Testovací a auto-evaluační framework pro evoluční algoritmy

Bc. Petr Kolář

Diplomová práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Petr Kolář**
Osobní číslo:	**A20709**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**Kombinovaná**
Téma práce:	**Testovací a auto-evaluační framework pro evoluční algoritmy**
Téma práce anglicky:	**Test and Auto-Evaluation Framework for Evolutionary Algorithms**

Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Proveďte návrh formátu výstupních dat pro další zpracování a statistické vyhodnocení.
3. Navrhněte a implementujte rozhraní pro auto-evaluační framework.
4. Otestujte funkcionalitu vytvořeného frameworku pro různé scénáře.
5. Věnujte pozornost zabezpečení aplikace.
6. Vytvořte dokumentaci auto-evaluačního frameworku.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. BARTZ-BEIELSTEIN, Thomas, Carola DOERR, Jakob BOSSEK, et al. Benchmarking in Optimization: Best Practice and Open Issues [online]. Kolín nad Rýnem, Německo, 2020. Dostupné také z: https://cos.bibl.th-koeln.de/frontdoor/index/index/docId/902. Preprint. Cologne University of Applied Sciences.
2. ZELINKA, Ivan. *Evoluční výpočetní techniky: principy a aplikace*. Praha: BEN – technická literatura, 2009, 534 s. ISBN 9788073002183.
3. KACPRZYK, Janusz a Witold PEDRYCZ, ed. *Springer handbook of computational intelligence*. Dordrecht: Springer, 2015, lvi, 1633 s. ISBN 9783662435045.
4. ZELINKA, Ivan, Václav SNÁŠEL a Ajith ABRAHAM, ed. *Handbook of optimization: from classical to modern approach*. Berlin: Springer, c2013, xii, 1100 s. Intelligent systems reference library. ISBN 9783642305030.
5. ŽÁRA, Ondřej. *JavaScript: programátorské techniky a webové technologie*. Brno: Computer Press, 2015, 180 s. ISBN 9788025145739. Dostupné také z: http://knihy.cpress.cz/K2209.
6. MELOUN, Milan a Jiří MILITKÝ. *Statistická analýza experimentálních dat*. Vyd. 2., upr. a rozš. Praha: Academia, 2004, 953 s. ISBN 8020012540.

Vedoucí diplomové práce: **doc. Ing. Roman Šenkeřík, Ph.D.**
Ústav informatiky a umělé inteligence

Konzultant diplomové práce: **Ing. Tomáš Kadavý**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **3. prosince 2021**
Termín odevzdání diplomové práce: **23. května 2022**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 23.5.2022

Petr Kolář, v. r.
podpis diplomanta

ABSTRAKT

Tato diplomová práce popisuje vytvoření rozhraní vlastního auto-evaluačního frameworku, který slouží pro ukládání dat k pozdějšímu statistickému zpracování. Teoretická část práce zahrnuje základní pojmy z matematické optimalizace, principy evolučních algoritmů a způsoby jejich testování a ohodnocení. Praktická část této práce se věnuje návrhu vstupních a výstupních dat, architektuře rozhraní, obsahuje popis jednotlivých částí rozhraní a otestování jeho funkční části na různých testovacích scénářích.

Klíčová slova: evoluční algoritmy, optimalizace, testování evolučních algoritmů, parsování dat, Python

ABSTRACT

This master thesis describes the creation of an interface of auto-evaluation framework, which is used for storing evaluation data for later statistical processing. The theoretical part of the thesis includes basic terms of mathematical optimization, describes the principles of evolutionary algoritms and its way of benchmarking and ranking. The practical part of the thesis describes the scheme of the input and output data, architecture of the interface, includes a description of parts of the interface and testing its functional part on various test cases.

Keywords: evolutionary algorithms, optimalisation, testing of evolutionary algorithms, data parsing, Python

Tímto bych chtěl poděkovat vedoucímu práce doc. Ing. Romanu Šenkeříkovi, Ph.D. a konzultantovi Ing. Tomáši Kadavému za vstřícný přístup, odbornou pomoc a cenné rady při zpracování diplomové práce.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V posledních desetiletích se k řešení rozsáhlých a výpočetně náročných optimalizačních úloh začalo masivně využívat evolučních algoritmů. Jedná se o algoritmy založené na evoluční teorii. S rostoucím výpočetním výkonem se tyto typy algoritmů dostaly do popředí při řešení optimalizačních problémů. Evoluční algoritmy jsou výkonné, avšak otázkou zůstává, které algoritmy dokáží podávat vyšší výkon než jiné.

Problematika testování evolučních algoritmů a porovnávání výsledků na různých problémech optimalizace se stala cílem mnoha experimentů a studií. Ať už se jedná o každoroční soutěže, nebo představení nových přístupů k volbě strategie evolučních algoritmů, potřeba testování výsledků zůstává nadále.

Cílem této práce je v teoretické části popsat optimalizaci, evoluční algoritmy, jejich testování a problémy s nimi spojenými. V praktické části je zase cílem navrhnout rozhraní pro auto-evaluační framework tak, aby usnadňoval práci s daty a dokázal archivovat výsledky jednotlivých algoritmů na určených problémech pro další zpracování ve vhodném datovém formátu.

I. TEORETICKÁ ČÁST

1 OPTIMALIZACE

Optimalizací rozumíme takový proces, při kterém dochází k výběru nejlepší varianty z množiny všech možných řešení daného problému. S takovými problémy se setkáváme v běžném životě dost často. Příkladem může být navigační systém, který hledá optimální trajektorii z jednoho bodu v prostoru do druhého. Není jednoduché však jednoznačně určit, které řešení je ze všech možností to nejlepší, protože množina všech řešení je často moc velká na nalezení optima.

Z matematického hlediska se optimalizace zabývá zkoumáním extrémů funkcí. Cílem optimalizace je tedy nalezení takových hodnot parametrů funkce, ve kterých daná funkce nabývá extrému neboli minimálních/maximálních funkčních hodnot. K nalezení extrému zvolené funkce existuje spousta optimalizačních metod. Ať už se jedná o deterministické metody, stochastické metody, náhodné vyhledávání, nebo třeba i evoluční algoritmy. [1]

Optimalizace nachází široké uplatnění v různých oborech. V ekonomii je velmi častým problémem například optimalizace výroby tak, aby za co nejmenší náklady byl vygenerován co největší zisk. Další možnosti využití optimalizace jsou podle Zelinky [2] například výpočty, které vedou k optimalizaci v dopravě, stavebnictví, energetické a raketové technice, kybernetice a podobně.

1.1 Účelová funkce

Účelovou funkcí rozumíme funkci *f(x)*, kdy při procesu optimalizace, tedy hledání minima či maxima, dojde k nalezení optimálních hodnot jejích argumentů. Většinou při minimalizovaných problémech se tato funkce nazývá jako cenová funkce z angl. *cost function*, která se značí jako $f_{cost}(x)$. [3]

V rámci této práce jsou implementovány poznatky z testovacích sad funkcí, kde je prioritně hledáno minimum, proto hledáním extrému funkce se v tomto kontextu bude rozumět právě minimum.

Pro zjištění minima lze využít rovnici č. 1:

$$f_{cost}(x) > f_{cost}(x_0) \qquad \text{(Rovnice 1)}$$

Předcházející vztah vyjadřuje, že funkce jedné reálné proměnné $f_{cost}(x)$ má v bodě x_0 ostré lokální minimum, jestliže existuje okolí bodu x_0, ve kterém neexistuje x s menší hodnotou účelové funkce $f_{cost}(x)$.

Pro zjištění minima funkce více reálných proměnných lze použít rovnici č. 2. Funkce $f(x_1, \dots, x_n)$ má v bodě $A = (a_1, \dots, a_n)$ lokální minimum, jestliže existuje okolí bodu A takové, že pro všechny body $(x_1, \dots, x_n)$ platí:

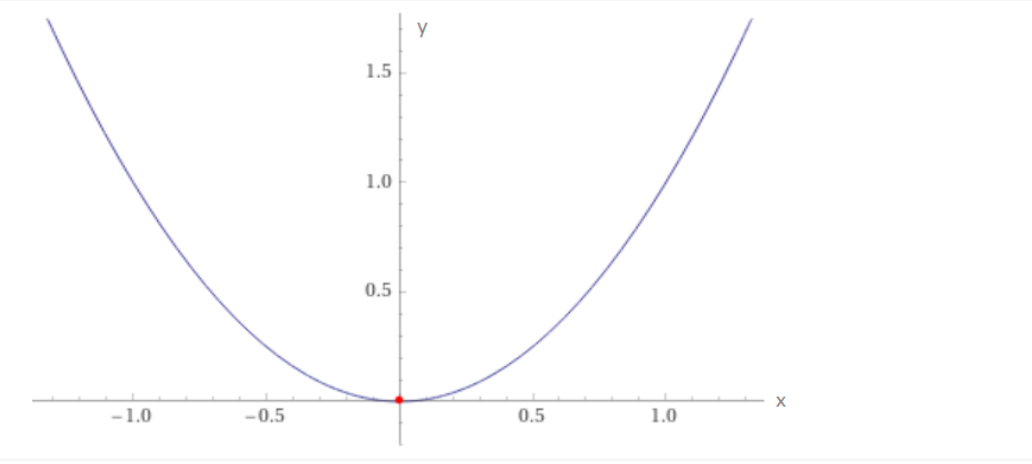
$$f_{cost}(x_1, \dots, x_n) > f_{cost}(a_1, \dots, a_n) \qquad \text{(Rovnice 2)}$$

Pokud jsou předcházející podmínky splněny nejen pro body z okolí extrému, ale pro všechny body z definičního oboru zvolené funkce, jedná se o globální extrémy.

Každá funkce reprezentuje geometrický problém, ve kterém hledáme pozici extrému na $n + 1$ rozměrné ploše. Takovéto ploše se také říká hyperplocha, nebo prostor možných řešení problému.

1.1.1 Unimodální funkce

Unimodální funkce se nazývá funkce, která má pouze jeden extrém. Tento extrém je zároveň i globálním extrémem. Příkladem může být funkce $f = x^2$, viz obrázek 1.



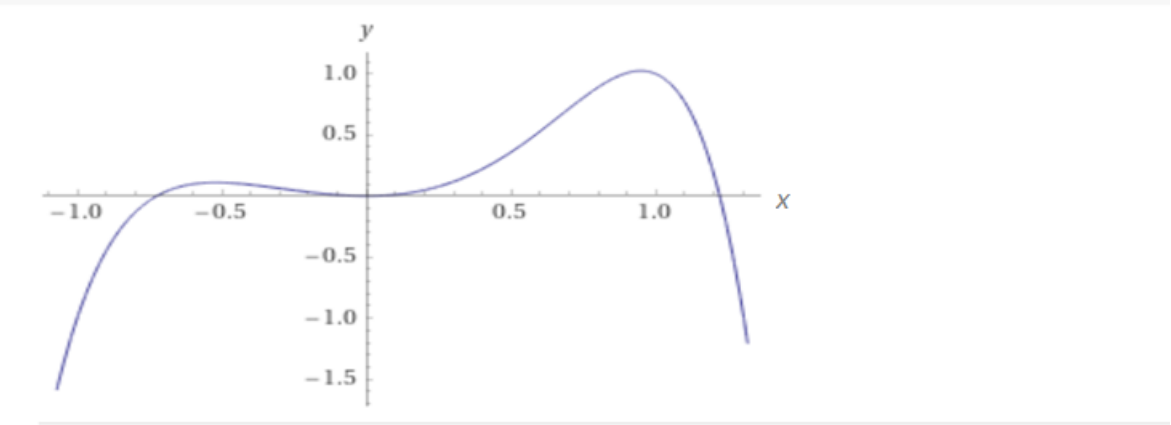
Obrázek 1: Příklad unimodální funkce

1.1.2 Multimodální funkce

Multimodální funkce je taková funkce, která má více než jeden lokální extrém. Můžeme rozlišit dva typy multimodálních funkcí podle počtu globálních extrémů.

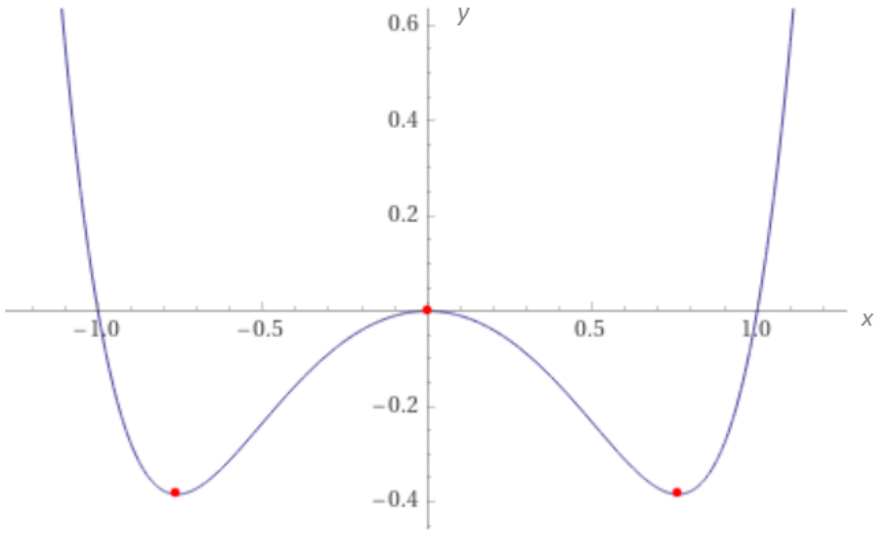
Na obrázku č. 2 je zobrazen graf funkce $f = x^3 + x^2 - (x^2)^3$. V grafu můžeme rozlišit jedno lokální minimum v bodě $x = 0$ a dvě lokální maxima. Jedno pro zápornou část definičního oboru a druhé pro kladnou část definičního oboru.

Globální maximum je pouze jedno. V tomto případě se jedná o lokální maximum kladné části definičního oboru.



Obrázek 2: *Multimodální funkce s jedním globálním maximem*

Druhou možností je funkce $f = -x^2 + (x^2)^3$, viz. obrázek 3.



Obrázek 3: Multimodální funkce se dvěma globálními minimy

V grafu můžeme pozorovat jedno lokální maximum v bodě $x = 0$ a dvě lokální minima, která jsou zároveň globální minima.

1.2 Optimalizační algoritmy

Optimalizační algoritmus je postup, který se používá k řešení různých optimalizačních úloh. Tyto algoritmy využívají rozdílné přístupy k řešení problémů, a tak je můžeme rozdělit do několika kategorií podle principů, na kterých na kterých jsou založeny, podle třídy problémů, které řeší. Obecně lze optimalizační algoritmy rozdělit následovně: [2; 3]

- **Enumerativní** – algoritmy provádí výpočet všech možných kombinací zvoleného problému. Enumerativní přístup k optimalizaci je vhodný pro řešení těch problémů, u kterých jsou účelové funkce diskrétní a mají malé množství argumentů. Výpočetní doba nalezení nejoptimálnějšího řešení narůstá s množstvím argumentů a není pro efektivní pro složité problémy například typu obchodní cestující.
- **Deterministické** – takové algoritmy stojí na základech klasické matematiky. Pro svou činnost vyžadují omezující podmínky či předpoklady. Taková omezení zvyšují efektivitu algoritmu a mohou tak podávat lepší výsledky díky menší hyperploše. Omezením může být předem známý analytický tvar funkce, známý gradient, linearita, konkávnost funkce a další. Příkladem algoritmu může být horolezecký algoritmus.
- **Stochastické** – je to skupina takových algoritmů, která využívá prvky náhody pro nalezení extrému. Argumenty účelové funkce se tedy vyhledávají náhodně, a proto jsou obvykle pomalé a nevhodné pro rozměrné prostory řešení. Dají se kombinovat i s jinými přístupy pro lepší výsledky. Například náhodné vyhledávání a následné spuštění horolezeckého algoritmu na nejlepší sadu argumentů ze stochastického algoritmu.
- **Smíšené** – jak bylo naznačeno v předcházejícím bodu, smíšené algoritmy jsou kombinací stochastických a deterministických metod optimalizace. Tato spolupráce rozdílných přístupů poskytuje dobré výsledky, které získá již za relativně malé množství ohodnocení účelové funkce. Do této skupiny algoritmů také spadají evoluční algoritmy.

1.3 Optimalizační testovací funkce

Za účelem testování optimalizačních algoritmů se používají takzvané testovací funkce. Můžeme rozlišit dva rozdílné přístupy. [2; 4]

V prvním případě vycházíme již z existujících případů, které již byly řešeny jinými algoritmy. Výsledky vlastního algoritmu, který takto testujeme, se porovnají s výsledky již existujícími. Druhý způsob zahrnuje použití množinu testovacích funkcí, které mají různé vlastnosti jako nelinearita, či obsahují patologie typu rovina okolo extrémů aj. Díky tomu, že k takovým funkcím známe analytické vztahy, je jednodušší správně vypočítat hodnotu extrému pro libovolnou dimenzi a porovnat tak výkon, robustnost, přesnost a konvergenci sledovaného optimalizačního algoritmu. Různorodost takových funkcí je vhodná pro porovnání

i mezi jednotlivými algoritmy a často je tak součástí testovacích sad. Obsah testovacích sad se různí podle potřeby a zaměření testování. V praktické části jsou využity testovací funkce pro vytvoření malé testovací sady frameworku. Pro názornost jsou uvedeny některé z nich na obrázcích 4, 5, 6 a rovnicích 3–8.

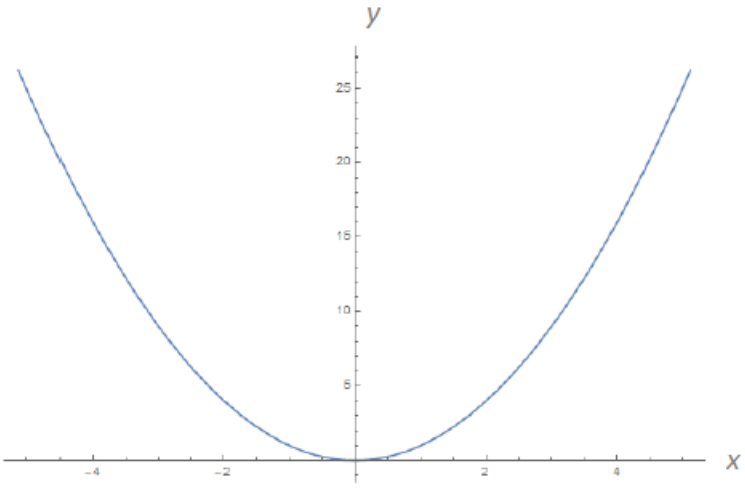
1.3.1 První De Jongova funkce

Analytický zápis:

$$f(x) = \sum_{i=1}^{D} x_i^2 \tag{3}$$

Globální minimum:

$$f(x_1 \dots x_n) = 0 \tag{4}$$



Obrázek 4: První De Jongova funkce

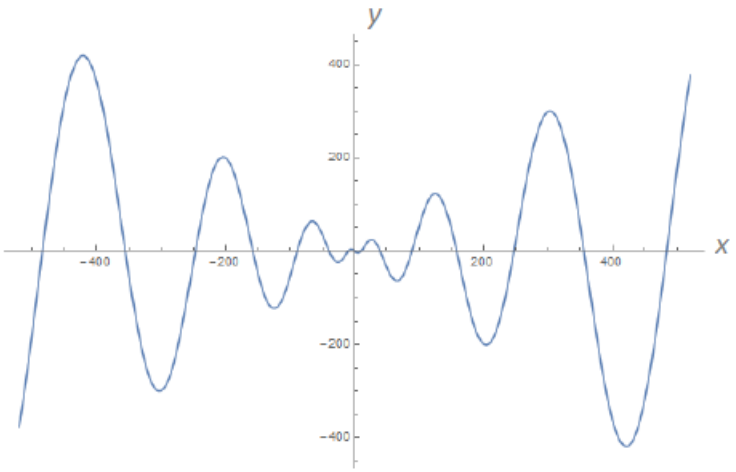
1.3.2 Schwefelova funkce

Analytický zápis:

$$f(x) = 418{,}9829 * D \sum_{i=1}^{D} - x_i \sin\left(\sqrt{|x_i|}\right) \tag{5}$$

Globální minimum:

$$f(x_0 \dots x_n) = 0 \; pro \; x = (420{,}9687 \dots 420{,}9687) \tag{6}$$



Obrázek 5: Schwefelova funkce

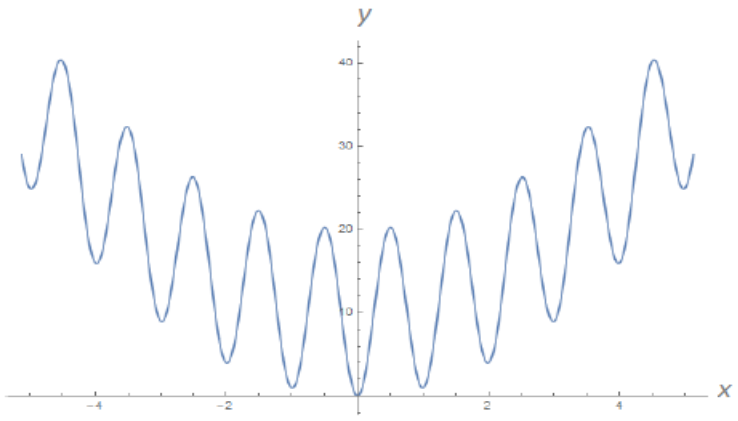
1.3.3 Rastriginova funkce

Analytický zápis:

$$f(x) = 10 * D \sum_{i=1}^{D} [x_i^2 - 10 * \cos(2\pi x_i)] \quad (7)$$

Globální minimum:

$$f(x_0 \dots x_n) = 0 \; pro \; x = (420{,}9687 \dots 420{,}9687) \quad (8)$$



Obrázek 6: Rastriginova funkce

2 EVOLUČNÍ ALGORITMY

Evoluční algoritmy spadají do kategorie smíšených optimalizačních algoritmů. Jsou součástí celku nazvaného Evoluční výpočetní techniky. Mezi tyto techniky patří například genetické algoritmy, evoluční strategie nebo hejnové algoritmy. Evoluční algoritmy představují metody řešení komplexních problémů, pro které je řešení konvenčními postupy příliš časově náročné. Evoluční algoritmy také nezávisí na gradientních informacích, a proto jsou též vhodné pro problémy, kde jsou takové informace nedostupné. [2]

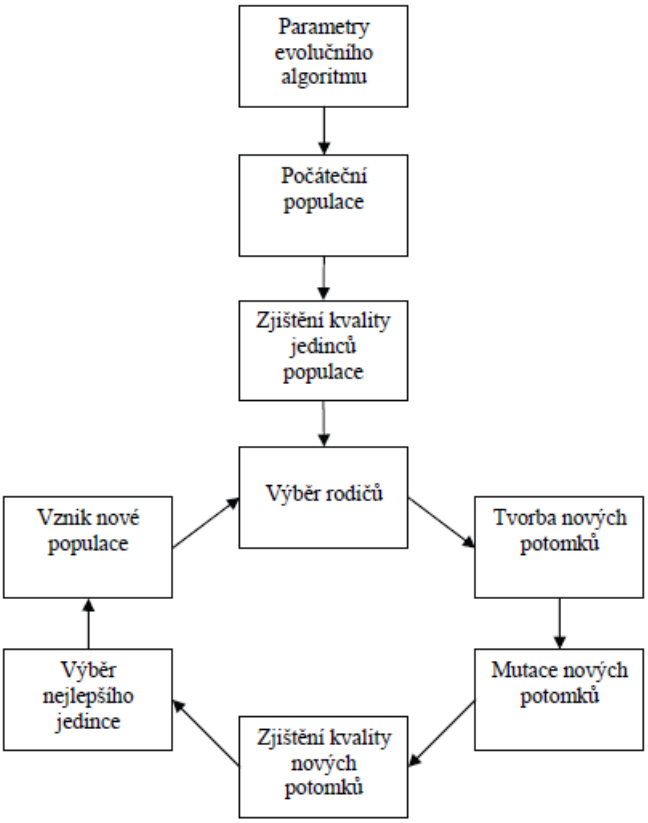
Evoluční algoritmy jsou robustní, protože nejsou závislé na počátečních podmínkách a díky evolučním principům dokážou uniknout z lokálního extrému. [3]

Hlavní myšlenkou těchto algoritmů je předávání rodičovského genomu novým potomkům a následném uvolnění místa v populaci pro další generace, nebo využití vzorů kolektivního chování jedinců v hejnech a modelování této komunikace, interakce a rozhodování na základě omezených znalostí.

2.1 Princip evolučních algoritmů

Tyto algoritmy pracují s populací tvořenou jedinci, kteří reprezentují kandidátní řešení optimalizovaného problému. Populací tedy rozumíme množinu všech jedinců dané generace (nebo iterace v rámci hejnových algoritmů). Princip je patrný na obrázku 7.

Na počátku se musí vymezit parametry, které řídí běh algoritmu. Patří sem například ukončovací kritéria, velikost populace, práh mutace a další. Nastavení není univerzální a každý algoritmus může mít jiné parametry. Následně se vygeneruje prvotní populace. Tento krok se často provádí stochasticky. Provede se zjištění kvality celé populace pomocí ohodnocení účelovou funkcí. Populace se postupně vyvíjí podle evolučních principů tak, že se z rodičů (z pravidla křížením) vytváří noví jedinci. Volba, kteří jedinci se budou účastnit křížení, probíhá na základě ohodnocení účelové funkce a selekce, což je mechanismus zajišťující variabilitu populace tím, že selekcí mohou projít (zpravidla s menší pravděpodobností) i jedinci s menším ohodnocením. Nově vzniklí jedinci mohou podléhat mutacím, které dále diverzifikují populaci. Tento krok bývá obecně stochastický. Poté nejslabší jedinci vymírají cyklicky po generacích (cyklus generace je znázorněn na obrázku dole) a uvolňují místo novým a lepším potomkům, kteří se stávají novými rodiči podle přesně definovaných pravidel. [2; 3]



Obrázek 7: Princip evolučních algoritmů; převzato z [2]

Způsob reprezentace jedinců v populaci určuje, o jaký evoluční algoritmus se jedná. V dnešní době existuje velké množství takových algoritmů a jejich variant. Žádný ale není univerzální nebo nejlepší pro všechny typy problémů (no free lunch teorém). [4]

Důležitou součástí těchto algoritmů je takzvaný elitismus. Jedná se o princip, který zajišťuje, že v každé následující generaci budou minimálně stejně kvalitní jedinci jako v předchozí generaci. Tímto způsobem účelová rychleji konverguje ke globálnímu extrému. [4]

V kontextu evolučních algoritmů se můžeme setkat s názvem *cost function* nebo *fitness function*, což bychom do češtiny mohli přeložit jako cenu nebo vhodnost. V rámci evolučních algoritmů tak hodnota *fitness* funkce vyjadřuje míru schopnosti jedince přežít, adaptovat se a předat svou genetickou informaci dalším generacím. Tato funkce *fitness* se může zaměnit s účelovou funkcí, avšak oproti ní se jedná o konverzi účelové funkce do intervalu *<0,1>*. Toho se využívá především u genetických algoritmu, kde je použita binární reprezentace jedinců. [3]

2.2 Vybraní zástupci evolučních algoritmů

Následující algoritmy byly implementovány v rámci praktické části této práce.

2.2.1 SOMA

SOMA je zkratka pro Samoorganizující se migrační algoritmus. Od ostatních evolučních algoritmů se odlišuje především tím, že noví potomci nevznikají křížením rodičů. Jeho princip je založen na sociální spolupráci mezi jedinci při migraci jedinců v prostoru řešení daného optimalizačního problému. Jedná se tedy o hejnový algoritmus s funkcí mutace. [2]

Mutace je řízena perturbačním vektorem, což je stochastická složka pohybu, která ovlivňuje výpočet trajektorie směrem k vybranému jedinci. Pohyb je prováděn v podobě několika skoků napříč prostorem hledání.

Cyklus jedné populace se nenazývá generace, ale z povahy algoritmu se nazývá migrace. V průběhu migrace dochází k samoorganizaci populace tak, že se jednotliví jedinci navzájem ovlivňují, vznikají shluky jedinců, které migrují společně. Toto chování bylo inspirováno obdobou v přírodě, kdy skupiny jedinců spolupracují při hledání potravy. [2]

2.2.1.1 Nastavení parametrů

Před spuštěním je potřeba nadefinovat následující parametry:

- **PathLength** – délka cesty určuje, v jaké vzdálenosti od leadera se migrující jedinec zastaví. Vzdálenost mezi dvěma jedinci je vynásobena tímto parametrem, a proto při hodnotě 1 se aktivní jedinec zastaví přesně na stejné pozici, jako se nachází leader, což degeneruje proces migrace.
- **Step** – nastavuje počet mezizastávek před dosažením požadované délky cesty. Při každém kroku se vypočítává hodnota účelové funkce. Vyšší hodnota parametru tak urychluje algoritmus a zvyšuje pokrytí plochy řešení. Zároveň ale zvyšuje riziko uváznutí v lokálním extrému.
- **D** – dimenze neboli počet argumentů účelové funkce
- **PopSize** – počet jedinců v populaci
- **Prt** – perturbace
- **Migrace** – počet migračních kol

2.2.1.2 Strategie algoritmu

Existuje několik strategií migrace:

- **All to One** – všichni jednotlivci z populace migrují k jednomu vybranému leaderovi. Leaderem je zpravidla nejvhodnější jedinec z populace (na základě hodnoty účelové funkce)
- **All to Random** – Leader je vybrán náhodně (bez ohledu na hodnotu účelové funkce) ze všech jedinců na začátku každé migrace.
- **All to All** – Každý jedinec migruje směrem ke všem ostatním jedincům. Pozice se aktualizují až po dokončení migrace všemi jedinci.
- **All to All Adaptive** – Stejné jako předchozí varianta. Jediným rozdílem je, že se pozice upravují průběžně po každém jednotlivém pohybu aktivního jedince. Díky tomu může migrovat k dalšímu jedinci již z aktualizované pozice.

.

3 TESTOVÁNÍ EVOLUČNÍCH ALGORITMŮ

Po vytvoření nového algoritmu nebo modifikaci učících a adaptivních mechanismů ve stávajících algoritmech je nasnadě provést testy ke zjištění jeho efektivity v hledání řešení napříč různě složitými a omezenými problémy tak, jak bylo naznačeno v kapitole optimalizace.

Stejně jako neexistuje algoritmus, který by zvládal jakoukoli optimalizační úlohu lépe než všechny ostatní algoritmy, existuje také velké množství rozdílných testovacích funkcí a sad, které se hodí na testování specifických problémů a neexistuje nejvhodnější testovací set, který by vyhovoval pro každý zkoumaný problém. Specifikace cílů testovacích studií, které se používají na hodnocení výkonnosti algoritmů, jsou tedy klíčové pro správný experimentální přístup k testování evolučních algoritmů. [5]

Zlepšení algoritmu pro jednu skupinu problémů s velkou pravděpodobností zhorší výsledky daného algoritmu pro jiné problémy. [5]

Před vytvořením testovací knihovny a samotným testováním je potřeba specifikovat sadu účelových funkcí, sadu evolučních algoritmů, které se budou účastnit testování a ohodnocení, je potřeba definovat evaluační kritéria a výstup, kterým mohou být hrubá statistická data, výsledek porovnání jednotlivých algoritmů pomocí statistických metod, nebo porovnání na bázi ranku.

3.1 Výkonnost algoritmu

Pro testování se dají rozlišit dva rozdílné přístupy. V prvním vycházíme z již existujících reálných optimalizačních případů, které již byly řešeny jinými algoritmy. Výsledky právě testovaného algoritmu porovnáme s již existujícími výsledky.

Druhý způsob testování evolučních algoritmů spočívá v použití množiny testovacích funkcí neboli benchmarkovací sady, která obsahuje funkce s různými vlastnostmi. V tomto případě využíváme statistický základ. Jelikož se jedná o heuristiky, je potřeba provést několik (min. 30) opakování, aby se ověřila statistická úspěšnost a výkon daného algoritmu a předešlo se výchylkám, které jsou způsobeny stochastickými prvky jako je mutace nebo náhodná počáteční populace. [5]

Práce se zaměřuje na druhý typ, tedy využití testovacích sad. Pro ty můžeme rozlišit dva způsoby auto-evaluace evolučních algoritmů.

3.1.1 Target type

Tento typ testování definuje ukončovací podmínky tak, že testování končí, pokud je nalezen předem definovaný cíl, anebo je dosažena požadovaná numerická přesnost hodnoty účelové funkce. Nehraje zde roli omezení času či počtu ohodnocení účelové funkce. Výhodu zde mají algoritmy, které mají implementovanou mechaniku re-inicializace populace po nalezení dílčího cíle nebo požadované přesnosti.

Příklady tohoto benchmarku jsou:

- IEEE CEC 2019 100 digits competiotion
- BBOB test bed

3.1.2 Budget type

Tento typ testování uvažuje, že ohodnocení účelové funkce je časově náročné. Proto nehledá čistě nejlepší řešení bez ohledu na čas (jako u předcházejícího typu), ale hledá nejlepší řešení do určeného počtu ohodnocení účelové funkce. Jinými slovy se hledá algoritmus, který podává nejlepší výsledky efektivně a v omezeném počtu ohodnocení.

Vyhodnocují se zde statistické vlastnosti výsledků opakovaného spouštění. Dle [bart] se za centrální prvky statistické analýzy považuje minimum, maximum, průměr, medián, směrodatnou odchylku, 25% a 75% kvantil. Každá účelová testovací funkce ze sady má vlastní statistické vlastnosti a účelem ohodnocení je spočítat kvalitu řešení napříč všemi funkcemi ve všech iteracích. Data mohou být zatížena odchylkami vlivem stochastické části algoritmu, nízkým počtem maximálního ohodnocení účelové funkce, nebo také nevhodně zvolenými parametry. Tyto odchylky je pak potřeba vzít na vědomí při analýze výsledků. Je to také důvod, proč analytici preferují použití mediánu (více robustní) před průměrem. [5].

Minimum a maximum poskytují informaci o rozpětí populace a ve statistickém měřítku také o kvalitě algoritmu.

Příkladem mohou být benchmarkovací sady:

- IEEE CEC 2011 competition
- IEEE CEC 2015 competition
- IEEE CEC 2017 competition
- IEEE CEC 2021 competition

3.2 Způsoby vyhodnocení

Pro porovnání výsledků můžeme použít statistické neparametrické testy. Ke srovnání dosažených výsledků dvojice algoritmů používáme Wilcoxon Sign/Sum Rank test podle toho, zda je počáteční populace rozdílná, nebo stejná. Pokud porovnáváme více algoritmů, používáme Friedman Rank testy a dodatečné post-hoc testy. [5]

Další možností je vyhodnocování na základě míry úspěšnosti daného algoritmu. Počítá se podíl úspěšných evaluací na testovací sadě, přičemž úspěch lze definovat různým způsobem. Například může jít dosažení požadované přesnosti řešení za určitý počet ohodnocení.

Případně lze sledovat průměrný počet ohodnocení účelové funkce, který vedl k získání úspěšného řešení za daný počet opakování na jednotlivých testovacích funkcích.

3.3 Testovací sady CEC

Testovací sady CEC (Congress on Evolutionary Computation) jsou vytvářeny pod záštitou organizace IEEE. Jedná se o testovací sady, které jsou výzkumníky široce používány na testování evolučních algoritmů. Každý rok se vytváří nová sada o různé velikosti a zaměření. Funkce se obměňují anebo upravují každý rok. Dále probíhá soutěž, kdy se hledá nejlepší algoritmus pro danou sadu funkcí.

Problémem těchto testových sad zůstává omezení počtu ohodnocení účelové funkce (FEZ), protože i zdánlivě neefektivní či stagnující algoritmus může podávat dobré výsledky při vyšším počtu FEZ. [6].

Dalším problémem je, že výsledky jsou platné jen pro danou testovací sadu a nemohou být generalizovány nebo použity pro porovnání s jinou sadou. Anebo problém vynechání množiny vítězných algoritmů z další verze testovací sady, jenž fungují jako dobrý referenční ukazatel výkonnosti pro nové algoritmy. [7]

II. PRAKTICKÁ ČÁST

4 ARCHITEKTURA ROZHRANÍ

Pro implementaci rozhraní byl zvolen jazyk Python, jelikož se jedná o jeden z nejpoužívanějších jazyků v rámci data science a analýzy dat. [8] Obsahuje spoustu volitelných modulů a balíčků, které po instalaci rozšíří Python o nové funkcionality. Za zmínku stojí především tyto:

- **NumPy** – open source knihovna usnadňující práci s polem, maticemi, lineární algebrou a Fourierovou transformací. Tato knihovna umožňuje mnohem výkonnější práci s poli oproti standardním poli v Pythonu, a proto je široce využívána v oblasti data science. Knihovna je také optimalizována na nejnovější CPU architektury. Více informací o knihovně v [9]
- **Matplotlib** – open source vizualizační knihovna na zprostředkování statických, animovaných a interaktivních informací
- **SciPy** – open source knihovna, která využívá knihovnu **NumPy**. Jedná se o knihovnu, která obsahuje široké spektrum funkcí pro optimalizaci, statistiku a zpracování signálů. Tato knihovna také obsahuje funkce pro provádění Friedmanova nebo Wilcoxonova testu. [10]
- **Pandas** – blíže bude popsán v následující pod-kapitole

4.1 Informace o použitých knihovnách a formátech

4.1.1 Pandas

Knihovna pandas je open source knihovna, která slouží pro manipulaci s daty a jejich analýzu. Představuje optimalizovaný a efektivní objekt DataFrame, který je vhodný pro manipulaci s daty a obsahuje integrované indexování pro rychlejší práci s daty. Zobrazení dat je provedeno tabulkou.

DataFrame objekt z knihovny pandas lze dále nativně exportovat do dalších formátů dle potřeby. Může jím být například excel, csv, xml, html, latex, SQL aj. Součástí jsou také funkce pro načítání dat do formátu DataFrame, což činí z této knihovny univerzální nástroj pro práci s daty napříč různými strukturami. [11]

Tato knihovna obsahuje funkce na seskupení, řazení, spojování a re-modelování data setu. Taktéž je ošetřena proti chybějícím datům, které automaticky (ve formě „N/A“) doplňuje do své struktury. Součástí je také integrace časových řad.

Jelikož pandas interně využívá knihovnu NumPy, je tato knihovna a její datový formát podporován mnoha moduly pro data science. Knihovny jako NumPy, Matplotlib a SciPy umí pracovat přímo s formátem DataFrame jako datovým objektem.

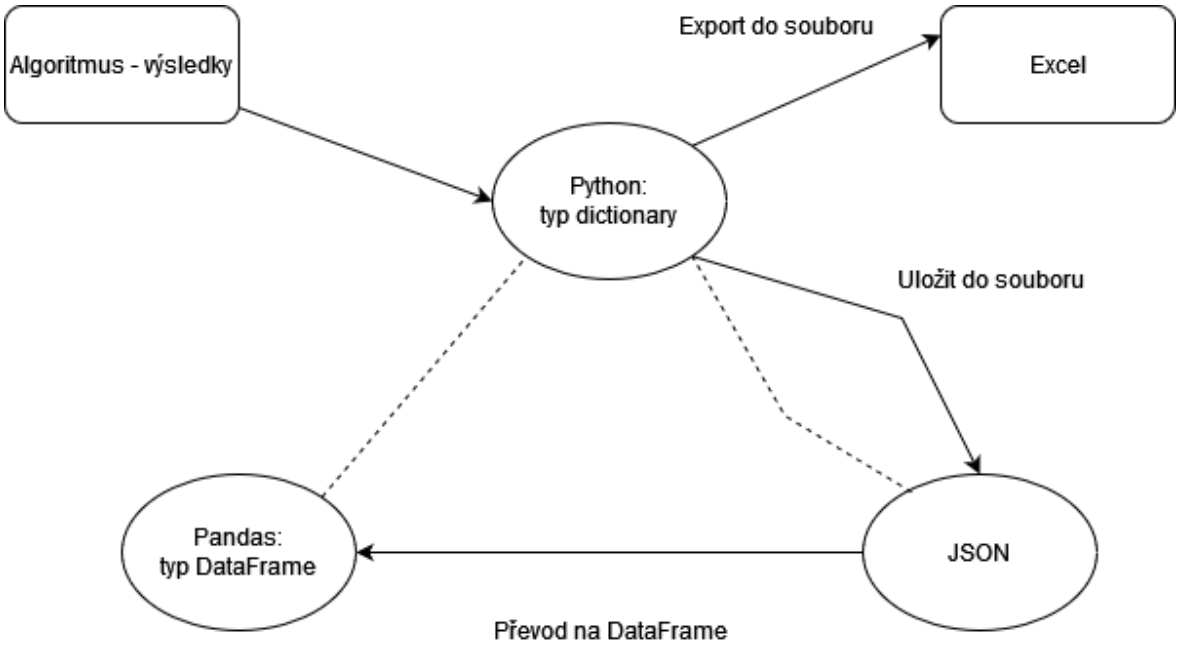
Z tohoto důvodu je tato knihovna využita pro rozhraní auto-evaluačního frameworku. Datový analytik tak může pomocí pandas vybrat a organizovat vhodná data, která použije jako například vstup do funkcí knihovny SciPy pro následné statistické vyhodnocení daného data setu. Ohodnocení lze tak provádět pomocí knihoven (např. SciPy) nad objektem DataFrame. Popřípadě lze tyto data interpretovat pomocí výše uvedených knihoven.

4.1.2 JSON

JSON (JavaScript Object Notation) je jednoduchý formát, který se využívá pro přenos a uchování informací. Díky své jednoduché, ale efektivní struktuře je dobře čitelný a lze parsovat a číst v mnoha modulech napříč programovacími jazyky. Je založen na párech *klíč: hodnota*, které se mohou v daném jazyce implementovat jako hash tabulka, list, nebo slovník. Pro ukládání výsledků algoritmů byl zvolen právě tento formát. [12]

4.2 Schéma rozhraní

Obrázek č. 8 popisuje tok dat frameworku:



Obrázek 8: Schéma rozhraní

Popis schématu:

- **Algoritmus** – tato část reprezentuje funkci daného algoritmu. Poskytuje výsledky evaluace algoritmu a ukládá je do nativního objektu jazyka Python, což je typ dictionary.
- **Python** – tato část reprezentuje data v paměti IDE. Z ní lze exportovat přímo do excelu pomocí vytvořených funkcí, nebo do formátu JSON.
- **Excel** – jedná se o jednorázový export buď statistických dat dané evaluace (spuštění algoritmu pro daný set parametrů a nastavení), nebo dat všech opakovaných průběhů algoritmu na všech zvolených testovacích funkcích
- **JSON** – data z evaluace se uloží do jednoho souboru JSON. Předchozí evaluace se nachází v témže souboru. Pokud soubor neexistuje, vytvoří se nový. Soubor tak slouží jako databáze všech provedených evaluací. Soubor má specifickou strukturu, která bude popsána níže.
- **DataFrame** – jedná se o část frameworku, která se stará o formátování dat, načtení dat ze souboru JSON a vizualizaci v tabulkách. Dále tento formát lze použít pro další statistické zpracování a slouží jako výchozí datový formát pro další práci s daty.

4.3 Struktura souboru JSON

Jak již bylo zmíněno, soubor formátu JSON je databází celého frameworku a uchovává data jednotlivých výsledků každého algoritmu. Při návrhu struktury bylo vycházeno z předpokladu, že není určen experiment, který bychom požadovali testovat. Nejsou dopředu známy účelové funkce, ani počet dimenzí pro testování. Proto byl zvolen, robustní přístup, kdy se data z nových evaluací neukládají sekvenčně za sebou, ale podle jednotlivých dimenzí a počtu ohodnocení účelové funkce.

Tímto způsobem je možné přistoupit k uloženým datům i později a separovat je mnohem rychleji podle nového nastavení. Data tak nebudou závislá na jednom nastavení pro testovací sadu (jako například v CEC test bedu). Tento přístup k ukládání dat v databázi oddělil data s jiným počtem dimenzí od dat s jiným počtem ohodnocení. Poté je již na datovém analytikovi, jak navrhne experiment nad vlastě určenou sadou.

Struktura dat je následující:

```
{
  "lastId": 4,
  "dimensions": [{
      "numberOfDimensions": 1,
      "fez": [{
          "numberOfFez": 100,
          "evaluations": [{
              "id": 0,
              "nameOfAlgorith": "SOMA",
              "executionDateTime": "19.05.2022 06:06",
              "parameters": {
                "pathLength": 20,
                  "step": 1,
                  "populationSize": 10,
                  "prt": 5 },
              "numberOfReruns": 4,
              "bestCostPerRun": {
                  "Dejong_1": [1, 2, 0.4, 1],
                  "Schweffel": [],
                                    ...
```

Objekt JSON obsahuje *lastId*, které je pravidelně aktualizováno při vkládání nových dat do souboru. Toto id slouží jednoznačné identifikaci evaluace jednoho algoritmu pro danou dimenzi a hodnotu FEZ (maximální počet ohodnocení účelové funkce). Může se použít pro přístup na konkrétní evaluaci a její data.

Dále obsahuje *dimensions*, což je objekt, ve kterém jsou uloženy všechny existující dimenze, ve kterých existuje alespoň jedna evaluace. Pokud dimenze neexistuje a nový algoritmus tuto dimenzi spouští prvně (v kontextu JSON dat), přidá se nový objekt s touto dimenzí.

Dimensions obsahuje *numberOfDimensions*, což je parametr určující velikost dimenze a FEZ, který reprezentuje všechny maximální počty ohodnocení účelové funkce, které pro danou dimenzi existují.

Struktura objektu FEZ je totožná s objektem *dimensions* (pouze jiný kontext dat). Každý objekt FEZ obsahuje *evaluations*, což je soubor všech provedených evaluací pro danou dimenzi a FEZ.

Evaluations objekt obsahuje *„meta data“*, která vyjadřují název algoritmu, id evaluace, datum a čas provedení, parametry algoritmu a počet opakovaných spouštění algoritmu. Také obsahuje *bestCostPerRun*, což je objekt, ve kterém jsou obsaženy formou *klíč: hodnota* všechny účelové/testovací funkce, ke kterým algoritmus hledal extrém. Výsledky jsou uvedeny v poli, a to v pořadí v jakém bylo postupně získáno nejlepší řešení pro každé opakování algoritmu.

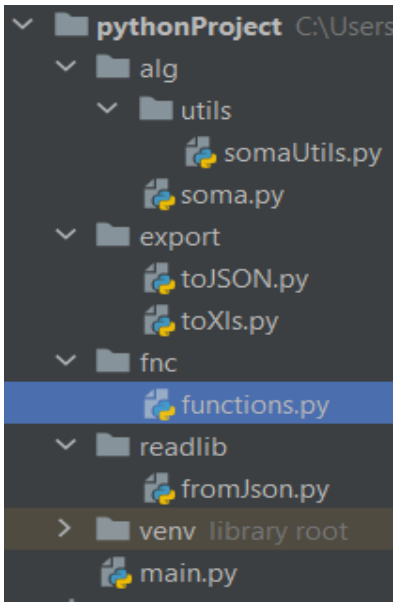
5 DOKUMENTACE

Tato kapitola pojednává o souhrnu funkcí frameworku a jeho ovládání.

5.1 Struktura projektu

Obrázek 9 ukazuje strukturu projektu:

- **alg** – složka s implementovanými algoritmy
  - **utils** – složka s dodatečnými funkcemi pro konkrétní algoritmus
- **export** – složka s implementacemi funkcí pro ukládání do souboru JSON a Excel
- **fnc** – složka se souborem definic účelových/testovacích funkcí
- **readlib** – složka s funkcemi pro parsování a načítání dat z JSON do formátu DataFrame
- **venv** – složka interpreteru. Jsou zde uloženy nainstalované knihovny – vytvářeno automaticky Pythonem
- **main.py** – hlavní funkce a zároveň pracovní prostředí, ze kterého se používá rozhraní frameworku



Obrázek 9: Struktura adresáře

5.2 Export

Export obsahuje dva soubory *toJSON.py* a *toXLS.py*.

5.2.1 Soubor toJSON.py

Tento soubor obsahuje pouze jednu funkci:

- **create_json**(*fileName, dimension, fezNumber, resultsPerFnc, reruns, algName, paramsDict*)

Popis: Funkce zpracuje výsledky spouštěného algoritmu do výše popsané struktury JSON a vytvoří/aktualizuje soubor JSON s příslušnými daty.

Argumenty funkce:

- *fileName* – je název souboru, který se má otevřít. Pokud soubor není nalezen v kořenovém adresáři projektu, soubor se vytvoří právě s tímto názvem.
- *Dimensions* – je to počet dimenzí, pro který se testovaný algoritmus spouštěl. Tento parametr se přidává do objektu *numberOfDimensions* ve struktuře JSON.
- *fezNumber* – je to počet ohodnocení účelové funkce, pro který se testovaný algoritmus spouštěl. Tento parametr se přidává do objektu *numberOfFez* ve struktuře JSON.
- *resultsPerFnc* – je to objekt typu dictionary, která obsahuje výsledky všech běhů algoritmu. Jsou následujícího formátu:

```
{
    'Dejong_1': [1.0598146951379338e-09, 0.06021491974340745],
    'Schweffel': [-837.9657745382847, -817.9367998306649],
    'Dejong_2': [3.509698641225608e-08, 91.6553635299764],
    'Rastrigin': [10.000000000000002, 10.321796236579733]
}
```

  Klíčem je název účelové funkce a hodnotou je pole všech výsledků (nejlepší hodnota účelové funkce z poslední generace/iterace). Zde jsou zobrazeny dvě opakování algoritmu, proto má pole pouze dvě hodnoty.

- *reruns* – jedná se o počet opakování běhu algoritmu nad testovacími funkcemi
- *algName* – je to název algoritmu, který bude uložen do souboru
- *paramsDict* – je to také objekt typu dictionary, který obsahuje všechny parametry, se kterými byl spouštěn algoritmus. Zde je ukázka parametrů z algoritmu SOMA.

```
{
    'pathLength': 3,
    'step': 0.33,
    'populationSize': 50,
```

```
    'prt': 0.3
}
```

Výstup funkce: Funkce nevrací žádný objekt, pouze zobrazí na konzoli postup ukládání, nebo chybovou hlášku.

5.2.2 Soubor toXls.py

Tento soubor obsahuje dvě veřejné funkce:

- **export_statistics**(*costFncsDict, bestResults, reruns, nameFile*)
- **export_results**(*costFncsDict, bestResults, reruns, nameFile*)

Popis: Funkce *export_statistics* vytváří soubor excelu s tabulkou obsahující statistické charakteristiky všech nejlepších hodnot účelových funkcí. Funkce *export_results* vytváří soubor excelu, který obsahuje všechny nejlepší hodnoty pro všechna opakování daných účelových funkcí. Obě mají stejné argumenty.

Argumenty funkce:

- *costFncsDict* – je to objekt typu dictionary, který obsahuje účelové funkce. Klíčem je pořadí účelové funkce. Hodnotou je její název:

```
costFncsDict = {
    1: 'Dejong_1',
    2: 'Schweffel',
    3: 'Dejong_2',
    4: 'Rastrigin'
}
```

- *bestResults* – je to objekt typu dictionary, který obsahuje výsledky všech běhů algoritmu. Objekt je následujícího formátu:

```
{
    'Dejong_1': [1.0598146951379338e-09, 0.06021491974340745],
    'Schweffel': [-837.9657745382847, -817.9367998306649],
    'Dejong_2': [3.509698641225608e-08, 91.6553635299764],
    'Rastrigin': [10.000000000000002, 10.321796236579733]
}
```

- *reruns* – jedná se o počet opakování běhu algoritmu nad testovacími funkcemi
- *fileName* – název souboru, který se má vytvořit

Výstup funkce: Funkce nevrací žádný objekt, pouze vytváří soubor *.xlsx* viz obrázek 10 a 11.

Reruns_20	MIN	MAX	MEAN	MEDIAN	STR DEV
Dejong_1	1,54178E-09	55,37167879	36,15992695	36,95912594	12,04677295
Schweffel	-3538,407135	-930,4903159	-1448,479167	-1298,175296	596,648147
Dejong_2	2,06351E-07	6252,17315	3493,015648	3437,247989	1360,023901
Rastrigin	90	95,64079719	91,13010902	90,7279242	1,33982073

Obrázek 10: Soubor statistics.xlsx

Reruns20	1	2	3	4	5
Dejong_1	3,38298E-09	41,75721951	42,76657261	46,31292501	42,21848475
Schweffel	-3972,689203	-1412,754709	-1293,538373	-1603,054609	-1472,722076
Dejong_2	1,6356E-07	4592,625849	4964,893249	3048,12623	3388,706313
Rastrigin	90	91,00232874	91,39105869	90,17564794	93,26233936

Obrázek 11: Soubor results.xlsx

5.3 Readlib

Tato složka obsahuje jeden soubor s mnoha funkcemi pro parsování z formátu JSON na formát DataFrame. Obsaženy jsou také funkce pro formátování, výpočet statistických charakteristik a výpis souboru JSON. V následujících podkapitolách jsou uvedeny jednotlivé funkce.

5.3.1 Funkce last_id_used(fileName, display=False)

Popis: Funkce vrací *lastId* z kořenové struktury objektu JSON.

Argumenty funkce:

- *fileName* – název souboru, kde se má najít *lastId*
- *display* – je to nepovinný argument. Pokud je nastaven na hodnotu *True*, výsledek funkce se zároveň vypíše i na konzoli. Ve všech ostatních funkcích má stejný význam.

5.3.2 Funkce get_dimensions(fileName, display=False)

Popis: Funkce vrací objekt DataFrame v kontextu JSON struktury *dimensions*.

Argumenty funkce:

- *fileName* – název souboru, odkud se má vrátit *dimensions*.
- *display* – je to nepovinný argument – výsledek zobrazí na konzoli viz obr. 12.

```
   numberOfDimensions
0                  10
1                  20  [{'numberOfFez': 3000, 'evaluations': [{'id': 2, 'nameOfAl
```

Obrázek 12: Výstup get_dimensions()

5.3.3 Funkce get_all_fez_in_dimension(fileName, dimension, display=False)

Popis: Funkce vrací objekt DataFrame a index v kontextu JSON struktury *fez*.

Argumenty funkce:

- *fileName* – název souboru, odkud se má vrátit *fez*.
- *dimension* – dimenze, ze které se má *fez* získat
- *display* – je to nepovinný argument – výsledek zobrazí na konzoli viz. obr. 13.

```
   numberOfFez
0        10000  [{'id': 0, 'nameOfAlgorith': 'SOMA-1', 'executionDateTim
```

Obrázek 13: Výstup get_all_fet_in_dimension()

5.3.4 Funkce get_all_evaluations_object(fileName, dimension, fez, display=False)

Popis: Funkce vrací objekt DataFrame a index v kontextu JSON struktury *evaluations*.

Argumenty funkce:

- *fileName* – název souboru, odkud se má vrátit *evaluations*.
- *dimension* – dimenze, ze které se má *evaluations* získat
- *fez* – fez, ze které se má *evaluations* získat
- *display* – je to nepovinný argument – výsledek zobrazí na konzoli viz. obr. 14.

```
   id nameOfAlgorith executionDateTime                                                    parameters
0   0         SOMA-1  23.05.2022 19:10  {'pathLength': 3, 'step': 0.33, 'populationSize': 50, 'prt': 0.3}
1   1         SOMA-2  23.05.2022 19:10  {'pathLength': 2, 'step': 0.33, 'populationSize': 50, 'prt': 0.2}

Process finished with exit code 0
```

Obrázek 14: Výstup get_all_evaluations_object()

5.3.5 Funkce get_evaluation_by_id(fileName, id, returnBestCostsPerRun=False, display=False)

Popis: Funkce vrací objekt DataFrame a index v kontextu JSON struktury *evaluations* pro zvolené ID.

Argumenty funkce:

- *fileName* – název souboru, odkud se má vrátit *evaluations*.
- *id* – jedná se o id evaluace, která se má získat
- *returnBestCostsPerRun* – je to nepovinný argument. Pokud je nastaven na *True*, vrátí se jenom objekt *bestCostsPerRun* s výsledky.
- *display* – je to nepovinný argument – výsledek zobrazí na konzoli viz. obr. 15.

	0	1	2	3	4	5
Dejong_1	1.236855e-09	35.075072	47.798765	57.355888	38.430376	43.124101
Schweffel	-3.834513e+03	-1287.787426	-1374.617057	-1398.503395	-947.781061	-1158.009441
Dejong_2	1.900110e-07	4531.678950	3459.778692	4523.185824	5233.117567	4658.505465
Rastrigin	9.000000e+01	91.045279	91.135187	90.997648	90.653845	90.735182

Obrázek 15: Výstup get_evaluation_by_id()

5.3.6 Funkce get_all_evaluations_formatted(fileName, dimension, fez, returnBestCostsPerRun=False, normalised=True, display=False)

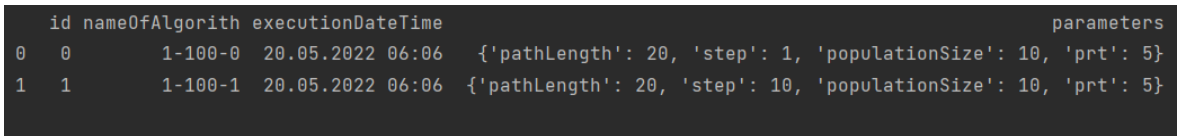
Popis: Funkce vrací objekt DataFrame v kontextu JSON struktury *evaluations*.

Argumenty funkce:

- *fileName* – název souboru, odkud se má vrátit *evaluations*.
- *dimension* – dimenze, ze které se má *evaluations* získat
- *fez* – fez, ze které se má *evaluations* získat
- *returnBestCostsPerRun* – je to nepovinný argument. Pokud je nastaven na *True*, vrátí se jenom objekt *bestCostsPerRun* s výsledky.
- *normalised* – je to nepovinný argument. Pokud je nastaven na *True*, proběhne parsování i vnitřní struktury *bestCostsPerRun*.
- *display* – je to nepovinný argument – výsledek zobrazí na konzoli viz. obr. 16, 17, 18.

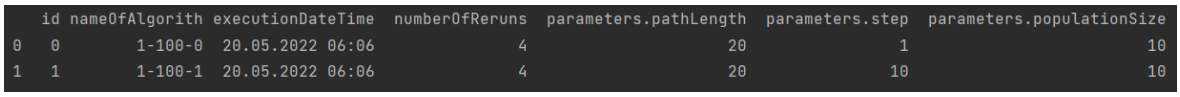
Parametr *normalised* využívá normalizační funkci z knihovny pandas, která upravuje složitou objektovou strukturu formátu JSON na „plochou“ tabulku. Výsledkem je tak přehledná

tabulka se správnou identifikací objektů, avšak dochází zde ke ztrátě informace JSON struktury, která je vhodná pro výběr dat. Z tohoto důvodu je normalizace uvedena jako volitelný parametr, aby bylo možné pracovat s DataFrame objektem i jako s objektem JSON a zachovala se robustnost rozhraní. Pro zobrazení dat v tabulce je normalizace vhodná a pro výběr dat je zase výhodnější normalizaci vynechat.



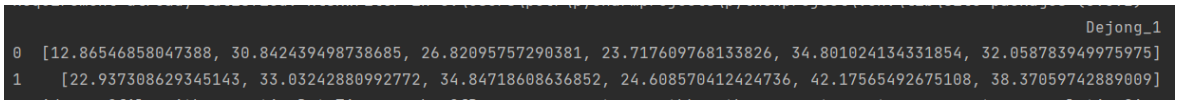
	id	nameOfAlgorith	executionDateTime	parameters
0	0	1-100-0	20.05.2022 06:06	{'pathLength': 20, 'step': 1, 'populationSize': 10, 'prt': 5}
1	1	1-100-1	20.05.2022 06:06	{'pathLength': 20, 'step': 10, 'populationSize': 10, 'prt': 5}

Obrázek 16: Výstup get_all_evaluations_formatted() pro normalised=False



	id	nameOfAlgorith	executionDateTime	numberOfReruns	parameters.pathLength	parameters.step	parameters.populationSize
0	0	1-100-0	20.05.2022 06:06	4	20	1	10
1	1	1-100-1	20.05.2022 06:06	4	20	10	10

Obrázek 17: Výstup *get_all_evaluations_formatted() pro normalised=True*



	Dejong_1
0	[12.86546858047388, 30.842439498738685, 26.82095757290381, 23.717609768133826, 34.801024134331854, 32.058783949975975]
1	[22.937308629345143, 33.03242880992772, 34.84718608636852, 24.608570412424736, 42.17565492675108, 38.37059742889009]

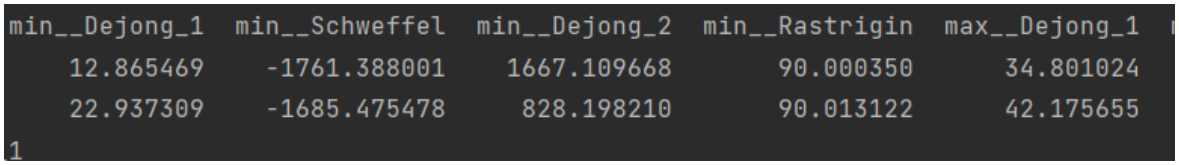
Obrázek 18: Výstup pro returnBestCostsPerRun=False

5.3.7 Funkce get_all_evaluations_with_statistics(fileName, dimension, fez, display=False)

Popis: Funkce vrací objekt DataFrame v kontextu JSON struktury *evaluations*. Součástí je také tabulka se statistickým vyhodnocením jednotlivých algoritmů.

Argumenty funkce:

- *fileName* – název souboru, odkud se má vrátit *evaluations*.
- *dimension* – dimenze, ze které se má *evaluations* získat
- *fez* – fez, ze které se má *evaluations* získat
- *display* – je to nepovinný argument – výsledek zobrazí na konzoli viz. obr. 19.



	min__Dejong_1	min__Schweffel	min__Dejong_2	min__Rastrigin	max__Dejong_1
	12.865469	-1761.388001	1667.109668	90.000350	34.801024
1	22.937309	-1685.475478	828.198210	90.013122	42.175655

Obrázek 19: Výstup get_all_evaluations_with_statistics()

5.3.8 Funkce print_all_data(fileName)

Popis: Funkce nic nevrací. Dochází pouze k výpisu celého JSON souboru na konzoli viz obrázek 20.

Argumenty funkce:

- *fileName* – je název souboru, který se má parsovat a vypsat na konzoli

```
######################
Dimension:  1

------------------
    Fez:  100
   id nameOfAlgorith executionDateTime
0   0        1-100-0  20.05.2022 06:06   {'pathLength': 20, 'step': 1, 'populationSize':
1   1        1-100-1  20.05.2022 06:06  {'pathLength': 20, 'step': 10, 'populationSize':
------------------
    Fez:  200
   id nameOfAlgorith executionDateTime
0   2        1-200-2  20.05.2022 06:07  {'pathLength': 20, 'step': 10, 'populationSize':
------------------
```

Obrázek 20: Výstup print_all_data()

5.4 Main

Funkce main je hlavní pracovní prostředí pro práci s rozhraním frameworku. Je koncipováno tak, že datový analytik bude právě zde využívat funkce frameworku a pracovat s datovým formátem DataFrame. Main je prostředník mezi rozhraním, zpracováním a testováním.

Tato hlavní funkce obsahuje všechny importy knihoven, které vznikly za účelem tohoto frameworku. Na obrázku 21 je ukázka použití readlib funkcí. Na existující JSON soubor stačí zavolat funkci dle dokumentace, nebo testů a spustí se s nastavením podle argumentů.

```
import numpy as np
from alg.soma import soma
from export.toJSON import create_json
import readlib.fromJson as rd
import export.toXls as xls
import pathlib
import os


#Using functions
rd.get_dimensions("json_test1.json", True)
rd.get_all_fez_in_dimension("json_test1.json", 10, True)
rd.get_all_evaluations_object("json_test1.json", 10, 3300, True)
```

Obrázek 21: Použití funkcí

Použití rozhraní je ukázáno na obrázku 22 a 23.

```
#Usage of interface
fez = 3000
dimension = 20
reruns = 10

costFncsDict = {
    1: 'Dejong_1',
    2: 'Schweffel',
    3: 'Dejong_2',
    4: 'Rastrigin'
}

paramsDict = {
    'pathLength': 3,
    'step': 0.33,
    'populationSize': 50,
    'prt': 0.3
}
```

Obrázek 22: Inicializace objektů

```
resultsPerFnc1 = {}

for costFncId, costFncName in costFncsDict.items():
    result = soma(dimension, costFncId, reruns, fez, paramsDict)
    resultsPerFnc1[costFncName] = result

create_json("json_test1.json", dimension, fez, resultsPerFnc1, reruns, 'SOMA', paramsDict)
create_json("json_test1.json", dimension, fez, resultsPerFnc1, reruns, 'SOMA', paramsDict)
create_json("json_test1.json", dimension, fez, resultsPerFnc1, reruns, 'XYZ', paramsDict)

rd.get_dimensions("json_test1.json", True)
rd.get_all_evaluations_object("json_test1.json", dimension, fez, True)
rd.get_all_fez_in_dimension("json_test1.json", dimension, True)
```

Obrázek 23: Spouštění funkcí a ukládání do souboru

Uživatel nadefinuje dva objekty jako na obrázku 22. Prvním je seznam testovacích funkcí. Jedná se o seřazenou sadu, která je postupně spouštěna na testovaný algoritmus. Druhým je objekt s parametry, který vstupuje do funkce z readlib a do funkce algoritmu.

Aby framework mohl pracovat s libovolným algoritmem a jeho unikátní sadou parametrů, musí se tyto parametry předávat jako objekt dictionary z obrázku 22. Tímto krokem se řeší problematika parametrů, kdy každý algoritmus má různý počet jinak pojmenovaných argumentů. V prvním případě je možné upravit testované algoritmy, aby podporovaly tento způsob předávání argumentů. Ve druhém případě je možné objekt vytvořit pouze pro účely rozhraní. Implementací více algoritmů a předání parametrů se práce nezabývá, ale jsou frameworkem podporovány.

Na obrázku 23 je patrné, jak práce s rozhraním probíhá. Cyklus iteruje přes objekt testovacích funkcí a postupně na každou funkci používá zvolený algoritmus. Výsledky každé testovací funkce se ukládají do objektu dictionary způsobem *klíč* = název funkce: *hodnota* = pole nejlepších výsledků. Následně stačí zavolat funkci pro uložení do souboru a používat knihovnu readlib pro získání požadovaných dat ve formátu DataFrame.

Výsledný framework samostatně nepodporuje přímé porovnání více algoritmů pomocí ranku, nebo váhového ohodnocení. Lze jej ale rozšířit pomocí knihovny SciPy, která umožňuje spouštět statistické testy na objekty DataFrame, které framework vytváří, a pomocí pandas s nimi manipulovat. [13]

6 ZABEZPEČENÍ

Jelikož výsledný framework nekomunikuje přes webové rozhraní, ani jinak nepoužívá webové služby a data jsou ukládána lokálně, šifrování a zabezpečení proti webovým hrozbám nebylo implementováno.

Zabezpečení ve smyslu ochrany dat před souběhem a uváznutím při práci se souborem JSON bylo zavedeno nativní formou jazyka Python.

6.1 Kontext

Python obsahuje zabezpečení proti chybám, kterému se říká kontextový manažer, který spravuje zdroje (anglicky context manager). Context manager pomáhá ovládat výjimky v aplikaci a předchází souběhu tím, že po přiřazení kontextu (přístup do souboru) daný zdroj uzamkne. Pokud si některý jiný proces vyžádá přístup k souboru, je přístup odepřen. Kontext se vytváří pomocí slova *with*.

```
with open('file_path', 'w') as file:
    file.write('hello world !')
```

Po dokončení práce se souborem kontext zaniká a automaticky zavírá daný zdroj, což zabezpečuje soubor proti ztrátě dat. Zároveň tak kontext chrání paměť automatickým uvolňováním zdrojů.

6.2 Správa výjimek

Dalším způsobem ošetření proti chybám je klauzule *try* a *except*. Pokud dojde při provádění kódu v části označenou *try* k výjimce vlivem chybného vstupu, nebo některá část kódu selže, vrátí se výjimka popisující chybu a příkaz se neprovede.

V kódu frameworku je zahrnuta vestavěná třída pro správu výjimek *OSError*, která poskytuje informace o chybě. Ošetřeny jsou takto i stavy, kdy uživatel zadá argument (například id, nebo dimenzi), který ještě není v souboru JSON.

6.3 Další zabezpečení

Při vytváření frameworku byla použita verze Pythonu 3.9, která má bezpečnostní podporu do roku 2025 a tedy je aktuální a obsahuje všechny bezpečnostní záplaty. Dále pak je využita absolutní cesta k importovaným modulů, jelikož se jedná o přímou a bezpečnou variantu používání knihoven.

7 TESTOVACÍ SCÉNÁŘE

Funkcionalita rozhraní frameworku byla testována manuálně. Nejprve byly provedeny unit testy pro každou vytvořenou funkci s různou sadou argumentů a možností. Poté byly provedeny komplexnější testové scénáře užití na daný framework pro otestování celkové funkčnosti.

Všechny provedené testy (unit testy i uživatelské scénáře) jsou zahrnuty ve zdrojovém kódu *main.py*, což umožňuje jejich opětovné spuštění. Pro spuštění zvoleného testu stačí odstranit komentářový blok a spustit funkci main. V následujících podkapitolách jsou uvedeny vybrané případy scénářů.

7.1 Test navrácení struktury

Cíl testu: Otestování základních *get* funkcí, které vrací požadovanou strukturu ve formátu DateFrame, pokud je v JSON souboru nalezena.

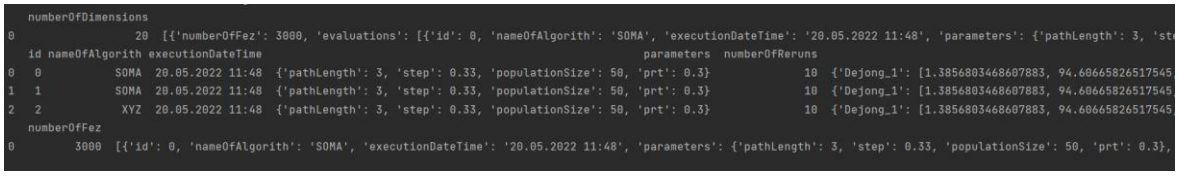
Popis scénáře: Uživatel vytvoří objekt s parametry a spustí vlastní testovaný algoritmus na sadu testovacích funkcích. Následně zavolá funkci pro uložení do souboru. Poté si chce prohlédnout všechny dimenze v souboru. Následně si jednu vybere a zobrazí všechny hodnoty FEZ dané dimenze. Pak zavolá funkci pro zobrazení všech evaluací pro zvolenou dimenzi a FEZ.

Očekávaný výsledek: Tři rozdílné tabulky v kontextu dimenze, FEZ a evaluace jsou zobrazeny. Každá funkce vrátí DataFrame objekt.

Provedení: Na obr. 24 jsou počáteční nastavení před spuštěním testu. Na obr. 25 jsou patrné výsledky všech tří *get* funkcí.

```
resultsPerFnc1 = {}

for costFncId, costFncName in costFncsDict.items():
    result = soma(dimension, costFncId, reruns, 3000, paramsDict)
    resultsPerFnc1[costFncName] = result

create_json("json_test1.json", dimension, 3000, resultsPerFnc1, reruns, 'SOMA', paramsDict)
create_json("json_test1.json", dimension, 3000, resultsPerFnc1, reruns, 'SOMA', paramsDict)
create_json("json_test1.json", dimension, 3000, resultsPerFnc1, reruns, 'XYZ', paramsDict)
rd.get_dimensions("json_test1.json", True)
rd.get_all_evaluations_object("json_test1.json", dimension, 3000, True)
rd.get_all_fez_in_dimension("json_test1.json", dimension, True)
```

Obrázek 24: Nastavení testu – navrácení struktury



Obrázek 25: Výsledek testu – navrácení struktury

Výsledek: Test proběhl úspěšně.

7.2 Test JSON souboru

Cíl testu: Otestovat všechny způsoby přidání do souboru:

1. Soubor neexistuje – vytvoření
2. Dimenze neexistuje – vytvoření objektu dimenze v JSON souboru
3. FEZ neexistuje – vytvoření objektu FEZ v objektu dimenze
4. Daná evaluace je přidána do souboru přímo

Popis scénáře: Uživatel ukládá více dat, přičemž některé vnořené objekty zatím nejsou v souboru JSON uloženy. Provádí se vytvoření příslušných objektů.

Očekávaný výsledek: Všechny způsoby vložení fungují a uživatel je informován o způsobu vložení na konzoli. Vytvořený soubor obsahuje všechna data, která uživatel ukládal.

Provedení: Na obr. 26 jsou počáteční nastavení před spuštěním testu. Na obr. 27 je porovnání výsledné a očekávané struktury JSON.

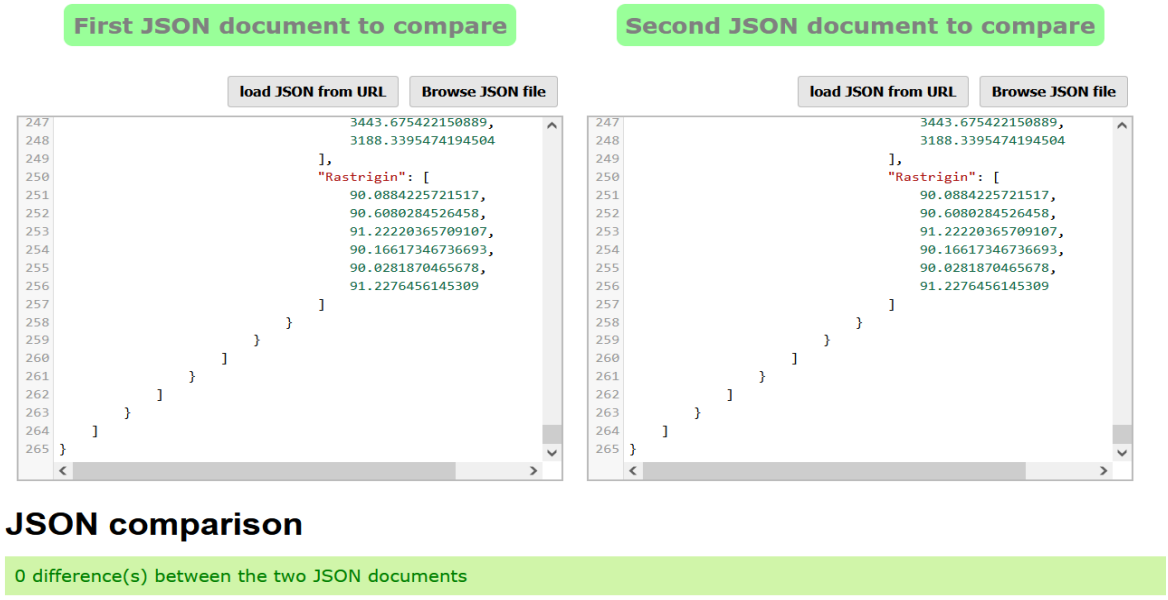
```
for costFncId, costFncName in costFncsDict.items():
    result = soma(dimension, costFncId, reruns, 1000, paramsDict)
    resultsPerFnc[costFncName] = result

create_json('json_data.json', 1, 100, resultsPerFnc, 4, '1-100-0', paramsDict)
create_json('json_data.json', 1, 100, resultsPerFnc, 4, '1-100-1', paramsDict)
create_json('json_data.json', 1, 200, resultsPerFnc, 4, '1-200-2', paramsDict)
create_json('json_data.json', 1, 300, resultsPerFnc, 4, '1-300-3', paramsDict)
create_json('json_data.json', 2, 100, resultsPerFnc, 4, '2-100-5', paramsDict)
```

Obrázek 26: Nastavení testu – JSON soubor

Výstup z konzole:

```
File json_test5.json has been created, evaluation id = 0 was initiated.
Found DIM: 1 -> FEZ: 100... Adding new evaluation data now
Found DIM: 1 but no FEZ: 200... Creating new FEZ for selected DIM + add evaluation data
Found DIM: 1 but no FEZ: 300... Creating new FEZ for selected DIM + add evaluation data
```

```
DIM: 2 not found... Creating new DIM with FEZ: 100 + add evaluation data
Process finished with exit code 0
```



Obrázek 27: Výsledek testu – JSON soubor

Výsledek: Test proběhl úspěšně.

7.3 Test dvou algoritmů

Cíl testu: Otestovat běh dvou algoritmů a ověřit správnost uložených dat. Dále se testuje vyhodnocení statistických charakteristik.

Popis scénáře: Uživatel spouští dva algoritmy s různými parametry. Vytvoří se soubor a správně se do něj uloží data obou algoritmů. Následně se vypíše statistická charakteristika obou algoritmů.

Očekávaný výsledek: Oba algoritmy jsou korektně uloženy se správnými parametry. Zobrazí se tabulka výsledků a tabulka statistiky.

Provedení: Na obr. 28 a 29 jsou počáteční nastavení před spuštěním testu. Na obr. 30 jsou výsledné tabulky.

```
paramsDict = {
    'pathLength': 3,
    'step': 0.33,
    'populationSize': 50,
    'prt': 0.3
}
paramsDict2 = {
    'pathLength': 2,
    'step': 0.33,
    'populationSize': 50,
    'prt': 0.2
}
```

Obrázek 28: Nastavení testu – parametry

```
resultsPerFnc1 = {}
resultsPerFnc2 = {}

for costFncId, costFncName in costFncsDict.items():
    result = soma(dimension, costFncId, reruns, 10000, paramsDict)
    result2 = soma(dimension, costFncId, reruns, 10000, paramsDict2)
    resultsPerFnc1[costFncName] = result
    resultsPerFnc2[costFncName] = result2

create_json("json_test2.json", dimension, 10000, resultsPerFnc1, reruns, 'SOMA-1', paramsDict)
create_json("json_test2.json", dimension, 10000, resultsPerFnc2, reruns, 'SOMA-2', paramsDict2)
rd.get_all_evaluations_with_statistics("json_test2.json", dimension, 10000, True)
#os.remove("json_test2.json")
```

Obrázek 29: Nastavení testu – dva algoritmy

```
File json_test2.json has been created, evaluation id = 0 was initiated.
Found DIM: 10 -> FEZ: 10000... Adding new evaluation data now
   id nameOfAlgorith executionDateTime                                                parameters  numberOfReruns  min__Dejong_1
0   0         SOMA-1  23.05.2022 22:19  {'pathLength': 3, 'step': 0.33, 'populationSize': 50, 'prt': 0.3}              6   1.991369e-09
1   1         SOMA-2  23.05.2022 22:19  {'pathLength': 2, 'step': 0.33, 'populationSize': 50, 'prt': 0.2}              6   1.112482e-08
                                                                                                                 Dejong_1
0  [1.9913693422962843e-09, 52.94887172136391, 42.32258167337125, 17.632272919929814, 17.623971235505127, 34.38908629045577]    [-3617.37
1  [1.1124821320204885e-08, 30.790899265921567, 37.5502253735923, 30.240137170953506, 23.966010192408085, 41.00439079995441]  [-3360.7275
```

Obrázek 30: Výsledek testu – dva algoritmy

Výsledek: Test proběhl úspěšně.

ZÁVĚR

Tato práce se zabývá problematikou optimalizace z pohledu evolučních algoritmů a jejich testování na testovacích sadách, které představují různé problémy, se kterými si algoritmy musí umět poradit.

V teoretické části diplomové práce byla popsána optimalizace, evoluční algoritmy a jejich princip. Také byla rozebrána komplexní problematika testování a ohodnocování evolučních algoritmů. Byly nastíněny způsoby vyhodnocení testů a rozdílné přístupy k jejich ukončovacím podmínkám.

V praktické části je popsána tvorba rozhraní robustního frameworku, který není omezen na určité optimalizační funkce, ani na předem definovaný počet iterací či dimenzí. Výsledný framework poskytuje rozhraní mezi prostředím jazyka Python, knihovnou pro datovou analýzu Pandas a daty, uloženými v souboru formátu JSON. Součástí práce je také dokumentace a vybrané testovací scénáře, které jsou k dispozici na odzkoušení ve funkci main.

Při navrhování datové struktury jsem vycházel z předpokladu, že si uživatel bude moci vybrat jaká data bude porovnávat. Framework tak nelimituje výběr a je možné ukládat všechna data v jednou souboru pro pozdější analýzu. Problémem se ale ukázalo příliš obecné řešení povahy dat. Zjistil jsem, že bez uživatelského zásahu, který specifikuje striktnější kritéria pro porovnávaná data, je velmi obtížné zachovat tento robustní přístup pro každý případný algoritmus nad veškerou množinou dat. Výsledný framework tak přímo nepracuje se statistickými testy typu Friedman a post-hoc, ale díky použitým knihovnám a zvolenému formátu DataFrame pro analýzu dat jde vhodně rozšířit například o knihovnu SciPy, která tento formát umí zpracovat do statistických testů.

SEZNAM POUŽITÉ LITERATURY

[1] V. PRICE, Kenneth. *Differential Evolution: A Practical Approach to Global Optimization. Springer*, 2005. ISBN: 9783540209508.

[2] ZELINKA, Ivan. *Evoluční výpočetní techniky – principy a aplikace*. Praha: BEN – technická literatura, 2008. ISBN 9788073002183.

[3] KACPRZYK, Janusz. *Springer Handbook of Computational Intelligence. Dordrecht: Springer*, 2015. ISBN 9783662435045.

[4] ZELINKA, Ivan. *Handbook of Optimization From Classical to Modern Approach*. Berlin: Springer, 2013. ISBN 9783642305030.

[5] BARTZ-BEIELSTEIN, Thomas. *Benchmarking in Optimalization: Best Practice and Open Issues* [online]. Kolín nad Rýnem. Německo, 2020 [cit. 2022-05-23]. Dostupné z: https://d-nb.info/1214641016/34

[6] KAZIKOVA, Anezka, Michal PLUHACEK a Roman SENKERIK. *How Does the Number of Objective Function Evaluations Impact Our Understanding of Metaheuristics Behavior?*. IEEE Access [online]. 2021 [cit. 2022-05-23]. Dostupné z: doi:https://doi.org/10.1109/ACCESS.2021.3066135

[7] MOLINA, Daniel. *Analysis among winners of different IEEE CEC competitions on real-parameters optimization: Is there always improvement?*. IEEE Access [online]. 2017 [cit. 2022-05-23]. Dostupné z: doi:10.1109/CEC.2017.7969392

[8] LUNA, Javier. Top programming languages for data scientists in 2022. Datacamp [online]. 2022 [cit. 2022-05-23]. Dostupné z: https://www.datacamp.com/blog/top-programming-languages-for-data-scientists-in-2022

[9] HARRIS, Charles R. *Array programming with NumPy*. Nature [online]. 2020 [cit. 2022-05-23]. Dostupné z: doi:https://doi.org/10.1038/s41586-020-2649-2

[10] *Friedman test using Python (with examples and code)*. Reneshbedre [online]. 2022 [cit. 2022-05-23]. Dostupné z: https://www.reneshbedre.com/blog/friedman-test-python.html

[11] *Analýza dat v Pythonu*. Nauč se Python! [online]. 2017 [cit. 2022-05-23]. Dostupné z: https://naucse.python.cz/lessons/intro/pandas/

[12] ŽÁRA, Ondřej. *JavaScript: programátorské techniky a webové technologie*. Brno: Computer Press, 2015. ISBN 9788025145739.

[13] *Statistics in Python*. Scipy-lectures [online]. 2022 [cit. 2022-05-23]. Dostupné z: https://scipy-lectures.org/packages/statistics/index.html#the-pandas-data-frame

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

CEC Congress on Evolutionary Computation

FEZ Maximální počet ohodnocení účelové funkce

IEEE Institute of Electrical and Electronics Engineer

JSON JavaScript Object Notation

SOMA Samoorganizující se migrační algoritmus

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

CD

PŘÍLOHA P I: CD

Obsah CD:

- Diplomová práce v elektronické podobě
- Zdrojové kódy