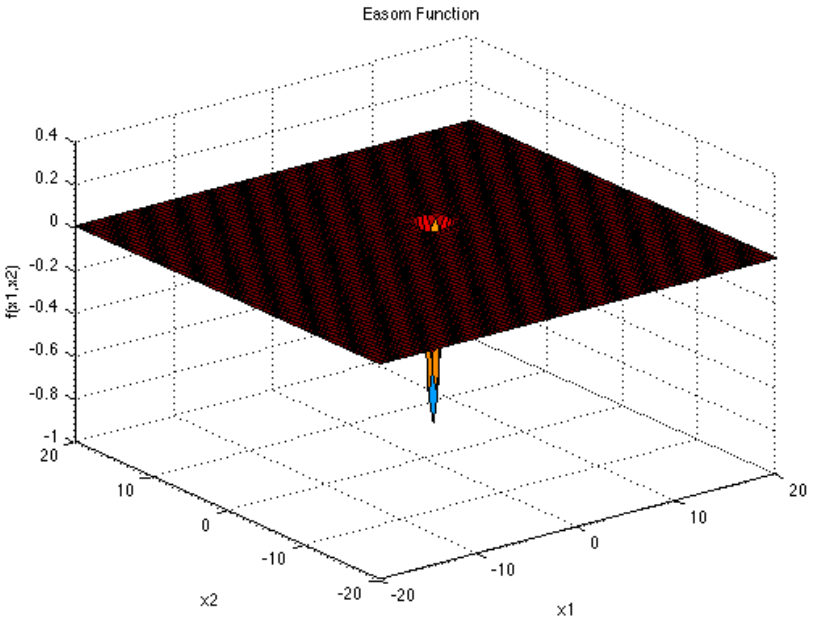
```
bounds = [(-500, 500)]*5
popsize = 50
max_gen= 1000

result = differential_evolution(
    schwefel,
    bounds,
    popsize=popsize,
    maxiter=max_gen,
    strategy='best1bin',
    tol=1e-6,
    mutation=(0.5, 1),
    recombination=0.7,
    seed=None,
    updating='immediate'
)
print("Optimized parameters:", result.x)
print("Objective function value:", result.fun)
```

```
Optimized parameters: [420.96874608 420.96874779 420.96874795 420.96874756 420.9687467 ]
Objective function value: 6.363783222695929e-05
```

Obrázok 13 - Ilustračný náhľad kódu diferenciálnej evolúcie

Ako vidíme na *obrázku 13 optimized parameters* nám vrátila táto funkcia velmi podobné hodnoty ako pri predchádzajúcej metóde pomocou knižnice PyGad. Hodnota *Objective function value* taktiež vyšla podobná ako hodnota funkcie fitness v predchádzajúcom riešení úlohy.

4.2 Minimalizácia Easomovej funkcie



Obrázok 14 - Vizualizácia Easomovej funkcie s lokálnym extrémom

Ďalšou praktickou úlohou je minimalizácia Easomovej funkcie, ktorej priebeh pre dve premenné v rozsahu <-20; 20> môžeme vidieť na *obrázku 14*. Táto funkcia je štandardne minimalizovaná v rozsahu definičného oboru

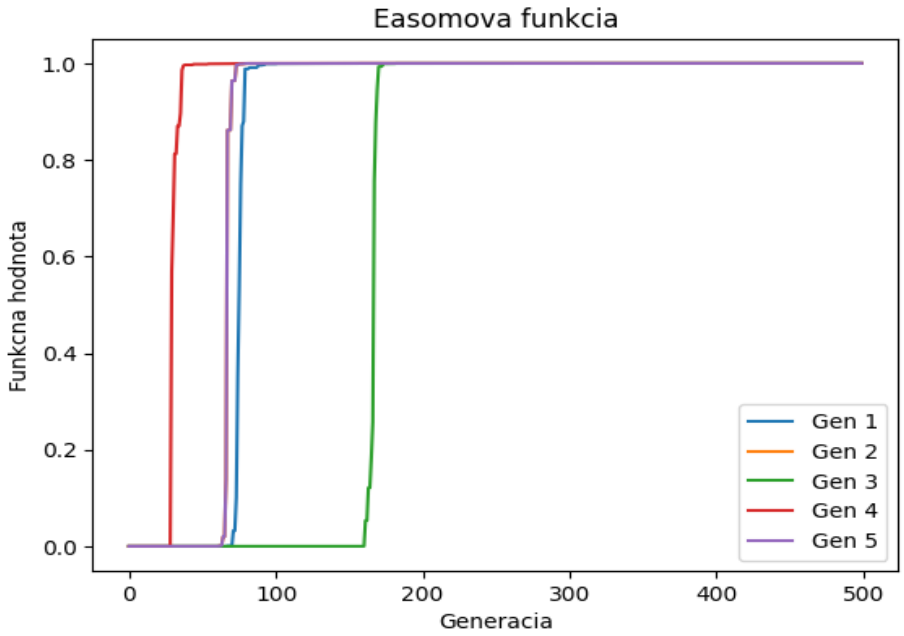
<-100;100>, kde pre dve premenné dosahuje táto funkcia minimálnu hodnotu -1 v bodoch $x_1 = \pi; x_2 = \pi$.

Easomova funkcia je daná predpisom:

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

Pri tejto funkcii budeme skúmať schopnosť nami navrhnutého algoritmu efektívne prehľadávať priestor ohraničenia riešenia. Z dôvodu nulovej derivácie funkcie na väčšinovej časti definičného oboru nemá genetický algoritmus dostatočné informácie o tom, ako dobre sa blíži k minimu funkcie. Z toho dôvodu je v tomto prípade kladený zvýšený dôraz na diverzitu populácie, za cenu nižšieho selektívneho tlaku. Takto nastavený genetický algoritmus zabezpečí široké rozdelenie génov jednotlivých chromozómov po celej ploche definičného oboru funkcie, čo zvyšuje pravdepodobnosť výskytu génov v okolí minima funkcie.

4.2.1 Priebeh optimalizácie



Obrázok 15 - Priebeh optimalizácie Easomovej funkcie

Na *obrázku 15* ktorý reprezentuje priebeh minimalizácie Easomovej funkcie v piatich behoch môžeme pozorovať tendenciu optimalizačného algoritmu na uviaznutie v lokálnom extréme, konkrétne na jednosmernej časti funkcie v hodnote 0. Taktiež môžeme pozorovať prakticky okamžitú konvergenciu algoritmu do

optimálnej hodnoty akonáhle chromozóm nadobudol hodnoty z blízkeho okolia minima tejto funkcie.

Nakoľko nami zvolená knižnica PyGAD funguje na princípe maximalizácie, účelovej hodnote sme museli obrátiť znamienko tak, aby optimálna hodnota účelovej funckie ležala v kladnej časti súradnicového systému.

Genetickému algoritmu sa podarilo dosiahnuť optimálnu hodnotu 1 v každom behu.

V behu č. 3 sme znížili faktor mutácie na 10 %, čo taktiež zapríčinilo neskoršie dosiahnutie optimálnej hodnoty účelovej funkcie.

4.3 Optimálna trajektória

V poslednej praktickej časti zvolíme ako predmet optimalizácie problém optimálnej trajektórie. Úlohou genetického algoritmu je nájdenie čo najkratšej trasy potrebnej na prechod medzi vopred definovanými bodmi. Účelovú funkciu reprezentuje suma euklidovských vzdialeností medzi jednotlivými prejdenými bodmi. Euklidovskú vzdialenosť rátame medzi aktuálnym a posledným prejdeným bodom, pre všetky prejdené body podľa vzťahu (X), kde N reprezentuje počet bodov prechodu:

Vzťah opisujúci euklidovskú vzdialenosť medzi aktuálnym a predošlým bodom, pre všetky body prechodu:

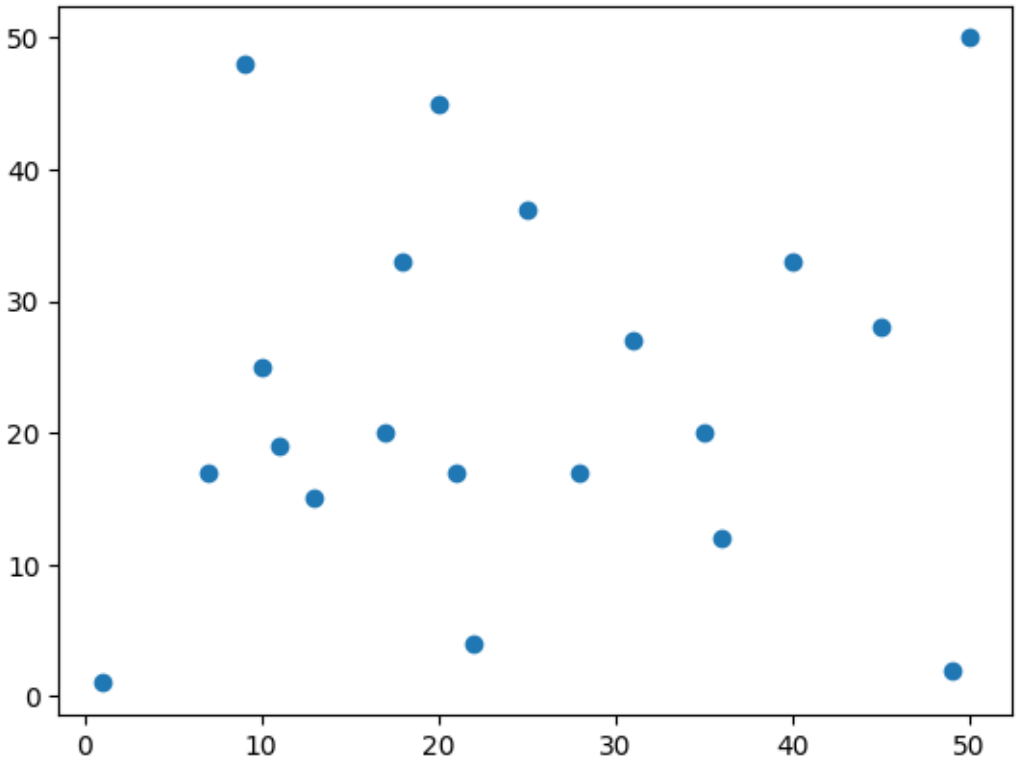
$$d = \sum_{k=1}^{N} \sqrt{\left((x[k] - x[k-1])^2 + (y[k] - y[k-1]^2)\right)}$$

Množinu nami zvolených bodov tvorí usporiadaná N-tica dvadsiatich rovinných koordinátov X a Y s tým, že začiatok našej trasy musí vždy začínať v bode [0,0] a koniec trasy je vždy v bode [50,50].

Úloha optimálnej trajektórie je optimalizačný problém s permutačným krížením. To znamená že na jednotlivé chromozómy nie je možné aplikovať klasické metódy kríženia, nakoľko by nastala situácia, kedy by algoritmus konvergoval takým spôsobom, že všetky body by sa zmenili na počiatočný bod a celková suma prejdenej vzdialenosti by bola 0. Takáto dynamika nášho optimalizačného algoritmu je neprípustná a tým pádom musíme použiť permutačný

typ kríženia. Operácia permutačného kríženia dovoľuje krížiť gény chromozómu iba vrámci toho istého chromozómu, tj. po riadku.

Operáciu mutácie taktiež nie je možné aplikovať na tento druh problému, nakoľko sú naše body cez ktoré má algoritmus prejsť pevne dané a ich mutovanie by nedávalo zmysel.



Obrázok 16 - Vizualizácia kontrolných bodov

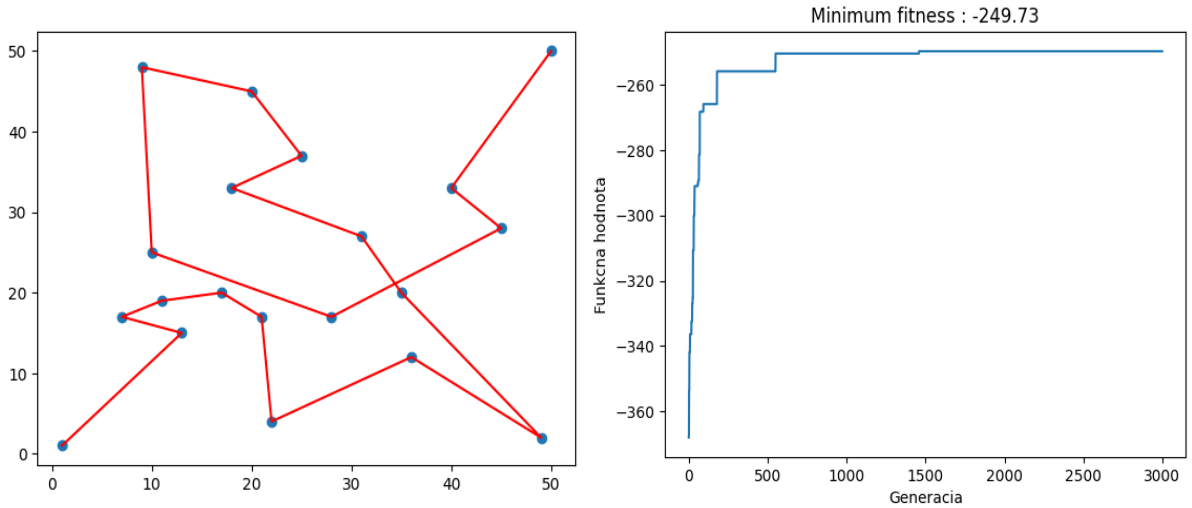
Na *obrázku 16* vidíme zobrazenie kontrolných bodov, cez ktoré všetky musí algoritmus prejsť aby sa úspešne dostal do cieľa. Tým že počiatočný a koncový bod sa nemenia, genetický algoritmus má na výber 18 faktoriál možností ako prejsť túto trasu, čo činí viac ako 6.4x10^15 spôsobov prechodu.

4.3.1 Optimalizačné parameter

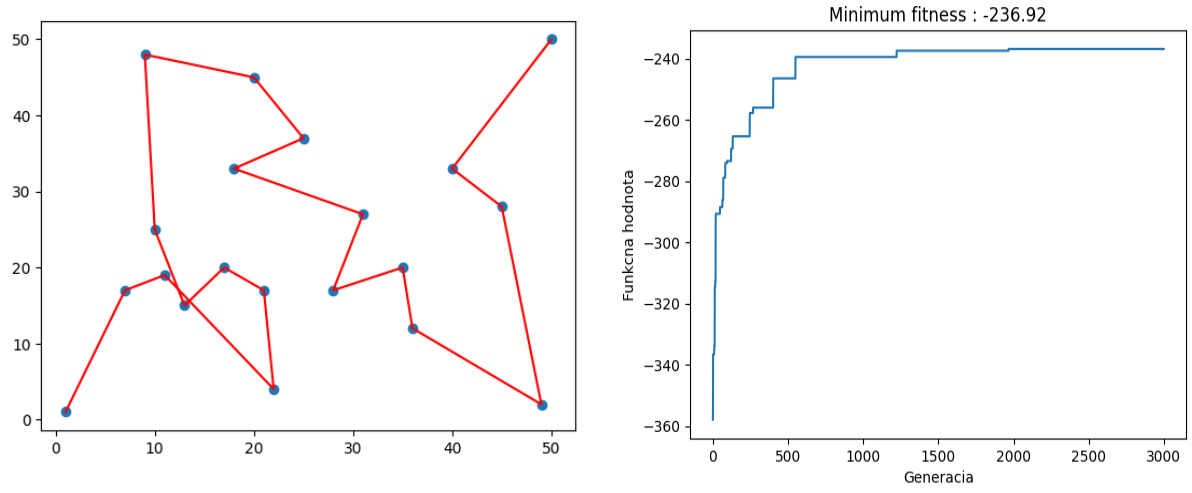
Pri riešení tejto praktickej úlohy využijeme zmiešavaciu metódu mutácie, ktorá bude vymieňať gény vždy medzi jedným a tým istým chromozómom. Pravdepodobnosť zmiešavacieho mutácie nastavíme v prvom behu na 15 %. Počet generácií zvýšime na 3000 a veľkosť populácie taktiež zvýšime na 100 jedincov, nakoľko sa jedná o komplexnejšiu úlohu v porovnaní s predošlími riešenými úlohami v tejto práci.

Nakoľko knižnica PyGAD pracuje na princípe maximalizácie, musíme výslednej sume zmeniť znamienko na záporné, aby logika nášho programu vyhovovala PyGAD implementácii. Tým pádom hodnota účelovej funkcie bližšia k nule reprezentuje vhodnejšie riešenie nášho problému.

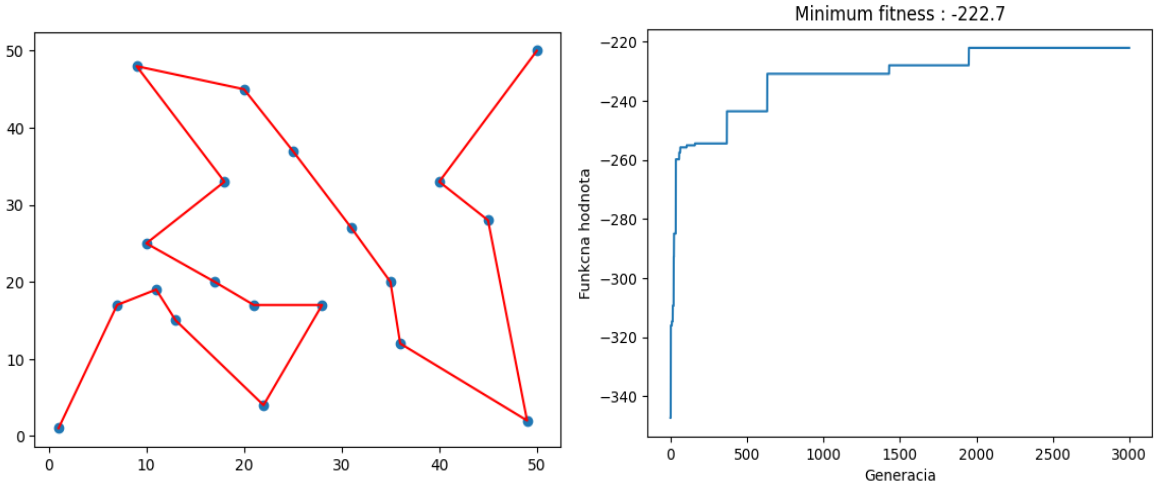
4.3.2 Priebeh optimalizácie



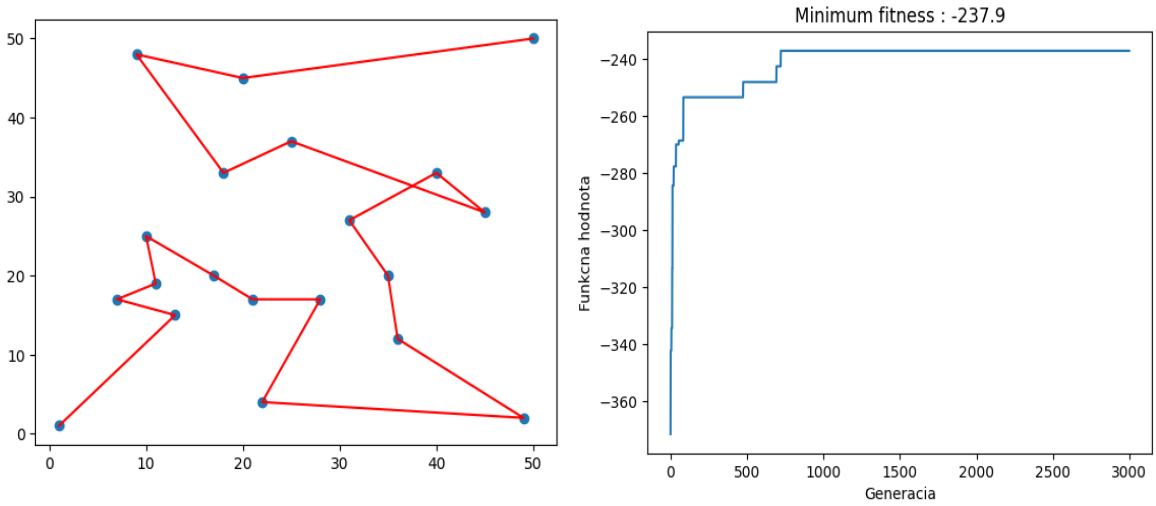
Obrázok 18 - Optimálna trajektória beh č. 1



Obrázok 17 - Optimálna trajektória beh č. 2



Obrázok 19 - Optimálna trajektória beh č.3



Obrázok 20 -Optimálna trajektória beh č.4

Z jednotlivých priebehov 1,2,3,4 môžeme pozorovať značnú mieru nekonzistencie sub-optimálneho algoritmu. Medzi jednotlivými behmi sme menili parametre genetického algoritmu.

V prvom behu bolo bez modifikáciu presunutých iba päť jedincov zo sto, ostatní podliehali operácii zmiešavacieho kríženia. Tento beh dopadol najhoršie, s výslednou hodnotou účelovej funkcie -249.7.

V druhom behu sme do nasledovnej generácie presunuli bez modifikácií 15 jedincov zo 100 a zvýšili sme kríženie z pôvodných 15 % na 20 %. To mierne zlepšilo naše výsledky a v tomto behu sme dosiahli hodnotu účelovej funkcie -236.9.

V nasledujúcom treťom behu sme hodnotu pravdepodobnosti kríženia nechali na 20% a do nasledovnej generácie sme navýšili počet jedincov putujúcich

do nasledovnej generácie bez modifikácií, a to na 30. V tomto behu sme dosiahli najlepšiu hodnotu účelovej funkcie a to -222.7 a trajektória sa na pohľad javí ako optimálna.

V poslednom behu sme pokračovali s tridsiatimi jedincami putujúcimi do nasledovnej generácie bez modifikácií, akurát sme tu zvýšili mieru kríženia na 30%, čo predstavuje pomerne vysokú hodnotu. V tomto behu sme dosiahli hodnotu účelovej funkcie -237.9 čo nepredstavuje žiadnu formu zlepšenia.

Najlepší dosiahnutý beh bol pri dvadsať percentnom krížení s tridsiatimi jedincami putujúcimi do nasledovnej generácie bez modifikácií. Pri tejto konfigurácii sme taktiež skúšali navýšiť aj počet generácií, čo však viedlo ku stagnácii účelovej funkcie bez významného zlepšenia.

V úlohách minimalizácie Schweffelovej a Easomovej funkcie sa podarilo dosiahnuť minimálnu hodnotu účelovej funkcie veľmi blízku globálnemu optimu funkcie.

V prípade Schweffelovej funkcie sme overovali schopnosť evolučného algoritmu vymaniť sa z lokálneho extrému funkcie, ktorá má v danom definičnom obore vysoký počet lokálnych extrémov, čo má za následok tendenciu algoritmu uviaznuť v týchto extrémoch.

V úlohe minimalizácie Easomovej funkcie sme testovali schopnosť algoritmu efektívne prehľadávať priestor definičného oboru, kde bola hodnota účelovej funkcie vo väčšine priestoru konštantná. Tento algoritmus nemal v častiach konštantnej účelovej funkcie spätnú väzbu o úspešnosti riešenia. Tento nedostatok sme vykompenzovali zvýšenou mierou diverzity populácie, kde sa nám podarilo dosiahnuť hodnoty veľmi blízke globálnemu optimu 1.

V úlohe minimalizácie trajektórie presnú hodnotu globálneho optima nepoznáme, no v behu č. 3 sa nám podarilo dosiahnuť trajektóriu, ktorá podľa vizuálnej intuície najviac pripomínala najkratšiu trasu. V tomto bola dosiahnutá hodnota účelovej funkcie rovná -222.7, čo je zároveň najlepšia hodnota účelovej funkcie spomedzi všetkých nami uskutočnených behov.

Záver

V tejto bakalárskej práci sme sa zaoberali genetickými algoritmami a ich využitím pri optimalizácii nami zvolených úloh. Pre získanie požadovaného výsledku sme používali prostredie programovacieho jazyka Python a jeho známu knižnicu Pygad. Python je jeden z najviac populárnych programovacích jazykov vo svete, medzi užívateľmi. Python je open-source programovacím jazykom. Programovanie v jazyku Python má jednoduchú a ľahko použiteľnú syntax, vďaka čomu sa radí medzi tie jazyky, vhodné pre začínajúcich programátorov.

V práci sme sa okrem charakteristiky a základných pojmov v rámci programovacieho jazyka Python naučili aj pojmy akými sú gén, chromozóm, fitness funkcia, kríženie či selekcia. Ukázali sme si ako pomocou knižnice PyGad a jej nástrojov vieme prísť k požadovanému výsledku, maximalizácii alebo minimalizaácii v úlohách, v krátkom čase a tým efektívne šetriť čas.

V tejto práci sme vykonali minimalizáciu troch funkcií pomocou prístupov genetického algoritmu a diferenciálnej evolúcie. Minimalizovali sme Schweffelovu a Easomovu funkciu, ktoré sú vhodné na testovanie optimalizačných algoritmov. Taktiež sme minimalizovali funkciu optimálnej trajektórie, ktorej výsledná hodnota sa odvíjala od poradia jednotlivých bodov, ktorých spojnica tvorila trajektóriu. Pri tejto úlohe sme presnú hodnotu minima účelovej funkcie nepoznali, no dosiahli sme trajektóriu ktorá na pohľad pripomínala najviac intuitívne riešenie.

V prvých dvoch úlohách sme pre každú úlohu vykonali päť behov, kde sme počas každého behu menili parametre algoritmu za účelom jeho odladenia. Všetky behy v prvých dvoch úlohách konvergovali do 200=tej generácie, čo svedčí o vhodne zvolených parametroch algoritmu.

V tretej úlohe sme vykonali štyri behy minimalizácie trajektórie, kde sme značne navýšili počet generácií. Hodnota účelovej funkcie viditeľne konvergovala aj v záverečných častiach optimalizačného procesu, t.j. v intervale 1500 až 2500 tej generácie, čo svedčí o pomerne vysokej náročnosti tejto úlohy.

Pri minimalizácii Schweffelovej funkcie pomocou metódy diferenciálnej evolúcie sme dosiahli výsledky veľmi porovnateľné s výsledkami získanými pomocou genetického algoritmu.

Evolučné algoritmy sú v dnešnom svete, v ktorom výpočtový výkon nie je prekážkou, veľmi populárnym nástrojom v procese optimalizácie. Za ich popularitu môže najmä ich pomerne nenáročná implementácia do úloh, kde gradientné optimalizačné metódy môžu byť náročné na implementáciu.

Zoznam použitej literatúry

[1] KANDRÁČ, J. *Úvod do algoritmov umelej inteligencie #1 - HillClimbing* [elektronický zdroj]. Slovensko, [2022], online. [cit. 2023-02-17]. Dostupné na: https://www.goodrequest.com/sk/blog/algoritmy-umelej-inteligencie-part-1-hillclimbing

[2] BAZARAA, S. M. - SHERALI, D. H. - SHETTY, C. M. *Nonlinear Programming: Theory and Algorithms,* 3. vydanie, WILEY 2006 872 s. ISBN: 978-0-471-48600-8

[3] MSG, L. *vs Golang vs Python programovanie. Aký je rozdiel?* [elektronický zdroj]. Slovensko, [2022], online. [cit. 2023-03-19]. Dostupné na: https://msg-life.sk/clanky/digitalizacia/java-vs-golang-vs-python-programovanie/?gclid=Cj0KCQiAz9ieBhCIARIsACB0oGK9H0O5JFwrXsoU-NdtL8DK-QpiXeo_HiiEZUfSrjJH3ceTw7Cg2W8aAugPEALw_wcB

[4] BLAHO, A. *Programovanie v Pythone* [elektronický zdroj]. Slovensko, [2022], online. [cit. 2023-03-20]. Dostupné na: https://python.input.sk/z/01.html

[5] NATIONAL GEOGRAPHIC SOCIETY *Theory of Evolution* [elektronický zdroj]. Britain, [2022], online. [cit. 2023-03-31]. Dostupné na: https://education.nationalgeographic.org/resource/theory-evolution/

[6] GOYAL, A. *Genetic algorithms: An overview of how biological systems can be represented with optimization functions* [elektronický zdroj]. Britain, [2021], online. [cit. 2023-04-01]. Dostupné na: https://aggietranscript.ucdavis.edu/genetic-algorithms-an-overview-of-how-biological-systems-can-be-represented-with-optimization-functions/

[7] ABBAS ALBADR, M. a kol. *Genetic Algorithm Based on Natural Selection Theory for Optimization Problems* [elektronický zdroj]. Britain, [2021], online. [cit. 2023-04-01]. Dostupné na: https://www.mdpi.com/2073-8994/12/11/1758

[8] GCX11 *Lekce 3 – Proměnné, typový systém a parsovaní v Pythonu* [elektronický zdroj]. Česká republika, [2022], online. [cit. 2023-03-30]. Dostupné na: https://www.itnetwork.cz/python/zaklady/python-tutorial-promenne-zakladni-datove-typy-a-funkce

[9] KABIL, A. *PyGAD – Python Genetic Algorithm* [elektronický zdroj]. , [2020], online. [cit. 2023-04-01]. Dostupné na: https://pygad.readthedocs.io/en/latest/

[10] SEKAJ, I. *Evolúcia v počítači alebo evolučné a genetické algoritmy* [elektronický zdroj]. Slovensko, [2009], online. [cit. 2023-04-02]. Dostupné na: http://www.posterus.sk/?p=3364

[11] CONTRIBUTOR, T. *Evolutionary computation* [elektronický zdroj]. USA, [2019], online. [cit. 2023-04-03]. Dostupné na: https://www.techtarget.com/whatis/definition/evolutionary-computation

[12]SZÖLLÖSI,T. 2012. *Evoluční algoritmy: Diplomová práca.* Vysoké učení technické v Brně. 2012. [cit. 2023-04-04]

[13] STORN, R.-PRICE, K. *Differential evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces* [elektronický zdroj]. Netherlands, [1997], online. [cit. 2023-04-04]. Dostupné na: https://www.metabolic-economics.de/pages/seminar_theoretische_biologie_2007/literatur/schaber/Storn1997JGlobOpt11.pdf

[14] ABRAHAM, A.-JAIN, L. *Evolutionary Multiobjective Optimization* [elektronický zdroj] [2020], online. [cit. 2023-04-06]. Dostupné na: https://towardsdatascience.com/how-to-define-a-fitness-function-in-a-genetic-algorithm-be572b9ea3b4

[15] TUTORIALSPOINT *Genetic Algorithms - Population* [elektronický zdroj]. USA,[2023], online. [cit. 2023-04-07]. Dostupné na: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_population.htm

[16] MALLAWAARACHCHI, V. *How to define a Fitness Function in a Genetic Algorithm* [elektronický zdroj]. Slovensko, [2017], online. [cit. 2023-04-07]. Dostupné na: https://towardsdatascience.com/how-to-define-a-fitness-function-in-a-genetic-algorithm-be572b9ea3b4

[17] DUTTA, A. *Crossover in Genetic Algorithm* [elektronický zdroj]. USA, [2023], online. [cit. 2023-04-10]. Dostupné na: https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/

[18] ALGODAILY *Swap Mutation* [elektronický zdroj], [2023], online. [cit. 2023-04-20]. Dostupné na: https://algodaily.com/lessons/introduction-to-genetic-algorithms-in-python/swap-mutation