VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

APLIKACE ALGORITMŮ ZALOŽENÝCH NA PSO

APPLICATIONS OF PSO-BASED ALGORITHMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. PATRÍCIA HUDECOVÁ**
AUTHOR

VEDOUCÍ PRÁCE **Doc.Ing. MICHAL BIDLO, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

155794

Ústav:	Ústav počítačových systémů (UPSY)
Studentka:	**Hudecová Patrícia, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Inteligentní systémy
Název:	**Aplikace algoritmů založených na PSO**
Kategorie:	Umělá inteligence
Akademický rok:	2023/24

Zadání:

1. Nastudujte doporučenou literaturu zabývající se optimalizačními technikami založenými na Particle Swarm Optimization (PSO).
2. Zpracujte přehledovou studii zaměřenou na vybrané relevantní metody PSO (např. Gravitational Search Algorithm, Black Hole Algorithm apod.) aplikované v různých oblastech.
3. Zvolte alespoň dva optimalizační problémy s cílem jejich řešení pomocí vybraných metod PSO. Implementujte jejich řešení ve zvoleném prostředí a analyzujte získané výsledky.
4. Na základě výsledků analýzy z bodu 3 vytipujte potenciální části optimalizačních algoritmů vhodné k modifikaci. Vhodně rozšiřte původních algoritmy, s cílem vylepšení řešení zvolených problémů
5. Realizujte sadu experimentů řešících zvolené úlohy s využitím různých verzí algoritmů navržených v bodu 4.
6. Dosažené výsledky srovnejte s těmi původními (získanými v bodu 3), zhodnoťte vlastnosti navržených rozšíření algoritmů a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího projektu.

Při obhajobě semestrální části projektu je požadováno:

- Bez požadavků (SEP splněn).

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Bidlo Michal, doc. Ing., Ph.D.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2023
Termín pro odevzdání:	17.5.2024
Datum schválení:	30.10.2023

Abstrakt

Cieľom tejto práce bolo naštudovanie už existujúcich optimalizačných algoritmov z oblasti algoritmov inšpirovaných kolektívnym správaním a ich aplikovanie na problémy využiteľné v praxi. Následná snaha o vylepšenie už existujúcich riešení možnými modifikáciami a vyhodnotenie výsledkov. Pričom ako algoritmy boli zvolené algoritmus optimalizácie hejnom častíc, algoritmus gravitačného vyhľadávania a algoritmus čiernych dier. Optimalizačné problémy boli problém obchodného cestujúceho a problém batohu, ktoré sú aplikovateľné v rôznych praktických oblastiach.

Abstract

This work aimed to study already existing optimization algorithms from the field of algorithms inspired by collective behavior and apply them to problems usable in practice in practice. Subsequent efforts to improve existing solutions with possible modifications and evaluation of the results. The algorithms chosen were particle swarm optimization, gravitational search algorithm, and black hole algorithm. The optimization problems were the traveling salesman problem and knapsack, which are applicable in various practical fields.

Klíčová slova

algoritmus optimalizácie hejnom častíc, algoritmus gravitačného vyhľadávania, algoritmus čiernych dier, problém obchodného cestujúceho, problém batohu, optimalizácie

Keywords

particle swarm optimization, gravitational search algorithm, black hole algorithm, traveling salesman problem, knapsack, optimization

Citace

HUDECOVÁ, Patrícia. *Aplikace algoritmů založených na PSO*. Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc.Ing. Michal Bidlo, Ph.D.

Aplikace algoritmů založených na PSO

Prohlášení

Prehlasujem, že som túto prácu vypracovala samostatne pod vedením pána Doc. Ing. Michala Bidla, Ph.D. uviedla som všetky literárne zdroje, z ktorých som čerpala.

.....
Patrícia Hudecová
16. května 2024

Poděkování

Týmto by som sa chcela poďakovať pánovi Doc. Ing. Michalovi Bidlovi Ph.D. za vedenie tejto práce, jeho cenné rady, trpezlivosť a ochotu pri vedení práce. Tiež by som chcela poďakovať mojej rodine a priateľom za podporu a motivovanie počas môjho štúdia.

Obsah

Kapitola 1

Úvod

V súčasnej dobe sa optimalizáciám prikladá v dnešnej spoločnosti veľmi veľký význam. Asi neexistuje odvetvie, ktoré by nebolo nejakým spôsobom optimalizované. Či už sa jedná o cesty do práce alebo školy, kde sa optimalizuje doprava a preprava osôb tak, aby sa v značnej miere zamedzilo dopravným zápcham, nakupovanie, pri ktorom obchodné reťazce hľadajú spôsoby, ako tento proces urýchliť. Optimalizácie energií v budovách rôznych inštitúcii vzhľadom na ekológiu aj financie. Optimalizácie v preprave materiálov a rôznych produktov. Optimalizácie v rôznych výrobných procesoch atď... Vzhľadom na tieto skutočnosti ponúka oblasť optimalizácií širokú škálu výskumných možností.

Pri hľadaní optimálneho riešenia inžinierske postupy narážajú na problém exponenciálnej zložitosti riešenia problému, niektoré problémy nie je možné pomocou bežných metód riešiť v reálnom čase. Preto vzniklo veľké množstvo stochastických a metaheuristických algoritmov na riešenie týchto úloh.

Jedným z príkladov týchto problémov je problém obchodného cestujúceho (travelling salesman problem), ktorý má veľmi široké využitie v praxi, ako napríklad pri optimalizácii dráhy nástroja pri vŕtaní dosiek plošných spojov. Taktiež sa dá využiť pri probléme s trasovaním vozidla (vehicle routing problem), ktorý pri dobrom naplánovaní trasy vedie k zníženiu nákladov na prepravu, zvýšeniu kvality životného prostredia a zrýchleniu prepravy tovaru. Ďalším príkladom takéhoto problému je problém batohu (knapsack), ktorý sa v praxi dá využiť napríklad v oblasti kryptografie a mnohých iných oblastiach.

Táto práca je zameraná na optimalizáciu oboch problémov vzhľadom na ich široké praktické využitie. Cieľom tejto práce je využiť existujúce optimalizačné algoritmy založené na inteligencii skupiny, a to konkrétne algoritmus časticových systémov (PSO)[27], algoritmus gravitačného vyhľadávania (GSA)[39] a algoritmus čiernych dier (BH)[22] na riešenie vyššie zmienených problémov. Jedným z ďalších cieľov práce je overiť už existujúce modifikácie algoritmu PSO pri riešení problému TSP[18], ďalším cieľom je modifikovať niekoľkými spôsobmi algoritmus čiernych dier a aplikovať ho pri riešení problému batohu[3].

Štruktúra práce je nasledujúca. V druhej kapitole môžeme vidieť stručný prehľad stochastických optimalizačných algoritmov. Ich delenie do rôznych kategórii, ako napríklad evolučné algoritmy, algoritmy inšpirované kolektívnym správaním, optimalizácie inšpirované fyzikou a iné princípy. V jednotlivých častiach sú uvedení reprezentanti daných kategórii so stručným popisom. Tretia kapitola je zameraná na podrobný popis algoritmu PSO, ktorý je prvým reprezentantom algoritmov založených na inteligencii skupiny a bol inšpiráciou pre vznik mnohých ďalších algoritmov. V úvode tejto kapitoly je popísaná jeho pôvodná verzia a spôsob akým vznikol, následne je popísaný základný algoritmus a niektoré možnosti jeho využitia. V štvrtej kapitole sú podrobne popísané algoritmy, ktoré čerpajú inšpiráciu

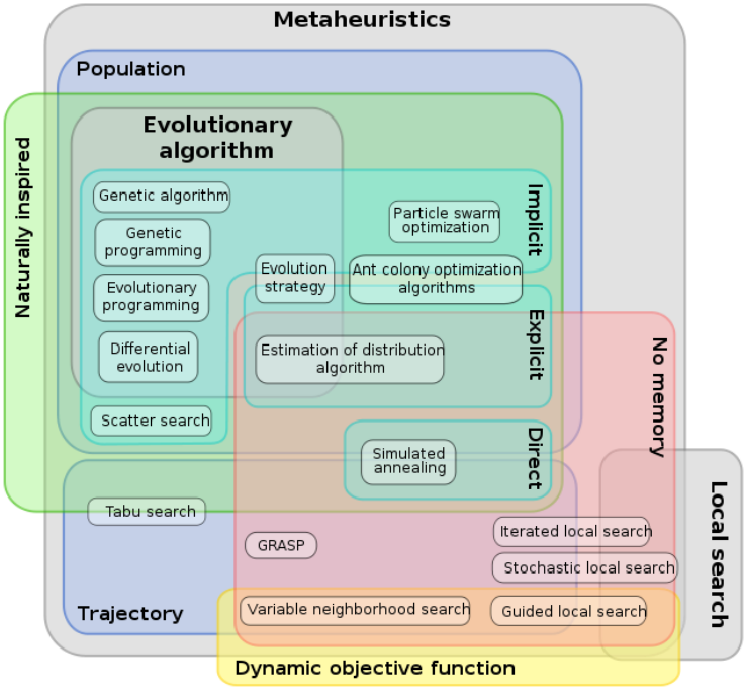
z gravitácie a zaraďujeme ich medzi algoritmy inšpirované fyzikou, konkrétne sa jedná o GSA a BH. Tieto algoritmy sú následne popísané a tiež sú podrobne popísané fenomény, ktoré inšpirovali ich vznik, a to gravitácia a čierne diery. Následne je uvedených niekoľko príkladov ich využitia v praxi v nedávnej dobe. Piata kapitola je zameraná na optimalizáciu NP-ťažkých úloh a ich dvoch predstaviteľov, a to problém obchodného cestujúceho a problém batohu. Oba sú podrobne popísané, sú uvedené niektoré ich varianty a možnosti využitia v praxi. Šiesta kapitola sa zaoberá podrobným popisom aplikácie a modifikácií, ktoré boli naimplementované. V siedmej kapitole je podrobne popísaný priebeh experimentovania a zhodnotenie jeho výsledkov. V závere je zhrnutie práce, jej výsledkov a stručne spomenuté možnosti pokračovania a prípadných rozšírení.

Kapitola 2

Stochastické optimalizačné algoritmy

V tejto kapitole sú stručne popísané niektoré z prístupov k optimalizáciám, ktoré sa v dnešnej dobe využívajú. Prístupov k optimalizáciám rovnako tak aj miest, kde optimalizačné algoritmy hľadajú inšpiráciu je veľké množstvo. Tieto algoritmy sú delené do rôznych kategórii a oblastí s rôznymi kritériami delenia. Nižšie budú stručne popísané evolučné algoritmy, algoritmy inšpirované kolektívnym správaním a algoritmy inšpirované fyzikou. Okrem zmienených oblastí, časť algoritmov čerpá inšpiráciu z ľudského správania, zo športu, z chemických javov, prírodných javov, správania živočíchov a mnohých iných[5].

Jeden zo spôsobov delenia metaheuristických algoritmov je znázornený na obrázku 2.1. Pre účely tejto práce sú najzaujímavejšie algoritmy označované ako metaheuristické. Čo sú v podstate obecné šablóny, pri ktorých konkrétne špecifikácie niektorých častí a parametrov umožňujú prispôsobenie daného algoritmu pre riešenie určitej úlohy. V tejto práci sa budeme zaoberať algoritmami, ktoré sa na obrázku 2.1 nachádzajú v časti prírodou inšpirovaných algoritmov a zároveň algoritmov založených na populácii. Konkrétne bude táto práca zameraná na časticové systémy, ktoré sú popísané v časti 7.10 a algoritmy inšpirované gravitáciou, ktoré sú popísané v kapitole 4, konkrétne algoritmus gravitačného zrýchlenia v podkapitole 4.1 a algoritmus čiernych dier v podkapitole 4.3. V ďalšej sekcii zmienime evolučné algoritmy, z ktorých niektoré princípy ďalších metód vychádzajú.



Obrázek 2.1: Delenie metaheuristických algoritmov na základe rôznych kritérií [5]

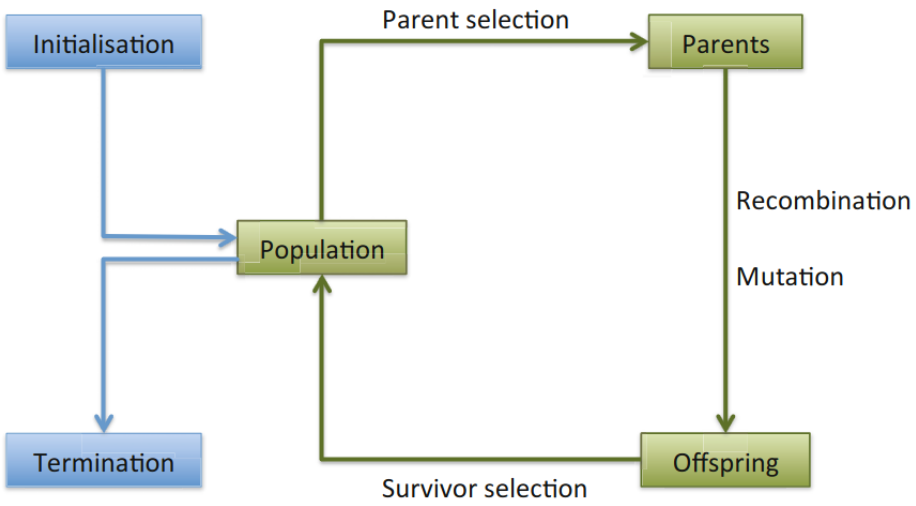
2.1 Evolučné algoritmy

Evolučné algoritmy patria medzi veľmi rozšírené a používané metaheuristiky v mnohých oblastiach, pričom inšpiráciu čerpajú z Darwinovej teórie o prirodzenom výbere a dedičnosti. V 20. storočí boli jeho poznatky rozšírené molekulárnymi biológmi o existenciu DNA, ktorá má dôležitú rolu pri určovaní dedičných vlastností.[8] Darwinova teória a nové informácie o DNA stáli za vznikom evolučných algoritmov. Tieto algoritmy vychádzajú z prirodzeného výberu a dedičnosti, pričom dedičnosť je zaručená prirodzeným výberom kvalitnejších riešení. Ich potomkovia sú odvodení pomocou genetických operátorov ako mutácia alebo kríženie z kópií svojich rodičov. Vďaka kopírovaniu informácii od rodičov k potomkom dochádza k dedičnosti. Kríženie a mutácia umožňujú jedincom sa vyvíjať a zlepšovať.

Evolučné algoritmy majú mnoho komponent, procedúr a operátorov, ktoré musia byť špecifikované avšak najdôležitejšie komponenty sú:

1. reprezentácia (definícia jednotlivcov),
2. hodnotiaca funkcia (alebo fitness funkcia),
3. populácia,
4. mechanizmus výberu rodičov,
5. variačné operátory, rekombinácia a mutácia,
6. mechanizmus výberu preživších (náhrada).[14]

Obecnú schému evolučného algoritmu môžme vidieť na 2.2.



Obrázek 2.2: Schéma evolučného algoritmu
[14]

Postupom času bolo vytvorených mnoho variant evolučných algoritmov, pričom najznámejšie sú:

- evolučné programovanie,
- evolučné stratégie,
- genetické algoritmy,
- genetické programovanie.

Vyššie uvedené evolučné algoritmy sú dané historicky, nakoľko tieto algoritmy boli vyvíjané nezávisle od seba. Avšak, v dnešnej dobe sa tieto algoritmy rôzne prepájajú a rozdiely medzi nimi sa postupne zmazávajú. [8]

2.2 Algoritmy inšpirované kolektívnym správaním

Potom, ako evolučné algoritmy preukázali svoje možnosti uplatnenia v rôznych oblastiach, boli objavené ďalšie prírodné fenomény, ktorými sa inšpiruje množstvo neskôr vyvinutých metaheuristických algoritmov. A to napríklad mravčie algoritmy, časticové systémy a iné. Tieto techniky zdieľajú niektoré dôležité aspekty evolučných algoritmov, hlavne prvky populácie jedincov a dedičnosti. V tejto sekcii sa ďalej zameriame na algoritmy inšpirované kolektívnym správaním rôznych prírodných spoločenstiev a systémov.

Algoritmy inšpirované kolektívnym správaním (swarm intelligence) sa zameriavajú na kolektív jedincov. Sledujú správanie viacerých jedincov, ktorí spolu interagujú na základe istých pravidiel. Každý jedinec (agent) je sám považovaný za neinteligentného, pričom skupina viacerých jedincov ukazuje známky samo-organizovaného správania, čo pôsobí ako kolektívna inteligencia. Hlavné vlastnosti algoritmov inšpirovaných inteligenciou skupiny je možné zhrnúť nasledovne[21]:

- zdieľanie informácií medzi viacerými agentmi,
- agenti sú samo-organizovaný a spoločne sa učia,
- je veľmi efektívny pre ich schopnosť vzájomného učenia,
- môže byť jednoducho paralelizovaný pre praktické problémy a problémy v reálnom čase (real-time problémy).

Do tejto kategórie patrí mnoho algoritmov, ako napríklad:

- algoritmus časticových systémov (particle swarm optimization),
- algoritmus kolónie mravcov (ant colony optimization),
- algoritmus umelej včelej kolónie (artificial bee colony),
- netopierí algoritmus (bat algorithm),
- algoritmus vlčej svorky (wolf search),
- algoritmus mačacej svorky (cat swarm algorithm),
- algoritmus svätojánskych mušiek (firefly algorithm).

V nasledujúcej časti prehľadovo popíšeme hlavné princípy najdôležitejších algoritmov z vyššie uvedených, ktoré poskytujú základ pre realizáciu algoritmov v rámci tejto práce.

Algoritmus optimalizácie hejnom častíc je jedným z prvých optimalizačných algoritmov založených na inteligencii skupiny. Inšpiráciu jeho autori čerpali z vtáčieho hejna, konkrétne zo spôsobu, akým vtáci vyhľadávajú potravu. Autori algoritmov živé tvory (vtáky) označujú všeobecne ako častice, odtiaľ vznikol názov časticové systémy. Tento algoritmus je podrobne popísaný v nasledujúcej kapitole 7.10.[27]

Algoritmus kolónie mravcov sa inšpiroval spôsobom, akým hľadajú mravce potravu v prírode. Mravce sa rozdelia a náhodne prehľadávajú priestor, pričom počas cesty nechávajú feromónové stopy pomocou ktorých komunikujú s ostatnými mravcami. V prípade, že mravec nájde zdroj potravy, zanechá na danej ceste väčšie množstvo feromónov. Zvyšné mravce túto cestu vďaka feromónom nájdu a tiež po nej začnú chodiť k danému zdroju potravy, kým nejaký ďalší mravec nenájde lepší zdroj potravy.[13]

Algoritmus umelej včelej kolónie vychádza zo správania včiel pri hľadaní potravy. Rozdeľuje včely do troch kategórii:

1. včely pracovnice (employed bees)
2. včely skauti (scouts)
3. pozorujúce včely (onlookers bees).

Úlohou pracujúcich včiel je získavať potravu zo známeho zdroja potravy. V prípade, že tento zdroj prestane byť vhodný, sa z včely stáva skautská včela a vydáva sa hľadať nový, lepší, zdroj potravy. Pozorujúce včely čakajú, kým sa z nich stanú včely skauti alebo včely pracovnice.[25]

Netopierí algoritmus čerpá inšpiráciu z echolokácie, ktorú využívajú mikronetopiere pri hľadaní potravy a lietaní. Echolokácia funguje na princípe odrážania zvukových vĺn od okolitých objektov. A to tak, že netopier vyšle zvukový signál, tento signál sa odrazí od objektov v okolí. Na základe tohto signálu, ktorý sa vráti, je možné určiť veľkosť, vzdialenosť a polohu objektov, a teda ju využívajú pri hľadaní potravy a na zlepšenie kvality letu.[48]

2.3 Optimalizácie inšpirované fyzikou

Optimalizačné algoritmy môžu čerpať inšpiráciu aj z fyzikálnych javov, ako napríklad gravitácia, elektrický náboj a podobne. [45] Patria sem algoritmy, ako napríklad:

- Metropolisov algoritmus,
- algoritmus simulovaného žíhania,
- algoritmus čiernych dier,
- algoritmus gravitačného vyhľadávania,
- algoritmus elektromagnetickej optimalizácie.

Algoritmus čiernych dier a algoritmus gravitačného vyhľadávania sú podrobne popísané v kapitole 4 a oba čerpajú inšpiráciu z gravitácie.

V nasledujúcej časti stručne popíšeme hlavné princípy najznámejších algoritmov inšpirovaných fyzikou.

Metropolisov algoritmus pôvodne vznikol, ako vylepšenie metódy Monte Carlo a je zaradený medzi desať najúspešnejších algoritmov 20. storočia. Je založený na náhodnom generovaní vzoriek.[6]

Algoritmus simulovaného žíhania[1] je globálny optimalizačný algoritmus, ktorý je inšpirovaný žíhaním kovov. Žíhanie kovov je proces, pri ktorom sa kovy zahrievajú na vysokú teplotu a následne ochladzujú tak, aby sme sa zbavili ich nežiadúcich vlastností. V algoritme sa proces tepelného žíhania využíva na hľadanie minima funkcie. Funguje tak, že na začiatku je nastavená nejaká teplota, ktorá sa v priebehu prehľadávania znižuje. Týmto sa zužuje oblasť prehľadávacieho priestoru na priestor so sľubnejšími riešeniami, čo zamedzuje uviaznutiu v lokálnych extrémoch na rozdiel od algoritmu horolezeckého vyhľadávania.[29]

Algoritmus čiernych dier je jedným z algoritmov, ktoré čerpajú inšpiráciu z gravitačnej sily, konkrétne fenoménu čiernych dier. Jedná sa o jav, pri ktorom sú hviezdy a iné vesmírne telesá priťahované gravitačnou silou k čiernej diere, ktorá ich následne pohltí. Tento algoritmus je podrobne popísaný v časti 4.3.[36]

Algoritmus elektromagnetickej optimalizácie je inšpirovaný elektromagnetickými javmi. Funguje na princípe pohybu častíc, na ktoré pôsobia elektromagnetické sily, pričom častice sa medzi sebou navzájom priťahujú a odpudzujú. [10]

2.4 Ostatné princípy

Mimo vyššie uvedené kategórie existuje veľké množstvo iných oblastí, z ktorých je možné čerpať inšpiráciu pri tvorbe optimalizačných algoritmov. Algoritmy môžu byť inšpirované taktiež chémiou, ako napríklad algoritmus chemickej reakcie(chemical reaction optimisation), hudbou, ako napríklad algoritmus harmonického vyhľadávania(harmony search). Ďalšou z možností je ľudské správanie, kde, ako príklad môžeme uviesť:

- sociálno-emočný optimalizačný algoritmus(social emotional optimization),
- optimalizácia anarchickej spoločnosti(anarchic society optimization).

[1] vychádza z Metropolisovho algoritmu

V nasledujúcej časti sú stručne popísané niektoré zaujímavé algoritmy z kategórie algoritmov inšpirovaných ostatnými princípmi.

Algoritmus optimalizácie anarchickej spoločnosti, je prvým kolektívnym algoritmom inšpirovaným správaním človeka. Je založený na abnormálnom správaní ľudí. Jeho jedinci vykazujú známky anarchického správania na zlepšenie svojej životnej situácie. Jedinci sú nestáli a ich nepravidelnosť sa zvyšuje so zhoršujúcou situáciou. Toto zabraňuje uviaznutiu v lokálnych extrémoch a zabezpečuje dôkladné prehľadávanie stavového priestoru. [40]

Algoritmus ligy majstrov, je algoritmom, ktorého inšpiráciou sa stal šport, zároveň sa jedná o prvý algoritmus, ktorý čerpal inšpiráciu z oblasti športu. Funguje na princípe turnaja, kde tímy hrajú medzi sebou zápasy počas niekoľkých týždňov (týždne sú reprezentované iteráciami). Na základe výsledkov sa určuje ďalší rozpis turnaja. Ich herná sila určuje ich fitness hodnotu a formácie tímov považujeme za riešenia. Na základe analýzy zápasov sa robia zmeny vo formáciách, čo odpovedá generáciám nových riešení na nasledujúce zápasy. Na konci sezóny je možný transfer hráčov, ktorý urýchli možnosť globálnej konvergencie. Turnaj prebieha, kým nenastanú koncové podmienky. [24]

Imperialistický súťažný algoritmus je inšpirovaný myšlienkou súťaženia krajín o nadvládu. V tomto algoritme jedincov predstavujú krajiny, ktoré rozdeľujeme do dvoch kategórií, a to na kolónie a imperialistov. Imperialisti a kolónie tvoria ríše, ktoré medzi sebou súťažia. Slabé ríše prídu o svoje kolónie, ktoré sú pridelené víťazom. Na konci algoritmu ostáva len jedna víťazná ríša, ktorej súčasťou sú všetky kolónie a predstavuje najlepšie riešenie. [4]

2.5 Zameranie tejto práce

Táto práca je zameraná na algoritmy inšpirované kolektívnym správaním. Avšak, do tejto kategórie patria aj niektoré algoritmy z kategórie algoritmov inšpirovaných fyzikou. A to napríklad algoritmus gravitačného vyhľadávania a algoritmus čiernych dier, ktoré sú podrobne popísané v ďalších kategóriách. Medzi algoritmy, ktoré sú založené na populácii patrí napríklad aj algoritmus optimalizácie anarchickej spoločnosti a mnohé iné algoritmy. Zameranie tejto práce na algoritmy založené na populácii je hlavne z dôvodu ich širokej využiteľnosti v mnohých oblastiach. Taktiež je dôvodom aj veľké množstvo ich modifikácií a rôznych hybridných algoritmov, ktoré z nich vznikli. Či už sa jedná o hybridné algoritmy algoritmov založených na populácii alebo kombináciu algoritmu založeného na populácii a nejakého iného algoritmu. Zameranie na samotný algoritmus PSO, ktorý je podrobne popísaný v nasledujúcej kapitole, je z dôvodu, že sa jedná o prvý algoritmus zameraný na populáciu a inšpiroval vznik ostatných algoritmov.

Kapitola 3

Particle swarm optimization

Táto kapitola sa zaoberá najznámejším algoritmom z oblasti kolektívnej výpočetnej inteligencie, a to optimalizáciou hejnom častíc (particle swarm optimization - PSO). Jedná sa o prvý optimalizačný algoritmus založený na inteligencii skupiny a stal sa inšpiráciou pre mnohé iné algoritmy. Algoritmus je veľmi využívaný pri riešení rôznych problémov, existuje mnoho jeho variant, hybridných algoritmov a podobne. V nasledujúcej kapitole je podrobne popísaný pôvodný algoritmus a príklady jeho využitia.

3.1 Hlavné rysy a vlastnosti PSO

Pôvodne bol algoritmus PSO navrhnutý na optimalizáciu spojitých multimodálnych funkcií, pričom vychádza zo simulácie zjednodušeného sociálneho modelu. Pôvodné PSO vychádza z dvoch metodológií, a to z umelého života (A-life), správania hejna vtákov, hejna rýb a všeobecnej teórie rojenia sa. Druhou je súvislosť s evolučným počítaním a genetickými algoritmami.

Pôvodný algoritmus mal jednoduchú implementáciu a bol výpočetne nenáročný. Jeho úspešnosť bola otestovaná pri trénovaní váh neurónových sietí a hľadania globálnych extrémov.

Pred vznikom algoritmu bolo vykonaných veľa simulácií pohybu organizmov v hejne ako synchronizácia vtákov v kŕdľoch a správania sa rýb v rojoch. Sociobiológ E. Wilson tvrdil, že jednotlivý členovia rybieho roja môžu aspoň teoreticky profitovať z predchádzajúcich skúseností ostatných členov pri hľadaní potravy, čím naznačil, že sociálne zdieľanie informácií ponúka evolučnú výhodu. Jeho hypotéza bola základom pre vznik algoritmu PSO.

Pôvodne boli za agentov považovaní vtáci, ktorí sa vyhýbali vzájomným kolíziám a pôvodným zámerom bolo simulovať choreografiu vtáčieho hejna. Pôvodná simulácia sa spoliehala na dva faktory:

1. najbližšieho suseda,
2. operátor šialenstva (craziness)[1].

Pomocou najbližšieho suseda sa vtákom/agentom priraďovala rýchlosť podľa jeho najbližšieho suseda, čo viedlo k synchrónnemu pohybu. Avšak, tento pohyb sa rýchlo ustálil, a preto bol vytvorený operátor šialenstva, ktorý zaručil stochasticitu simulácie. A to práve tak, že v každej iterácii bola pridaná nejaká zmena k náhodným rýchlostiam.

[1] bol navrhnutí autormi pôvodného článku o PSO, ale v praxi sa nepoužíva

Operátor šialenstva bol neskôr nahradený cornfieldovým vektorom, čo bol 2D vektor súradníc, pričom každý agent si pamätal najlepšiu hodnotu a jej pozíciu (súradnicu vektora), ktorá sa nazývala *pbest*. Vďaka tomu, že každý agent poznal svoju najlepšiu pozíciu, bolo možné určiť pozíciu jedinca so zatiaľ najlepšou nájdenou hodnotou účelovej funkcie *gbest*. Na základe týchto hodnôt bola upravovaná simulácia pohybu agentov.

Algoritmus mal na začiatku simulovať správanie kŕdľa, ale počas vylepšení v priebehu vývoja začal vykazovať skôr správanie hejna (swarm). Znakom hejna je, že je splnených nasledujúcich päť princípov[27]:

1. princíp blízkosti: jedinci by mali byť schopní vykonávať jednoduché priestorové a časové výpočty,
2. princíp kvality: jedinci by mali byť schopní reagovať na faktory kvality v prostredí,
3. princíp rôznorodej reakcie: jedinci by nemali vykonávať svoje aktivity neprimerane úzkymi kanálmi,
4. princíp stability: jedinci by nemali meniť stav svojho správania pri každej zmene prostredia,
5. princíp prispôsobivosti: jedinci by mali zmeniť svoje správanie, keď je to výhodné.

Hlavnou črtou algoritmov založených na kolektívnom správaní je, že vykazujú znaky samo-organizovaného správania. Pričom samo-organizácia v hejnách je definovaná pomocou troch základných zložiek[49]:

1. silná dynamická nelinearita: pomocou pozitívnej a negatívnej spätnej väzby sa vytvárajú vhodné štruktúry a stabilizuje sa kolektívny vzorec,
2. vyváženie medzi exploitáciou a exploráciou: balans medzi nimi zabezpečuje prehľadanie stavového priestoru,
3. viacnásobné interakcie: agenti využívajú informácie prichádzajúce od susedných agentov, vďaka čomu sa informácie šíria po celej sieti.

3.2 Popis algoritmu PSO

Algoritmus PSO je prvým optimalizačným algoritmom založeným na inteligencii skupiny, zároveň sa stal inšpiráciou pre vznik ďalších algoritmov tohto typu. Algoritmus čerpá z myšlienky Kennedyho a Eberharta, ktorí, ako prví prišli s myšlienkou optimalizácie pomocou hejna častíc. Ich pôvodný návrh je podrobne popísaný v podkapitole 3.1. Aj napriek tomu, že algoritmus z prvého návrhu sa veľmi nepoužíval, vzniklo mnoho jeho úspešných variant. Jedna z jeho základných variant je popísaná nižšie.

Jedinci, respektíve agenti sú označovaní ako častice. Zmeny ich polohy sú založené na sociálno-psychologickej tendencii jednotlivcov napodobňovať úspech iných jednotlivcov. Zmeny častíc sú, teda ovplyvnené skúsenosťami alebo znalosťami jej susedov. Častice sa v podstate snažia napodobňovať úspechy susediacich častíc a zároveň svoje vlastné. Dôsledkom tohto správania je, že sa častice stochasticky vracajú k prechádzajúcim úspešným územiam vo vyhľadávacom priestore. Takéto skupinové správanie umožňuje objavenie optimálnych oblastí vo vysokodimenzionálnom prehľadávacom priestore.

Algoritmus PSO pozostáva z hejna častíc[2], pričom každá častica predstavuje potenciálne riešenie. Častice sa pohybujú vo viacrozmernom vyhľadávacom priestore na základe svojich skúseností a skúseností susedných častíc. Nech $x_i(t)$ je pozícia častice i v prehľadávacom priestore v čase t[3], poloha častice sa mení pridaním rýchlosti častice $v_i(t)$ k jej súčasnej pozícii, tento vzťah je popísaní pomocou rovnice 3.1, kde $x_i(0) \sim U(x_{min}, x_{max})$.

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{3.1}$$

Optimalizácia je riadená pomocou vektoru rýchlosti, ktorý obsahuje poznatky častice (experiential knowledge) a sociálne vymieňané informácie z okolia častice. Poznatky častice sú označované ako kognitívna komponenta, ktorá predstavuje vzdialenosť častice od jej vlastnej najlepšej pozície od prvého časového kroku. Sociálne vymieňané informácie sú označované ako sociálna komponenta. Pôvodne boli navrhnuté dva algoritmy PSO, ktoré sa líšia veľkosťou svojho susedstva. Tieto algoritmy označujeme ako *gbest* a *lbest* PSO.

Globálne najlepšie PSO (gbest) má ako okolie pre jednotlivé častice celý prehľadávací priestor, pričom je reprezentované hviezdicovou topológiou. V okolí s hviezdicovou topológiou, sociálna komponenta, predstavuje informácie získané od všetkých častíc hejna, pričom sociálna komponenta predstavuje najlepšiu pozíciu v celom hejne a je označovaná ako $\hat{y}(t)$.

Rýchlosť častice i sa v algoritme *gbest* počíta pomocou rovnice 3.2

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)], \tag{3.2}$$

kde $v_{ij}(t)$ je rýchlosť častice i v dimenzii $j - 1, \ldots, n_x$ v čase t, $x_{ij}(t)$ je pozícia častice i v čase t, $c1$ a $c2$ sú pozitívne konštanty zrýchlenia, využívané na škálovanie prínosu kognitívnej a sociálnej komponenty, $r_{1j}(t), r_{2j}(t) \sim U(0,1)$ sú náhodné hodnoty v rozsahu $[0,1]$ z rovnomerného rozdelenia.

Najlepšia osobná pozícia častice y_i spojená s časticou i predstavuje jej najlepšiu hodnotu od prvého časového kroku. Pre problémy minimalizácie sa jej hodnota v čase $t+1$ počíta podľa rovnice 3.3.

$$y_i(t+1) = \begin{cases} y_i(t), & \text{if } f(x_i(t+1)) \geq f(y_i(t)) \\ x_i(t+1), & \text{if } f(x_i(t+1)) < f(y_i(t)), \end{cases} \tag{3.3}$$

kde $f : \mathbb{R}_x^n \to \mathbb{R}$ je fitness funkcia. Fitness funkcia meria, ako blízko je dané riešenie optimu, a teda určuje výkonnosť alebo kvalitu častice.

Globálna najlepšia pozícia $\hat{y}(t)$ v čase t je definovaná ako:

$$\hat{y}(t) \in \{y_o(t), \ldots, y_{n_s}(t)\} \mid f(\hat{y}(t)) = min\{f(y_0(t)), \ldots, f(y_{n_s}(t))\}, \tag{3.4}$$

kde n_s je počet všetkých častíc v hejne.

Algoritmus *gbest* PSO je popísaný formou pseudokódu v 1.

[2] Analogicky s evolučnými algoritmami hejno predstavuje populáciu a častica predstavuje jedinca.
[3] t sa odohráva v diskrétnom čase, ak nie je definované inak

Algorithm 1 Pseudokód algoritmu *gbest* PSO

Vytvoríme a inicializujeme n_x-dimenzionálne hejno;
repeat
    for *každú časticu* $i = 1, \ldots, n_s$ **do**
        //nastavíme najlepšiu osobnú hodnotu častice
        if $f(x_i) < f(y_i)$ **then**
            $y_i = x_i$;
        end if
        // nastavíme globálnu najlepšiu pozíciu
        if $f(y_i) < f(\hat{y})$ **then**
            $\hat{y} = y_i$;
        end if
    end for
    for *každú časticu* $i = 1, \ldots, n_s$ **do**
        Aktualizujeme rýchlosť pomocou rovnice 3.2;
        Aktualizujeme polohu pomocou rovnice 3.1;
    end for
until nie sú splnené ukončujúce podmienky;

Lokálne najlepšie PSO (lbest) je reprezentované kruhovou topológiou. Na rozdiel od *gbest* rozdeľuje prehľadávací priestor na menšie okolia pre jednotlivé častice. Sociálna komponenta predstavuje informácie vymenené medzi časticami v rámci jedného okolia, a teda predstavuje lokálne znalosti prostredia. Rýchlosť častice sa počíta podľa rovnice 3.5.

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)], \tag{3.5}$$

kde $\hat{y}_{ij}$ je najlepšia pozícia častice i v jej okolí v dimenzii j. Osobná najlepšia hodnota častice $\hat{y}_i$ je najlepšia pozícia častice v okolí $\mathcal{N}_i$ definovaná ako 3.6, susedstvo s veľkosťou $n_{\mathcal{N}_i}$ je definované ako 3.7.

$$\hat{y}_i(t+1) \in \{\mathcal{N}_i \,|\, f(\hat{y}_i(t+1)) = \min\{f(x)\}, \forall x \in \mathcal{N}_i\} \tag{3.6}$$

$$\mathcal{N}_i = \{y_{i-n_{\mathcal{N}_i}}(t), y_{i-n_{\mathcal{N}_i+1}}(t), \ldots, y_{i-1}(t), y_i(t), y_{i+1}(t), \ldots, y_{i+n_{\mathcal{N}_i}}(t)\} \tag{3.7}$$

Susedstvá v algoritme *lbest* PSO sa navzájom prekrývajú. Vďaka tomuto je možné zdieľať informácie naprieč susedstvami, a tak nájsť najlepšiu globálnu časticu v celom prehľadávacom priestore. Algoritmus *gbest* PSO je považovaný za špeciálny prípad algoritmu *lbest* PSO s okolím $n_{\mathcal{N}_i} = n_s$.

Pseudokód algoritmu *lbest* PSO je možné vidieť na 2.

Algorithm 2 Pseudokód algoritmu *lbest* PSO

```
1:  Vytvoríme a inicializujeme n_x-dimenzionálne hejno;
2:  repeat
3:      for každú časticu i = 1, ..., n_s do
4:          //nastavíme najlepšiu osobnú hodnotu častice
5:          if f(x_i) < f(y_i) then
6:              y_i = x_i;
7:          end if
8:          // nastavíme najlepšiu pozíciu susedstva
9:          if f(y_i) < f(ŷ) then
10:             ŷ = y_i;
11:         end if
12:     end for
13:     for každú časticu i = 1, ..., n_s do
14:         Aktualizujeme rýchlosť pomocou rovnice 3.6;
15:         Aktualizujeme polohu pomocou rovnice 3.5;
16:     end for
17: until nie sú splnené ukončujúce podmienky;
```

Obe verzie PSO sú veľmi podobné. Hlavné rozdiely medzi nimi sú:

- vďaka väčšiemu prepojeniu medzi časticami algoritmus *gbest* konverguje k riešeniu rýchlejšie. Avšak, táto rýchlejšia konvergencia prináša nižšiu diverzitu, ako má algoritmus *lbest*,
- vďaka vyššej diverzite algoritmus *lbest* prehľadáva väčšiu časť prehľadávacieho priestoru a zároveň má nižšiu pravdepodobnosť uviaznutia v lokálnych extrémoch. Vo všeobecnosti môžeme tvrdiť, že kruhová topológia algoritmu *lbest* zlepšuje výkonnosť.

Výpočet rýchlosti v oboch algoritmoch pozostáva z troch častí, a to z predchádzajúcej rýchlosti, kognitívnej komponenty a sociálnej komponenty.

Predchádzajúca rýchlosť $v_i(t)$ slúži, ako pamäť predchádzajúcich smerov pohybu častice a bráni častici drasticky zmeniť smer, túto zložku tiež označujeme ako zotrvačná komponenta.

Kognitívna komponenta $c_1 r_1(y_i = x_i)$ kvantifikuje výkonnosť častice, vzhľadom na predošlé výkony. Pripomína individuálnu pamäť polohy, ktorá bola pre časticu najlepšia. Vďaka tomu sú častice priťahované do svojich najlepších pozícií.

Sociálna komponenta $c_2 r_2(\hat{y} - x_i)$[4], $c_2 r_2(\hat{y}_i - x_i)$[5] kvantifikuje výkonnosť častice, vzhľadom na skupinu častíc alebo susedstvo. Sociálna zložka predstavuje skupinový štandard, ktorý sa jednotlivci snažia dosiahnuť. Vďaka tomuto sú častice zároveň priťahované k najlepším nájdeným pozíciám v rámci okolia.[15]

[4] pre *gbest*
[5] pre *lbest*

3.3 Využitie PSO

Algoritmus PSO sa súčasne vyskytuje v rôznych modifikáciách, prípadne rôznych formách hybridných algoritmov. Dosť časté je jeho prepojenie s rôznymi klasifikátormi, prípadne neurónovými sieťami. Algoritmus sa využíva pri riešení mnohých problémov v rôznych oblastiach.

Je možné ho využiť napríklad v **doprave**, a to rôznymi spôsobmi. V spojení s hĺbkovým učením (deep learning) sa využíva na *klasifikáciu vozidiel*. Klasifikácia vozidiel patrí do výskumnej oblasti spracovania obrazu a počítačového videnia, konkrétne do časti klasifikácie obrazu. Klasifikácia vozidiel sa v praxi využíva v oblasti bezpečnosti, analýzy dopravnej situácie a pri autonómnych vozidlách. Algoritmus PSO v kombinácii s predtrénovanou neurónovou sieťou bol využitý na klasifikáciu autonómnych vozidiel. PSO bolo použité na redukovanie a optimalizáciu atribútov (features). Algoritmus PSO redukoval počet vlastností obrázku z 1024 na 509.[2] Ďalším spôsobom využitia algoritmu PSO v kombinácii s neurónovými sieťami je *automatické detekovanie incidentov*[6] v doprave.[42] V doprave sa tiež algoritmus PSO používa pri *optimalizácii nastavovania semaforov*, s ohľadom na zvýšenie bezpečnosti chodcov, zníženie spotreby energie, plynulé riadenie toku dopravy, z čoho plynú aj výhody pre životné prostredie. Algoritmus PSO bol na riešenie tohto problému využitý viackrát vo viacerých mestách, napríklad v meste Kumamoto v Japonsku, v Malage v Španielsku a v meste Bahía Blanca v Argentíne. V oboch prípadoch sa PSO využilo na simuláciu dopravy, aby sa vedela určiť optimálna dĺžka trvania zelenej na jednotlivých semaforoch. V tomto prípade každý jedinec PSO predstavoval určité riešenie dopravnej situácie a vyberali sa najlepšie riešenia. V rámci danej optimalizácie sa skrátila priemerná doba trvania jednotlivých trás a zvýšila plynulosť dopravy.[47][16]

Ďalšou z oblastí využitia algoritmu PSO je oblasť **medicíny**, kde sa binárny algoritmus PSO využíva na *klasifikáciu medicínskych dát*. Tu rovnako, ako pri klasifikácii vozidiel PSO slúži na redukciu vlastností a určenie ich dôležitosti. PSO bolo úspešne použité napríklad pri klasifikácii rakoviny pľúc. Redukcia vlastností prebieha na základe ich využiteľnosti v histórii, korelácii medzi jednotlivými vlastnosťami a kategóriami. Klasifikácia snímok MR s využitím PSO sa tiež používa pri identifikácii poškodenia pečene, rakovine pŕs a iných.[9]Taktiež sa využíva pri *selekcii atribútov* zo snímok EKG, napríklad na klasifikáciu spánkových cyklov na základe variability srdcovej frekvencie.[43]

Ďalšou z oblastí je napríklad **geotechnické inžinierstvo**, kde bol algoritmus PSO využitý napríklad pri *analýze stability svahu*. Čo je jedným z najväčších prírodných problémov v horských oblastiach. Pričom tento problém môže viesť k serióznym ekonomickým stratám, poškodeniu majetku a ohrozeniu ľudského života. Algoritmus PSO sa v tomto prípade využíva tak, že častica predstavuje klzký povrch, ktorý sa následne klasifikuje na základe čoho sa určuje miera jeho nebezpečia. Ďalej sa využíva pri *návrhu stĺpov a základov*, hlavne na predikciu správania osovo zaťažených stĺpov a na maximálnu nosnosť základov. Taktiež sa využíva pri *mechanike hornín*, na určenie profilov drsnosti hornín, identifikácia štruktúry hornín a rozpoznávanie štruktúry zmenených hornín.[20]

Ďalšou zaujímavou oblasťou využitia je oblasť **energetiky**, kde sa algoritmus PSO využíva napríklad pri *chladení a vykurovaní budov*. Optimalizácia chladenia a vykurovania budov vedie k ekonomickým úsporám, úsporám energie a redukovaniu znečistenia. Konkrétne sa jedná o optimalizáciu systému BCHP(building cooling heating and power system).[46] Taktiež je ho možné použiť na *nájdenie optimálnej veľkosti komponent solárnych inštalácii budov*. V tomto prípade sa skúmala veľkosť troch komponent, a to veľkosť kolektora, ob-

[6] za incidenty sa považujú dočasné obmedzenia v doprave, problémy so semaformi a podobne

jem nádrže a veľkosť jednotky APU[7]. Hľadanie optimálnej veľkosti komponent prebiehalo pomocou simulácie modelu solárneho kombinovaného systému za pomoci simulačného systému Polysun pre stredne veľký rodinný dom v Zurichu vo Švajčiarsku. V rámci simulácie sa zistilo, že veľkosť kolektora má najväčší vplyv na spotrebu energie a náklady na inštaláciu. Ukázalo sa, že objem nádrže má tiež významný podiel pri nákladoch na inštaláciu a jednotka APU má malý vplyv aj na spotrebu, aj na náklady, ktoré sa týkajú inštalácie.[7]

Ďalšou zaujímavou oblasťou v ktorej bol algoritmus PSO využitý, je oblasť **financií**, v ktorej bol použitý pre *problémy klasifikácie*. Tu bol využitý pri hľadaní vhodných atribútov klasifikačného modelu, podobne, ako pri probléme klasifikácie vozidiel. Správna voľba atribútov má za dôsledok redukciu šumu, redukciu času, redukciu nákladov potrebných na implementáciu modelu, zjednodušenie výsledného modelu a uľahčenie používania a aktualizácie daného modelu. Algoritmus PSO bol aplikovaný na riešenie dvoch finančných klasifikačných problémov. A to hodnotenie úverového rizika, kde algoritmus PSO redukoval pôvodný počet atribútov 26 na 10. Druhá úloha sa týkala auditu identifikácie firiem, ktoré dostávajú kvalifikované audítorské správy, pričom aj pri riešení tejto úlohy, algoritmus PSO úspešne redukoval počet atribútov zo 16 na 10.[30] Ďalšou možnosťou jeho využitia v oblasti financií je pri *predpovedaní cien akcií* v kombinácii s SVM(support vector machine).[26]

Algoritmus PSO sa využíva aj v mnohých iných oblastiach, rovnako tak, aj pri riešení rôznych iných problémov. Jedná sa len o stručný výpis niektorých zaujímavých oblastí jeho využitia a stručný opis riešenia niektorých problémov, na ktorých bol aplikovaný.

[7] pomocná energetická jednotka

Kapitola 4

Algoritmy inšpirované gravitáciou

Táto kapitola sa zaoberá algoritmami, ktoré vznikli na základe inšpirácie gravitáciou, a to konkrétne algoritmom gravitačného vyhľadávania (Gravitational Search Algorithm - GSA) a algoritmom čiernych dier (Black Hole Algorithm - BH). Oba algoritmy čerpajú inšpiráciu z gravitácie, avšak, každý iným spôsobom. Algoritmus gravitačného vyhľadávania čerpá inšpiráciu z gravitačnej sily, ktorá pôsobí na telesá s rôznou hmotnosťou a vychádza z Newtonovho gravitačného zákona, ktorý je detailne popísaný v podkapitole 4.1. Zatiaľ, čo algoritmus čiernych dier sa inšpiroval fenoménom čiernych dier, ktoré vznikajú pri procese gravitačného zrútenia a ich gravitačná sila je obrovská. Tento algoritmus je podrobne popísaný v podkapitole 4.3.

4.1 Princíp optimalizácie využívajúci pôsobenie gravitácie

Pod pojmom gravitácia rozumieme tendenciu hmoty zrýchľovať k sebe navzájom (priťahovať sa). Je jednou zo štyroch fundamentálnych interakcií[1] v prírode. Medzi ďalšie patrí elektromagnetická sila, slabá nukleárna sila a silná nukleárna sila. Každá častica vo vesmíre priťahuje každú ďalšiu časticu, pričom gravitácia je všade. Spôsob, akým sa Newtonova gravitačná sila správa sa nazýva "akcia na diaľku". To znamená, že medzi oddelenými časticami gravitácia pôsobí bez akéhokoľvek sprostredkovateľa alebo oneskorenia.

V **Newtonovom gravitačnom zákone** každá častica priťahuje všetky ostatné častice pomocou gravitačnej sily. Gravitačná sila medzi dvoma časticami je priamo úmerná súčinu ich hmotností a nepriamo úmerná druhej mocnine vzdialenosti medzi nimi. Je daná vzťahom:

$$F = G\frac{M_1 M_2}{R^2}, \tag{4.1}$$

kde F je veľkosť gravitačnej sily, G je gravitačná konštanta, M_1 a M_2 sú hmotnosti prvej a druhej častice a R je vzdialenosť medzi týmito dvoma časticami. Z druhého Newtonovho zákona vyplýva, že keď na časticu pôsobí sila F, jej zrýchlenie a závisí iba od tejto pôsobiacej sily a hmotnosti M danej častice, čo je vyjadrené pomocou vzťahu:

$$a = \frac{F}{M}. \tag{4.2}$$

Na základe vyššie uvedených vzťahov, vieme, že existuje príťažlivá gravitačná sila medzi všetkými časticami vo vesmíre. Pričom veľkosť tejto sily závisí na hmotnosti danej častice

[1] pojmom fundamentálne interakcie (niekedy sa používa pojem fundamentálne sily) sa vo fyzike označujú interakcie, ktoré ďalej nie sú redukovateľné

a vzdialenosti medzi časticami. Na častice s väčšou hmotnosťou a menšou vzdialenosť má táto sila väčší vplyv. Inak povedané, čím sú častice od seba ďalej a čím ľahšie sú, tým je táto sila slabšia.

Hodnota gravitačnej konštanty G závisí od aktuálneho veku vesmíru, jej výpočet je daný nasledovným vzťahom:

$$G(t) = G(t_0) \times (\frac{t_0}{t}^{\beta}, \beta^2 < 1, \tag{4.3}$$

kde $G(t)$ je hodnota gravitačnej konštanty v čase t. $G(t_0)$ je hodnota gravitačnej konštanty na začiatku v čase t_0.

Existujú tri druhy hmotnosti:

1. aktívna gravitačná hmotnosť,
2. pasívna gravitačná hmotnosť,
3. zotrvačná hmotnosť.

Aktívnu gravitačnú hmotnosť označujeme M_a, a je to miera sily gravitačného poľa konkrétneho objektu. Gravitačné pole objektu s malou aktívnou gravitačnou hmotnosťou je slabšie ako gravitačné pole objektu s veľkou aktívnou gravitačnou hmotnosťou.

Pasívnu gravitačnú hmotnosť označujeme M_p, a je mierou sily interakcie objektu s gravitačným poľom. Vnútri rovnakého gravitačného poľa, objekt s menšou pasívnou gravitačnou hmotnosťou pôsobí menšou silou ako objekt s väčšou pasívnou gravitačnou hmotnosťou.

Zotrvačnú hmotnosť označujeme M_i, a je mierou odporu objektu voči zmene jeho pohybu pri pôsobení sily. Objekt s veľkou zotrvačnou hmotnosťou mení svoj pohyb pomalšie a objekt s malou zotrvačnou hmotnosťou ho mení rýchlejšie.

Gravitačná sila F_{ij}, ktorá pôsobí na hmotnosť i hmotnosťou j, je priamo úmerná súčinu aktívnej gravitačnej hmotnosti j a pasívnej gravitačnej hmotnosti i a nepriamo úmerná štvorcovej vzdialenosti medzi nimi. a_i je priamo úmerné F_{ij} a nepriamo úmerné zotrvačnej hmotnosti i. Na základe vyššie zmieneného rozdelenia hmotností, prepisujeme vzťahy z Newtonových zákonov do nasledujúcej podoby:

$$F = G\frac{M_{aj} \times M_{pi}}{R^2}, \tag{4.4}$$

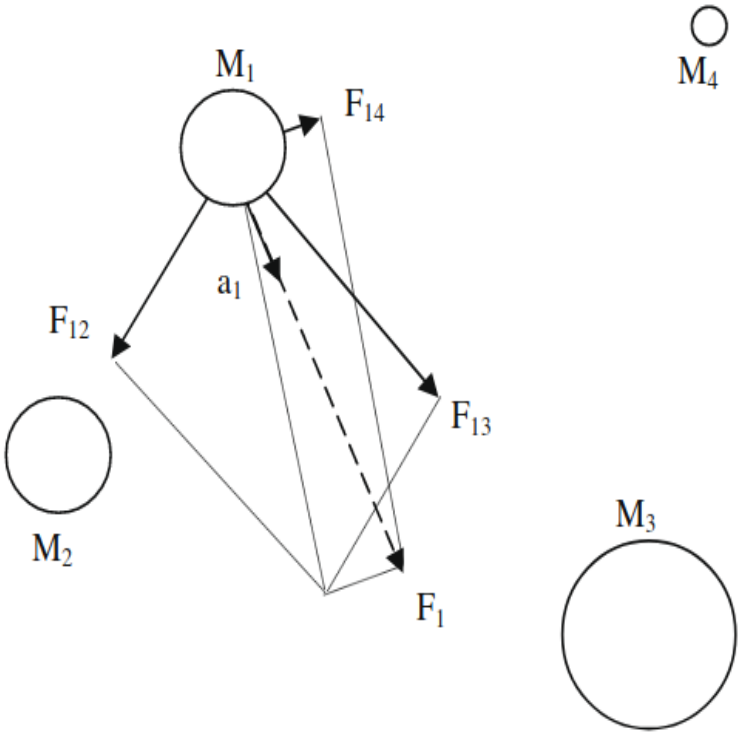
$$a = \frac{F_{ij}}{M_{ii}}, \tag{4.5}$$

kde M_{aj} a M_{pi} predstavujú aktívnu gravitačnú hmotnosť častice i a pasívnu gravitačnú hmotnosť častice j, respektíve M_{ii} predstavuje zotrvačnú hmotnosť častice i.

Hoci zotrvačná hmotnosť, pasívna gravitačná hmotnosť a aktívna gravitačná hmotnosť sú koncepčne odlišné, žiadny experiment nikdy jednoznačne nepreukázal medzi nimi nejaký rozdiel. Teória všeobecnej relativity je založená na predpoklade, že zotrvačná a pasívna gravitačná hmotnosť sú ekvivalentné. Toto je známe ako princíp slabej ekvivalencie. Štandardná všeobecná relativita tiež predpokladá ekvivalenciu zotrvačnej hmotnosti a aktívnej gravitačnej hmotnosti, táto ekvivalencia je niekedy nazývaná ako princíp silného ekvivalentu.[39]

[2] jej hodnota nie je presne definovaná, ale samotný algoritmus ju nepoužíva

Pôsobenie gravitačnej sily na teleso je možné vidieť na obrázku 4.1.



Obrázek 4.1: Pôsobenie gravitačnej sily telesa na okolité telesá[39]

4.2 Popis algoritmu gravitačného vyhľadávania

Algoritmus gravitačného vyhľadávania je inšpirovaný gravitačnou silou a vychádza z vyššie uvedených Newtonových zákonov. Zároveň patrí aj medzi optimalizačné algoritmy založené na populácii jedincov. Za agentov/jedincov sú v tomto algoritme považované objekty. Kvalita týchto objektov je meraná pomocou ich hmotností, inak povedané, kvalita riešenia vyjadrená účelovou funkciu je analogická hmotnosti objektu. Takže väčšie objekty poskytujú lepšie riešenia a pohybujú sa pomalšie, čím je zaručená exploitácia. Všetky objekty sa navzájom priťahujú pomocou gravitačnej sily, pričom táto sila spôsobuje globálny pohyb všetkých objektov smerom k objektom s ťažšou hmotnosťou. Interakcia medzi jednotlivými jedincami prebieha pomocou gravitačnej sily. Každý objekt je popísaný pomocou štyroch špecifikácií:

1. pozícia,
2. aktívna gravitačná hmotnosť,
3. pasívna gravitačná hmotnosť,
4. zotrvačná hmotnosť.

Pozícia objektu odpovedá riešeniu úlohy. Jedná sa o vektor parametrov, pričom ich počet a význam závisí na riešenej úlohe a jej gravitačné a zotrvačné hmotnosti sú určené pomocou fitness funkcie. Každý objekt predstavuje riešenie a algoritmus sa riadi správnym nastavením gravitácie a zotrvačnej hmotnosti. S odstupom času očakávame, že objekty budú priťahované k objektom s najväčšou hmotnosťou. Táto hmotnosť bude predstavovať optimum riešenia vo vyhľadávacom priestore.

Algoritmus gravitačného zrýchlenia sa riadi mierne upravenými Newtonovmi zákonmi:

- zákon príťažlivosti (law of gravity): každá častica priťahuje každú ďalšiu časticu a gravitačná sila medzi dvoma časticami je priamo úmerná súčinu ich hmotností a nepriamo úmerná vzdialenosti medzi nimi (používame R namiesto R^2, kvôli lepším výsledkom),
- zákon pohybu (law of motion): aktuálna rýchlosť akejkoľvek hmoty sa rovná súčtu zlomku jej predchádzajúcej rýchlosti a zmeny rýchlosti. Zmena rýchlosti alebo zrýchlenia akejkoľvek hmoty sa rovná sile pôsobiacej na systém delené hmotnosťou zotrvačnosti.

Algoritmus pracuje na základe nižšie uvedených vzťahov:

$$X_i = (x_i^1, \ldots, x_i^d, \ldots, x_i^n) for i = 1, 2, \ldots, N, \tag{4.6}$$

kde N je počet jedincov a x_i^d je hodnota na pozícii d vo vektore jedinca s indexom i.

Vzťah pre výpočet sily, ktorá v čase t pôsobí na objekt s hmotnosťou i z objektu s hmotnosťou j:

$$F_{ij}^d(t) = G(t)\frac{M_{pi}(t) \times M_{aj}(t)}{R_{ij}(t) + \varepsilon}(x_j^d(t) - x_i^d(t)), \tag{4.7}$$

kde M_{aj} je aktívna gravitačná hmotnosť agenta j, M_{pi} je pasívna gravitačná hmotnosť agenta i, $G(t)$ je gravitačná konštanta v čase t, ε je malá konštanta[3], $R_{ij}(t)$ je Euklidovská

[3] jej hodnotu autor algoritmu presne nedefinoval, ale v existujúcich implementáciách sa využívajú hodnoty 0,0001 a nižšie

vzdialenosť medzi agentmi i a j, pričom ju počítame na základe nasledujúceho vzťahu:

$$R_{ij}(t) = \|X_i(t), X_j(t)\|_2. \tag{4.8}$$

Nasledujúci vzťah slúži na zaručenie stochastického správania algoritmu, kde predpokladáme, že celková sila, ktorá pôsobí na agenta i v dimenzii d, je náhodne vážený súčet $d-tych$ zložiek síl vyvíjaných inými agentami:

$$F_i^d(t) = \sum_{j=1, j\neq i}^{N} rand_j F_{ij}^d(t), \tag{4.9}$$

kde $rand_j$ je náhodné číslo na intervale [0,1]. Ďalej je potrebné vypočítať zrýchlenie agenta i v čase t a smer d, na základe nasledujúceho vzťahu:

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)}, \tag{4.10}$$

kde M_{ii} je zotrvačná $i-teho$ agenta. Rýchlosť a poloha agenta i sa vypočíta nasledovne:

$$v_i^d(t+1) = rand_i \times v_i^d(t) + a_i^d(t), \tag{4.11}$$

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1), \tag{4.12}$$

kde $rand_i$ je číslo získané z rovnomerného náhodného rozloženia na intervale $[0, 1]$.
Gravitačná konštanta G sa inicializuje na začiatku a jej hodnota časom klesá, kvôli presnosti vyhľadávania:

$$G(t) = G(G_0, t). \tag{4.13}$$

Gravitačné a zotrvačné hmotnosti sa počítajú na základe hodnotenia spôsobilosti (hodnoty fitness). Jedná sa o hodnotu, ktorá určuje úspešnosť jedinca pri prehľadávaní stavového priestoru, pričom vyššia hmotnosť predstavuje lepšieho jedinca. Aktualizujeme ich nasledovným spôsobom:

$$M_{ai} = M_{pi} = M_{ii} = M_i, i = 1, 2, \ldots, N, \tag{4.14}$$

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)}, \tag{4.15}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}, \tag{4.16}$$

kde $fit_i(t)$ reprezentuje fitness hodnotu agenta i v čase t. $Worst(t)$ a $best(t)$ sú definované nasledovne a predstavujú najlepšie (najhoršie) riešenie:[39]

$$best(t) = min fit_j(t), j\epsilon\{1, \ldots, N\}^4 \tag{4.17}$$

$$worst(t) = max fit_j(t), j\epsilon\{1, \ldots, N\}^5 \tag{4.18}$$

$$best(t) = max fit_j(t), j\epsilon\{1, \ldots, N\}^6 \tag{4.19}$$

$$worst(t) = min fit_j(t), j\epsilon\{1, \ldots, N\}^7 \tag{4.20}$$

[4] Definované pre problémy hľadania minima
[5] Definované pre problémy hľadania minima
[6] Definované pre problémy hľadania maxima
[7] Definované pre problémy hľadania maxima

Priebeh algoritmu je popísaný pomocou pseudokódu 3:

Algorithm 3 Pseudokód algoritmu gravitačného vyhľadávania

1: Náhodne inicializujeme populáciu jedincov a identifikujeme prehľadávací priestor
2: **while** nie sú splnené ukončujúce podmienky **do**
3: Vypočítame fitness hodnotu pre každého jedinca
4: Aktualizujeme hodnoty $G(t)$ 4.13, $best(t)$ 4.17, $worst(t)$ 4.18 a $M_i(t)$ 4.16 pre $i = 1, 2, ..., N$.
5: Vypočítame celkovú silu v rôznych smeroch podľa vzťahu 4.9
6: Vypočítame zrýchlenie a rýchlosť podľa vzťahov 4.10 a 4.11
7: Aktualizujeme pozície agentov podľa vzťahu 4.12
8: **end while**

4.3 Black Hole Algorithm

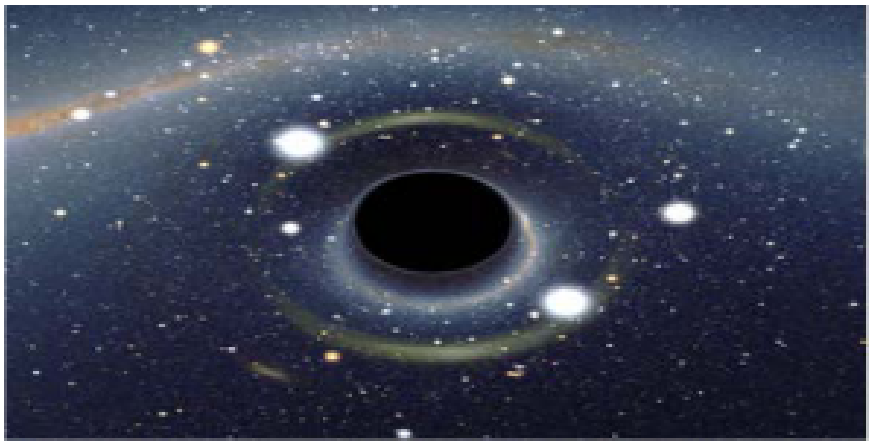
Algoritmus čiernych dier [8] je jedným z algoritmov inšpirovaných fyzikálnymi a chemickými javmi. Prvýkrát bol použitý v roku 2008, kde bol považovaný skôr za rozšírenie algoritmu PSO. Keďže sa ukázal, ako efektívny algoritmus aj pri riešení viacrozmerných funkcií, viedlo to k jeho širšiemu skúmaniu a mnohým variantám.

Čierna diera (môžeme vidieť na obrázku 4.2) je oblasť časopriestoru, ktorej gravitačné pole je také silné, že nič, čo doň vstúpi sa už nedostane von, dokonca ani svetlo. Podľa teórie všeobecnej relativity čierna diera vzniká tak, že dostatočne kompaktná hmotnosť deformuje časopriestor a vytvorí čiernu dieru. Podľa inej teórie čierna diera vzniká vďaka procesu, ktorý nazývame gravitačné zrútenie. Vo väčšine prípadov k nemu dochádza u hviezd. Gravitačné zrútenie nastane vtedy, keď vnútorný tlak objektu nie je dostatočne veľký, aby odolal vlastnej gravitácii. Pri hviezdach k nemu môže dôjsť napríklad, keď má hviezda príliš málo paliva na udržanie svojej teploty. Akonáhle sa proces gravitačného zrútenia spustí, nie je možné ho zastaviť žiadnym spôsobom.

Okolo čiernej diery je matematicky definovaný povrch nazývaný horizont udalostí, ktorý označuje bod, odkiaľ niet návratu. Ak sa objekt dostane blízko horizontu udalostí, respektíve prekročí Schwarzschildov polomer, bude absorbovaný do čiernej diery a natrvalo zmizne.

Čierne diery nie je možné pozorovať, nakoľko ich existenciu je možné zistiť len tak, že objekt, ktorý sa dostane do jej blízkosti zmizne. Svoje meno dostala na základe toho, že absorbuje všetko svetlo, ktoré dopadne na horizont udalostí. [36]

[8] Existujú dva typy čiernych dier, a to rotujúce a statické čierne diery, pričom algoritmus je inšpirovaný statickými čiernymi dierami



Obrázek 4.2: Čierna diera

Popis algoritmu čiernych dier

Algoritmus čiernych dier patrí medzi metódy založené na populácii. Rovnako ako iné algoritmy z tejto kategórie, generuje populáciu kandidátnych riešení, ktoré distribuuje náhodne v priestore.

Terminológia

- čierna diera(black hole): pojmom čierna diera označujeme najlepšieho jedinca z pomedzi všetkých jedincov, pričom sa určuje v každej iterácii,
- hviezdy(stars): všetci jedinci(s výnimkou čiernej diery) sú označovaní ako hviezdy, pričom čierna diera je jednou z hviezd,
- pohyb(movement): všetci jedinci sa snažia priblížiť smerom k čiernej diere na základe ich aktuálnej polohy a náhodného čísla.

Na začiatku algoritmus čiernych dier pracuje s počiatočnou populáciou kandidátnych riešení optimalizačného problému a účelovou funkciou, ktorá je pre nich vypočítaná.

V každej iterácii je vybraný najlepší jedinec, ktorého označujeme ako "čierna diera", ktorý je vybratý z pomedzi všetkých hviezd. Ostatní jedinci reprezentujú hviezdy. Po fáze inicializácie začne čierna diera priťahovať ostatné hviezdy do svojej blízkosti.

Keď sa hviezda dostane príliš blízko k čiernej diere, prekročí Schwartszildov polomer, je čiernou dierou pohltená. Pohltená hviezda sa nahradí novou hviezdou náhodne vygenerovanou vo vyhľadávacom priestore.

Výpočet hodnoty fitness funkcie

Na začiatku máme populáciu $P(x) = \{x_1^t, x_2^t, x_3^t, \ldots, x_n^t\}$ náhodne vygenerovaných kandidátnych riešení(hviezd), ktoré sú náhodne umiestnené v prehľadávacom priestore alebo nejakom probléme.

Následne zistíme celkovú fitness hodnotu populácie podľa nasledujúcich vzťahov:

$$f_i = \sum_{i=1}^{popSize} eval(p(t) \tag{4.21}$$

$$f_{BH} = \sum_{i=1}^{popSize} eval(p(t) \tag{4.22}$$

kde f_i a f_{BH} sú fitness hodnoty čiernej diery a f_{tej} hviezdy v inicializovanej populácii.

Hodnoty hviezd v populácii sú odhadnuté a zvolí sa kandidát s najlepšou hodnotou fitness f_i, ktorý reprezentuje čiernu dieru. Zvyšní kandidáti sú hviezdy. Čierna diera absorbuje hviezdy v svojom okolí a tie sú nahradené novými jedincami. Po inicializácii prvej čiernej diery sa hviezdy pohybujú smerom k nej, pričom hviezdy, ktoré prekročia Schwarzschildov polomer sú ňou pohltené. Priebežne sa počítajú nové fitness hodnoty jednotlivých hviezd. V prípade, že dostaneme novú najlepšiu fitness hodnotu, daná hviezda nahradí čiernu dieru a sama sa ňou stane.

Miera absorpcie hviezd pomocou čiernej diery

Čierna diera absorbuje hviezdy okolo seba, zároveň sa hviezdy k nej približujú na základe vzťahu:

$$X_i(t) = X_i(t) + rand \times (X_{BH} - X_i(t)) \sum_{i=1}^{popSize} eval(p(t), \tag{4.23}$$

kde $i = 1, 2, 3, \ldots, n$, X_i^t a X_i^{t+1} sú pozície i_{tej} hviezdy v iterácii t respektíve $t + 1$. X_{BH} je pozícia čiernej diery vo vyhľadávacom priestore, $rand$ je náhodné číslo na intervale $[0, 1]$ a n je počet hviezd. Kým sa hviezdy presúvajú smerom k čiernej diere, nejaká iná hviezda môže mať lepšiu hodnotu ako aktuálna čierna diera. V takomto prípade ju nahradí, stáva sa novou čiernou dierou a začne priťahovať hviezdy v svojom okolí.

Pravdepodobnosť prekročenia horizontu udalostí počas pohybu hviezd

Pravdepodobnosť prekročenia horizontu udalostí slúži na získanie lepších výsledkov. Každá hviezda, ktorá prekročí horizont udalostí zanikne a je nahradená novou hviezdou, ktorá je náhodne umiestnená v priestore. Počet hviezd musí byť vždy rovnaký. Prekročenie horizontu udalostí sa počíta pomocou Schwarzschildovho polomeru, ktorý je daný vzťahom:

$$R = \frac{f_{BH}}{\sum_{i=1}^{N} f_i} \tag{4.24}$$

kde f_i a f_{BH} sú fitness hodnoty i_{tej} hviezdy a čiernej diery, N je počet hviezd. Keď je vzdialenosť medzi hviezdou a čiernou dierou menšia ako Schwarzschildov polomer (R), hviezda je pohltená čiernou dierou a nahradená novou náhodne vygenerovanou hviezdou v priestore.[36][1]

Pseudokód algoritmu je možné vidieť na 4:

Algorithm 4 Pseudokód algoritmu čiernych dier

1: Náhodne inicializujeme populáciu hviezd v prehľadávacom priestore $P(x) = \{x_1^t, x_2^t, x_3^t, \ldots, x_n^t\}$. Náhodne vygenerovaná populácia kandidátnych riešení (hviezd) je umiestnená v prehľadávacom priestore nejakého problému alebo funkcie
2: **loop**
3: Pre každú hviezdu vypočítame jej hodnotiacu funkciu podľa vzťahu 4.21 a 4.22
4: V populácii nájdeme hviezdu s najlepšou hodnotou fitness funkcie a označíme ju ako čierna diera
5: Vymeníme pozície všetkých hviezd na základe rovnice 4.23
6: **if** hviezda dosiahne pozíciu s lepšou hodnotou ako má čierna diera **then**
7: vymeníme ich pozície
8: **end if**
9: Vypočítame Schwarzschildov polomer podľa vzťahu 4.24
10: **if** hviezda prekročí horizont udalostí čiernej diery **then**
11: nahradíme ju novou hviezdou náhodne vygenerovanou v prehľadávacom priestore
12: **end if**
13: **if** je splnená ukončujúca podmienka (maximálny počet iterácii alebo dostatočne dobrá hodnota fitness funkcie) **then**
14: koniec cyklu
15: **end if**
16: **end loop**

4.4 Praktické využitie algoritmov inšpirovaných gravitáciou

Oba algoritmy inšpirované gravitáciou majú mnoho modifikácií, prípadne z nich vznikli rôzne hybridné algoritmy, ktoré je možné použiť v rôznych oblastiach. Nižšie je uvedených niekoľko zaujímavých oblastí a problémov, pri ktorých boli tieto algoritmy využité.

Algoritmus gravitačného vyhľadávania, je relatívne nový algoritmus, ale aj napriek tomu má širokú oblasť využitia. Jednou z oblastí je napríklad oblasť **stavebníctva**, kde bol použitý na riešenie *optimálneho dizajnu oporných múrov*. Konkrétne sa jedná o železobetónovú opornú stenu, pri ktorej konštrukcii je potrebné dbať na to, aby konštrukcia bezpečne podopierala zásypovú zeminu, bola stabilná voči možnosti prevrátenia a iné. Algoritmus bol zameraný na optimalizáciu hmotnosti, nákladov a množstva emisií zabudovaného CO_2.[28]

Ďalšou oblasťou využitia algoritmu je oblasť **chémie**, kde bol algoritmus využitý na *riešenie optimalizácie problémov chemickej kinetiky*. V tomto prípade bol algoritmus využitý na vyhľadávanie kinetických parametrov dvoch chemických procesov, predreformovanie propánu na Ni-katalyzátore a katalytická izomerizácia pentán-hexánovej frakcie. Algoritmus gravitačného vyhľadávania bol pri riešení týchto problémov porovnávaný s algoritmom harmonického vyhľadávania. Pričom algoritmus GSA sa ukázal ako efektívnejší, zvlášť pri riešení druhého problému.[41]

Ďalšou oblasťou využitia je **zdravotníctvo**, kde bol algoritmus využitý napríklad na *voľbu atribútov biomedicínskych dát*. Tieto dáta sa ďalej používajú na predikciu ochorení. Pri využití algoritmu gravitačného vyhľadávania došlo k redukcii atribútov o 66%.[35]

Ďalšou možnou oblasťou využitia je oblasť **energetiky**, kde bol algoritmus využitý napríklad pri *správe obnoviteľných energií* vo vzdialených oblastiach, pričom testovacou oblasťou bola malá odľahlá zóna v Nigérii. Pracovalo sa s turbínovým čerpadlom, ktoré je hybridným systémom, ktoré slúži na dodávku vody aj elektriny a uskladnenie vody v

nádrži.[17] Ďalším príklad bolo jeho využitie na nájdenie optimálneho riešenia pre *problém kombinovaného ekonomického a emisného dispečingu*. Tento problém je postavený na tom, že plánovanie generátorov, by malo byť vykonané tak, aby sme pracovali s najmenšími možnými nákladmi a najnižšími emisiami a zároveň je potrebné dodržať obmedzujúce podmienky. Algoritmus GSA sa pri riešení tohto problému ukázal ako veľmi úspešný.[19]

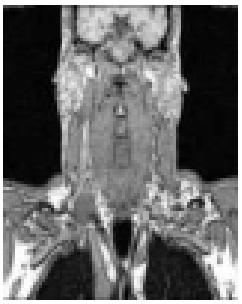
Algoritmus čiernych dier je taktiež možné využiť v rôznych iných oblastiach. Úspešne bol využitý napríklad v oblasti **medicíny**, kde sa využíva napríklad na *zlepšenie rozpoznávania emócií zo snímok EEG*. Prvým krokom spracovania signálu EEG je eliminácia šumu, ktorá sa dá robiť manuálne alebo automatizovane. Ďalším krokom je určenie vektoru atribútov(feature vector), ktorý slúži na vytvorenie klasifikačného modelu. Algoritmus čiernych dier bol využitý na elimináciu šumu a určenie vektoru atribútov pre SVM, pričom pri testovaní dosiahol úspešnosť viac ako 92%. [34] Ďalej bol modifikovaný algoritmus BH využitý na *vylepšenie medicínskych snímok* s nízkym kontrastom, konkrétne na odhadnutie najlepších hodnôt parametrov prenosovej funkcie. Na obrázku 4.3 je možné vidieť snímok pred použitím a po použití algoritmu. [37] Taktiež bol algoritmus BH využitý pri *diagnostikovaní rakoviny prostaty*, pomocou snímok MR. Konkrétne na segmentáciu vstupného obrazu na základe viacúrovňovej techniky prahovania, vďaka čomu je možné na obrázku identifikovať oblasti v ktorých by sa nádor mohol nachádzať.[44]

Ďalšou oblasťou v ktorej bol algoritmus čiernych dier použitý, je oblasť **energetiky**, kde bol využitý napríklad na *optimalizáciu kombinovaného chladiaceho a vykurovacieho systému pomocou tepelného čerpadla*. Po optimalizácii tohto systému pomocou algoritmu čiernych dier došlo k úspore energie vo výške 5.5%, zníženiu oxidu uhličitého o 3.5% a zníženie nákladov dosahuje 3.5%.[11] Ďalším zaujímavým spôsobom jeho využitia je štúdia zameraná na *komerčné straty* v Brazílii, kde sa pomocou binárneho algoritmu čiernych dier charakterizujú zlodeji energie. Podľa Brazílskeho úradu pre elektrickú energiu stratila Brazília v roku 2011 asi 4 miliardy dolárov na komerčných stratách za elektrickú energiu. Za komerčné straty sa považujú straty netechnického charakteru, kde je energia dodaná koncovým zákazníkom, ale nie je za ňu zaplatené. Jedná sa o celosvetový problém, akurát veľkosť týchto strát sa líši v jednotlivých krajín na základe príjmu, vzdelania, miery násilia a podobne. Pre porovnanie v Brazílii sa komerčné straty pohybujú na úrovni okolo 17 %, vo Veľkej Británii sa tieto straty pohybujú na úrovni okolo 1 %. Hlavnými príčinami komerčných strát sú krádeže elektromerov a manipulácia s nimi. Avšak, je veľmi náročné detekovať a kontrolovať, kde a kedy takéto situácie nastanú. Algoritmus BH bol v tomto prípade využitý na voľbu relevantných atribútov pre rozhodovanie. Pri voľbe atribútov pomocou algoritmu čiernych dier sa počet správne detekovaných nezrovnalostí pri komerčných spotrebiteľoch zlepšil z 58.87 % na 64.81 % a pri priemyselných spotrebiteľoch zo 64.14 % na 83.76 %.[38]

Ďalšie využitie algoritmu čiernych dier môžeme vidieť v **armáde**, kde bol algoritmus použitý pri *plánovaní misií bezpilotných vojenských lietadiel*.[23]

Ďalšou oblasťou využitia algoritmu je oblasť **financií**, konkrétne pri *výbere aktív a tvorbe portfólia aktív*, pričom hlavným cieľom je maximalizovať očakávaný výnos a minimalizovať riziko. Algoritmus čiernych dier bol pri riešení tohto problému použitý na Iránskom trhu s akciami.[32]

(a) Pôvodný snímok



(b) Snímok po aplikovaní algoritmu čiernych dier

Obrázek 4.3: Porovnanie snímok pred použitím algoritmu čiernych dier a po jeho použití

Kapitola 5

Optimalizácia NP-ťažkých úloh

V tejto kapitole sú popísané NP-ťažké úlohy, ktorých optimalizácia je cieľom tejto práce. Konkrétne sa jedná o veľmi časté úlohy, ktoré je možné v praxi využiť v rôznych smeroch. A to práve problém obchodného cestujúceho a problém plnenia batohu (knapsack problem) a spôsob, akým sú využívané pri optimalizáciách v praxi.

NP-ťažké úlohy reprezentujú problémy, ktoré nie je možné riešiť v polynomiálnom čase. Na riešenie NP-ťažkých problémov neexistujú obecné optimalizačné algoritmy, ktoré by tieto problémy vedeli riešiť v reálnom čase. Avšak, riešenie mnohých NP-problémov je užitočné v praxi, a preto vzniklo veľké množstvo metaheuristických a stochastických algoritmov, ktoré sú schopné tieto problémy optimalizovať.

Pod pojmom optimalizácia si predstavujeme hľadanie riešenia s prijateľnými nárokmi na výpočetné technológie v prijateľnom čase, avšak, nemáme zaručené, že nájdené riešenie je najlepšie možné. Inak povedané, je to najlepšie možné riešenie v danom čase za daných podmienok výpočetnej techniky. Je pravdepodobné, že pri väčšom množstve času a lepšej technológii, by bolo možné nájsť lepšie riešenie.

V tejto práci je skúmané využitie optimalizačných algoritmov založených na inteligencii skupiny. A to konkrétne pri riešení už zmienených úloh, pri rôznych dátových sadách a rôznych modifikáciách za účelom zvýšenia kvality optimalizácie.

5.1 Problém obchodného cestujúceho

Problém obchodného cestujúceho (traveling salesman problem - TSP), bol skúmaný už v 18. storočí Írskym matematikom Wiliamom Hamiltonom a britským matematikom Thomasom Kirkmanom. Je jedným z najpopulárnejších problémov z množiny NP, taktiež je jedným z najťažších. Jedná sa o problém kombinatorickej optimalizácie. Riešenie tohto problému je aplikovateľné v mnohých praktických oblastiach, čo veľmi zvyšuje potrebu jeho efektívneho riešenia. Keďže jeho deterministické riešenia majú exponenciálnu zložitosť je snaha o nájdenie riešení pomocou stochastických algoritmov.

Problém obchodného cestujúceho pozostáva zo zoznamu miest [1] a vzdialeností medzi nimi. Pričom cieľom je nájsť takú trasu, pri ktorej bude každé mesto navštívené len raz a cestujúci skončí v meste, z ktorého vychádzal a zároveň, aby táto vzdialenosť bola čo najkratšia. Jeho komplexnosť je pre n miest definovaná ako počet miest, ktoré sa majú navštíviť. Pričom celkový počet možných trás, ktoré pokrývajú všetky mestá, možno vyjadriť ako súbor prijateľných riešení TSP a je daný ako $(n-1)!/2$.

[1] môžu byť zadané svojimi menami alebo súradnicami

Problém obchodného cestujúceho sa ďalej klasifikuje ako symetrický problém obchodného cestujúceho (sTSP), asymetrický problém obchodného cestujúceho (aTSP) a viacrozmerný problém obchodného cestujúceho (mTSP). Cieľom tejto práce bolo skúmať symetrický problém obchodného cestujúceho.

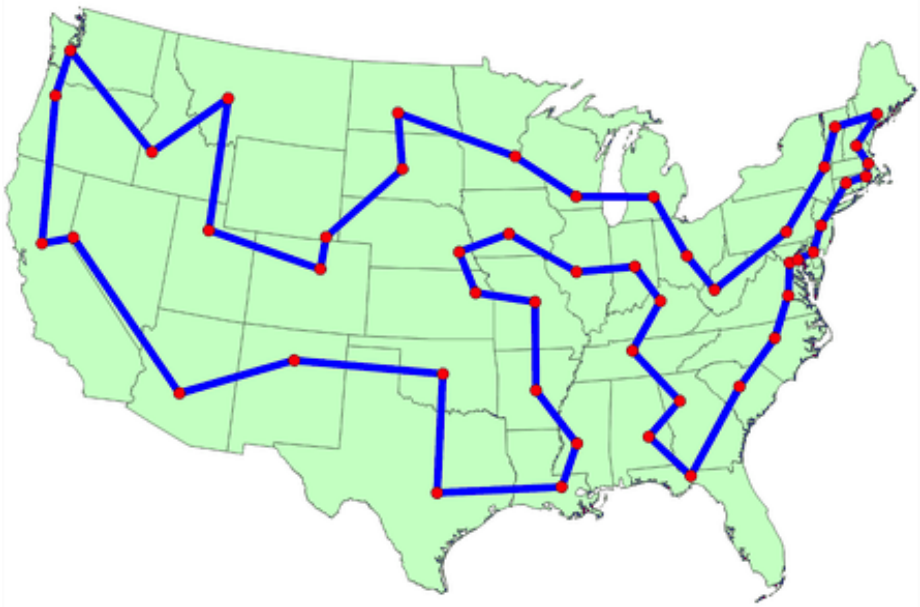
Symetrický problém obchodného cestujúceho **sTSP** je definovaný ako $V = \{v_1, ..., v_n\}$, ktoré predstavuje súbor miest, $A = \{(r, s) : r, s \in V\}$ ktoré predstavuje súbor hrán, a $d_{rs} = d_{sr}$ ktoré predstavuje mieru nákladov spojenú s hranou $(r, s) \in A$. Cieľom symetrického problému obchodného cestujúceho je nájsť minimálnu dĺžku trasy, pričom sa každé mesto navštívi len raz.

O asymetrický problém obchodného cestujúceho **aTSP** sa jedná v prípade, ak $d_{rs} \neq d_{sr}$ pre aspoň jednu dvojicu (r, s), a teda vzdialenosť medzi dvomi mestami v rôznych smeroch môže byť odlišná, prípadne na jednom smere nemusí cesta existovať vôbec.

Viacrozmerný problém obchodného cestujúceho **mTSP** je definovaný tak, že v danom súbore uzlov je umiestnených m obchodníkov v jednom depe. Zvyšné uzly (mestá), ktoré majú byť navštívené, označujeme prostredné uzly (inetermediate nodes). Potom viacrozmerný problém obchodného cestujúceho spočíva v nájdení trás pre všetkých m obchodníkov, ktorí začínajú a končia v depe. Takže každý prostredný uzol je navštívený práve raz a celkové náklady na návštevu všetkých uzlov sú minimalizované. Metrika nákladov sa môže definovať vo forme vzdialenosti, času, atď.

Viacrozmerný problém obchodného cestujúceho má viacero variant. Napríklad môžeme pracovať s jedným alebo viacerými depami, kde v prípade jedného depa všetci obchodníci ukončia svoje trasy na jednom bode. Zatiaľ, čo v prípade viacerých dep sa môžu obchodníci, buď vrátiť do svojho pôvodného depa alebo sa vrátiť do akéhokoľvek depa. Pričom počet obchodníkov v každom depe zostáva rovnaký. Taktiež môžeme mať pevný alebo premenný počet obchodníkov. V niektorých prípadoch sa využíva takzvaný časový rámec, kde musia byť niektoré uzly navštívené v určitých časových obdobiach nazývaných časové okná. V tomto prípade sa jedná o rozšírenie mTSP a označuje sa ako viacrozmerný problém obchodného cestujúceho so špecifikovaným časovým rámcom (mTSPTW). Aplikácia mTSPTW sa veľmi dobre prejavuje v problémoch s plánovaním letov. Pracuje sa aj s inými obmedzeniami ako počet uzlov, ktoré každý obchodník môže navštíviť, maximálna alebo minimálna vzdialenosť, ktorú obchodník precestuje.[31]

Ukážku problému TSP je možné vidieť na obrázku 5.1.



Obrázek 5.1: Ukážka riešenia TSP pre 48 miest USA
dostupné z: Cornell

Problém obchodného cestujúceho je aplikovateľný v mnohých oblastiach. Využíva sa napríklad pri vŕtaní spojovacích dosiek, a to nasledovným spôsobom. Na prepojenie vodiča na jednej vrstve s vodičom na inej vrstve alebo na umiestnenie pinov integrovaných obvodov, kde sa musia vŕtať diery cez dosku, pričom môžu mať rôzne veľkosti. Pri vŕtaní dier s rôznymi priemermi sa musí hlava stroja vrátiť k nástrojovému boxu a vymeniť vŕtacie zariadenie. Keďže sa jedná o časovo náročný proces, optimalizácia spočíva v tom, že si vyberieme určitý priemer, vyberieme všetky diery rovnakého priemeru a až následne vymieňame vŕtací prístroj. Následne tento postup opakujeme. Problém obchodného cestujúceho sa v tomto prípade aplikuje tak, že pre každý priemer máme mestá, ktoré reprezentujú pozície dier, ktoré môžeme vŕtať tým istým vŕtacím nástrojom. Vzdialenosť medzi dvomi mestami je určená časom potrebným na pohyb vŕtacej hlavy z jednej pozície na druhú, pričom cieľom je minimalizovať čas cesty pre vŕtaciu hlavu stroja.

Ďalšou z oblastí využitia problému obchodného cestujúceho je pri analýze štruktúry kryštálov pomocou rontgenového difraktometra. Informácie o štruktúre kryštálu sa získavajú tak, že sa meria intenzita rontgenového odrazu z rôznych pozícií. Meranie nie je časovo náročná operácia, ale existuje značný náklad na čas na umiestnenie, pretože pre niektoré experimenty musí byť realizovaných až niekoľko stoviek tisíc pozícií. Optimalizácia v tomto prípade spočíva v tom, že sa hľadá taká postupnosť snímania, ktorá minimalizuje čas vynaložený na umiestnenie.

Ďalšími oblasťami, v ktorých je možné tento problém aplikovať, je napríklad problém manipulácie materiálu v sklade. Tento problém úzko súvisí s problémom smerovania vozidiel (vehicle routing problem), ktorý používame napríklad v prípade, že chceme doručovať nejaké zásielky. Pričom tieto zásielky chceme doručiť v najkratšom možnom čase s najmenším možným počtom áut.

Ďalšou z oblastí, kde bol tento problém aplikovaný v reálnom živote, je pri revízii plynových turbínových motorov lietadiel. Taktiež bol aplikovaný pri pripájaní komponent na počítačovú dosku. Využíva sa aj pri optimalizácií postupov vo výrobe, ako napríklad v elek-

tronike alebo automobilovom priemysle. Existujú tiež jeho rôzne variácie, ktoré je možné aplikovať v reálnom živote, ako napríklad problém obchodného cestujúceho s obmedzenými zdrojmi, problém orientačného behu.[18]

Viacrozmerný problém obchodného cestujúceho je taktiež možné aplikovať na rôzne oblasti. Napríklad problém s plánovaním tlače, plánovanie trasy školského autobusu, problém s plánovaním pohovoru, problém plánovania valcovania za tepla a mnohých iných. [31]

5.2 Problém batohu

Problém batohu (knapsack) je jedným z najštudovanejších kombinatorických problémov. Rovnako ako problém obchodného cestujúceho, patrí medzi do množiny NP problémov. Jeho zložitosť pri riešení deterministickými algoritmami je exponenciálna. Problém batohu má mnoho variant a rozšírené použitie v reálnom živote.

Pôvodný problém je postavený na tom, že máme n položiek, pričom každá má svoju pridruženú hmotnosť a zisk. Cieľom je vybrať podmnožinu n položiek, ktorá maximalizuje zisk a zároveň sa neprekračuje kapacitu batohu. Názov problému je odvodený zo situácie, v ktorej by sa horolezec mal rozhodnúť pre určitý počet položiek, ktoré má zahrnúť do svojho batohu a ktoré by mohol použiť počas svojej cesty. Problém spočíva v tom, že horolezec by mal vybrať položky s väčším ziskom, ale zároveň by nemal prekročiť kapacitu batohu.

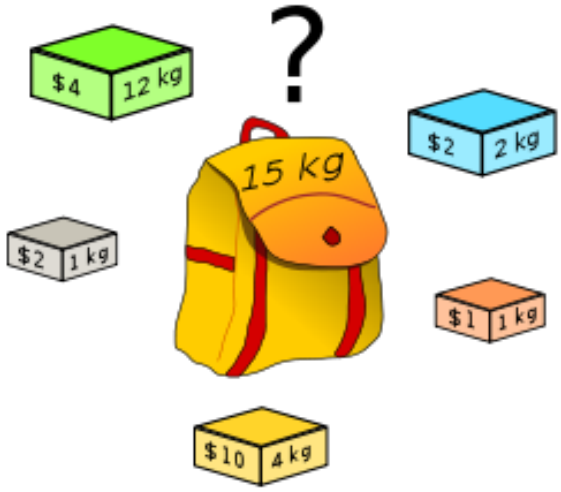
Rovnako, ako pri probléme obchodného cestujúceho, existuje aj pri probléme batohu viacero variácií, z ktorých každý sa líši v rôznych obmedzeniach a podmienkach. Medzi najznámejšie patria binárny problém batohu (0-1 Knapsack Problem), ktorého skúmanie bolo cieľom tejto práce. Predpokladajme, že máme n položiek, pričom každá položka j má hodnotu v_j a váhu w_j. Nech x_j je počet kópií vybranej položky j. V binárnej variante problému je každá položka buď vybraná raz alebo ani raz. Preto môže x_j nadobúdať hodnotu nula alebo jedna. Predpokladajme, že W je maximálna váha, ktorú je možné niesť v batohu a, že váhy a hodnoty položiek sú nezáporné, cieľom je maximalizovať súčet hodnôt vybraných položiek bez prekročenia limitu batohu, v tomto prípade W.

Ďalšie varianty problému batohu sú neobmedzený problém batohu (Unbounded Knapsack Problem), kde môžeme zahrnúť ľubovoľný počet kópií každej položky. Problém viacnásobného batohu (Multiple Knapsack Problem), kde máme viacero batohov a snažíme sa vyplniť ich všetky s položkami tak, aby sme maximalizovali zisk. Obmedzený problém batohu (Bounded Knapsack Problem), ktorý je podobný neobmedzenému problému batohu, ale s obmedzením na maximálny počet kópií každej položky. Problém čiastočného batohu (Fractional Knapsack Problem), kde je možné zahrnúť časti položky do batohu, umožňujúc čiastočný výber. Problém súhrnných súčtov (Subset Sum Problem), ktorý sa snaží nájsť podmnožinu prvkov s celkovou hodnotou rovnajúcou sa danému cieľu.

Problém batohu má široké využitie pre rôzne praktické situácie v reálnom živote. Je možné ho využiť napríklad pri ukladaní dátových súborov tak, že máme n položiek, ktoré reprezentujú n vypočítaných dátových súborov, ktoré treba uložiť a vieme, že je k dispozícii iba W bytov úložného priestoru. Preto je cieľom vybrať podmnožinu súborov tak, aby celková veľkosť neprekročila hmotnosť W a celkový výpočtový čas uložených súborov bol maximalizovaný. Ďalšou možnosťou je daný problém využiť pri investovaní v prípade, že akcionár disponuje kapitálom x dolárov a chce investovať do niektorých z v možných investícií. Vzhľadom na to, že poznáme očakávaný zisk a náklady na každú investíciu, jedná sa o binárny problém batohu. Ďalšou aplikáciou binárneho problému batohu je problém nakladania nákladových kontajnerov. Taktiež sa využíva v oblasti kryptografie, pri viacná-

sobnom plánovaní procesora, pridaní úloh medzi autonómnych agentov, aukciách, plánovaní výroby a iných oblastiach. [3]

Ukážku tohto problému je možné vidieť na obrázku 5.2.



Obrázek 5.2: Ukážka problému plnenia batohu
dostupné z: Wikipedia

Kapitola 6

Implementácia a vybrané modifikácie

Táto kapitola je zameraná na popísanie implementácie a modifikácií, ktoré boli využité pri experimentovaní. V časti 6.1 je popísaný spôsob implementácie aplikácie, implementácia jednotlivých algorimov, spôsob načítania a reprezentácie problému obchodného cestujúceho a problému batohu. Ďalej sú popísané grafy, ktoré aplikácia využíva. V časti 6.2 sú popísané jednotlivé modifikácie spolu s popisom, pri ktorých algoritmoch ich je možné použiť.

6.1 Implementácia

Pred implementáciou aplikácie bolo najskôr potrebné previesť analýzu požiadaviek. Z analýzy ďalej vychádzajú väčšie celky, na ktoré sa bolo pri implementácii potreba zamerať:

- optimalizátory,
- konfigurácia,
- načítanie a reprezentácia problémov,
- grafový výstup.

Na základe analýzy požiadaviek a existujúcich riešení bol na implementáciu zvolený jazyk `Python`, nakoľko sa jedná o veľmi používaný programovací jazyk. Jeho častá použiteľnosť je spôsobená jeho jednoduchosťou, ľahko čitateľnou syntaxou a zároveň jednoduchou údržbou kódu. Vzhľadom na jeho častú použiteľnosť má rozsiahly systém knižníc, mnohé z nich sú veľmi vhodné pri riešení optimalizačných úloh. Jedná sa napríklad o knižnicu `NumPy`[1], ktorá bola využitá v tejto práci a poskytuje efektívne nástroje pre numerické výpočty. Jazyk `Python` má taktiež dobrú podporu pre vizualizáciu dát. V tejto práci boli konkrétne použité knižnice `Matplotlib`[2] a `Seaborn`[3]. Zároveň som narazila na už existujúce implemntácie algoritmu optimalizácie hejnom častíc a algoritmu gravitačného vyhľadávania v jazyku `Python`, pričom tieto implementácie boli s úpravami využité v aplikácii.

[1] https://numpy.org/
[2] https://matplotlib.org/
[3] https://seaborn.pydata.org/

Analýza problému

Pre návrh aplikácie boli kľúčové tieto parametre:

- aplikácia by mala mať spoločný spúšťač pre všetky optimalizátory,
- parametre aplikácie aj optimalizátorov je možné jednoducho a rýchlo meniť a daný experiment spustiť znova,
- pre vytvorenie box plotov je mnohokrát nutné spustiť algoritmus s rovnakými parametrami a vypísať výsledky z každého behu,
- aplikácia bude vykresľovať do grafu aktuálnu fitness hodnotu optimálneho riešenia,
- pre problém obchodného cestujúceho aplikácia vykresľuje aj vizualizáciu nájdenej cesty.

Ako prvé bolo treba nájsť spoločné črty vybraných optimalizačných algoritmov pre návrh rozhrania optimalizátorov. Vďaka spoločnému rozhraniu je možné algoritmy spúšťať rovnakým spôsobom. Všetky algoritmy pracujú so stavovým priestorom, v ktorom sú objekty ohodnotené fitness funkciou. Rozdielom sú parametre, s ktorými algoritmy pracujú. Pre zjednodušenie rozhrania bolo rozhodnuté, že konfigurácia optimalizátora bude predaná dátovou triedou špecifickou pre každý optimalizátor.

Táto konfigurácia bola rozšírená na celú aplikáciu. Vďaka tomu je možné spustiť v jednom behu aplikácie všetky optimalizátory pre všetky problémy, čo umožňuje rýchlejšie vykonávanie experimentov. Aby bolo možné aplikáciu spustiť viackrát rovnakým spôsobom, prípadne strojovo generovať rôzne konfigurácie, je možné ju načítať zo súboru.

Keďže táto práca rieši dva rôzne problémy, bolo ich tiež potrebné abstrahovať. V predchádzajúcom odstavci bolo načrtnuté, že ich spoločnou vlastnosťou je ich ohodnotenie. Objekt pre samotné úlohy spracovávaný optimalizátormi, teda bude obsahovať fitness hodnotu a kompozíciou objekt špecifický pre reprezentáciu riešenej úlohy. Objekt pre reprezentáciu úlohu, ale tiež bude mať spoločné rozhranie na výpočet fitness hodnoty, výpis riešenia a jeho nahradenie [4]. Metódy na aktualizáciu fitness hodnoty a nahradenie riešenia sú potrebné na aktualizáciu parametrov kandidátneho riešenia počas behu algoritmu. Metóda na výpis riešenia sa zase použije pre vizualizáciu a výpis výsledku.

Aby aplikácia mohla vykresľovať grafy z každého optimalizátora, je nutné v každej iterácii získať aktuálne riešenie úlohy. Toto bolo vyriešené tak, že kontrola nad iteráciami algoritmu je zabezpečená spúšťacím programom a každý optimalizátor poskytuje metódu na vykonanie jednej iterácie. Po každej iterácii je vrátené najlepšie riešenie, z ktorého sa následne vyberú všetky údaje potrebné pre vykreslenie grafov a výpis výsledkov.

Optimalizátory

Každý optimalizátor je implementovaný vo vlastnom module balíka optimizers a dedí rozhranie z abstraktnej triedy `BaseOptimizer`, ktoré už bolo načrtnuté v predchádzajúcej podkapitole. Okrem toho ešte toto rozhranie obsahuje virtuálnu metódu pre inicializáciu populácie, ktorá sa vykonáva zo spúšťača, keďže každý inicializátor má len metódu pre spustenie jednej iterácie. Pre porovnanie, všetky pôvodné implementácie algoritmov obsahovali metódu pre beh celého algoritmu. Tie zahŕňali aj incilizáciu, ale v tejto aplikácii má optimalizátor iba metódu, ktorá sa vykoná v každej iterácii, kde sa nemôže vykonať

[4] nachádza sa v `Optimizable v module problems/base.py`

inicializácia. Optimalizátor algoritmu optimalizácie hejnom častíc bol implementovaný na základe už existujúcej implementácie [5], ktorá bola prispôsobená pre použitie v tejto aplikácii. Optimalizátor algoritmu čiernych dier bol naimplementovaný na základe odbornej literatúry spomínanej v 4.3. Optimalizátor pre algoritmus gravitačného vyhľadávania, bol naimplementovaný na základe dostupnej implementácie [6]. Táto implementácia bola inšpirovaná implementáciou, ktorú zverejnil autor pôvodného článku o algoritme gravitačného vyhľadávania a rovnako, ako v prípade PSO bola prispôsobená pre potreby aplikácie.

Pretože máme dva typy problémov, každý vyžaduje iný spôsob inicializácie. Preto tiež rozhranie obsahuje metódy `init_knapsack a init_tsp`, ktoré implementuje načítanie problému zo súboru podľa typu úlohy, ktorú má práve optimalizátor riešiť. Tu je dôležité poznamenať, že problém batohu je maximalizačný problém pri čom problém obchodného cestujúceho je minimalizačný problém. Preto optimalizátory tiež implementujú rôzne porovnávacie metódy pre každý typ problému, ktoré sa nastavia pri inicializácii.

Pre **spúšťanie optimalizátorov** je použitý spoločný spúšťací mechanizmus implementovaný v súbore `main.py`. Jeho prvou úlohou je spracovať vstupy od používateľa - v obmedzenej miere sa na to používajú argumenty, ale väčšinovo je zabezpečený konfiguračným súborom. Načítanie argumentov je zabezpečené základnou knižnicou `argparse`[7]. Bola zvolená hlavne preto, že syntax parametrov nebola zložitá a umožňuje automatické generovanie pomocníka.

Ďalšou úlohou je načítanie konfigurácie, ktorá je podrobne popísaná v časti konfigurácie. Ďalší dôvod pre použitie dátovej triedy na reprezentáciu konfigurácie je zjednotenie validácie argumentov. Pre reprezentáciu sa používajú triedy `FileConfig` a `CmdLineConfig`. Trieda `FileConfig` obsahuje priamo parametre pre beh aplikácie. Trieda `CmdLineConfig` okrem argumentov obsahuje aj konfiguráciu zo súboru `FileConfig`. Keďže argumenty majú vyššiu dôležitosť ako konfigurácia zo súboru, prepíšu konfiguráciu načítanú zo súboru. Najprv sa prevedie konverzia na správne dátové typy, aby bol zabezpečený konzistentný prevod z reťazcov, ktoré sú získavané zo súboru aj z `argparse`. Ďalším krokom je validácia správnosti, aj v tomto prípade sa využíva spojenie oboch zdrojov parametrov, aby validácia prebehla naraz. Výhodou je aj to, že všetky nájdené chyby v konfigurácii budú užívateľovi vrátené naraz.

V ďalšom kroku sa incializujú triedy optimalizátorov podľa toho, ktoré sa majú spustiť. Aplikácia umožňuje spustenie všetkých optimalizátorov v jednom behu pre zníženie počtu potrebných spustení pri vysokom počte experimentov. Potom sa spustí optimalizátor pre problém obchodného cestujúceho a následne znova pre problém batohu. Každý typ úlohy tiež vykreslí grafy za pomoci modul `graphs`, ktorý obsahuje spoločnú implementáciu grafu fitness hodnôt a pre problém obchodného cestujúceho aj vizualizáciu cesty. Výsledky sú tiež zapisované do súboru `tsp.out` a `knapsack.out`.

Konfigurácia

Pre konfiguráciu bola pre jednoduchosť najprv vyhodnotená vhodnosť balíkov základnej knižnice `Pythonu`. Jasnými kandidátmi sú v tomto prípade balíky `json` a `configparser`[8]. Konfigurácia vo formáte `JSON` má výhodu jednoduchého strojového spracovania, ale keďže

[5] Implementácia algoritmu optimalizácie hejnom častíc je dostupná na: https://github.com/rameziophobia/Travelling_Salesman_Optimization

[6] Implementácia algoritmu gravitačného vyhľadávania je dostupná na: https://github.com/alireza-soheilmahmodi/gravitational-search-algorithm/tree/master

[7] https://docs.python.org/3/library/argparse.html

[8] https://docs.python.org/3/library/configparser.html

konfiguračný súbor má byť aj prívetivý pre použitie človekom, nie je úplne vhodná na tento účel.

Lepšie z tohto porovnania vychádza knižnica `configparser`, ktorá spracováva `ini` súbory. Tieto súbory sa veľmi často vyskytujú ako konfiguračné súbory v operačných systémoch `Linux` aj `Windows`. Majú veľmi jednoduchý formát, ktorý je ľahko pochopiteľný a veľa ľudí s ním má skúsenosti.

Konfigurácia sa skladá z 3 hlavných sekcií:

1. `app` (aplikácia) - nastavuje správanie aplikácie - textový výpis, grafy, viacnásobné spustenie optimalizátorov
2. `optimizers` (optimalizátory) - zapnutie a nastavenia optimalizátorov
3. `problems` (úlohy) - zapnutie a cesta k súboru so zadaním úlohy

Konfigurácia sa načítava väčšinovo zo súboru. Aby nebolo nutné riešiť poskytnutie nekompletnej konfigurácie užívateľom, program obsahuje súbor `default.ini`, ktorý zároveň slúži ako vzorová konfigurácia. Program pri spustení automaticky tiež načítava súbor `config.ini` s konfiguráciou užívateľa, ktorý bude umiestnený v koreni aplikácie. Ďalej je možné, ako argument špecifikovať ďalší konfiguračný súbor.

Tieto súbory sú zoradené podľa dôležitosti. Každý ďalší súbor je vyhodnotený ako dôležitejší, takže vo výslednej konfigurácii prepíše rovnaké kľúče načítané z predchádzajúceho súboru. Ako bolo spomenuté v predchádzajúcej podkapitole, ak existujú rovnaké parametre zadané ako argumenty, majú najvyššiu prioritu.

Reprezentácia konfigurácie v aplikácii je vyriešená s použitím dátových tried `dataclass`[9]. Podstatou tohto riešenia je mať typovú kontrolu nad jednotlivými možnosťami, a jednoduchú prácu s konfiguráciou vďaka práci s objektami. Tiež sa použitím dátových tried redukuje redundantný kód pre inicializáciu tried.

Nie celá konfigurácia je vyskladaná z dátových tried. Hlavne v samotných sekciách konfigurácie, ktoré už obsahujú jednotlivé možnosti, sú použité klasické triedy obohatené o metódy na konverziu na správny dátový typ zdedené z triedy `KwargsConverter`. Táto trieda obsahuje tiež triednu premennú na zachytávanie chýb z konverzie, ktoré nastanú ak používateľ do konfigurácie vloží nesprávny typ dát. Vďaka tomu je možné vypísať všetky chyby v konfigurácii súčasne, nie opravovať po každom spustení aplikácie jednu chybu.

Podobne je riešená validácia parametrov, akurát tá môže byť dedená sekciami konfigurácie z triedy `Validator`. Pokročilé validácie porovnávajúce kombinácie parametrov sú implementované v triede `CmdLineConfig`, ktorá taktiež využíva zdedenú triednu premennú `validation_errors`. Tieto pokročilé validácie sa spúšťajú externe metódou `validate`. Má význam ju spúšťať až potom čo sa overí, že pri typovej konverzii dát zo súboru nenastali žiadne chyby. Opäť budu spustené všetky validácie a chyby sa zapíšu do triednej premennej `validation_errors`, ktorú metóda `validate` aj rovno vráti.

Načítanie a reprezentácia problémov

Pre načítanie problémov bolo rozhodujúce, aké dáta sú dostupné pre testovanie. Pre problém obchodného cestujúceho existuje štandard `TSPLIB 95`, ktorý zapuzdruje problém a aj optimálne riešenie. Pre prácu so súbormi štandardu `TSPLIB 95` existuje rovnomenná

[9] https://docs.python.org/3/library/dataclasses.html

`Python` knižnica `tsplib95`[10]. V aplikácii, preto bola použitá pre načítanie dát z `TSP` súborov v module `problems.TSP.tsp_loader`.

Pre problém batohu nebol nájdený formalizovaný štandard. Boli nájdené testovacie dáta v jednoduchom formáte, pre ktorých načítanie bol vytvorený modul. Súbory nazvané `kf`[11] obsahujú iba riadky reprezentujúce jednotlivé položky v batohu. Prvé číslo na riadku reprezentuje *váhu (weight)* a druhé oddelené neviditeľnými znakmi reprezentuje *zisk (profit)*.

Reprezentácia problému obchodného cestujúceho potrebuje vedieť zoznam miest, ktoré musia obsahovať súradnice. Pre jednoduchší výpis boli týmto mestám priradené aj názvy[12]. Je možné použiť akékoľvek reťazce, no pre načítanie zo súboru bolo jednoducho použité poradie mesta v súbore. Mesto je reprezentované triedou `City`.

Pre reprezentáciu trasy slúži trieda `Route`, ktorá už je o niečo zložitejšia. Okrem povinných metód zo zdedenej triedy `Optimizable` obsahuje tiež inicializačné metódy, ktoré optimalizátory používajú na vygenerovanie kandidátnych riešení. Jedná sa o metódy `make_random` a `make_greedy`. Okrem nich je tu ešte metóda `two_opt`, ktorá sa používa práve na použitie tejto modifikácie. Keďže objekt `Route` používa každý algoritmus, dávalo význam túto modifikáciu implementovať tu, kde ju každý algoritmus môže kedykoľvek použiť.

Pri probléme batohu máme položky reprezentované triedou `Item`, ktorá premennými `weight` a `profit` reprezentuje váhu a ohodnotenie položky. Opäť bola pridaná možnosť pre pomenovanie položky akýmkoľvek reťazcom a, ako pri problém obchodného cestujúceho sa pri načítaní zo súboru použije poradie položky.

Na reprezentáciu batoha je implementovaná trieda Knapsack. Oproti problém obchodného cestujúceho je rozdiel v tom, že batoh je obmedzený kapacitou, ktorá je obsiahnutá v tejto triede pre potreby algoritmov, ale hlavne inicializácie. Pre naplnenie batohu je základnou inicializačnou metódou opäť `make_random`, ktorá ale v tomto prípade tiež kontroluje, že nebola naplnená kapacita batoha. Zaujímavosťou je postup dosiahnutia náhodného, ale správneho naplnenia batohov, najprv sa "vložia" všetky položky bez ohľadu na kapacitu. Potom sa náhodne premiešajú a následne sa z batoha odstraňujú od poslednej pokým jeho naplnenosť nezodpovedá kapacite.

Zaujímavosťou je aj metóda `make_all`, ktorá iba vloží do batoha všetky položky načítané zo súboru. Je nutná pre algoritmus gravitačného vyhľadávania, ktorý má špecifickú implementáciu a počas svojho behu rôzne manipuluje s batohmi. V tomto prípade je `Knapsack` objekt použitý iba na reprezentáciu načítaných dát, nie kandidátnych riešení.

Grafový výstup

Poslednou časťou je vykresľovanie grafov, ktoré bolo oddelené do samostatného modulu. Aplikácia podporuje dva typy grafov. Jeden je spoločný pre oba algoritmy, graf fitness funkcie. Druhý špecifický pre problém obchodného cestujúceho, vizualizácia trasy. Oba typy grafov, ale vznikajú z viacerých optimalizátorov, a to je dôvod prečo ich implementovať abstraktne raz a všade využiť túto jednotnú implementáciu.

Grafy boli implementované podľa už existujúcej vizualizácie z algoritmu PSO[13]. Táto implementácia používala na vykresľovanie knižnicu `matplotlib`. Vizualizácie boli jemne vylepšené oproti pôvodnej implementácii, ktorá obsahoval duplicitné volania alebo nejednotné volania niektorých funkcií knižnice `matplotlib`.

[10] https://pypi.org/project/tsplib95/

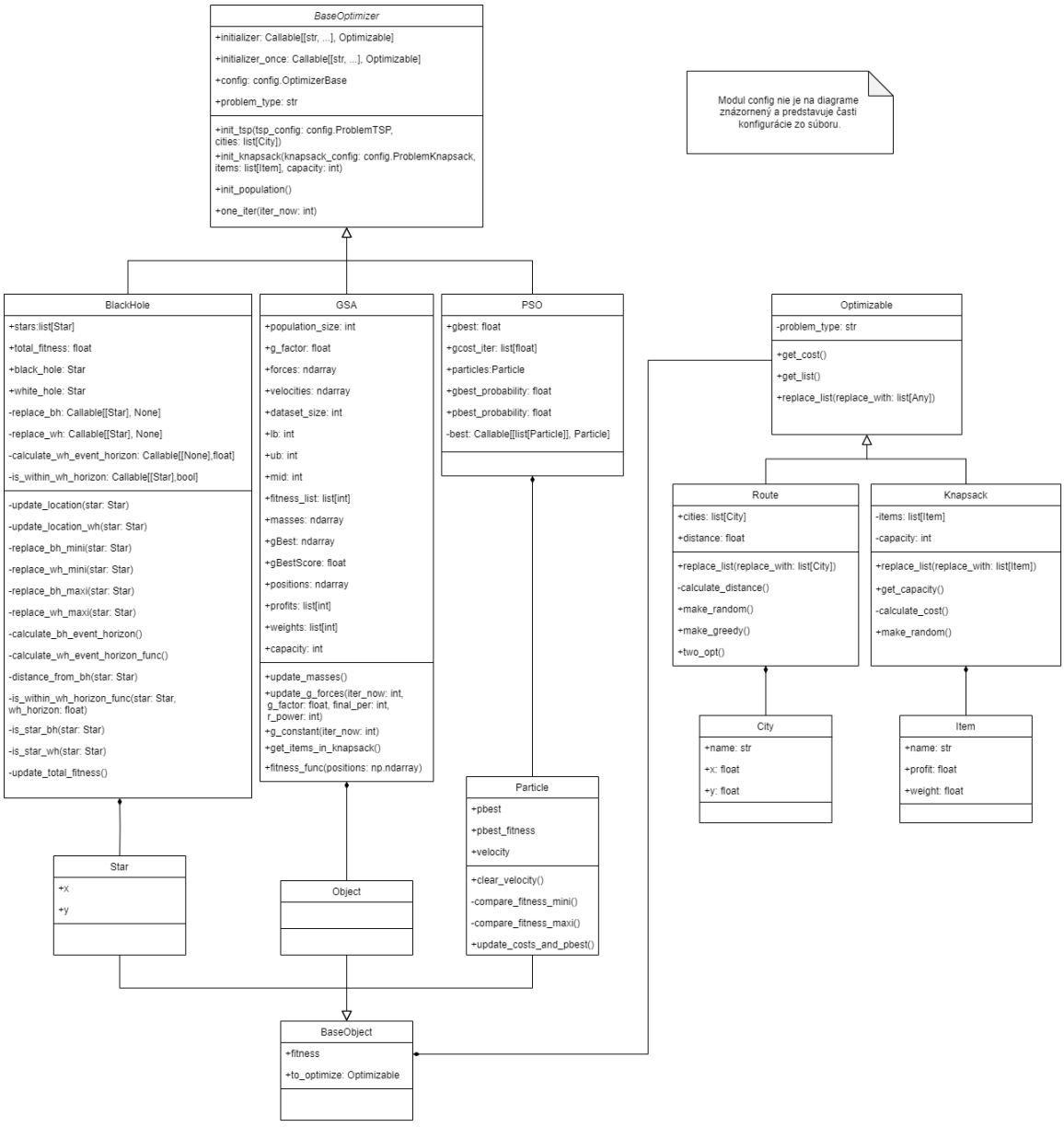
[11] ako Knapsack file

[12] názvy sú reprezentované pomocou čísel

[13] Obe implementácie boli prevzaté a upravené z https://github.com/rameziophobia/Travelling_Salesman_Optimization

Graf fitness funkcie tiež obsahuje v názve aktuálnu iteráciu, aby bolo možné interaktívne sledovať priebeh optimalizácie. Po dokončení algoritmu je tiež, teda z grafu hneď jasné pre koľko iterácii algoritmus bežal. Pre obzvlášť pomalé algoritmy, ako GSA a PSO s 2opt optimalizáciou bol v konfigurácii zavedený parameter pre vzorkovanie grafu. Ovplyvňuje ako často sa graf prekresľuje a tým aj koľko výsledných bodov bude mať. Pre pomalé algoritmy je, teda rýchlejšie viditeľné, ako prebieha optimalizácia. Vykreslovanie krabicového diagramu (boxplotu), nie je súčasťou aplikácie, ale boli naň využité knižnice `Seaborn` a `Matplotlib`.

Na obrázku 6.1 je možné vidieť diagram tried, ktorý znázorňuje štruktúru aplikácie.



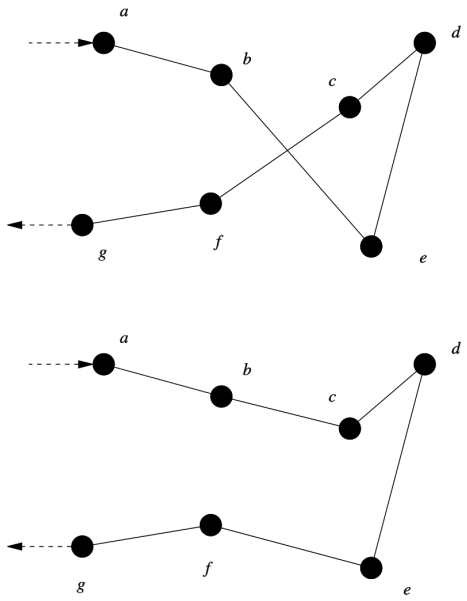
Obrázek 6.1: Diagram tried pre danú aplikáciu

6.2 Modifikácie

Bolo naimplementovaných niekoľko modifikácií, ktoré boli využité pre algoritmus optimalizácie hejnom častíc a pre algoritmus čiernych dier. Pre algoritmus gravitačného vyhľadávania neboli naimplementované modifikácie, vzhľadom na jeho nízku efektivitu a vysokú výpočetnú náročnosť, čo je popísané v časti 7.2. Implementované modifikácie som pri algoritme optimalizácie hejnom častíc zvolila na základe inšpirácie preštudivanej literatúry. Pre algoritmus čiernych dier som zvolila jednu modifikáciu, ktorou som sa inšpirovala na základe štúdia literatúry. V ďalších dvoch prípadoch sa jednalo o mnou navrhnutú modifikáciu.

V prípade **algoritmu optimalizácie hejnom častíc** som zvolila dve varianty. Prvá varianta bola inšpirovaná algoritmom hladného vyhľadávania (greedy search). Táto varianta by mala zabrániť uviaznutiu v lokálnych extrémoch, čo by malo viesť k hľadaniu lepších globálnych riešení. Implementácia tejto modifikácie bola naimplementovaná na základe dostupnej implementácie [14], pričom bola upravená pre potreby danej aplikácie.

Ako druhá modifikácia bola zvolená kombinácia s heuristikou *2opt*. V prípade tejto heuristiky sa jedná o jednoduchú lokálnu prehľadávaciu metódu, ktorá sa často využíva pri riešení problému obchodného cestujúceho. Funguje tak, že iteratívne prehľadáva možnosti pre vylepšenie existujúceho riešenia tým, že vymieňa dvojice hrán v ceste a kontroluje či takýto výmenný krok zlepšuje celkovú dĺžku cesty. Ak áno, použije sa táto zmena a proces sa opakuje, kým sa už nedá zlepšiť žiadne riešenie. Je populárna pre svoju jednoduchosť a efektivitu pri riešení problémov obchodného cestujúceho a často sa používa ako súčasť iných optimalizačných algoritmov alebo metaheuristík. Jej názov "2-opt"pochádza z toho, že vymieňa dvojice hrán v ceste. Ukážku použitia metódy 2opt je možné vidieť na obrázku 6.2.[12]Implementácia tejto metódy[15] je voľne dostupná a bola prerobená pre účely aplikácie.



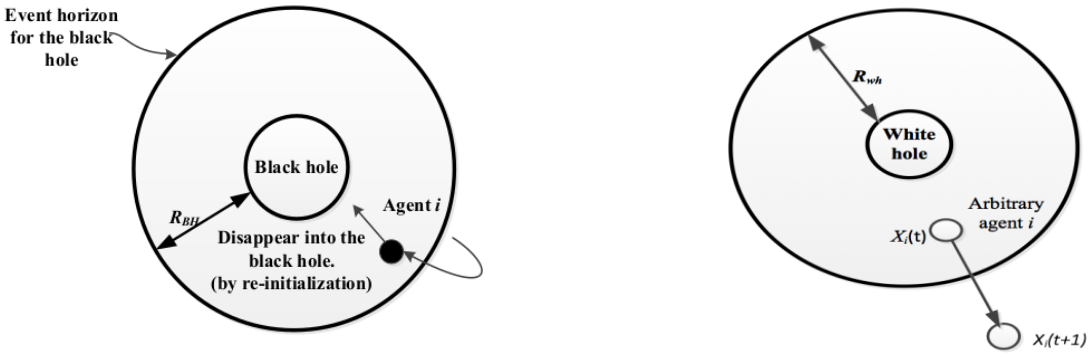
Obrázek 6.2: Ukážka pred a po použití metódy 2opt
dostupné z: Wikipedia

[14] https://github.com/rameziophobia/Travelling_Salesman_Optimization/blob/master/pso.py
[15] https://stackoverflow.com/questions/53275314/2-opt-algorithm-to-solve-the-travelling-salesman-problem-in-python

Pri **algoritme čiernych dier** boli zvolené dve, respektíve tri modifikácie. Tretia modifikácia je kombináciou prvých dvoch modifikácií. Prvá mnou navrhnutá modifikácia je postavená na tom, že sa suma hodnôt fitness funkcií vynásobí konštantou `(bh_radius_modifier)`, ktorá je ľubovoľné desatinné číslo. Táto hodnota je využitá pri výpočte honoty horizontu udalostí. Horizont udalostí sa používa na určenie toho, kedy má byť hviezda pohltená čiernou dierou a následne je vygenerovaná nová hviezda. Táto modifikácia by mala zaručiť zlepšenie výsledkov, nakoľko pri správne zvolenej hodnote konštanty, by malo dochádzať k častejšiemu pohlcovaniu hviezd a častejšiemu generovaniu nových hviezd. Očakáva sa, že tento jav nastane pre hodnoty menšie ako jedna, pričom hodnota jedna je vzhľadom na operáciu násobenia neutrálny prvok. Táto modifikácia nebola prebratá zo žiadnej odbornej literatúry, a teda ani jej implementácia.

Druhá modifikácia bola inšpirovaná existujúcou variantou z odbornej literatúry. Jedná sa o modifikáciu, kde bola pridaná aj biela diera. Biela diera je opakom čiernej diery. Jedná sa o oblasť časopriestoru, do ktorej nie je možné vstúpiť zvonku, ale hmota a svetlo z nej môžu uniknúť. Kým čiernej diere priraďujeme najlepšiu hodnotu spomedzi všetkých hodnôt, v prípade bielej diery jej priraďujeme hodnotu najhoršieho jedinca. Rovnako, ako v prípade čiernej diery aj pri bielej diere počítame horizont udalostí. Avšak, v tomto prípade, nie je hviezda pohltená a generovaná nanovo, ale hviezda je ňou odpudená, čo vedie k prehľadávaniu priestoru s lepšími riešeniami. Implementácia tejto modifikácie vychádza z informácií uvedených v článku. Porovnanie horizontu udalostí pre čiernu dieru a pre bielu dieru je možné vidieť na 6.3. [33]

Tretia modifikácia je prepojením prvých dvoch. Algoritmus bol použitý s modifikáciou bielej diery, ale horizont udalostí pre bielu `(wh_radius_modifier)` aj čiernu dieru `(bh_radius_modifier)` bol vynásobený ľubovoľnou desatinnou hodnotou konštanty, pričom vzhľadom na operáciu násobenia je hodnota jedna považovaná za neutrálny prvok.



(a) Čierna diera (b) Biela diera

Obrázek 6.3: Porovnanie horizontu udalostí pri čiernej a bielej diere

Kapitola 7

Experimentálne výsledky

V tejto kapitole sú popísané výsledky experimentov, ktoré boli vykonané pomocou algoritmu optimalizácie hejnom častíc, algoritmu čiernych dier a algoritmu gravitačného vyhľadávania. Tieto algoritmy boli využité pri riešení problému obchodného cestujúceho a problému batohu. Výsledky a priebeh experimentov problému obchodného cestujúceho sú podrobne popísané v podkapitole 7.1. Výsledky riešenia problému batohu a priebeh experimentov sú podrobne popísané v podkapitole 7.2.

Keďže sa jedná o rôzne algoritmy, pri ktorých sa pracuje s rôznymi parametrami, bolo experimentovanie zamerané na porovnanie úspešnosti algoritmu na základe rôznych hodnôt jeho parametrov a zároveň na porovnávanie úspešnosti algoritmov medzi sebou. Jediné parametre, ktoré boli pre všetky algoritmy rovnaké, bol počet jedincov a počet iterácií. V jednotlivých algoritmoch majú jedinci rôzne pomenovanie. V prípade algoritmu čiernych dier hovoríme o hviezdach. V prípade algoritmu optimalizácie hejnom častíc hovoríme o časticiach. V prípade algoritmu gravitačného vyhľadávania hovoríme o objektoch. Napriek odlišnému názvosloviu sa vo všetkých prípadoch jedná o jedincov.

Každý problém bol skúmaný pomocou dvoch rôznych algoritmov, pričom na základe vyhodnotenia výsledkov bol úspešnejší algoritmus ďalej modifikovaný. V prípade obchodného cestujúceho bol využitý algoritmus optimalizácie hejnom častíc a algoritmus čiernych dier. Pri probléme batohu bol taktiež použitý algoritmus čiernych dier a algoritmus gravitačného vyhľadávania.

7.1 Problém obchodného cestujúceho

Experimentovanie pri probléme obchodného cestujúceho bolo vykonané pomocou algoritmu optimalizácie hejnom častíc a pomocou algoritmu čiernych dier. Experimentovanie bolo vykonané na datasete *TSPLIB 95*, ktorý je jedným z najvyužívanejších datasetov na testovanie problému obchodného cestujúceho a je dostupný na TSPLIB. TSPLIB je knižnica, ktorá obsahuje vzorové inštancie pre problém obchodného cestujúceho. V tejto knižnici je možné nájsť aj inštancie pre iné problémy, ako je napríklad problém Hamiltonovskej kružnice, problém s kapacitným smerovaním vozidla a iné. Pre problém obchodného cestujúceho knižnica obsahuje datasety pre symetrický problém obchodného cestujúceho a pre asymetrický problém obchodného cestujúceho. Táto práca sa zaoberá symetrickým problémom obchodného cestujúceho.

Formát dátového súboru určeného na testovanie sa skladá z dvoch častí, zo špecifikačnej a dátovej časti. Špecifikačná časť obsahuje meno súboru, komentár, typ problému, dimenziu

[1] a váhový typ hrany (edge weight type). Existuje viacej variant váhového typu hrany, ale experimenty boli vykonané len na datasetoch, ktorých váhový typ hrany bol definovaný pomocou Euklidovskej vzdialenosti. Dátová časť súboru obsahuje súradnice uzlov, s tým, že v prvom stĺpci sa nachádza číslo uzla a v zvyšných dvoch súradnice x a y.

Experimenty prebehli na piatich rôzne veľkých inštanciách z tohto datasetu. Konkrétne sa jednalo o *eil51*, *eil101*, *rat195*, *rat575* a *pr1002*, pričom čísla predstavujú počet miest. Nakoľko sa jedná o často využívaný dataset sú pre všetky inštancie, na ktorých boli vykonané experimenty už vopred známe optimálne hodnoty. Vďaka tomuto je možné porovnať úspešnosť daných algoritmov pri riešení tohto problému. Optimálne hodnoty pre použité inštancie je možné vidieť v tabuľke 7.1.

Názov inštancie	**Optimálna hodnota**
eil51	426
eil101	629
rat195	2323
rat575	6773
pr1002	259045

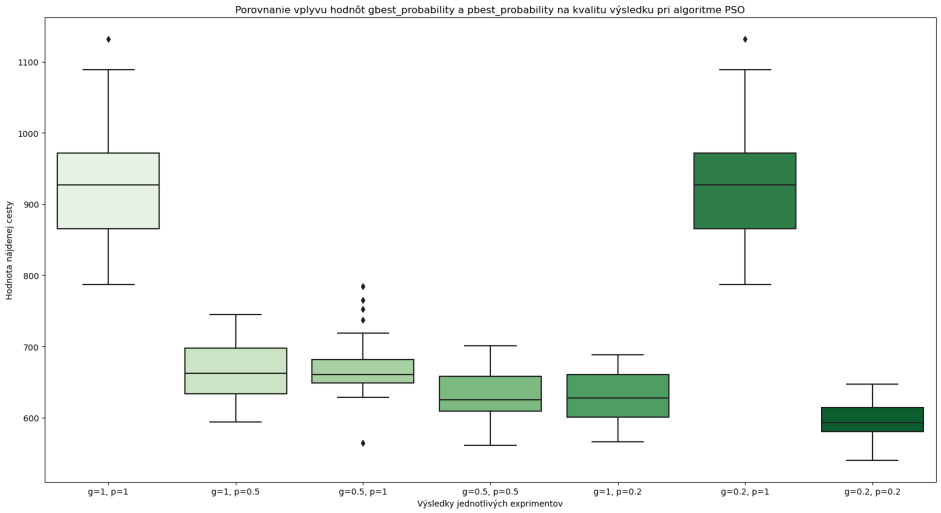
Tabulka 7.1: Optimálne hodnoty pre testované inštancie

Na riešenie problému obchodného cestujúceho boli vykonané série štyroch rôznych experimentov. Je dôležité zdôrazniť, že na rozdiel od problému batohu sa jedná o problém minimalizácie, a preto je potrebné nájsť čo najmenšiu hodnotu. Takže platí, že čím menšiu hodnotu algoritmy našli, tým úspešnejšie boli. Avšak, vždy je dôležité vziať do úvahy aj časovú náročnosť výpočtu.

Cieľom **prvého experimentu** bolo zistiť vplyv hodnôt *gbest probability* a *pbest probability* na výkonnosť algoritmu optimalizácie hejnom častíc. Tento experiment prebiehal len na algoritme optimalizácie hejnom častíc, nakoľko algoritmus čiernych dier nemá žiadne nastaviteľné parametre mimo počtu iterácií a počtu jedincov. Na základe výsledkov tohto experimentu prebiehali ďalšie experimenty. Počet iterácií a počet jedincov bol pri tomto experimente nastavený na tisíc, menili sa len hodnoty *gbest probability* a *pbest probability*. Pre všetky hodnoty, s ktorými sa experimentovalo, bolo vykonaných tridsať behov programu. Hodnoty *gbest probability* a *pbest probability* boli nastavené na základe hodnôt, ktoré využívali pri experimentovaní autori v článkoch o optimalizácií pomocou hejna častíc a ich rôzne kombinácie. Výsledky tohto experimentu sú znázornené na grafe 7.1. Z grafu je vidieť, že najlepšie hodnoty dosiahol algoritmus v prípade, že hodnoty *gbest probability* a *pbest probability* boli nadstavené na hodnotu 0.2. Zároveň je z grafu možné vyčítať, že najhoršie výsledky mal algoritmus v prípade, že bola hodnota *pbest probability* nastavená na hodnotu jedna. Z tohto vyplýva, že hodnota *pbest probability* má na výsledok väčší vplyv ako hodnota *gbest probability*.

Druhý experiment bol zameraný na vplyv počtu iterácií a jedincov na výsledky. Tento experiment pracoval s algoritmom optimalizácie hejnom častíc a algoritmom čiernych dier. Tento experiment vychádzal z prvého experimentu, z ktorého boli prebraté najlepšie hodnoty *gbest probability* a *pbest probability*. Cieľom tohto experimentu bolo zároveň porovnať aj úspešnosť algoritmov medzi sebou, čoho výsledky boli použité pre ďalšie experimentovanie. Pri oboch algoritmoch sa pracovalo s počtom iterácií a počtom jedincov nastavených na hodnotu tisíc. V ďalšom kroku sa pracovalo s počtom iterácií nastavených na hodnotu

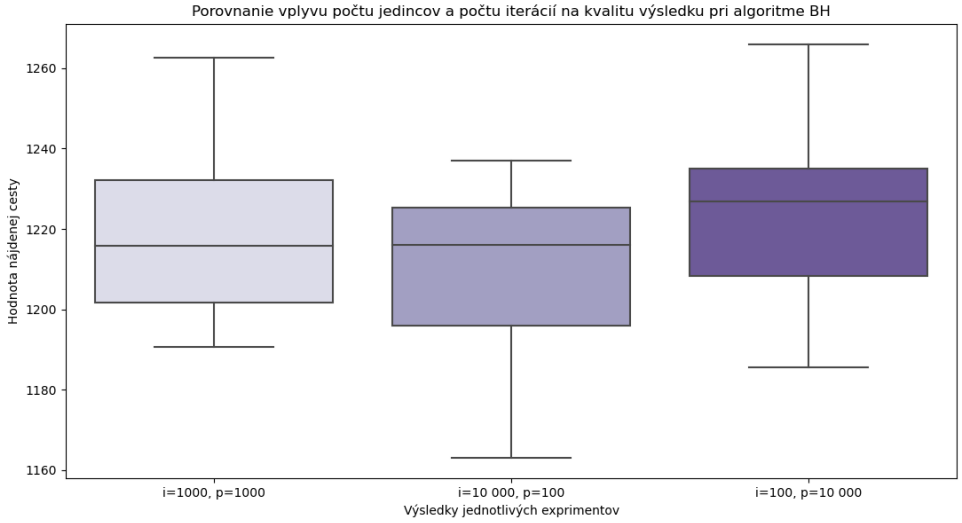
[1] dimenzia predstavuje počet miest



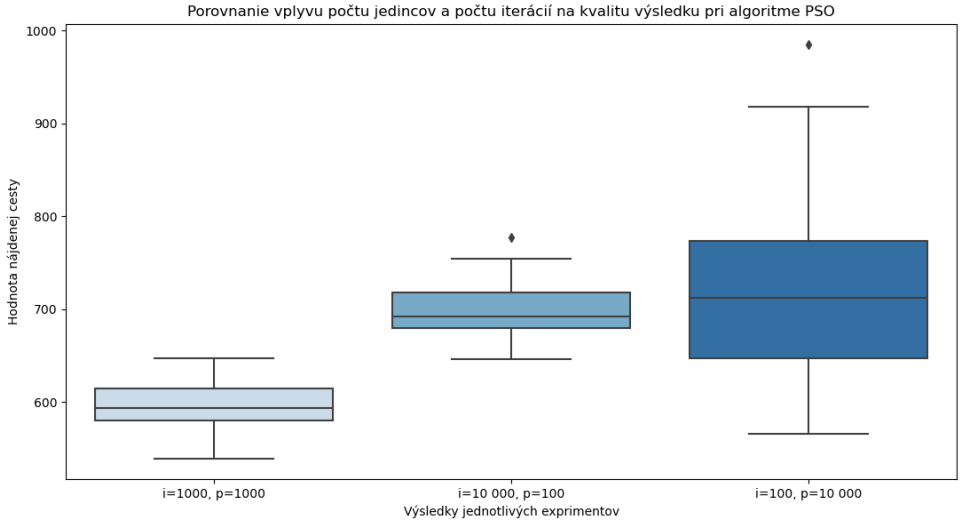
Obrázek 7.1: Porovnanie výsledkov pre rôzne hodnoty pbest a gbest probability

desaťtisíc a počtom jedincov nastavených na hodnotu sto. V poslednom kroku sa pracovalo s počtom iterácií sto a počtom jedincov desaťtisíc. Pre všetky alternatívy bolo vykonaných tridsať behov programu pre oba algoritmy. Na grafe 7.2 je možné vidieť výsledky experimentov pre algoritmus čiernych dier. Graf ďalej znázorňuje, že algoritmus čiernych dier dosahoval najlepšie výsledky pri vyššom počte iterácií a nižšom počte jedincov. Zároveň je možné si všimnúť, že vyšší počet jedincov mal na výsledok algoritmu najhorší vplyv. Na grafe 7.3 môžeme vidieť výsledky experimentu pri algoritme optimalizácie hejnom častíc. Algoritmus dosiahol najlepšie výsledky pre počet iterácií a počet jedincov nastavených na hodnotu tisíc. Rovnako, ako v prípade algoritmu čiernych dier aj tu bol najhorší výsledok v prípade, že sme mali vysoký počet jedincov a nízky počet iterácií. Na základe výsledkov môžeme tvrdiť, že počet iterácií má väčší vplyv na výsledok ako počet jedincov. Graf 7.4 znázorňuje porovnanie úspešnosti algoritmov medzi sebou. Z grafu je zrejmé, že algoritmus optimalizácie hejnom častíc dosahoval oveľa lepšie výsledky ako algoritmus čiernych dier. Preto sa v ďalších experimentoch pracovalo s algoritmom optimalizácie hejnom častíc.

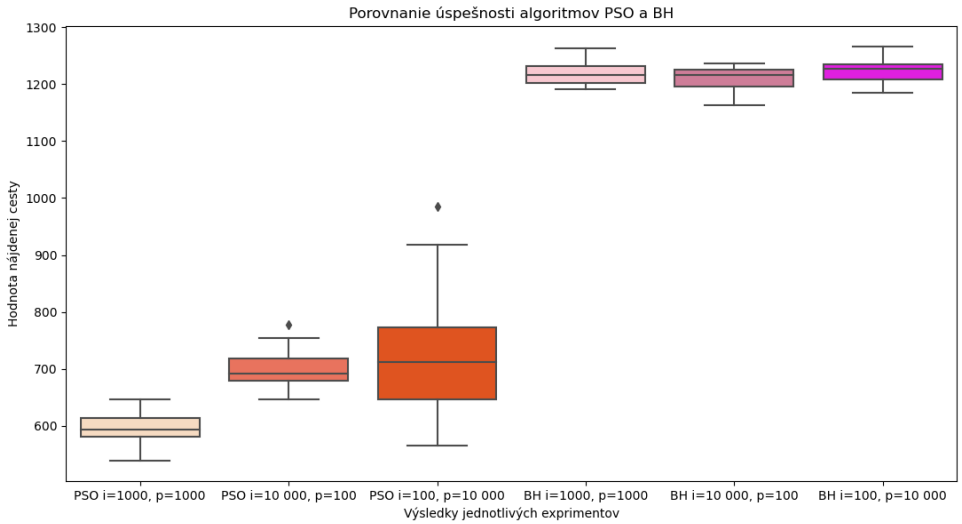
Cieľom **tretieho experimentu** bolo zaviesť modifikáciu do algoritmu optimalizácie hejnom častíc a porovnať jej úspešnosť s pôvodným algoritmom. Zároveň bolo cieľom tohto experimentu otestovať algoritmus optimalizácie hejnom častíc a modifikovaný algoritmus na väčšej sade miest. Nakoľko v odbornej literatúre bol tento algoritmus testovaný len na inštanciách zhruba o sto mestách. V tomto experimente bol algoritmus testovaný na inštanciách o veľkosti päťdesiatjeden až tisícdva miest. V tomto experimente bola použitá modifikácia, ktorá využíva algoritmus hladného prehľadávania (greedy search). Experiment vychádzal z druhého experimentu a pracoval s najlepšími hodnotami aký algoritmus optimalizácie hejnom častíc dosiahol. A to práve s počtom iterácií a počtom jedincov nastavených na hodnotu tisíc, zároveň s hodnotami *gbest probability* a *pbest probability* nastavenými na hodnotu 0.2. Rovnako, ako v predošlých experimentoch bolo pre všetky alternatívy vykonaných tridsať behov programu, avšak pri vyššom počte miest sa hodnoty výsledkov veľmi nelíšili. Na grafe 7.5 je možné vidieť porovnanie úspešnosti základného a modifikovaného algoritmu, pričom z grafu je jasne poznať rozdiel v ich úspešnosti. Modifikovaný algorti-



Obrázek 7.2: Porovnanie vplyvu počtu iterácií a jedincov na algoritmus BH



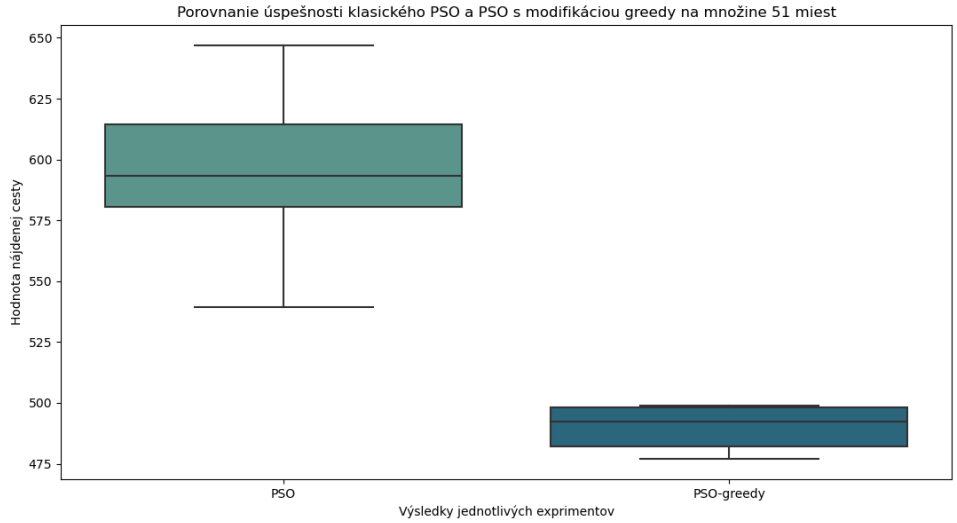
Obrázek 7.3: Porovnanie vplyvu počtu iterácií a jedincov na algoritmus PSO



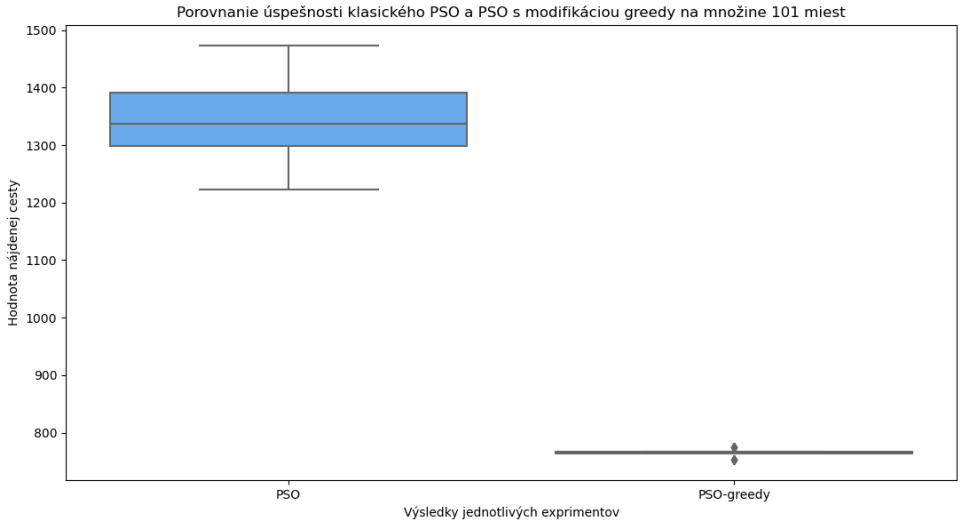
Obrázek 7.4: Porovnanie úspešnosti algoritmov BH a PSO medzi sebou

mus síce nedosiahol známu optimálnu hodnotu, ale bol sa jej schopný priblížiť. Najlepšia hodnota, ktorú dosiahol sa rovnala 477, pričom optimálna hodnota je 426. Pre porovnanie, algoritmu bez modifikácie dosiahol najlepšiu hodnotu 539. Graf 7.6 zobrazuje porovnanie algoritmov pre 101 miest. Aj v tomto prípade graf jasne znázorňuje rozdiel medzi modifikovaným a základným algoritmom. Optimálna hodnota pre inštanciu o veľkosti 101 miest je 629. Najlepšia hodnota, ktorú modifikovaný algoritmus našiel je 753. Je zrejmé, že pri vyššom počte miest sa algoritmus aj s pomocou modifikácie vzďaľuje optimálnej hodnote. Zároveň je vidieť väčší rozdiel medzi základnou a modifikovanou verziou algoritmu. Rovnaký jav si môžeme všimnúť aj pri väčších inštanciách o veľkosti 195 mestách na grafe 7.7 a 575 mestách na grafe 7.8. Vzhľadom na prehlbujúce sa rozdiely v úspešnosti základného a modifikovaného algoritmu, bol pri inštancii o veľkosti 1002 miest využitý len modifikovaný algoritmus. Pri všetkých tridsiatich behoch našiel len jednu hodnotu, čo je znázornené na grafe 7.9.

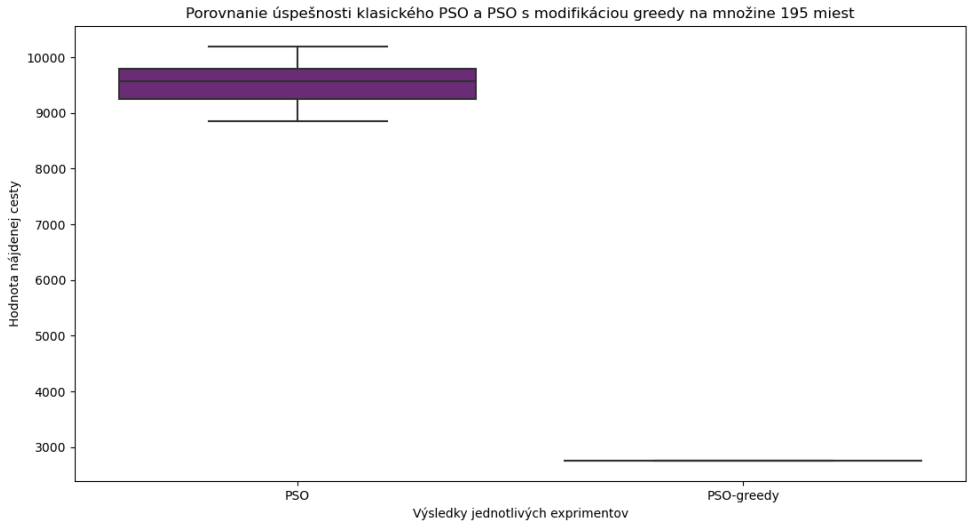
Cieľom **štvrtého experimentu** bolo otestovať úspešnosť algoritmu optimalizácie hejnom častíc s pridaním modifikácie 2opt. Testovanie prebiehalo na základe predošlých experimentov, a teda aj v tomto prípade, boli hodnoty počtu iterácií a počtu jedincov nastavené na hodnotu 1000 a parametre *gbest probability* a *pbest probability* nadstavené na hodnotu 0.2. Algoritmus s touto modifikáciou pri inštancii o veľkosti 51 miest dosiahol veľmi dobré výsledky. Táto modifikácia bola úspešnejšia ako modifikácia hladného prehľadávania. Avšak, čas výpočtu bol v porovnaní s predošlými experimentami extrémne dlhý. Jeden beh algoritmu trval zhruba 10 hodín. Po úpravách beh algoritmu trval o niečo menej, avšak, stále niekoľkonásobne dlhšie v porovnaní s ostatnými variantami. Pre túto skutočnosť, sa experimenty na vyššom počte miest neuskutočnili, vzhľadom na kompromis medzi hodnotou, ktorú bol algoritmus schopný nájsť a časom, ktorý na to potreboval. Graf 7.10 znázorňuje cestu, ktorú našiel algoritmus optimalizácie hejnom častíc vo svojej základnej verzii. Na grafe 7.11 je vidieť cestu, ktorú našiel algoritmus s modifikáciou hladného prehľadávania. Graf 7.12 zobrazuje cestu, ktorú našiel algoritmus s modifikáciou 2opt. Pri porovnaní je zrejmé, že algoritmus s modifikáciou 2opt našiel cestu bez križovania sa. Algoritmus s



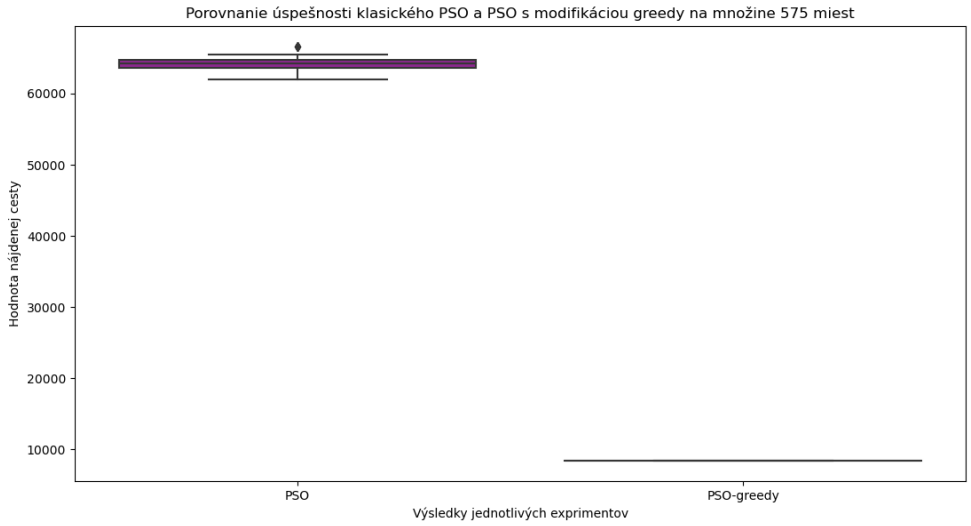
Obrázek 7.5: Porovnanie algoritmu PSO s greedy modifikáciou pri algoritme PSO pri 51 mestách



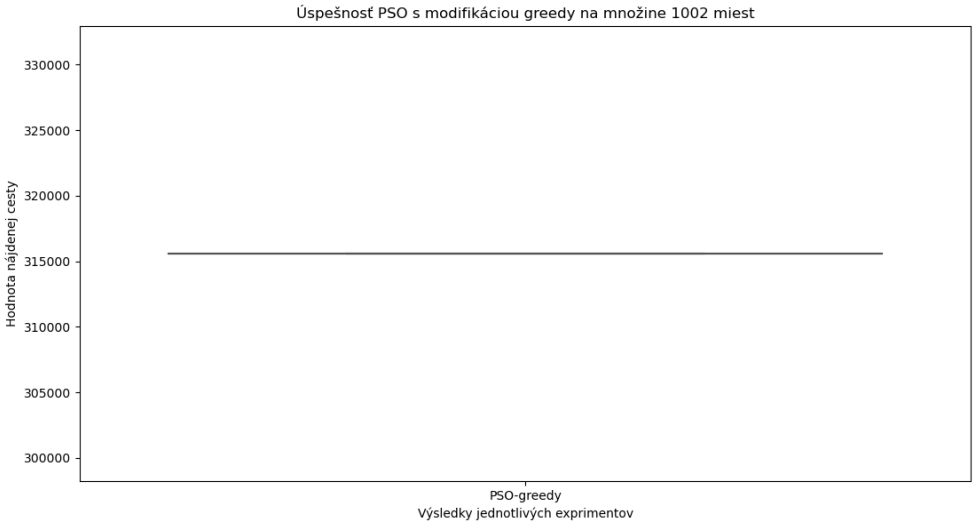
Obrázek 7.6: Porovnanie algoritmu PSO s greedy modifikáciou pri algoritme PSO pri 101 mestách



Obrázek 7.7: Porovnanie algoritmu PSO s greedy modifikáciou pri algoritme PSO pri 195 mestách

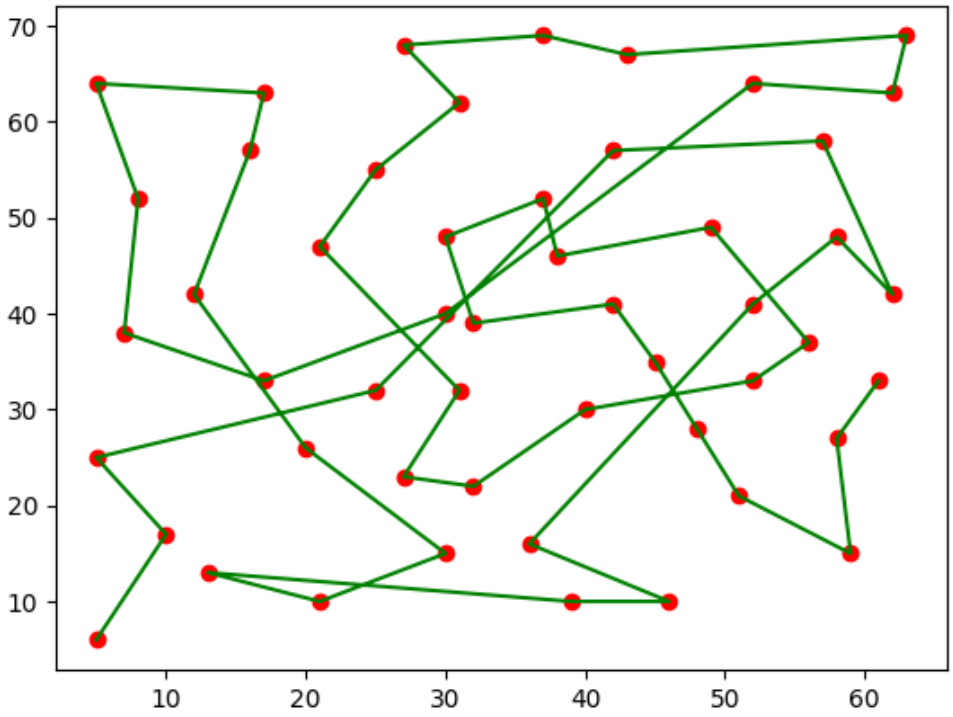


Obrázek 7.8: Porovnanie algoritmu PSO s greedy modifikáciou pri algoritme PSO pri 575 mestách

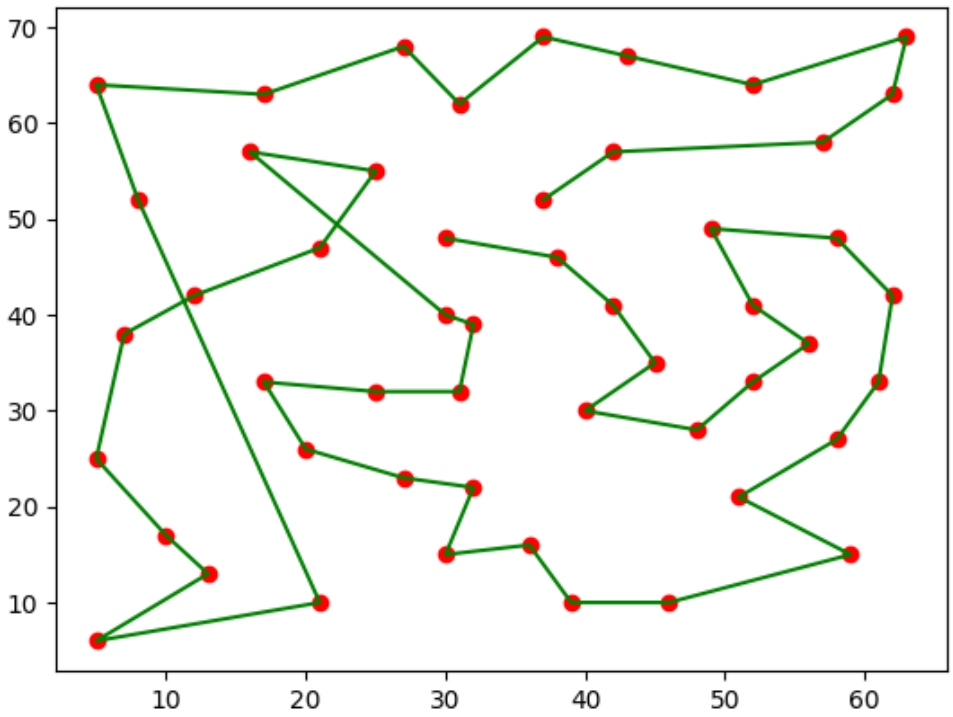


Obrázek 7.9: Výsledok algoritmu s greedy modifikáciou pre 1002 miest

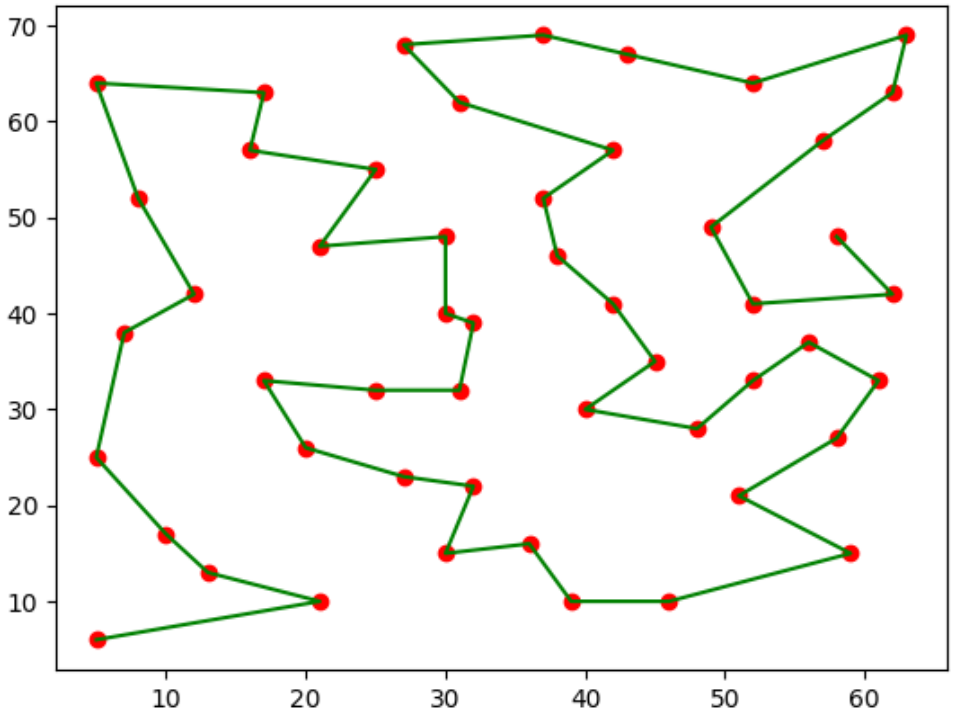
modifikáciou hladného prehľadávania mal nejaké cesty, ktoré sa križovali, ale aj tak bol v hľadaní relatívne úspešný. Zároveň si môžeme všimnúť, že algoritmus v základnej verzii mal dosť ciest, ktoré sa navzájom križovali, čo viedlo k vyššej hodnote výslednej cesty.



Obrázek 7.10: Cesta TSP pre 51 miest vygenerovaná základným PSO



Obrázek 7.11: Cesta TSP pre 51 miest vygenerovaná algoritmom PSO s modifikáciou greedy



Obrázek 7.12: Cesta TSP pre 51 miest vygenerovaná algoritmom PSO s modifikáciou 2opt

7.2 Problém batohu

Experimentovanie pri probléme batohu bolo vykonané pomocou algoritmu čiernych dier a algoritmu gravitačného vyhľadávania na datasete 0/1 Knapsack Problem dostupnom na Knapsack. Rovnako, ako v prípade obchodného cestujúceho aj pri probléme batohu boli experimenty vykonané s často používaným datasetom, ktorý poskytuje vopred známe optimálne hodnoty. Vďaka tomu môžeme porovnať úspešnosť našich algoritmov.

Názov súboru pri danom datasete má nasledujúci formát **kp__n__wmax**, kde *kp* predstavuje meno inštancie problému batohu, *n* predstavuje počet položiek a *wmax* predstavuje kapacitu batohu. Súbor sa skladá z dvoch stĺpcov. V prvom riadku prvého stĺpca je uvedený počet položiek, ktoré sa nachádzajú v súbore. V prvom riadku v druhom stĺpci je uvedená kapacita batohu. V zvyšných riadkoch prvého stĺpca sú uvedené hodnoty zisku pre danú položku. V zvyšných riadkoch druhého stĺpca sú uvedené váhy daných položiek. Inak povedané, s výnimkou prvého riadku, máme na všetkých riadkoch dvojice, ktoré predstavujú zisk a váhu danej položky.

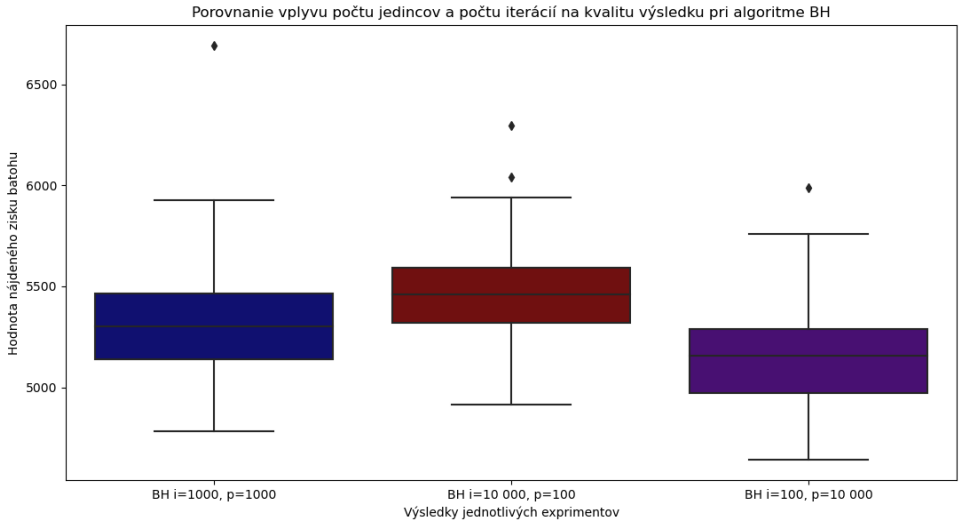
Experimenty prebehli na inštanciách o veľkosti desiatich, dvadsiatich a stých položkách, pričom ich optimálne hodnoty je možné vidieť v tabuľke 7.2. V prípade, že sa jednalo o inštancie o desiatich a dvadsiatich položkách, oba algoritmy našli optimálne hodnoty, preto ich v týchto prípadoch nebolo možné porovnať. Jediný rozdiel bol, že algoritmus gravitačného vyhľadávania potreboval na svoj beh viacej času ako algoritmus čiernych dier.

Názov inštancie	Optimálna hodnota
f1_l-d_kp_10_269	295
f2_l-d_kp_20_878	1024
knapPI_1_100_1000_1	9147

Tabulka 7.2: Optimálne hodnoty pre testované inštancie

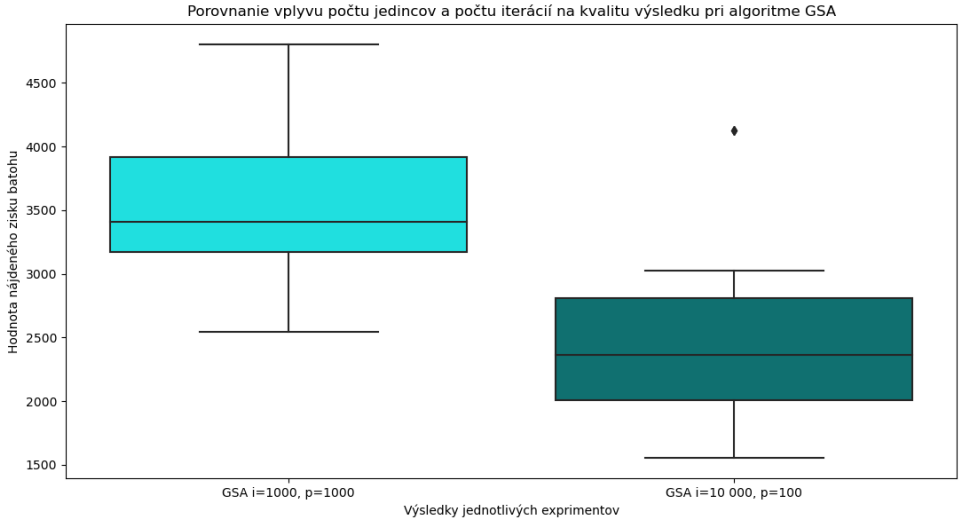
Pri riešení problému batohu boli vykonané série štyroch rôznych experimentov. Nakoľko pre inštancie s menším počtom položiek oba algoritmy dokázali nájsť optimálnu hodnotu, bolo experimentovanie vykonané na inštancii o veľkosti sto položiek. Problém batohu sa považuje za problém maximalizácie. Naším cieľom bolo nájsť, čo najvyššiu možnú hodnotu a zároveň bolo potrebné vziať do úvahy aj časovú náročnosť týchto výpočtov.

Cieľom **prvého experimentu** bolo zistiť vplyv počtu jedincov a iterácií na výsledok algoritmu a zároveň porovnať úspešnosť algoritmov medzi sebou. Algoritmy bežali pre všetky varianty tridsaťkrát. Graf 7.13 zobrazuje výsledky tohto experimentu pre algoritmus čiernych dier. Z grafu je možné zistiť, že algoritmus bol najúspešnejší pri vyššom počte iterácií a nižšom počte jedincov. Tento výsledok je rovnaký, ako sme dosiahli pri probléme obchodného cestujúceho. Výsledky tohto experimentu pre algoritmus gravitačného vyhľadávania sú znázornené na grafe 7.14. V tomto prípade nebol algoritmus testovaný pre počet jedincov 10 000 a 100 iterácií, nakoľko sa už pri prvom behu táto varianta ukázala, ako pomalá a dosahovala slabé výsledky. Porovnanie úspešnosti týchto algoritmov zobrazuje graf 7.15. Je zrejmé, že algoritmus čiernych dier je podstatne lepší, zároveň výsledky dosiahol v rýchlejšom čase. Vzhľadom na spomínané nedostatky algoritmu gravitačného vyhľadávania, nebol tento algoritmus použitý pri ďalších experimentoch.

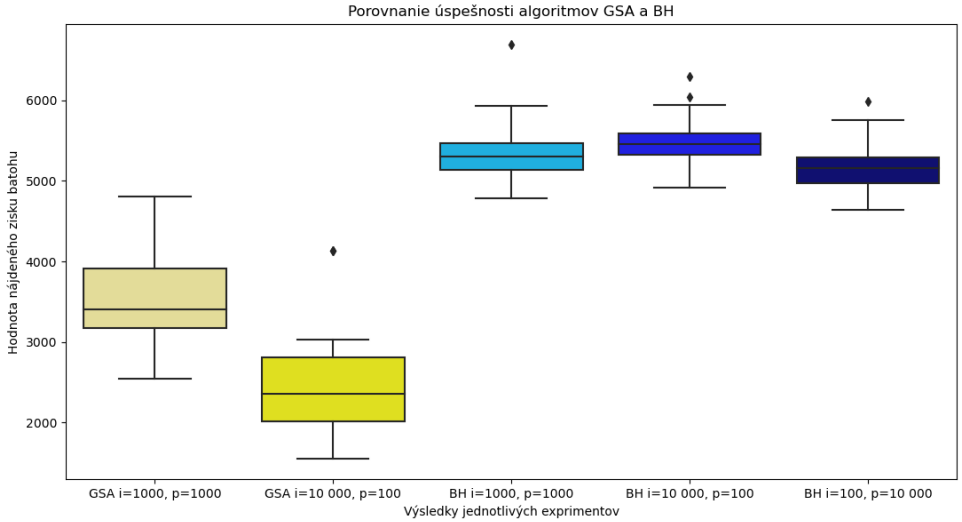


Obrázek 7.13: Porovnanie vplyvu počtu iterácií a jedincov na algoritmus BH

Druhý experiment bol zameraný na modifikáciu algoritmu čiernych dier, jej vplyv na hodnotu výsledku. Cieľom tejto modifikácie bolo pridať ľubovoľnú konštantu k výpočtu rádiusu horizontu udalostí. Tento experiment vychádzal z predošlého experimentu a na testovanie boli využité najlepšie zistené hodnoty z prvého experimentu. Boli testované rôzne

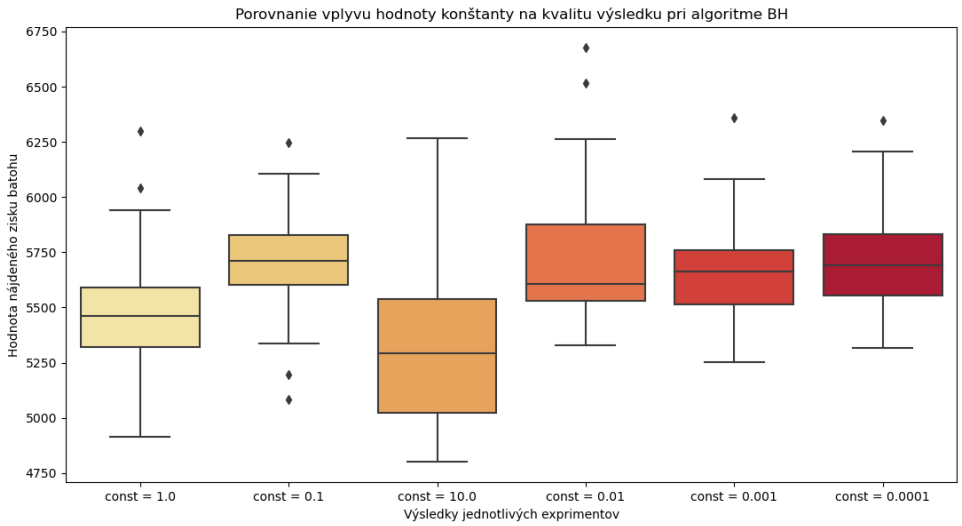


Obrázek 7.14: Porovnanie vplyvu počtu iterácií a jedincov na algoritmus GSA



Obrázek 7.15: Porovnanie úspešnosti algoritmov BH a GSA medzi sebou

hodnoty konštanty, ktoré boli následne porovnané medzi sebou. Rovnako, ako v predošlých prípadoch aj tu bolo pre každú možnosť vykonaných tridsať behov programu. Výsledky tohto experimentu sú znázornené na grafe 7.16, pričom je vidieť, že pre nižšie hodnoty konštanty algoritmus dosahoval lepšie výsledky. V prípade, že je konštanta nastavená na hodnotu jedna, sa v podstate jedná o pôvodný algoritmus bez modifikácií. Graf taktiež znázorňuje, že táto modifikácia bola úspešnejšia ako základný algoritmus.

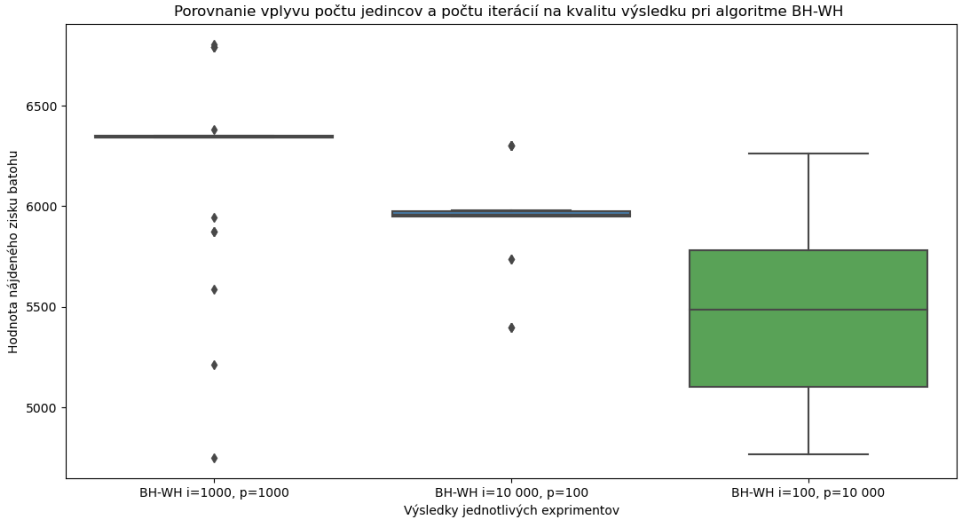


Obrázek 7.16: Porovnanie úspešnosti algoritmu BH pre rôzne hodnoty konštanty

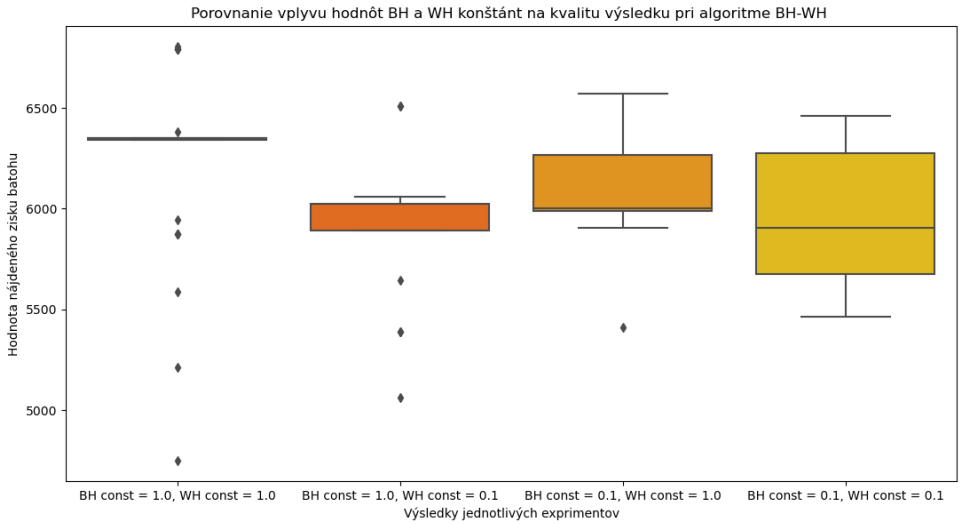
Cieľom **tretieho experimentu** bolo porovnať úspešnosť algoritmu čiernych dier s modifikáciou bielej diery. A to pre rôzne hodnoty počtu jedincov a počtu iterácií. Výsledky tohto experimentu je možné vidieť na grafe 7.17. Na rozdiel od základnej varianty algoritmu čiernych dier, v tomto prípade algoritmus dosahoval najlepšie hodnoty pri nadstavení počtu iterácií a počtu jedincov na hodnotu tisíc.

Štvrtý experiment sa zaoberal algoritmom čiernych dier s oboma modifikáciami z predošlých experimentov. V tomto prípade bola pridaná aj konštanta na výpočet horizontu udalostí pre bielu dieru. Výsledky tohto experimentu sú znázornené na grafe 7.18. Môžeme vidieť, že algoritmus dosiahol najlepšie výsledky v prípade, že sa hodnoty konštánt pre výpočet horizontu udalostí pre bielu aj čiernu dieru rovnali hodnote jedna, čo je neutrálny prvok v oboch prípadoch.

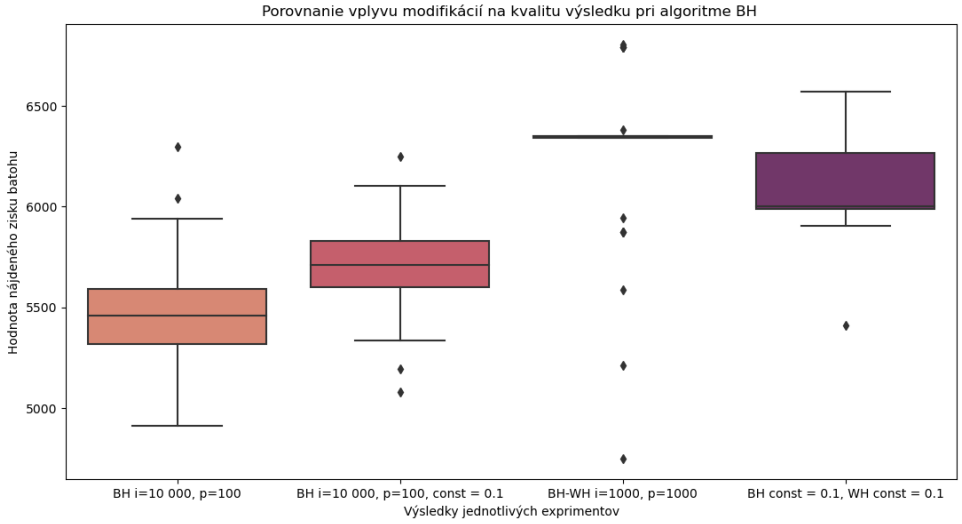
Na grafe 7.19 je zobrazené porovnanie najúspešnejších variant zo všetkých experimentov. Ako najúspešnejšia varianta sa ukázala modifikácia s bielou dierou, ktorá dosiahla najlepšie výsledky. Za najhoršiu variantu považujeme základný algoritmus čiernych dier bez modifikácií. Aj napriek úspešnosti modifikácie algoritmu čiernych dier s bielou dierou sa algoritmus nepriblížil známej optimálnej hodnote. Táto optimálna hodnota je rovná 9147, pričom najlepšia hodnota nášho experimentu dosiahla hodnotu 6804. Avšak algoritmus čiernych dier v odbornej literatúre nebol testovaný na takých veľkých inštanciách.



Obrázek 7.17: Porovnanie úspešnosti algoritmu BH-WH pre rôzne hodnoty iterácií a jedincov



Obrázek 7.18: Porovnanie úspešnosti algoritmu BH-WH pre rôzne hodnoty konštánt



Obrázek 7.19: Porovnanie úspešnosti algoritmu BH a jeho modifikácií medzi sebou

Kapitola 8

Záver

Cieľom tejto diplomovej práce bolo preskúmať optimalizačné algoritmy a otestovať ich úspešnosť pri riešení NP-ťažkých problémov. Bol zvolený algoritmus optimalizácie hejnom častíc, algoritmus čiernych dier a algoritmus gravitačného vyhľadávania. Všetky algoritmy patria medzi algoritmy založené na populácii, pričom posledné dva patria do kategórie algoritmov inšpirovaných fyzikálnymi javmi, v našom prípade konkrétne gravitáciou.

Ako problémy, na ktorých bola testovaná úspešnosť týchto algoritmov, boli zvolené problém obchodného cestujúceho, ktorý má široké využitie v praxi a patrí medzi často študované problémy. Ako ďalší bol zvolený problém batohu, ktorý je tiež uplatniteľný v praxi a často študovaný. Oba problémy patria medzi ťažké, čo znamená, že nie sú riešiteľné v reálnom čase pomocou bežných metód. Zvlášť pre inštancie s vyšším počtom miest pre problém obchodného cestujúceho a položiek pre problém batohu.

Práca je delená nasledujúcim spôsobom. V druhej kapitole sú popísané rôzne stochastické optimalizačné algoritmy a ich delenie do rôznych kategórií, ako napríklad evolučné algoritmy, algoritmy inšpirované kolektívnym správaním, optimalizácie inšpirované fyzikou a iné princípy. V tretej kapitole je podrobne popísaný algoritmus optimalizácie hejnom častíc, ktorý je jedným z prvých algoritmov založených na inteligencii skupiny. V štvrtej kapitole sú popísané algoritmy inšpirované gravitáciou spolu s ich využitím pri riešení praktických problémov. Piata kapitola je zameraná na podrobný popis problému obchodného cestujúceho a problému batohu, pričom sú uvedené aj možnosti ich využitia v praxi. V šiestej kapitole je popísaná implementácia vybraných algoritmov a ich vybrané modifikácie. Siedma kapitola je zameraná na popis experimentov a zhrnutie výsledkov.

V budúcnosti je možné rozšíriť implementáciu o ďalšie optimalizačné algoritmy, ako napríklad genetické algoritmy alebo algoritmy kolónie mravcov, ktoré patria medzi často študované a používané optimalizačné algoritmy inšpirované prírodou. Tiež by bolo možné v budúcnosti rozšíriť implementáciu o ďalšie modifikácie, prípadne aplikovať algoritmy na iné problémy z množiny NP.

Literatura

[1] Abualigah, L., Elsayed Abd Elaziz, M., Sumari, P., Khasawneh, A., Al Shinwan, M. et al. Black hole algorithm: A comprehensive survey. *Applied Intelligence*. 2022, sv. 52, s. 1–24.

[2] Alhudhaif, A., Saeed, A., Imran, T., Kamran, M., Alghamdi, A. et al. A Particle Swarm Optimization Based Deep Learning Model for Vehicle Classification. *Computer Systems Science and Engineering*. 2022, sv. 40, s. 223–235.

[3] Assi, M. a Haraty, R. A. A Survey of the Knapsack Problem. In: *2018 International Arab Conference on Information Technology (ACIT)*. 2018, s. 1–6.

[4] Atashpaz Gargari, E. a Lucas, C. Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. In: *2007 IEEE Congress on Evolutionary Computation*. 2007, s. 4661–4667.

[5] Beheshti, Z. a Shamsuddin, S. M. H. A review of population-based meta-heuristic algorithms. *Int. J. Adv. Soft Comput. Appl.* 2013, sv. 5, č. 1, s. 1–35.

[6] Beichl, I. a Sullivan, F. The Metropolis Algorithm. *Computing in Science & Engineering*. 2000, sv. 2, č. 1, s. 65–69.

[7] Bornatico, R., Pfeiffer, M., Witzig, A. a Guzzella, L. Optimal sizing of a solar thermal building installation using particle swarm optimization. *Energy*. 2012, sv. 41, č. 1, s. 31–37. ISSN 0360-5442. 23rd International Conference on Efficiency, Cost, Optimization, Simulation and Environmental Impact of Energy Systems, ECOS 2010.

[8] Brabazon, A., O'Neill, M. a McGarraghy, S. *Natural Computing Algorithms*. Springer Berlin Heidelberg, 2015. Natural Computing Series. ISBN 9783662436318.

[9] Chen, Y., Wang, Y., Cao, L. a Jin, Q. An Effective Feature Selection Scheme for Healthcare Data Classification Using Binary Particle Swarm Optimization. In: *2018 9th International Conference on Information Technology in Medicine and Education (ITME)*. 2018, s. 703–707.

[10] Cuevas, E., Oliva, D., Zaldivar, D., Pérez Cisneros, M. a Sossa, H. Circle detection using electro-magnetism optimization. *Information Sciences*. 2012, sv. 182, č. 1, s. 40–55. ISSN 0020-0255. Nature-Inspired Collective Intelligence in Theory and Practice.

[11] Deng, Y., Liu, Y., Zeng, R., Wang, Q., Li, Z. et al. A novel operation strategy based on black hole algorithm to optimize combined cooling, heating, and

power-ground source heat pump system. *Energy*. 2021, sv. 229, s. 120637. ISSN 0360-5442.

[12] Dordevic, M., Tuba, M. a Djordjevic, B. Impact of grafting a 2-opt algorithm based local searcher into the genetic algorithm. In:. Srpen 2009, s. 485–490.

[13] Dorigo, M., Birattari, M. a Stutzle, T. Ant colony optimization. *IEEE computational intelligence magazine*. IEEE. 2006, sv. 1, č. 4, s. 28–39.

[14] Eiben, A. a Smith, J. *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg, 2015. Natural Computing Series. ISBN 978-3-662-44874-8.

[15] Engelbrecht, A. P. Particle Swarm Optimization. In: *Computational Intelligence*. John Wiley & Sons, Ltd, 2007, kap. 16, s. 289–358. ISBN 9780470512517.

[16] García Nieto, J., Olivera, A. C. a Alba, E. Optimal Cycle Program of Traffic Lights With Particle Swarm Optimization. *IEEE Transactions on Evolutionary Computation*. 2013, sv. 17, č. 6, s. 823–839.

[17] Ghavidel, S., Aghaei, J., Muttaqi, K. M. a Heidari, A. Renewable energy management in a remote area using Modified Gravitational Search Algorithm. *Energy*. 2016, sv. 97, s. 391–399. ISSN 0360-5442.

[18] Goyal, S. A Survey on Travelling Salesman Problem. In:. 2013.

[19] Güvenç, U., Sönmez, Y., Duman, S. a Yörükeren, N. Combined economic and emission dispatch solution using gravitational search algorithm. *Scientia Iranica*. 2012, sv. 19, č. 6, s. 1754–1762. ISSN 1026-3098.

[20] Hajihassani, M., Jahed Armaghani, D. a Kalatehjari, R. Applications of Particle Swarm Optimization in Geotechnical Engineering: A Comprehensive Review. *Geotechnical and Geological Engineering*. 2018, sv. 36, č. 2, s. 705–722. ISSN 1573-1529. Dostupné z: https://doi.org/10.1007/s10706-017-0356-z.

[21] Hassanien, A. E. a Emary, E. *Swarm intelligence: principles, advances, and applications*. CRC Press, 2016. ISBN 978-1-4987-4106-4.

[22] Hatamlou, A. Black hole: A new heuristic optimization approach for data clustering. *Information Sciences*. 2013, sv. 222, s. 175–184. ISSN 0020-0255. Including Special Section on New Trends in Ambient Intelligence and Bio-inspired Systems.

[23] Heidari, A. a Abbaspour, R. A Gravitational Black Hole Algorithm for Autonomous UCAV Mission Planning in 3D Realistic Environments. *International Journal of Computer Applications*. 2014, sv. 95.

[24] Husseinzadeh Kashan, A. League Championship Algorithm (LCA): An algorithm for global optimization inspired by sport championships. *Applied Soft Computing*. 2014, sv. 16, s. 171–200. ISSN 1568-4946.

[25] Karaboga, D., Gorkemli, B., Ozturk, C. a Karaboga, N. A comprehensive survey: artificial bee colony (ABC) algorithm and applications. *Artificial Intelligence Review*. 2014, sv. 42, č. 1, s. 21–57. ISSN 1573-7462.

[26] Karazmodeh, M., Nasiri, S. a Hashemi, S. Stock Price Forecasting using Support Vector Machines and Improved Particle Swarm Optimization. *Journal of Automation and Control Engineering*. 2013, sv. 1, s. 173–176.

[27] Kennedy, J. a Eberhart, R. Particle swarm optimization. In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. 1995, sv. 4, s. 1942–1948 vol.4.

[28] Khajehzadeh, M., Taha, M. a Eslami, M. Efficient gravitational search algorithm for optimum design of retaining walls. *Structural engineering & mechanics*. 2013, sv. 45, s. 111–127.

[29] Kirkpatrick, S., Gelatt Jr, C. D. a Vecchi, M. P. Optimization by simulated annealing. *Science*. American association for the advancement of science. 1983, sv. 220, č. 4598, s. 671–680.

[30] Marinakis, Y., Marinaki, M., Doumpos, M. a Zopounidis, C. Ant colony and particle swarm optimization for financial classification problems. *Expert Systems with Applications*. 2009, sv. 36, č. 7, s. 10604–10611. ISSN 0957-4174.

[31] Matai, R., Singh, S. a Mittal, M. Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches. In:. Listopad 2010. DOI: 10.5772/12909. ISBN 978-953-307-426-9.

[32] Mehrani, K., Mirshahvalad, A. a Abbasi, E. Portfolio Optimization Using Black Hole Meta Heuristic Algorithm. In:. 2019. Dostupné z: https://api.semanticscholar.org/CorpusID:221168335.

[33] Muhammad, B., Mohammed, S., Pebrianti, D., Aziz, N., Abdul Aziz, N. H. et al. A Black Hole Algorithm with White Hole Operators and Its Application in Parameter Tuning of Proportional-Integral-Derivative Controller. In:. Březen 2018.

[34] Munoz, R., Olivares, R., Taramasco, C., Villarroel, R., Soto, R. et al. Using Black Hole Algorithm to Improve EEG-Based Emotion Recognition. *Computational Intelligence and Neuroscience*. Hindawi. 2018, sv. 2018, s. 3050214. Dostupné z: https://doi.org/10.1155/2018/3050214.

[35] Nagpal, S., Arora, S., Dey, S. a Shreya. Feature Selection using Gravitational Search Algorithm for Biomedical Data. *Procedia Computer Science*. 2017, sv. 115, s. 258–265. ISSN 1877-0509. 7th International Conference on Advances in Computing & Communications, ICACC-2017, 22-24 August 2017, Cochin, India.

[36] Nemati, M., Salimi, R. a Bazrkar, N. Black Holes Algorithm: A Swarm Algorithm inspired of Black Holes for Optimization Problems. *IAES International Journal of Artificial Intelligence (IJ-AI)*. 2013, sv. 2.

[37] Pashaei, E. Medical Image Enhancement using Guided Filtering and Chaotic Inertia Weight Black Hole Algorithm. In: *2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. 2021, s. 37–42.

[38] Ramos, C. C. O., Rodrigues, D., Souza, A. N. de a Papa, J. P. On the Study of Commercial Losses in Brazil: A Binary Black Hole Algorithm for Theft Characterization. *IEEE Transactions on Smart Grid*. 2018, sv. 9, č. 2, s. 676–683.

[39] Rashedi, E., Nezamabadi pour, H. a Saryazdi, S. GSA: A Gravitational Search Algorithm. *Information Sciences*. 2009, sv. 179, č. 13, s. 2232–2248. ISSN 0020-0255. Special Section on High Order Fuzzy Sets.

[40] Shayeghi, H. Anarchic Society Optimization Based PID Control of an Automatic Voltage Regulator (AVR) System. *Electrical and Electronic Engineering*. 2012, sv. 2.

[41] Shvareva, E., Enikeeva, L. a Gizzatova, E. Harmony search algorithm for chemical kinetics optimization problems. In:. 2020, s. 1–4.

[42] Srinivasan, D., Loo, W. H. a Cheu, R. L. Traffic incident detection using particle swarm optimization. In: *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706)*. 2003, s. 144–151.

[43] Surantha, N., Lesmana, T. a Isa, S. Sleep stage classification using extreme learning machine and particle swarm optimization for healthcare big data. *Journal of Big Data*. 2021, sv. 8.

[44] Taghooni, S., Ramezanpour, M. a Khorsand, R. Prostate Cancer Detection through MR Images based on Black Hole Optimization Algorithm. *Jundishapur Scientific Medical Journal*. Ahvaz Jundishapur University of Medical Sciences. 2023, sv. 21, č. 6, s. 860–874. ISSN 2252-052X. Dostupné z: https://jsmj.ajums.ac.ir/article_174446.html.

[45] Vahidi, B. a Foroughi, A. Physical and Physic-Chemical Based Optimization Methods: A Review. *Journal of Soft Computing in Civil Engineering*. 2020, sv. 3, s. 12–27.

[46] Wang, J., Zhai, Z. J., Jing, Y. a Zhang, C. Particle swarm optimization for redundant building cooling heating and power system. *Applied Energy*. 2010, sv. 87, č. 12, s. 3668–3679. ISSN 0306-2619.

[47] Wijaya, I. G. P. S., Uchimura, K. a Koutaki, G. Traffic light signal parameters optimization using particle swarm optimization. In: *2015 International Seminar on Intelligent Technology and Its Applications (ISITIA)*. 2015, s. 11–16.

[48] Yang, X.-S. *A New Metaheuristic Bat-Inspired Algorithm* [online]. Nature Inspired Cooperative Strategies for Optimization, 2010 [cit. 2022-01-20]. Dostupné z: https://arxiv.org/pdf/1004.4170.pdf.

[49] Zhang, Y., Wang, S. a Ji, G. A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications. *Mathematical Problems in Engineering*. Hindawi Publishing Corporation. 2015, sv. 2015, s. 931256. ISSN 1024-123X. Dostupné z: https://doi.org/10.1155/2015/931256.