Analýza citlivosti implementace testovacích funkcí CEC benchmark setu

Jiří Opravil

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jiří Opravil**
Osobní číslo:	**A21706**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Analýza citlivosti implementace testovacích funkcí CEC benchmark setu**
Téma práce anglicky:	**Sensitivity Analysis of the CEC Benchmark Set Implementation**

Zásady pro vypracování

1. Seznamte se s problematikou CEC benchmarku pro jednoúčelovou optimalizaci s omezením hranic a popište ji.
2. Vytvořte přehled veřejně dostupných implementací CEC benchmarku.
3. Pro jednotlivé implementace otestujte jejich citlivost na přesnost vstupních parametrů.
4. Zhodnoťte a porovnejte dosažené výsledky.
5. Okomentujte možné implikace výsledků a vytvořte doporučení pro používání CEC benchmarku.

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

Seznam doporučené literatury:

1. GITHUB P-N-SUGANTHAN. *2022-SO-BO*. Online. 2022.
2. GITHUB P-N-SUGANTHAN. *2021-SO-BCO*. Online. 2021.
3. GITHUB P-N-SUGANTHAN. *2020-Bound-Constrained-Opt-Benchmark*. Online. 2020.
4. GITHUB P-N-SUGANTHAN. *CEC2019*. Online. 2019.
5. GITHUB P-N-SUGANTHAN. *CEC2017-BounContrained*. Online. 2017.

Vedoucí bakalářské práce: **Ing. Adam Viktorin, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

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

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

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

Ve Zlíně dne 5. ledna 2024

Jméno, příjmení: Jiří Opravil

Název bakalářské práce: Analýza citlivosti implementace testovacích funkcí CEC benchmark setu

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 bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- 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 – 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 13.5.2024

Jiří Opravil, v.r.
podpis studenta

ABSTRAKT

Bakalářská práce se zabývá tématem analýzy citlivosti implementace testovacích funkcí CEC benchmark setu. Práce je rozdělena na dvě části, a to teoretickou a praktickou. V teoretické části jsou v první kapitole představeny základní pojmy, jako CEC benchmark, jeho vznik a jednoúčelová optimalizace s omezením. Ve druhé kapitole je přehled veřejně dostupných implementací CEC benchmarku a jsou zde představeny jednotlivé ročníky. Praktická část je zaměřena na testování CEC benchmark setu, a to v ročnících CEC 2022 a CEC 2017. Cílem práce je zjistit, jak moc se může pozměněná (vypočítaná) hodnota lišit od pozice optima v každé dimenzi, tak aby byl výstup z funkce stále optimem nebo byl v přípustné odchylce. Současně bylo nutné otestovat všechny implementace a vzájemně je mezi sebou porovnat. Na základě testování a vyhodnocení výsledků je zřejmé, že se přesnosti pro různé implementační jazyky odlišují.

Klíčová slova: CEC benchmark, jednoúčelová optimalizace, optimalizace s omezením hranic, benchmarková funkce

ABSTRACT

The bachelor's thesis addresses the topic of sensitivity analysis of the CEC benchmark set implementation. The thesis is divided into two parts: theoretical and practical. In the theoretical part, the first chapter introduces basic concepts such as the CEC benchmark, its origin, and single objective bound constrained optimization. The second chapter provides an overview of publicly available implementations of the CEC benchmark and presents individual editions for every year. The practical part focuses on testing the CEC benchmark set, specifically in the CEC 2022 and CEC 2017 editions. The goal of the thesis is to determine how much the modified (calculated) value can differ from the optimum position in each dimension, such that the function output remains optimal or within an acceptable deviation. It was also necessary to test all implementations and compare them with each other. Based on testing and evaluation of the results, it is evident that the accuracies differ across various implementation languages.

Keywords: CEC benchmark, single objective optimization, bound constrained optimization, benchmark function

Velice děkuji panu Ing. Adamu Viktorinovi, Ph.D., vedoucímu práce, za podnětné a cenné rady, vstřícný přístup a ochotu odpovídat na moje otázky v kteroukoli denní nebo noční hodinu.

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

OBSAH

ÚVOD

Věda se posouvá stále dopředu, každý rok se objevuje množství nových objevů, vynálezů nebo programů, které mají usnadňovat práci a život. Vývoj se posouvá i v oblasti evolučních výpočtů a optimalizace. V tomto případě je potřeba se zaměřit na vývoj efektivních algoritmů, které jsou schopny řešit optimalizační problémy s omezeními. Tato problematika je mezi vědci oblíbená, každý rok se pořádá CEC kongres, který je plný přednášek, workshopů i ukázek nových postupů. Pravidelně se koná také soutěž, na které jsou testovány jednotlivé algoritmy a funkce. Těchto dat bylo využito v bakalářské práci.

Práce je rozdělena na dvě části, první z nich je část teoretická. Nejprve je nutné se seznámit se základními pojmy, proto je první kapitola zaměřena na problematiku CEC benchmarku pro jednoúčelovou optimalizaci s omezením hranic. V jednotlivých podkapitolách je nejdříve vysvětlen CEC benchmark, následně vznik CEC benchmarku, včetně prestižního CEC kongresu, a nakonec jednoúčelová optimalizace s omezením hranic.

Druhá kapitola je rozsáhlá a věnuje se veřejně dostupným implementacím CEC benchmarku. Obsáhlost kapitoly je dána tím, že jsou zde v jednotlivých podkapitolách představeny jednotlivé ročníky včetně souhrnu testovacích funkcí CEC benchmark setu z daného roku, který je uveden v tabulce. Dále jsou zde uvedeny ukázky kódu testovacích funkcí a v závěru druhé kapitoly najdeme podkapitolu věnující se jednotlivým typům funkcí – unimodální, multimodální, hybridní a kompozitní.

Druhá část bakalářské práce je část praktická, která je zaměřena na testování CEC benchmark setu. První podkapitola se věnuje testování CEC benchmark setu 2022. Odchylky testovacích funkcí jsou uvedeny v grafech, které jsou doplněny slovním komentářem. Dále jsou uvedeny tabulky, ve kterých jsou srovnány vyžadované přesnosti v jednotlivých jazycích a další tabulka představuje výpis vypočítaných odchylek hybridních funkcí.

Druhá podkapitola se zaměřuje na testování funkcí CEC benchmark setu 2017. I zde jsou výsledky odchylek testovacích funkcí uvedeny v grafu a slovně okomentovány. Kromě grafů v této části najdeme tabulky, které srovnávají přesnost jazyka C a MATLAB.

Cílem bakalářské práce je zjistit, jak moc se může pozměněná (vypočítaná) hodnota lišit od pozice optima v každé dimenzi, tak aby byl výstup z funkce stále optimem nebo byl v přípustné odchylce. Současně bylo nutné otestovat všechny implementace a vzájemně je

mezi sebou porovnat. Shrnutí odlišností jednotlivých implementačních jazyků je uvedeno za jednotlivými podkapitolami, na závěr praktické části je celkové shrnutí testování.

I. TEORETICKÁ ČÁST

1 PROBLEMATIKA CEC BENCHMARKU PRO JEDNOÚČELOVOU OPTIMALIZACI S OMEZENÍM HRANIC

Problematika spojená s CEC benchmarkem pro jednoúčelovou optimalizaci s omezením hranic představuje klíčový výzkumný směr v oblasti evolučních výpočtů a optimalizace. V tomto kontextu je hlavním cílem vývoj efektivních algoritmů, které jsou schopny řešit optimalizační problémy s omezeními. Tato problematika se zabývá výběrem vhodných testovacích funkcí, definicí omezení a vhodnými metrikami pro hodnocení výkonu algoritmů. Klíčové je zajistit, aby tyto metriky byly citlivé na vlastnosti problému a aby algoritmy měly schopnost dodržet stanovená omezení. Studium této problematiky má potenciál přinést nové poznatky a metody, které mohou významně přispět k rozvoji efektivnějších algoritmů určených k řešení praktických optimalizačních úloh, které mohou mít využití v různých aplikacích. [12; 16]

1.1 CEC benchmark

CEC (Congress on Evolutionary Computation) benchmark je standardizovaná sada optimalizačních testovacích funkcí a úloh, která je používána k hodnocení výkonu optimalizačních algoritmů. Tato sada testovacích funkcí je navržena tak, aby poskytovala objektivní systém pro porovnání různých optimalizačních algoritmů v různých optimalizačních scénářích.

[12; 16]

CEC benchmark se obvykle skládá z různorodých optimalizačních problémů (testovacích funkcí), které mají různé charakteristiky a obtížnost. Tyto testovací funkce mohou být unimodální, multimodální, hybridní nebo kompozitní a jejich cílem je simulovat různé reálné situace, se kterými se mohou algoritmy setkat v praxi. Cílem CEC benchmarku je poskytnout referenční zdroj pro srovnání a hodnocení různých algoritmů v různých podmínkách a umožnit další postup na poli vědeckého výzkumu v oblasti optimalizace. [1; 3; 9; 12]

1.2 Vznik CEC benchmarku

Začátky CEC benchmarku sahají do doby, kdy byly optimalizační algoritmy testovány na různých problémech, což často vedlo k nedostatečně spolehlivým a neobjektivním výsledkům. CEC benchmark byl vytvořen s cílem vyřešit tuto výzvu tím, že poskytuje

standardizovaný soubor testovacích funkcí, které pokrývají širokou škálu optimalizačních scénářů a obtížností. [12]

Díky kolektivní práci vědců a výzkumníků z celého světa se CEC benchmark stal uznávaným nástrojem v oblasti optimalizace, který je používán pro objektivní srovnání výkonu různých optimalizačních algoritmů. [4; 6]

1.2.1 CEC kongres

Rozhodující roli v rozvoji evolučního výpočtu a komputační inteligence hraje každoroční kongres CEC. Tento prestižní kongres je organizován IEEE Computational Intelligence Society a slouží jako klíčová událost pro prezentaci nejnovějších výzkumných výsledků a diskusi o aktuálních tématech v oboru. [15]

CEC představuje vyhledávanou platformu pro výzkumníky a odborníky z celého světa, kteří se zabývají evolučním výpočtem a souvisejícími oblastmi, jako je genetické programování, genetické algoritmy, evoluční strategie a další. Každý rok se kongres koná v jiném městě a přináší bohatý program, který zahrnuje přednášky, workshopy a další akce. [17]

1.3 Jednoúčelová optimalizace s omezením hranic

V rámci jednoúčelové optimalizace je snaha identifikovat hodnotu proměnné nebo sady proměnných, jež minimalizuje nebo maximalizuje dané kritérium. Toto kritérium může představovat náklady, časový úsek, výnos nebo jiný relevantní faktor v kontextu zkoumaného problému. Při provádění jednoúčelové optimalizace jsou dodrženy určitá omezení nebo podmínky, jež mohou být vyjádřeny formou rovnic, nerovnic nebo jiných matematických výrazů. [14]

Algoritmy jednoúčelové optimalizace jsou základem, na kterém jsou postaveny složitější metody, jako jsou víceúčelová optimalizace, optimalizační algoritmy s omezením, niching a další. Průběžná vylepšení jednoúčelových optimalizačních algoritmů jsou důležitá, protože mohou ovlivnit i jiné oblasti. Tato algoritmická vylepšení zčásti závisí na zpětné vazbě z experimentů provedených s benchmarkovými funkcemi jednoúčelové optimalizace, které jsou samy o sobě základními stavebními bloky pro složitější úkoly, mezi které patří problémy dynamické a výpočetně náročné nebo niching. Jak se algoritmy zdokonalují, je nutné vyvíjet stále náročnější funkce. Interakce mezi algoritmy a testovacími funkcemi zrychluje pokrok, a proto byly vyvinuty speciální soutěže na optimalizaci reálných parametrů, aby byla podpořena tato vzájemná spolupráce. [12; 13; 16]

2 PŘEHLED VEŘEJNĚ DOSTUPNÝCH IMPLEMENTACÍ CEC BENCHMARKU

Implementace jsou často k dispozici v otevřených repozitářích, jako je GitHub a nabízejí širokou škálu variant v různých programovacích jazycích včetně C/C++, Pythonu nebo Javy. Veřejně dostupné implementace podporují sdílení výsledků z CEC benchmarku, což přispívá k dalšímu rozvoji a inovacím nejen v této oblasti.

2.1 Přehled jednotlivých ročníků

Metody optimalizace a optimalizační problémy někdy vyžadují aktualizaci tradičních testovacích kritérií. V posledních letech bylo navrženo mnoho nových optimalizačních algoritmů k řešení jednoúčelových optimalizačních problémů s omezeními nabízených v CEC'05, CEC'13, CEC'14, CEC'17, CEC'20, CEC'21 CEC'22 speciálních soutěžích na optimalizaci reálných parametrů. Vždy se s ohledem na komentáře k sestavě testů z předešlých let zorganizovala nová soutěž pro jednoúčelovou optimalizaci reálných parametrů. [9]

Pro tyto soutěže byly vyvinuty testovací funkce s několika novými vlastnostmi, jako jsou nové základní testovací funkce, skládání testovacích funkcí extrakcí vlastností podle dimenzí z několika různých testovacích funkcí, stupňovaná míra vzájemných vazeb, rotované funkce s pastmi a podobně. [11]

Tyto speciální soutěže byly věnovány přístupům, algoritmům a technikám pro řešení jednoúčelové optimalizace s omezením hranic s reálnými parametry bez použití přesných rovnic testovacích funkcí, náhradních a meta-modelů. [7]

2.1.1 Definice testování funkcí

Pokud není uvedeno jinak, tak obecně platí:

- rozsah vyhledávání je omezen na interval $[-100; 100]^D$ pro všechny testovací funkce, zároveň uvnitř tohoto intervalu leží i posunuté globální optimum $[-80; 80]^D$;
- odlišná rotační matice je použita pro každou funkci ať už pro základní, nebo složenou;
- vzhledem k tomu, že v problémech reálného světa jen zřídkakdy existují vazby mezi všemi proměnnými, CEC soutěže náhodně dělí proměnné na podskupiny. Rotační

matice pro každou skupinu podskupin je generována z položek s normálním rozdělením pomocí Gram-Schmidtovy orto-normalizace s podmíněným číslem c, které je rovno 1 nebo 2;

- všechny testovací funkce jsou minimalizačními úlohami, dopředu jsou posunuty do optima a jsou škálovatelné;
- testování končí v okamžiku, kdy se hodnota vypočítané funkce liší od původní (ideální) hodnoty o méně než 10^{-8}, nebo pokud algoritmus dosáhne na hranici maximálního počtu vyhodnocení *(MaxFES),* vždy je uvedeno v tabulce;
- Fi^* udává ideální hodnotu (globální optimum), do které je funkce posunuta.

2.1.2 Rok 2022

CEC benchmark v roce 2022 obsahoval 12 testovacích funkcí. Tyto testovací funkce byly implementovány v jazyce C, Python a MATLAB. Bylo možné použít i jazyk C++, ale ten nebyl součástí oficiální dokumentace.

Tabulka 1 D a MaxFES pro rok 2022 [9]

Počet dimenzí (D=*)	MaxFES
10	200 000
20	1 000 000

Souhrn 12 testovacích funkcí CEC benchmarku z roku 2022

Tabulka 2 Souhrn testovacích funkcí CEC 2022 [10]

Typ funkce	**Číslo funkce**	**Název funkce**	F_i^*
Unimodální	1	Shifted and full Rotated Zakharov Function	300
Základní	2	Shifted and full Rotated Rosenbrock's Function	400
	3	Shifted and full Rotated Expanded Schaffer's F6 Function	600
	4	Shifted and full Rotated Non-Continuous Rastrigin's Function	800
	5	Shifted and full Rotated Levy Function	900
Hybridní	6	Hybrid Function 1 (N = 3)	1800

	7	Hybrid Function 2 (N = 6)	2000
	8	Hybrid Function 3 (N = 5)	2200
Kompozitní	9	Composition Function 1 (N = 5)	2300
	10	Composition Function 2 (N = 4)	2400
	11	Composition Function 3 (N = 5)	2600
	12	Composition Function 4 (N = 6)	2700

2.1.3 Rok 2021

V tomto soutěžním roce jsou benchmarkové funkce parametrizovány zahrnutím operátorů, jako jsou zkreslení, rotace a translace. Hlavním motivem za parametrizací je otestovat účinek všech kombinací operátorů na všechny benchmarkové funkce. Parametrizované vyhodnocování benchmarku je krokem k získání vícestranného pohledu do výkonnosti algoritmů a optimalizačních problémů. Pro tento účel je navrženo 10 škálovatelných benchmarkových funkcí s těmito binárními operátory. [7]

Funkce jsou implementovány v jazyce C a MATLAB.

Tabulka 3D a MaxFES pro rok 2021 [7]

Počet dimenzí (D=*)	MaxFES
10	200 000
20	1 000 000

Během soutěže bylo zjištěno, že přítomnost nebo absence posunutí transformace byla dominující. Například mít rotaci bez posunutí bylo shledáno jako bezvýznamné, protože globální řešení bylo vždy v centru vyhledávacího prostoru. Proto operátory, které zkreslují vyhledávání směrem k centru, mohou všechny problémy bez posunutí poměrně snadno vyřešit. A tak byly některé scénáře bez posunutí odstraněny. Byl zahrnut pouze jeden scénář bez posunutí. Tyto testovací funkce (bez posunutí) by neměly být použity k hodnocení algoritmů, místo toho by měly být použity k prozkoumání vlivu posunutí na algoritmus. [8]

Souhrn 10 testovacích funkcí CEC benchmark z roku 2021

Tabulka 4 Souhrn testovacích funkcí CEC 2021 [7;8]

Typ funkce	Číslo funkce	Název funkce	F_i^*
Unimodální	1	Shifted and Rotated Bent Cigar Function	100
Základní	2	Shifted and Rotated Schwefel's Function	1100
	3	Shifted and Rotated Lunacek bi-Rastrigin Function	700
	4	Expanded Rosenbrock's plus Griewangk's Function	1900
Hybridní	5	Hybrid Function 1 (N = 3)	1700
	6	Hybrid Function 2 (N = 4)	1600
	7	Hybrid Function 3 (N = 5)	2100
Kompozitní	8	Composition Function 1 (N = 3)	2200
	9	Composition Function 2 (N = 4)	2400
	10	Composition Function 3 (N = 5)	2500

2.1.4 Rok 2020

V předchozích ročnících soutěže maximální povolený počet vyhodnocení funkce – na rozdíl od složitosti problému – se exponenciálně nezvyšoval s rozměrem. Aby se vyřešila tato nerovnost, tak se významně zvýšil maximální povolený počet vyhodnocení funkce pro 10 škálovatelných benchmarkových problémů nad jejich předchozí soutěžní limity s cílem zjistit, do jaké míry se toto kritérium promítne do zlepšené přesnosti řešení. [5]

Funkce jsou implementovány v jazyce C a MATLAB. Pro funkce F1-F5 a F8-F10 se využívá D=5, D=10, D=15 a D=20. Pro funkce F6 a F7 platí D=10, D=15 a D=20. [5]

Tabulka 5 D a MaxFES pro rok 2020 [5]

Počet dimenzí (D=*)	MaxFES
5	50 000
10	1 000 000
15	3 000 000
20	10 000 000

Souhrn 10 testovacích funkcí CEC benchmarku z roku 2020

Tabulka 6 Souhrn testovacích funkcí CEC 2020 [6]

Typ funkce	Číslo funkce	Název funkce	F_i^*
Unimodální	1	Shifted and Rotated Bent Cigar Function	100
Základní	2	Shifted and Rotated Schwefel's Function	1100
	3	Shifted and Rotated Lunacek bi-Rastrigin Function	700
	4	Expanded Rosenbrock's plus Griewangk's Function	1900
Hybridní	5	Hybrid Function 1 (N = 3)	1700
	6	Hybrid Function 2 (N = 4)	1600
	7	Hybrid Function 3 (N = 5)	2100
Kompozitní	8	Composition Function 1 (N = 3)	2200
	9	Composition Function 2 (N = 4)	2400
	10	Composition Function 3 (N = 5)	2500

2.1.5 Rok 2019

Pro lepší pochopení chování evolučních algoritmů použitých na jednoúčelovou optimalizaci s omezením hranic byla představena výzva *100-digit Challenge*. [3]

Originální výzva *100-digit Challenge SIAM* byla vyvinuta v roce 2002 Nickem Trefethenem z Oxfordu ve spolupráci se Společností pro průmyslovou a aplikovanou matematiku (SIAM) jako test pro výpočty s vysokou přesností. Cílem bylo vyřešit 10 obtížných problémů s desetimístnou přesností, přičemž za každou správnou číslici byl udělen jeden bod, s maximálním skóre 100. Soutěžící mohli použít jakoukoliv metodu a na řešení měli neomezeně času. [4]

Proto byla pro rok 2019 navržena výzva *100-digit Challenge* podobně jako v SIAM verzi s 10 funkcemi k optimalizaci a s cílem dosáhnout desetimístné přesnosti bez časového omezení. Na rozdíl od SIAM verze výzva *100-digit Challenge* vyžadovala použití jednoho

algoritmu pro všechny úlohy s omezeným laděním parametrů. Skóre bylo hodnoceno jako průměr správných číslic v nejlepších 25 z 50 pokusů.[3]

Speciální soutěže v roce 2019 s jednoúčelovou optimalizací s omezením hranic měly přísná omezení na maximální počet vyhodnocení funkce, protože čas je v mnoha reálných situacích velmi cenný. Často také zakazovaly ladění algoritmů, což v některých situacích vedlo k preferenci rychlé konvergence na úkor přesnosti. Výzva *100-digit Challenge* se snažila řešit tento aspekt numerické optimalizace tím, že umožnila flexibilnější přístup k vyhodnocení algoritmů. Hodnocení nezáviselo na čase ani na úsilí, jež byly věnovány ladění. [3]

Jak bylo zmíněno výše, nedávné soutěže měřily hodnotu objektivní funkce ve "vertikálních" časových řezech konvergenčního grafu (hodnota funkce vs. počet vyhodnocení funkce). Výzva *100-digit Challenge* doplnila stávající testovací prostředí měřením funkčních hodnot v "horizontálních" řezech konvergenčního grafu. V dřívějších soutěžích "horizontální řez" znamenal dosažení určité hodnoty (VTR) a průměrný počet vyhodnocení funkce k dosažení VTR byl hlavním statistickým ukazatelem výkonnosti. Tato soutěž však nabídla graduálnější měření "horizontální" výkonnosti (přesnosti), protože dokonce i "neúspěchy" mohly mít některé správné číslice. [4]

Funkce jsou implementovány v jazyce C a MATLAB. Pro korektní podmínky testování je potřeba nastavit *počet dimenzí (D)*.

Funkce F4-F10 jsou platné pro kterékoliv $D>1$, zatímco funkce F1-F3 jsou platné pro $D=2n$, n2 a 3n kde n=1, 2, ... Zároveň funkce F4-F10 jsou posunuté pomocí matice *o* a otočené rotační maticí M, kdežto funkce F1-F3 nejsou ani otočené ani posunuté. [3]

Rozsah vyhledávání a počet dimenzí jsou specifické, a proto jsou uvedeny v tabulce. Mezi hlavní vlastnosti testovacích funkcí patří multimodalita a neseperabilita. [3]

Testování končí, pokud algoritmus dosáhne přesnosti na 10 číslic (9 desetinných míst) např. 1.000000000. Maximální počet vyhodnocení *(MaxFES)* je pro tuto soutěž vynechán. [3]

Souhrn 10 testovacích funkcí ve výzvě *100-digit Challenge*

Tabulka 7 Souhrn testovacích funkcí v rámci 100-digit Challenge [4]

Číslo	**Název funkce**	**D (dimenze)**	**Rozsah vyhledávání**	F_i^*

1	Storn's Chebyshev Polynomial Fitting Problem	9	[-8192, 8192]	1
2	Inverse Hilbert Matrix Problem	16	[-16384, 16384]	1
3	Lennard-Jones Minimum Energy Cluster	18	[-4,4]	1
4	Rastrigin's Function	10	[-100,100]	1
5	Griewangk's Function	10	[-100,100]	1
6	Weierstrass Function	10	[-100,100]	1
7	Modified Schwefel's Function	10	[-100,100]	1
8	Expanded Schaffer's F6 Function	10	[-100,100]	1
9	Happy Cat Function	10	[-100,100]	1
10	Ackley Function	10	[-100,100]	1

2.1.6 Rok 2017

CEC benchmark v roce 2017 obsahoval 30 testovacích funkcí. Po připomínkách byla testovací funkce F2 vyřazena a množina funkcí se zmenšila na 29. Sada je implementována v jazyce C a MATLAB. Pro rychlejší běh je možné použít i jazyk C++, který byl doplněn až dodatečně.

Tabulka 8 D a MaxFES pro rok 2017 [1]

Počet dimenzí (D=*)	MaxFES
10	100 000
30	300 000
50	500 000
100	1 000 000

Souhrn 29 testovacích funkcí v CEC benchmarku 2017

Tabulka 9 Souhrn testovacích funkcí CEC benchmarku 2017[1]

Typ funkce	**Číslo funkce**	**Název funkce**	F_i^*
Unimodální	1	Shifted and Rotated Bent Cigar Function	100
	2	Shifted and Rotated Zakharov Function	300
Jednoduché multimodální	3	Shifted and Rotated Rosenbrock's Function	400
	4	Shifted and Rotated Rastrigin's Function	500
	5	Shifted and Rotated Expanded Scaffer's F6 Function	600
	6	Shifted and Rotated Lunacek Bi_Rastrigin Function	700

	7	Shifted and Rotated Non Continuous Rastrigin's Function	800
	8	Shifted and Rotated Levy Function	900
	9	Shifted and Rotated Schwefel's Function	1000
Hybridní	10	Hybrid Function 1 (N=3)	1100
	11	Hybrid Function 2 (N=3)	1200
	12	Hybrid Function 3 (N=3)	1300
	13	Hybrid Function 4 (N=4)	1400
	14	Hybrid Function 5 (N=4)	1500
	15	Hybrid Function 6 (N=4)	1600
	16	Hybrid Function 7 (N=5)	1700
	17	Hybrid Function 8 (N=5)	1800
	18	Hybrid Function 9 (N=5)	1900
	19	Hybrid Function 10 (N=6)	2000
Kompozitní	20	Composition Function 1 (N=3)	2100
	21	Composition Function 2 (N=3)	2200
	22	Composition Function 3 (N=4)	2300
	23	Composition Function 4 (N=4)	2400
	24	Composition Function 5 (N=5)	2500
	25	Composition Function 6 (N=5)	2600
	26	Composition Function 7 (N=6)	2700
	27	Composition Function 8 (N=6)	2800
	28	Composition Function 9 (N=3)	2900
	29	Composition Function 10 (N=3)	3000

2.1.7 Rok 2014

Pro tuto soutěž byly vyvinuty testovací funkce s několika novými vlastnostmi, jako jsou nové základní testovací funkce, skládání testovacích funkcí extrahováním vlastností dimenzí z několika různých testovacích funkcí, stupňovité propojení, testovací funkce s rotovanými pastmi a podobně. [11]

CEC benchmark v roce 2014 obsahoval 30 testovacích funkcí. Sada testovacích funkcí je implementována v jazyce C, Java a MATLAB.

Tabulka 10 D a MaxFES pro rok 2014 [11]

Počet dimenzí (D=*)	MaxFES
10	100 000
30	300 000
50	500 000
100	1 000 000

Souhrn 30 testovacích funkcí CEC benchmark 2014

Tabulka 11 Souhrn testovacích funkcí CEC benchmarku 2014 [11]

Typ funkce	Číslo funkce	Název funkce	F_i^*
Unimodální	1	Rotated High Conditioned Elliptic Function	100
	2	Rotated Bent Cigar Function	200
	3	Rotated Discus Function	300
Jednoduché multimodální	4	Shifted and Rotated Rosenbrock's Function	400
	5	Shifted and Rotated Ackley's Function	500
	6	Shifted and Rotated Weierstrass Function	600
	7	Shifted and Rotated Griewank's Function	700
	8	Shifted Rastrigin's Function	800
	9	Shifted and Rotated Rastrigin's Function	900
	10	Shifted Schwefel's Function	1000
	11	Shifted and Rotated Schwefel's Function	1100
	12	Shifted and Rotated Katsuura Function	1200
	13	Shifted and Rotated HappyCat Function	1300
	14	Shifted and Rotated HGBat Function	1400
	15	Shifted and Rotated Expanded Griewank's plus Rosenbrock's Function	1500
	16	Shifted and Rotated Expanded Scaffer's F6 Function	1600
Hybridní	17	Hybrid Function 1 (N=3)	1700
	18	Hybrid Function 2 (N=3)	1800
	19	Hybrid Function 3 (N=4)	1900
	20	Hybrid Function 4 (N=4)	2000
	21	Hybrid Function 5 (N=5)	2100
	22	Hybrid Function 6 (N=5)	2200
Kompozitní	23	Composition Function 1 (N=5)	2300
	24	Composition Function 2 (N=3)	2400
	25	Composition Function 3 (N=3)	2500
	26	Composition Function 4 (N=5)	2600
	27	Composition Function 5 (N=5)	2700
	28	Composition Function 6 (N=5)	2800
	29	Composition Function 7 (N=3)	2900

	30	Composition Function 8 (N=3)	3000

2.1.8 Rok 2013

Do roku 2013 uběhlo 8 let od poslední speciální soutěže uspořádané na jednoúčelovou optimalizaci s reálnými parametry, proto přišlo zdokonalení testovací sady. [13]

CEC benchmark v roce 2013 obsahoval 28 testovacích funkcí. Sada testovacích funkcí je implementována v jazyce C, Java a MATLAB.

Tabulka 12 D a MaxFES pro rok 2013 [13]

Počet dimenzí (D=*)	MaxFES
10	100 000
30	300 000
50	500 000

Souhrn 28 testovacích funkcí v CEC benchmarku 2013

Tabulka 13 Souhrn testovacích funkcí CEC benchmarku 2013 [13]

Typ funkce	Číslo funkce	Název funkce	F_i^*
Unimodální	1	Shifted Sphere Function	-1400
	2	Shifted and Rotated Ellipsoidal Function	-1300
	3	Shifted and Rotated Bent Cigar Function	-1200
	4	Shifted and Rotated Discus Function	-1100
	5	Shifted Different Powers Function	-1000
Multimodální	6	Shifted and Rotated Rosenbrock's Function	-900
	7	Shifted and Rotated Schaffer's F7 Function	-800
	8	Shifted Rotated Ackley's Function	-700
	9	Shifted and Rotated Weierstrass's Function	-600
	10	Shifted Rotated Griewank's Function	-500
	11	Shifted Rastrigin’s Function	-400

	12	Shifted and Rotated Rastrigin's Function	-300
	13	Shifted and Rotated Non-continuous Rastrigin's Function	-200
	14	Shifted Schwefel's Function	-100
	15	Shifted and Rotated Schwefel's Function	100
	16	Shifted and Rotated Katsuura Function	200
	17	Shifted Lunacek Bi_rastrigin Function	300
	18	Shifted and Rotated Lunacek Bi_rastrigin Function	400
	19	Shifted and Rotated Griewank-Rosenbrock Function	500
	20	Shifted and Rotated Expanded Schaffer's F6 Function	600
Kompozitní	21	Composition Function 1	700
	22	Composition Function 2	800
	23	Composition Function 3	900
	24	Composition Function 4	1000
	25	Composition Function 5	1100
	26	Composition Function 6	1200
	27	Composition Function 7	1300
	28	Composition Function 8	1400

2.1.9 Rok 2005

Do roku 2005 bylo navrženo a použito mnoho různých druhů optimalizačních algoritmů k řešení problémů optimalizace funkcí s reálnými parametry. Některé z populárních přístupů zahrnovaly evoluční algoritmy s reálnými parametry, evoluční strategie (ES), diferenciální evoluci (DE), optimalizaci hejnem částic (PSO), evoluční programování (EP), klasické metody, jako byla kvazi-Newtonova metoda (QN), hybridní evolučně-klasické metody, další neevoluční metody, jako bylo simulované žíhání (SA), tabuové vyhledávání (TS) a další. [12]

V rámci každé kategorie existovalo mnoho různých metod s odlišnými operátory a principy fungování, jako byly například korelované ES a CMA-ES (Covariance Matrix

Adaptation Evolution Strategy). Ve většině takových studií byla zahrnuta podmnožina standardních testovacích funkcí a problémů (Sphere function, Schwefel's problem, Rosenbrock's function, Rastrigin's function atd.). I když byla v některých výzkumných studiích provedena porovnání, často byla zmatená a omezená na testovací problémy a funkce použité ve studii. [12]

V některých případech se testovací sada a vybraný algoritmus vzájemně doplňovaly a stejný algoritmus nemusel dobře fungovat u jiných funkcí. Určitě bylo třeba tyto metody hodnotit systematickým způsobem, specifikováním společného kritéria ukončení, velikosti problémů, schématu inicializace, spojení/rotace atd. Bylo také nutné provést studii škálovatelnosti, která ukázala, jak se počet vyhodnocení zvyšoval se zvětšením velikosti problému. [12]

Proto se v roce 2005 se uskutečnil první ročník soutěže CEC benchmark, která byla zahájena s cílem poskytnout platformu pro srovnávání výkonnosti různých optimalizačních algoritmů. Tento ročník byl klíčovým milníkem v oblasti evolučních algoritmů a optimalizace, protože formálně stanovil sadu standardizovaných testovacích funkcí a úloh, které mohly být použity pro porovnání výkonnosti algoritmů. [12]

CEC benchmark v roce 2005 obsahoval 25 testovacích funkcí. Sada testovacích funkcí je implementována v jazyce C, Java a MATLAB.

Tabulka 14 D a MaxFES pro rok 2005 [12]

Počet dimenzí (D=*)	MaxFES
10	100 000
30	300 000
50	500 000

Souhrn 25 testovacích funkcí CEC benchmark 2005

Tabulka 15 Souhrn testovacích funkcí CEC benchmarku 2005 [12]

Typ	**Složitost**	**Číslo (F*)**	**Název**
UNIMODÁLNÍ		1	Shifted Sphere Function
		2	Shifted Schwefel's Problem 1.2
		3	Shifted Rotated High Conditioned Elliptic Function
		4	Shifted Schwefel's Problem 1.2 with Noise in Fitness
		5	Schwefel's Problem 2.6 with Global Optimum on Bounds

Multimodální	**Základní**	6	Shifted Rosenbrok's Function
		7	Shifted Rotated Griewank's Function without Bounds
		8	Shifted Rotated Ackley's Function with Global Optimum on Bounds
		9	Shifted Rastrigin's Function
		10	Shifted Rotated Rastrigin's Function
		11	Shifted Rotated Weierstrass Function
		12	Schwefel's Problem 2.13
	Rozšířené	13	Expanded Extended Griewank's plus Rosenbrock's Function (F8F2)
		14	Shifted Rotated Expanded Scaffer's F6
	Hybridní	15	Hybrid Composition Function
		16	Rotated Hybrid Composition Function
		17	Rotated Hybrid Composition Function with Noise in Fitness
		18	Rotated Hybrid Composition Function
		19	Rotated Hybrid Composition Function with a Narrow Basin for the Global Optimum
		20	Rotated Hybrid Composition Function with the Global Optimum on the Bounds
		21	Rotated Hybrid Composition Function
		22	Rotated Hybrid Composition Function with High Condition Number Matrix
		23	Non-Continuous Rotated Hybrid Composition Function
		24	Rotated Hybrid Composition Function
		25	Rotated Hybrid Composition Function without Bound

2.2 Ukázka kódu testovacích funkcí

Pro ilustraci je ukázáno několik testovacích funkcí. Pro účely testování, ale musí zůstat skryty a je nutno s nimi nakládat jako s tzv *blackbox* problémy, takže je zakázáno je používat mimo účely CEC benchmarku např. k výpočtům gradientu.

2.2.1 Ukázka kódu elementárních funkcí

Elementární funkce jsou základem všech dalších složitějších funkcí.

```c
/* code to evaluate sphere function */
long double calc_sphere (long double *x)
{
    int i;
    long double res;
    res = 0.0;
    for (i=0; i<nreal; i++)
    {
        res += x[i]*x[i];
    }
    return (res);
}
```

Obrázek 1 Sphere function [18]

```c
/* Code to evaluate rastrigin's function */
long double calc_rastrigin (long double *x)
{
    int i;
    long double res;
    res = 0.0;
    for (i=0; i<nreal; i++)
    {
        res += (x[i]*x[i] - 10.0*cos(2.0*PI*x[i]) + 10.0);
    }
    return (res);
}
```

Obrázek 2 Rastrigin's function [18]

```c
/* Code to evaluate ackley's function */
long double calc_ackley (long double *x)
{
    int i;
    long double sum1, sum2, res;
    sum1 = 0.0;
    sum2 = 0.0;
    for (i=0; i<nreal; i++)
    {
        sum1 += x[i]*x[i];
        sum2 += cos(2.0*PI*x[i]);
    }
    sum1 = -0.2*sqrt(sum1/nreal);
    sum2 /= nreal;
    res = 20.0 + E - 20.0*exp(sum1) - exp(sum2);
    return (res);
}
```

Obrázek 3 Ackley's function [18]

```c
/* Code to evaluate griewank's function */
long double calc_griewank (long double *x)
{
    int i;
    long double s, p;
    long double res;
    s = 0.0;
    p = 1.0;
    for (i=0; i<nreal; i++)
    {
        s += x[i]*x[i];
        p *= cos(x[i]/sqrt(1.0+i));
    }
    res = 1.0 + s/4000.0 - p;
    return (res);
}
```

Obrázek 4 Griewank's function [18]

```c
/* Code to evaluate schwefel's function */
long double calc_schwefel (long double *x)
{
    int i, j;
    long double sum1, sum2;
    sum1 = 0.0;
    for (i=0; i<nreal; i++)
    {
        sum2 = 0.0;
        for (j=0; j<=i; j++)
        {
            sum2 += x[j];
        }
        sum1 += sum2*sum2;
    }
    return (sum1);
}
```

Obrázek 5 Schwefel's function [18]

```c
/* Code to evaluate rosenbrock's function */
long double calc_rosenbrock (long double *x)
{
    int i;
    long double res;
    res = 0.0;
    for (i=0; i<nreal-1; i++)
    {
        res += 100.0*pow((x[i]*x[i]-x[i+1]),2.0) + 1.0*pow((x[i]-1.0),2.0);
    }
    return (res);
}
```

Obrázek 6 Rosenbrok's function [18]

```
/* Code to evaluate weierstrass's function */
long double calc_weierstrass (long double *x)
{
    int i, j;
    long double res;
    long double sum;
    long double a, b;
    int k_max;
    a = 0.5;
    b = 3.0;
    k_max = 20;
    res = 0.0;
    for (i=0; i<nreal; i++)
    {
        sum = 0.0;
        for (j=0; j<=k_max; j++)
        {
            sum += pow(a,j)*cos(2.0*PI*pow(b,j)*(x[i]+0.5));
        }
        res += sum;
    }
    return (res);
}
```

Obrázek 7 Weierstrass function [18]

```c
/* Code to evaluate schaffer's function and rounding-off variables */
long double nc_schaffer (long double x, long double y)
{
    int i;
    int a;
    long double b;
    long double res;
    long double temp1, temp2;
    long double t1[2], t2[2];
    t1[0] = x;
    t1[1] = y;
    for (i=0; i<2; i++)
    {
        if (fabs(t1[i]) >= 0.5)
        {
            res = 2.0*t1[i];
            a = res;
            b = fabs(res-a);
            if (b<0.5)
            {
                t2[i] = a/2.0;
            }
            else
            {
                if (res<=0.0)
                {
                    t2[i] = (a-1.0)/2.0;
                }
                else
                {
                    t2[i] = (a+1.0)/2.0;
                }
            }
        }
        else
        {
            t2[i] = t1[i];
        }
    }
    temp1 = pow((sin(sqrt(pow(t2[0],2.0)+pow(t2[1],2.0)))),2.0);
    temp2 = 1.0 + 0.001*(pow(t2[0],2.0)+pow(t2[1],2.0));
    res = 0.5 + (temp1-0.5)/(pow(temp2,2.0));
    return (res);
}
```

Obrázek 8 Rounded Schaffer's function [18]

```
/* Code to evaluate rastrigin's function and rounding-off variables */
long double nc_rastrigin (long double *x)
{
    int i;
    int a;
    long double b;
    long double res;
    for (i=0; i<nreal; i++)
    {
        if (fabs(x[i]) >= 0.5)
        {
            res = 2.0*x[i];
            a = res;
            b = fabs(res-a);
            if (b<0.5)
            {
                temp_x4[i] = a/2.0;
            }
            else
            {
                if (res<=0.0)
                {
                    temp_x4[i] = (a-1.0)/2.0;
                }
                else
                {
                    temp_x4[i] = (a+1.0)/2.0;
                }
            }
        }
        else
        {
            temp_x4[i] = x[i];
        }
    }
    res = 0.0;
    for (i=0; i<nreal; i++)
    {
        res += (temp_x4[i]*temp_x4[i] - 10.0*cos(2.0*PI*temp_x4[i]) + 10.0);
    }
    return (res);
}
```

Obrázek 9 Rounded Rastrigin's function [18]

2.2.2 Ukázka kódu testovací funkce F1 Shifted Sphere Function z roku 2005

```
long double calc_benchmark_func(long double *x)
{
    long double res;
    transform (x, 0);
    basic_f[0] = calc_sphere (trans_x);
    res = basic_f[0] + bias[0];
    return (res);
}
```

Obrázek 10 Shifted Sphere Function [18]

Celé testování pomocí testovacích funkcí prochází několika zásadními fázemi. Nejprve se zavolá funkce *transform(x, 0)* (viz Obrázek 11), která mění vstupní data pomocí matic a vektorů uložených v globálních polích. Postupně odečítá prvek pole *o[count]* od každého prvku vstupního pole *x*, poté každý výsledek dělí odpovídajícím prvkem pole *lambda[count]*. Následně provede násobení matice *g* s výsledkem z předchozího dělení, a nakonec vynásobí matici *l* s výsledkem této operace. Výslednou hodnotu zapíše do globálního pole *trans_x*. Tímto způsobem funkce modifikuje vstupní data podle zadané specifikace.

```
void transform (long double *x, int count)
{
    int i, j;
    for (i=0; i<nreal; i++)
    {
        temp_x1[i] = x[i] - o[count][i];
    }
    for (i=0; i<nreal; i++)
    {
        temp_x2[i] = temp_x1[i]/lambda[count];
    }
    for (j=0; j<nreal; j++)
    {
        temp_x3[j] = 0.0;
        for (i=0; i<nreal; i++)
        {
            temp_x3[j] += g[i][j]*temp_x2[i];
        }
    }
    for (j=0; j<nreal; j++)
    {
        trans_x[j] = 0.0;
        for (i=0; i<nreal; i++)
        {
            trans_x[j] += l[count][i][j]*temp_x3[i];
        }
    }
    return;
}
```

Obrázek 11 Funkce transform [18]

Jako další se volá funkce *calc_sphere(trans_x)* (viz Obrázek 1), která vezme uložená data z pole *trans_x* a vypočítá hodnotu podle matematického předpisu. Výsledek zapíše do pole *basic_f*.

Nakonec se k vypočtené hodnotě funkce přičte přednastavená proměnná *bias*, která se inicializuje před počítáním testovací benchmarkové funkce. V tomto konkrétním případě se zavolá funkce *initialize()* (viz Obrázek 12). Obecně platí, že každá funkce má vlastní inicializační funkci, kde se nejprve otevře soubor se vstupními daty, která se uloží do matice *o*. Nastaví se hodnota *bias*, která představuje globální optimum účelové funkce. Zde představuje globální minimální hodnotu účelové funkce, a ta se rovná *-450*.

```
void initialize()
{
    int i, j;
    FILE *fpt;
    fpt = fopen("input_data/sphere_func_data.txt","r");
    if (fpt==NULL)
    {
        fprintf(stderr,"\n Error: Cannot open input file for reading \n");
        exit(0);
    }
    for (i=0; i<nfunc; i++)
    {
        for (j=0; j<nreal; j++)
        {
            fscanf(fpt,"%Lf",&o[i][j]);
            printf("\n O[%d][%d] = %LE",i+1,j+1,o[i][j]);
        }
    }
    fclose(fpt);
    bias[0] = -450.0;
    return;
}
```

Obrázek 12 Funkce initialize [18]

Pro správnou funkci programu je potřeba nastavit dvě globální proměnné nreal a nfunc. Proměnná *nreal* udává, kolik nezávislých proměnných (nebo rozměrů) má optimalizační problém. Každá proměnná může být interpretována jako jedna dimenze prostoru, ve kterém se hledá optimální řešení. Například pokud je počet reálných proměnných 2, pak je problém dvojdimenzionální (2D), a tedy vyhledáváme optimální řešení ve dvojdimenzionálním prostoru.

Proměnná *nfunc* udává, kolik různých funkcí je k dispozici v testovacím scénáři. Každá základní funkce může reprezentovat jiný typ optimalizačního problému nebo charakteristiku, která je testována.

2.3 Definice typů funkcí

Rozdělení funkcí na typy v jednotlivých ročnících CEC benchmarku pomáhá k rozvrstvení testování na jednotlivé podskupiny.

Existují čtyři hlavní skupiny: unimodální, multimodální (někdy označované jako základní), hybridní a kompozitní funkce.

2.3.1 Unimodální

Unimodální funkce se vyznačují jediným globálním optimem a už nemají žádná další lokální optima.

2.3.2 Multimodální

Multimodální funkce se vyznačují několika lokálními optimy a můžou mít i několik globálních.

2.3.3 Hybridní

Ve skutečných optimalizačních problémech mohou mít různé podkomponenty proměnných různé vlastnosti. Proměnné jsou náhodně rozděleny do několika podkomponentů a pak jsou pro jednotlivé podkomponenty použity odlišné základní funkce.

Hybridní funkce skládají několik funkcí dohromady. Mohou být složeny z unimodálních, ale hlavně z multimodálních funkcí. Pro každou funkci je vyznačená procentuální hodnota podílu na hybridní funkci.

Matematická definice:

$$F(x) = g_1(M_1 z_1) + g_2(M_2 z_2) + \cdots + g_N(M_N z_N) + F^*(x)$$

F(x): hybridní funkce

$g_i(x)$: i-tá základní funkce použitá k sestavení hybridní funkce

N: počet základních funkcí

$$z = [z_1, z_2, \ldots, z_N]$$

$z_1 = [y_{S_1}, y_{S_2}, \ldots, y_{S_n}]$, $z_2 = [y_{S_{n+1}}, y_{S_{n+2}}, \ldots, y_{S_{n+n_2}}]$, $\ldots$, $z_N = [y_{S_{N-n+1}}, y_{S_{N-n+2}}, \ldots, y_{S_p}]$

$y = x - o_i$, $S = \text{randperm}(1:D)$

p_i: používá se ke kontrole procentuálního vyjádření $g_i(x)$

n_i: dimenze pro každou základní funkci $\sum_{i=1}^{N} n_i = D$

$$n_1 = [p_1, D], \quad n_2 = [p_2, D], \quad \dots, \quad n_{N-1} = [p_{N-1}, D], \quad n_N = D - \sum_{i=1}^{N-1} n_i$$

[9]

2.3.4 Kompozitní

Nové kompozitní funkce jsou testovací funkce, které mají mnoho žádoucích vlastností.

Nápad spočívá ve skládání standardních benchmarkových funkcí za účelem vytvoření náročnější funkce s náhodně umístěným globálním optimem a několika náhodně umístěnými hlubokými lokálními optimy. Gaussovy funkce jsou použity ke kombinaci těchto benchmarkových funkcí a zvrásnění struktury jednotlivých funkcí.

Matematická definice:

$$F(x) = \sum_{i=1}^{N} \{ \omega_i^* [\lambda_i g_i(x) + bias_i] \} + F^*$$

$F(\mathbf{x})$: nová kompozitní funkce

$g_i(\mathbf{x})$: i-tá základní funkce použitá k sestavení kompozitní funkce

N: počet základních funkcí

o_i: nově posunutá optimální pozice pro každou $g_i(x)$, definuje pozici globálního a lokálního optima

$bias_i$: určuje, které optimum je globální optimum

σ_i: používá se ke kontrole rozsahu pokrytí každé $g_i(x)$ malý σ_i poskytuje úzký rozsah pro danou $g_i(x)$

λ_i: používá se ke kontrole výšky každé $g_i(x)$

ω_i: hodnota váhy pro každou $g_i(x)$, vypočítává se následovně:

$$\omega_i = \frac{1}{\sum_{j=1}^{D}(x_j - o_{ij})^2} \exp\left(-\frac{\sum_{j=1}^{D}(x_j - o_{ij})^2}{2D\sigma_i^2}\right)$$

Pak se váha normalizuje

$$\omega_i = \frac{w_i}{\sum_{i=1}^{n} w_i}$$

Takže když x= o_i,

$\omega_{j=}\begin{cases}1 & j = i\\ 0 & j \neq i\end{cases}$ pro j=1,2,.., $N, (f(x)) = \text{bias} + f^*$

Lokální optimum, které má nejmenší hodnotu zkreslení (bias), je globálním optimem. Kompozitní funkce lépe spojuje vlastnosti dílčích funkcí a udržuje spojitost kolem globálních/lokálních optim. Funkce $F_i' = F_i - F_i^*$ jsou používány jako g_i. Tímto způsobem jsou hodnoty funkcí globálních optim gi rovny 0 pro všechny kompozitní funkce. Všechny základní funkce použité v kompozitních funkcích jsou posunuté a rotované funkce.

[9]

II. PRAKTICKÁ ČÁST

3 TESTOVÁNÍ CEC BENCHMARK SETU

Praktická část bakalářské práce je zaměřena na testování funkcí. Testuje se hlavně citlivost testovacích funkcí na přesnost jejich vstupních parametrů. Cílem bylo zjistit, jak moc se může pozměněná (vypočítaná) hodnota lišit od pozice optima v každé dimenzi, tak aby byl výstup z funkce stále optimem nebo v přípustné odchylce (např menší než 10^{-8} pro rok 2022). Zároveň bylo potřeba otestovat všechny implementace a navzájem je mezi sebou porovnat.

Pro přehlednost je každý rok uveden v samostatné podkapitole.

Každý prvek vstupního vektoru jednotlivé funkce byl otestován samostatně ve všech dimenzích, které byly definované pro účely testování v jednotlivých ročnících. Výsledky jsou zaneseny do grafů a tabulek. Na závěr podkapitoly je vždy shrnuto, jak se mezi sebou jednotlivé implementační jazyky odlišují.

3.1 Testování funkcí

Pro každou funkci existují různá vstupní data, vždy zapsána v souboru shift_data_x.txt (x je číslo funkce). Vstupní data jsou v jednotlivých funkcích pro všechny dimenze stejná. Jako vstupní data se bere vektor posunutého globálního optima, který odpovídá ideální hodnotě funkce.

Testování probíhá v několika krocích. Jako první se nastaví výchozí stav testování. Použijí se vstupní data ze souboru shift_data_x.txt. Následně se ubírá přesnost po jednom desetinném místě, dokud se dílčí výsledek funkce nedostane mimo hranici přípustné odchylky. V této chvíli je pro vyšší přesnost použita metoda půlení intervalů, která osciluje mezi příliš malou hodnotou odchylky a již nepřípustnou odchylkou (mimo hranice přípustné odchylky). Půlení končí ve chvíli, kdy se hodnota funkce dostane do požadovaných mezí. Meze byly navrhnuty tak, aby se získala co největší přesnost. Např. pro rok 2022 je horní mez 10^{-8} a spodní mez $9{,}6^{-8}$.

Na závěr je získaná odchylka vyzkoušena ve výchozím stavu testování. Po zkoušce je získaná odchylka zapsána do výsledků. Pro účely vykreslení grafu je z jednotlivých odchylek funkcí vyčleněna pouze ta nejmenší, a to v každé dimenzi.

Celý algoritmus probíhá pro každý prvek vektoru, pro každou funkci ve všech dimenzích, které jsou definovány v jednotlivých ročnících. Vždy se vybírá prvních *D* prvků vektoru. Pokud D=10, pak se použije prvních 10 prvků vektoru ze souboru shift_data_x.txt.

U testování dochází k několika anomáliím. První anomálie "necitlivosti" se vyznačuje absolutní necitlivostí ke změně. Prvek vektoru je možné dokonce vynulovat, avšak výsledek funkce zůstává stále stejný. Těmto odchylkám je přiřazena nekonečná odchylka *(infinite)*. U druhé anomálie "extrémní citlivosti" dochází k takové míře citlivosti, že po odstranění jednoho či více desetinných řádů dochází k rozsáhlému zkreslení ideální hodnoty. Pomocí metody půlení intervalů se s vysokou přesností (25 desetinných míst) získaná odchylka nedokáže dostat do požadovaných mezí, proto se počítá odchylka znovu ze vstupního vektoru a v tomto případě může být přípustná hodnota získané odchylky nižší než spodní požadovaná mez.

Celé testování je zalgoritmizováno pomocí jazyka Python a je dostupné na GitHubu pro jednotlivé ročníky. Odkazy jsou v příloze.

Pro zobrazení výsledků testování je využito grafů. Pro každou použitou dimenzi v ročníku jeden. V grafu je pro lepší přehlednost konečných hodnot využito logaritmické osy. Pro funkci v dimenzi jsou využity výlučně nejnižší odchylky, a to z toho důvodu, aby byla dosažena maximální nutná přesnost.

Pro názornost byla vytvořena tabulka se srovnáním jednotlivých funkcí, jejich vyžadované přesnosti, jak se liší mezi implementačními jazyky. Do tabulky je zapsáno srovnání vyžadované přesnosti vždy dvojice implementačních jazyků. Prochází se všechny funkce a vyžadovaná přesnost u jednotlivých elementů vektoru pro danou dimenzi. Počet elementů se pokaždé rovná počtu dimenzí.

Pokud se mezi sebou přesnosti rovnají, pak vyžadují stejnou přesnost, zapsáno pomocí znaku pro rovnost. Pokud se vyžadované přesnosti mezi sebou liší, pak je nutné odlišit, který jazyk vyžaduje vyšší přesnost. To udává číslo za znakem plus, minus. Znaménko plus se váže s nutností vyšší přesnosti u prvního jazyku z dvojice. Znaménko mínus se váže s nutností nižší přesnosti prvního jazyku neboli nutností vyšší přesnosti u druhého jazyku. Např rok 2022, D=10, C/MATLAB funkce F12 – C potřebuje 1x (u jednoho prvku) vyšší přesnost, 9x se přesnosti rovnají.

3.1.1 Ukázka testování

Na testování bylo využito scriptů napsaných v jazyce Python. Celý proces probíhá pro každý prvek vektoru posunutého globálního optima všech funkcí ve všech dimenzích. Na otestování jednoho prvku vektoru se využívá skript z repozitáře *run-tests.py*, který přijímá 3

argumenty-číslo funkce, dimenzi a číslo prvku. Skript nejprve naimportuje originální data, tak aby se testovalo ze základní pozice. Následně spustí testovací běh hlavní funkce, aby získal momentální hodnotu výsledku benchmarkové funkce.

```
sys.argv = ['run-tests.py', arg1, arg2, arg3]
#import the original data

runpy.run_path('./import-original.py')
runpy.run_path('./run-main.py')

# Open the dimension file and func_num file and read its contents

with open(f'test_data/current_result_{argNum}.txt', 'r') as file:
    result = file.read()

# Find the number in the string
result = re.findall(r'\d+\.\d+', result)

# Convert to float when you need to do a numerical operation

result = str(result)
result = result.split('.')
#before the floating point
before = result[0]
#after the floating point
after = result[1]

# Now I want to get the first 8 digits from after
sevenDigits = after[:7]
```

Obrázek 13 Začátek skriptu run-tests.py

Pokud je výsledek benchmarkové funkce už od počátku mimo nastavené kritérium (10^{-8}), tak se ukončí běh programu. Pokud je výsledek v předpokládané hodnotě (ideální hodnota funkce), tak maže desetinná místa od posledního pomocí skriptu *delete-floating-point.py*, dokud výsledek není v odchylce větší než 10^{-8}.

```
if int(sevenDigits) != 0:
    sys.exit(f"The default value is less than 10e-08. Function:{argNum}, Dimension:{dimension}, Element:{arg3}")
lastTwoDigits = after[-2:]
counter_first = 0
while int(sevenDigits) == 0:
    runpy.run_path('./delete-floating-point.py')
    runpy.run_path('./run-main.py')
    with open(f'test_data/current_result_{argNum}.txt', 'r') as file:
        result = file.read()

    result = re.findall(r'\d+\.\d+', result)

    result = float(result[0])


    result = str(result)
    result = result.split('.')
    #before the floating point
    before = result[0]
    #after the floating point
    after = result[1]

    sevenDigits = after[:7]
    lastTwoDigits = after[-2:]
```

Obrázek 14 Cyklus s mazáním desetinného místa v run-tests.py

Jako další se provádí úprava čísel. Z vědeckého zápisu se přejde na zápis s desetinným místem, aby bylo možné provádět výpočty. To se provede pomocí skriptů *first-run.py* a *convert-e-to-float.py*. Příštím krokem je využití metody půlení intervalů na vypočítání odchylky. Odchylka se musí vejít do intervalu, který má hranice 10^{-8} a 9.6^{-8}. První zmíněný skript zároveň provede první půlení intervalů. Nastaví původní spodní a horní hranici.

```
while int(sevenDigits) > 0 or int(lastTwoDigits) < 96:

    runpy.run_path('./help-bisecting.py')

    runpy.run_path('./half.py')

    runpy.run_path('./run-main.py')
    with open(f'test_data/current_result_{argNum}.txt', 'r') as file:
        result = file.read()

    result = re.findall(r'\d+\.\d+', result)
    result = float(result[0])

    result = str(result)
    result = result.split('.')
    #before the floating point
    before = result[0]
    #after the floating point
    after = result[1]

    sevenDigits = after[:7]
    lastTwoDigits = after[-2:]
```

Obrázek 15 Cyklus s půlením intervalů

Následuje půlení intervalů pomocí skriptů *help-bisecting.py*, který vždy aktualizuje hranice podle toho, kam se hodnota posunula, a *half.py*, který podle stanovených hranic vypočítá hodnotu středu intervalu. Opět se kontroluje s hlavní funkcí, která udává hodnotu benchmarkové funkce.

Po dokončení tohoto procesu se vypočítá odchylka pomocí skriptu *deviation.py* a nakonec se zkontroluje odchylka s původními daty pomocí skriptu *inspection.py*.

Během testů dochází k anomáliím. K jedné anomálii dochází v případě, že v cyklu u mazání desetinného místa se celé číslo vymaže nebo se vymažou všechna desetinná místa. V tomto případě prvek vektoru nemá vliv na funkci a jeho výsledná odchylka se rovná nekonečnu (inf). Ke druhé odchylce dochází při půlení intervalů, kdy se průběh dokáže zacyklit, protože se nelze dostat k výsledné odchylce přesně. V tomto případě se zavolá skript *anomaly-deviation.py*, který nastaví referenční hodnotu odchylky.

```
counter = 0

while int(sevenDigits) > 0 or int(lastTwoDigits) < 96:

    runpy.run_path('./help-bisecting.py')

    runpy.run_path('./half.py')

    runpy.run_path('./run-main.py')
    with open(f'test_data/current_result_{argNum}.txt', 'r') as file:
        result = file.read()

    result = re.findall(r'\d+\.\d+', result)
    result = float(result[0])

    result = str(result)
    result = result.split('.')
    #before the floating point
    before = result[0]
    #after the floating point
    after = result[1]

    sevenDigits = after[:7]
    lastTwoDigits = after[-2:]
    counter += 1
    if counter == 50:
        runpy.run_path('./anomaly-deviation.py')
```

Obrázek 16 Řešení příliš citlivého prvku funkce

Obrázek 16 Řešení příliš citlivého prvku funkce

Pro řešení citlivého prvku funkce je použit speciální skript, který se pustí až po tom, co padesátkrát proběhne půlení intervalů. Nakonec se ve skriptu *anomaly-deviation.py* nastaví odchylka na rozdíl mezi momentální hodnotou prvku a původní hodnotou.

```
if int(sevenDigits) != 0:
    sys.exit(f"The default value is less than 10e-08. Function:{argNum}, Dimension:{dimension}, Element:{arg3}")
lastTwoDigits = after[-2:]
counter_first = 0
while int(sevenDigits) == 0:
    runpy.run_path('./delete-floating-point.py')
    runpy.run_path('./run-main.py')
    with open(f'test_data/current_result_{argNum}.txt', 'r') as file:
        result = file.read()

    result = re.findall(r'\d+\.\d+', result)

    result = float(result[0])


    result = str(result)
    result = result.split('.')
    #before the floating point
    before = result[0]
    #after the floating point
    after = result[1]

    sevenDigits = after[:7]
    lastTwoDigits = after[-2:]
    counter_first += 1
    if counter_first == 40:
        with open(f'test_data/result/result_data_{argNum}_dim_{dimension}_number_of_element_{number_of_element+1}.txt', 'w') as file:
            file.write(f"deviation:{float('inf')}")
        sys.exit(f"Function:{argNum}, Dimension:{dimension}, Element:{arg3} has a deviation of infinity")
```

Obrázek 17 Řešení nekonečné odchylky v run-tests.py

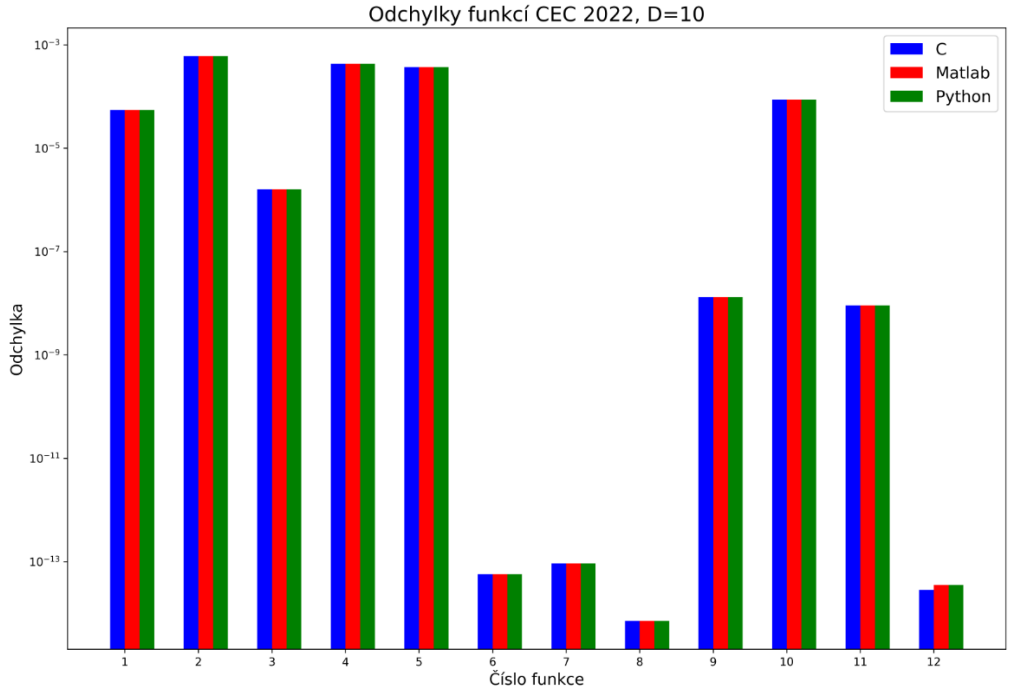
Ve skriptu *delete-floating-point.py* se ještě řeší naprosté vynulování prvku. Prvek by měl mít maximálně 20 desetinných míst, přesto skript s nekonečnou odchylkou končí až po čtyřiceti běžích.

3.2 Testování funkcí CEC benchmark 2022

V roce 2022 CEC benchmark obsahoval 12 testovacích funkcí. Pro testování citlivosti byly využity tři programovací jazyky z oficiální soutěže, a to Python, C a MATLAB. Počet dimenzí byl nastaven podle definice na 10 a 20. Maximální přijatelná odchylka měla být menší než 10^{-8}.

3.2.1 Výsledky testování 2022

Pro každou dimenzi, se kterou se v tomto ročníku pracovalo, je zobrazen graf se získanými odchylkami. U některých funkcí se získané odchylky velmi liší v rámci elementů vektoru v dimenzi. Kdyby se ovšem vybrala střední hodnota vypočítané odchylky v rámci funkce v dimenzi, tak by se výsledky u hybridních funkcí výrazně změnily.



Obrázek 18 Graf odchylek testovacích funkcí z CEC 2022, D=10

Z grafu je vidět, že implementace v různých jazycích se od sebe nepatrně liší. Kód v Pythonu a v MATLABu dosahuje stejných výsledků, zatímco C se v kompozitní funkci F12 odlišuje. Je ilustrativně vidět, že při používání jazyka C v rámci soutěže je požadována přesnost vyšší než při použití zbylých dvou jazyků.

Tabulka 16 Srovnání vyžadované přesnosti v jednotlivých jazycích CEC 2022, D=10

Číslo funkce (F*)	C/MATLAB	C/Python	MATLAB/Python
1	=10	=10	=10
2	=10	=10	=10
3	=10	=10	=10
4	=10	=10	=10
5	=10	=10	=10
6	=10	=10	=10
7	=10	=10	=10

8	=10	=10	=10
9	=10	=10	=10
10	=10	=10	=10
11	=10	=10	=10
12	+1/=9	+1/=9	=10

Pro valnou většinu funkcí jsou přesnosti dvojic implementačních jazyků naprosto stejné. Z tabulky je jasně patrné, že jazyk C z těchto tří implementačních jazyků vyžaduje nejvyšší přesnost. U F12 potřebuje jednou vyšší přesnost oproti MATLABu a Pythonu.

Pro jednotlivé skupiny funkcí se odlišují jejich vypočítané odchylky. U skupiny unimodální funkce a multimodálních funkcí (F1-F5) je vidět, že jejich získaná odchylka může být relativně vysoká a stále jsou splněny podmínky soutěže. U skupiny hybridních funkcí (F6-F8) je vidět, že jejich získané odchylky naopak nesmí být příliš vysoké, a to v mezích do 10^{-13}, jinak se již od optima příliš vzdalují. Pro poslední skupinu kompozitních funkcí (F9-F12) se získané odchylky velmi liší.

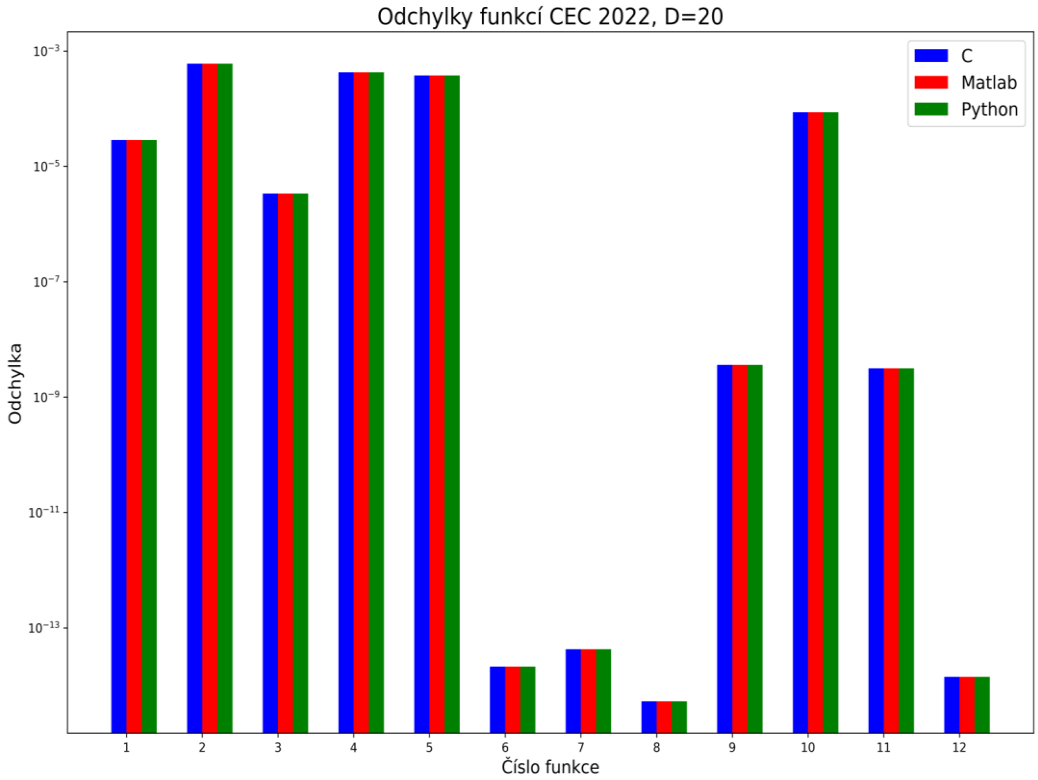
Z definice kompozitních funkcí je možno vyčíst, že jednotlivé funkce jsou složeny z několika základních funkcí, ale můžou se použít i hybridní funkce. Mnoho dalších atributů funkce se dá ovlivnit, tak aby funkce byla výzvou na vyřešení a lépe představovala skutečnost.

Tabulka 17 Výpis vypočítaných odchylek hybridních funkcí CEC 2022, d=10

	Hybridní funkce		
Číslo prvku	**F6**	**F7**	**F8**
1	6^{-12}	inf	7^{-15}
2	3^{-13}	1^{-7}	3^{-8}
3	6^{-13}	3^{-8}	2^{-8}
4	6^{-7}	inf	3^{-9}
5	9^{-14}	4^{-4}	7^{-15}
6	1^{-13}	2^{-9}	4^{-9}

7	3^{-7}	1^{-8}	1^{-2}
8	7^{-14}	8^{-5}	6^{-8}
9	3^{-7}	4^{-9}	4^{-8}
10	5^{-14}	9^{-14}	1^{-2}

U hybridních funkcí F6, F7 a F8 vypočítané odchylky pro jednotlivé prvky vektoru velmi kolísají. Jsou to právě ty funkce, u kterých byly objeveny anomálie. V tabulce výše jsou vidět vypočítané odchylky jednotlivých prvků pro funkce F6, F7, F8. Do tabulky jsou zapsány pouze řádově, aby byl vidět jejich rozsah. Je vidět, že funkce jsou náročné na vyřešení, protože každý prvek vektoru se liší svojí důležitostí v rámci funkce. Je to spojeno s tím, že hybridní funkce jsou složené z několika základních funkcí a je jim jednotlivě přiřazena váha. Zároveň odpovídají tomu, že pokud tomu nastavení funkce napovídá, jsou velmi složité na vyřešení.



Obrázek 19 Graf odchylek testovacích funkcí z CEC 2022, D=20

Z grafu je vidět, že se velmi podobá grafu s D=10, lze pozorovat pouze nepatrné nuance. Skupiny zůstávají naprosto totožné. Skupina základních funkcí (F1-F5) zůstává stále poměrně jednodušeji řešitelná a vypočítaná odchylka tomu odpovídá. Skupina kompozitních

funkcí (F9-F12) opět nabízí rozsáhlé výkyvy uvnitř skupiny. Skupina hybridních funkcí (F6-F8) nadále vyžaduje velmi vysokou přesnost.

Všechny funkce až na F12 kopírují stav grafu s D=10. Pro funkci F7 a F8 jsou vidět změny. U F1, F6, F7, F9, F11 se požadovaná přesnost zvedla s počtem dimenzí. U F3 a F12 se naopak o něco snížila. Je možno pozorovat, že pro všechny implementační jazyky je minimální získaná odchylka stejná, a to u všech funkcí.

Je nutno připomenout, že v grafu se zobrazují minimální získané odchylky, proto tabulka požadované přesnosti bude více vypovídající.

Tabulka 18 Srovnání vyžadované přesnosti v jednotlivých jazycích CEC 2022, D=20

Číslo funkce (F*)	**C/MATLAB**	**C/Python**	**MATLAB/Python**
1	=20	=20	=20
2	=20	=20	=20
3	=20	=20	=20
4	=20	=20	=20
5	=20	=20	=20
6	=20	=20	=20
7	=20	=20	=20
8	=20	=20	=20
9	=20	=20	=20
10	=20	=20	=20
11	=20	=20	=20
12	+4/=16	+4/=16	=20

Tabulka pro D=20 na první pohled připomíná tabulku pro D=10. Skupina bezmála všech funkcí (F1-F11) přesností odpovídá předchozímu zjištění, tedy že pro téměř všechny funkce veškeré implementační jazyky mají stejné vypočítané odchylky, a tudíž i požadovanou přesnost. Tomuto pravidlu se vymyká funkce F12, u které jazyk C vyžaduje vyšší přesnost oproti

MATLABu i Pythonu. Současně se dá pozorovat, že MATLAB a Python jsou si svými přesnostmi značně podobné.

Kompozitní funkce F12 patřila do skupiny funkcí, ve kterých se objevovaly anomálie, a to do skupiny extrémně citlivých funkcí. Pro tyto funkce se nedala exaktně určit jejich odchylka v přípustných mezích.

Z tabulky je zřejmé, že C vyžaduje vyšší přesnost v porovnání s dalšími dvěma jazyky, což může být z pohledu CEC 2022 zásadním poznatkem.

3.2.2 Závěr testování CEC 2022

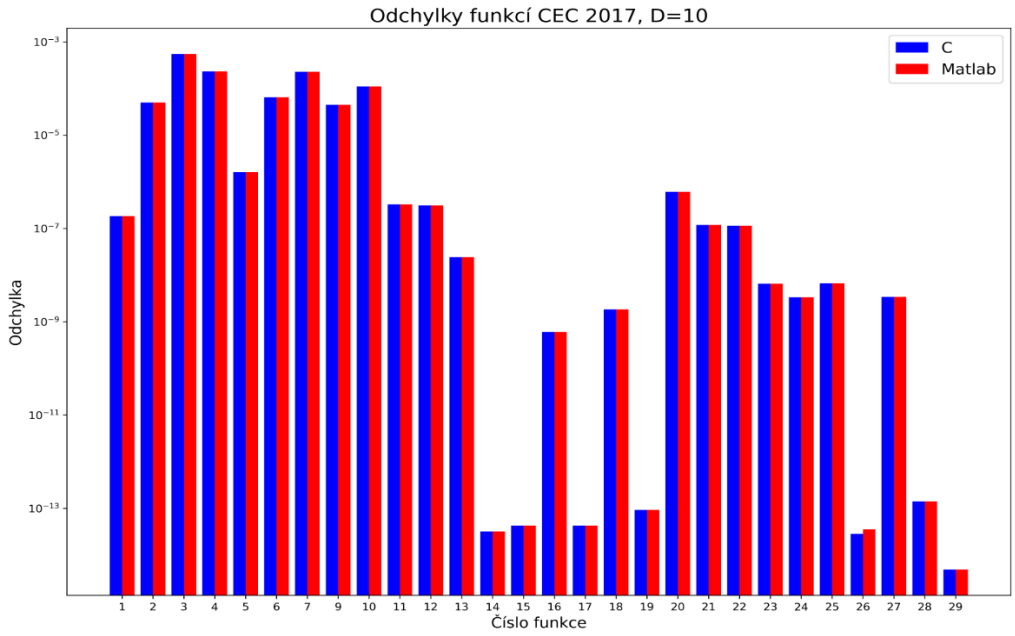
V rámci testovací sady funkcí CEC 2022 bylo 12 funkcí. Použity byly tři jazyky na jejich implementaci, a to MATLAB, C a Python. Z testování je jasně patrné, že se přesnosti pro různé implementační jazyky odlišují. Pokud si účastník soutěže 2022 zvolí jako implementační jazyk testovacích funkcí C, pak je znevýhodněn na funkci F12 oproti těm, kteří se rozhodli pro MATLAB nebo Python.

3.3 Testování funkcí CEC benchmarku 2017

V tomto ročníku bylo uvedeno 29 testovacích funkcí. Pro testování citlivosti bylo důležité využít oba programovací jazyky MATLAB a C zahrnuté v oficiální soutěži a porovnat výsledky funkcí mezi sebou. Nejdůležitějším nastavením základního bodu testování byl počet dimenzí. Výsledky pro D=10 a D=30 byly přijatelné v původní verzi. Pro finální verzi bylo nutné zahrnout výsledky i pro D=50 a D=100. Maximální přijatelná odchylka měla být menší než 10^{-8}.

3.3.1 Výsledky testování

Už v počátku testování byly objeveny funkce, pro které základní posunuté optimum leželo mimo hranice maximální přijatelné odchylky. Byly to funkce F8 a F26. U F8 se nepodařilo získat žádná výsledná data, protože pro oba implementační jazyky a všechny dimenze se získaná odchylka pohybuje nad určenou hranicí maximální přijatelné odchylky. Pro představu pro funkci F8 je očekávaná hodnota alespoň 900.00000009, ale vypočítaná hodnota, kde je nastaven výchozí vektor posunutého globálního optima a D=10, se rovná 901.442600987. Proto není funkce F8 uvedena ve výsledných grafech. Funkce F26 měla obdobný problém, ale pouze pro jazyk C a pro D=50, D=100.



Obrázek 20 Graf odchylek testovacích funkcí z CEC 2017, D=10

Z grafu se dá vyčíst, že se mezi sebou implementace nepatrně liší. Ve valné většině funkcí jsou přesnosti dílčích jazyků absolutně stejné. Výjimkou je kompozitní funkce F26, u které se přesnost odlišovala už od základního nastavení vstupního vektoru posunutého globálního optima.

Tabulka 19 Srovnání vyžadované přesnosti C/MATLAB CEC 2017, D=10

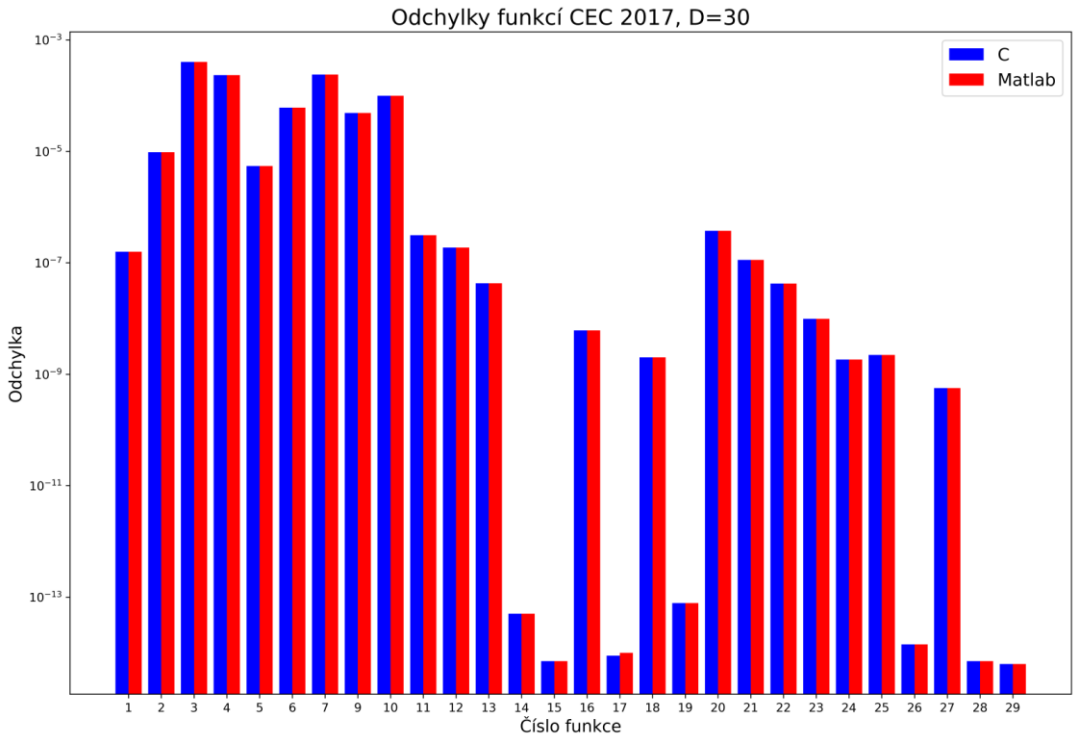
Číslo funkce (F*)	**C/MATLAB**
1-25 (bez F8)	=10
26	+1/=9
27	=10
28	=10
29	=10

Až na jednu funkci se přesnosti rovnají. Z tabulky je poznat, že jazyk C potřebuje vyšší přesnost než MATLAB u jednoho elementu vektoru. Je vidět, že se liší pouze jediná přesnost, a to u prvku, u kterého byla dosažena nejnižší odchylka, proto tabulka zcela odpovídá grafu.

Pro jednotlivé skupiny funkcí se odlišují jejich vypočítané odchylky. Relativně vysoké odchylky pro skupinu základních funkcí F1-F9, a to především pro multimodální

funkce, které se zdají být v těchto kritériích snadné k vyřešení. Pro skupinu hybridních funkcí F10-F19 i kompozitních funkcí F20-F29 se odchylky zřetelně odlišují.

I v této iteraci testování docházelo k anomáliím. Anomálie, kde se prvek vyznačuje absolutní necitlivostí, se nacházela u funkcí F13 a F19. Druhá anomálie, kde se prvek vyznačuje extrémní citlivostí, se vyskytovala nejen u hybridních funkcí F14, F15, F17 a F19, ale i u kompozitních funkcí. Funkce F28 a F29 byly předurčeny k výskytu anomálie, protože se skládají kromě základních funkcí i z několika hybridních funkcí a vždy alespoň u jedné hybridní funkce se anomálie objevila.



Obrázek 21 Graf odchylek testovacích funkcí z CEC 2017, D=30

Seskupeným typům funkcí zůstávají velikosti jejich získaných odchylek stejné jako u D=10, i když v rámci více dimenzí je u některých benchmarkových funkcí, jako jsou F2, F12, F15, F17, F22, F26, F28 vyžadována vyšší přesnost než v D=10. Naopak u funkcí F5, F16, F25 je potřeba nižší přesnost.

U grafu pro D=30 je možné rozeznat nepatrné změny u jiné funkce, než tomu bylo u D=10. Odchylka se objevuje u funkce F17, u které byly pozorovány anomálie. U této funkce je potřeba větší přesnosti pro jazyk C než pro MATLAB.

Tabulka 20 Srovnání vyžadované přesnosti C/MATLAB CEC 2017, D=30

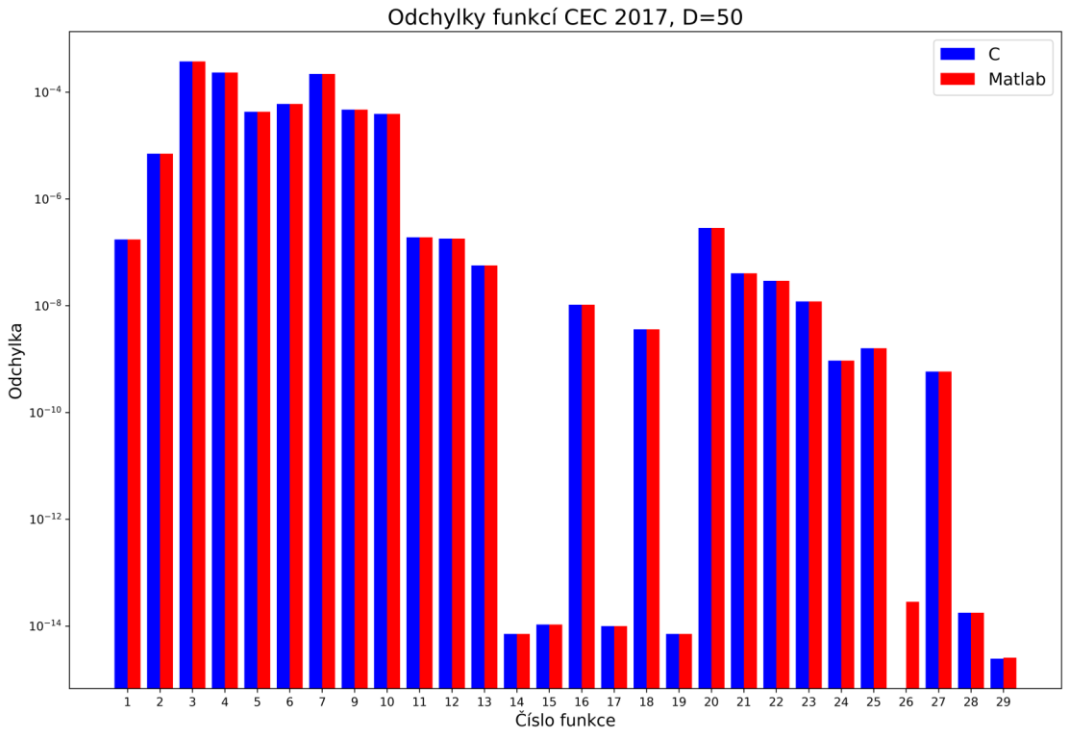
Tabulka 20 Srovnání vyžadované přesnosti C/MATLAB CEC 2017, D=30

Funkce	**C/MATLAB**
1	=30
2	=30
3	=30
4	=30
5	=30
6	=30
7	=30
9	=30
10	=30
11	=30
12	=30
13	=30
14	=30
15	=30
16	=30
17	+1/=29
18	=30
19	=30
20	=30
21	=30
22	=30
23	=30

24	=30
25	=30
26	+1/=29
27	=30
28	=29/-1

Z tabulky je patrné, že až na výjimky se přesnosti rovnají. U hybridní funkce F17 je potřeba vyšší přesnosti u jednoho elementu pro jazyk C, než je tomu pro MATLAB. U kompozitních funkcí F26 a F28 se vyžadovaná přesnost liší. Funkce F26 kopíruje v ohledu přesnosti jazyků funkci F17. Funkce F28 vyžaduje vyšší přesnost u jednoho prvku pro implementaci v MATLABu než pro implementaci v jazyce C. Toto je pozoruhodný poznatek, protože kompozitní funkce F28 se kromě jiných skládá i z hybridní funkce F17.

V tabulce se ukázalo, že použití jazyku C vyžaduje vyšší přesnost v porovnání s jazykem MATLAB.



Obrázek 22 Graf odchylek testovacích funkcí z CEC 2017, D=50

Pro skupiny funkcí zůstávají odchylky velmi podobné jako u D=30. Je tu jisté posunutí k větší přesnosti. To se dá pozorovat na logaritmické stupnici velikosti odchylky.

K zásadní změně ale došlo u funkce F26, kde se nedá určit řešení benchmarkové funkce jazyku C, protože již zmiňovaný výchozí vektor posunutí globálního optima se pohybuje mimo přípustnou odchylku. Výsledek funkce má být menší než 10^{-8}. Už u výchozí stavu je ale hodnota funkce větší než přípustná odchylka.

Zároveň vznikla odlišnost u funkce F29, kde jazyk C potřebuje vyšší přesnost než MATLAB. V porovnání grafu s grafem pro D=10 a D=30 je to třetí funkce, která má rozdílnou odchylku, pokud se nepočítá funkce F26.

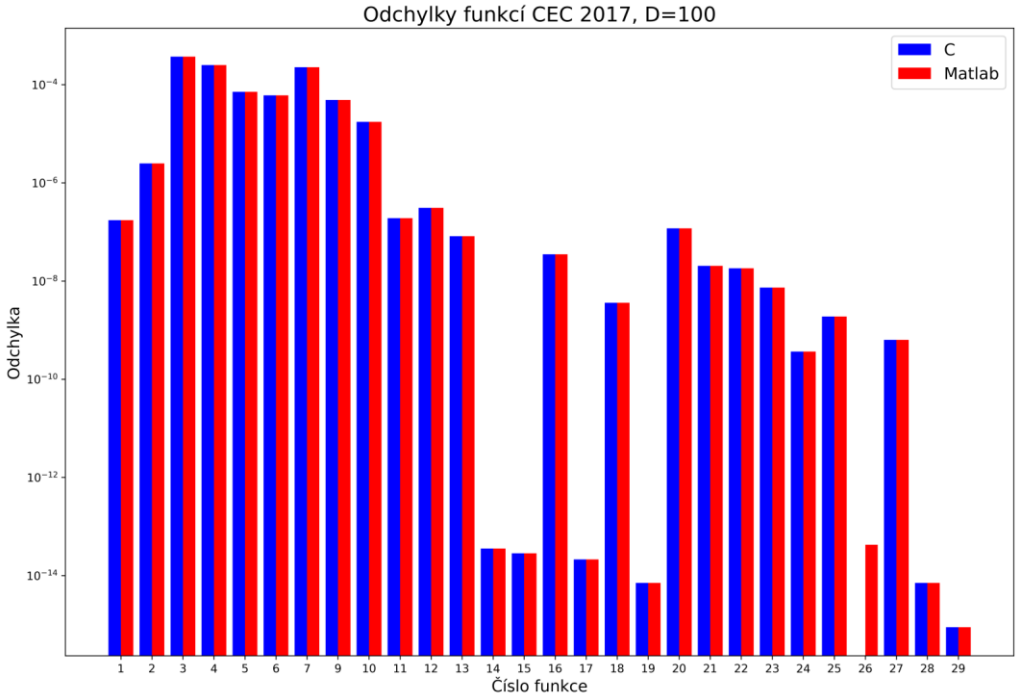
Tabulka 21 Srovnání vyžadované přesnosti C/MATLAB CEC 2017, D=50

Funkce	**C/MATLAB**
1	=50
2	=50
3	=50
4	=50
5	=50
6	=50
7	=50
9	=50
10	=50
11	=50
12	=50
13	=50
14	+27/=23
15	+30/=20
16	=50

17	+26/=24
18	=50
19	+14/=36
20	=50
21	=50
22	=50
23	=50
24	=50
25	=50
26	
27	=50
28	+24/=26
29	+3/=47

Je patrné, že dochází už k rozsáhlejším výkyvům přesností mezi jazyky C a MATLAB při D=50. Jazyk C vyžaduje vyšší přesnost u mnoha elementů u hybridních i kompozitních funkcí, u kterých docházelo k anomáliím. Ale hodnoty lišících se získaných odchylek pro jednotlivé prvky vektoru byly získány převážně z prvků, které netrpěly anomáliemi.

Pro účastníky soutěže tohle mohlo znamenat nesmírný problém. Pokud si zvolili jako implementační jazyk C, pak museli mnohokrát počítat na vyšší přesnost, než kdyby se rozhodli pro MATLAB. Zároveň účastníci soutěže nebyli schopni určit přesnou hodnotu funkce F26, proto na ukončení testování byli nuceni použít MaxFES.



Obrázek 23 Graf odchylek testovacích funkcí z CEC 2017, D=100

Oproti D=30 došlo ke stejným změnám jako u D=50. Posunutí přesnosti všech funkcí, které je viditelné díky větší logaritmické stupnici velikosti odchylky. U funkce F26 se nedá určit řešení benchmarkové funkce pro jazyk C.

U funkce F29 se už ovšem neobjevuje rozdíl mezi jazyky. Ovšem tento rozdíl se v grafu vyobrazuje jako minimální odchylka jednoho ze sta prvků vektoru, proto je potřeba srovnat vyžadované přesnosti i v tabulce.

Tabulka 22 Srovnání vyžadované přesnosti C/MATLAB CEC 2017, D=100

Funkce	C/MATLAB
1	=100
2	=100
3	=100
4	=100
5	=100
6	=100
7	=100

9	=100
10	=100
11	=100
12	=100
13	=100
14	+81/=19
15	+82/=18
16	=100
17	+80/=20
18	=100
19	+53/=47
20	=100
21	=100
22	=100
23	=100
24	=100
25	=100
26	
27	=100
28	+73/=26/-1
29	+2/=95/-3

Stejně jako u D=50, tak i u D=100 dochází ke značné odlišnosti přesností mezi jazyky C a MATLAB u prvků vektoru globálního optima. Nastává tomu tak u stejných benchmarkových funkcí jako u D=50 tj. F14, F15, F17, F19, F28 a F29, pokud se vynechá funkce F26.

Jazyk C viditelně potřebuje vyšší přesnost u většiny prvků, kde se rozdíly objevily. U F26 dochází k problému určení hodnoty funkce na sedm desetinných míst. To znovu vede k nerovnostem soutěže mezi jazykem MATLAB a C. Pro D=100 se jednoznačně vyplatí použít jazyk MATLAB.

3.3.2 Závěr testování CEC 2017

V rámci testovací sady funkcí CEC 2017 bylo 29 funkcí. Použity byly dva jazyky na jejich implementaci, a to MATLAB a C. Z testování je jasně patrné, že se přesnosti pro různé implementační jazyky odlišují. Pokud si účastník soutěže 2017 zvolí jako implementační jazyk testovacích funkcí C, pak je znevýhodněn na některých hybridních funkcích a některých kompozitních funkcích oproti těm, kteří se rozhodli pro MATLAB. Složitost přiblížení se optimu v C oproti MATLABu se zvyšováním počtu dimenzí netriviálně roste.

3.4 Závěry testování

Nastavení rovných podmínek pro všechny soutěžící je nezbytným kritériem pro všechny soutěže. Pokud je v soutěži zahrnuta výpočetní technika, tak to platí dvojnásob. Z testování CEC 2017 a CEC 2022 vyplynulo, že tomu tak u CEC zcela není. Účastník soutěže by si měl vybrat z identických implementací, které pouze představují jiný programovací jazyk. Mimo to naprostá nevědomost účastníků o jakékoliv výhodě vede ke znáhodnění a jistému znehodnocení výsledků soutěže.

Je potřeba zmínit, že s ohledem na poslední ročník byly podmínky soutěže přinejmenším zdokonaleny. Možnost vybrat si ze tří implementačních jazyků, které jsou si, co do přesnosti, velmi podobné ne-li stejné, je krok správným směrem. Taktéž se v CEC 2022 oproti CEC 2017 snížil počet dimenzí a funkcí, a tím i počet chyb spojených s různou mírou přesnosti.

ZÁVĚR

Bakalářská práce se věnovala analýze citlivosti implementace testovacích funkcí CEC benchmark setu. První část práce je teoretická a skládá se ze dvou kapitol. V první je popis CEC benchmarku, vznik CEC benchmarku a jednoúčelová optimalizace s omezením hranic. Druhá kapitola se zabývá veřejně dostupnými implementacemi CEC benchmarku a popisem jednotlivých ročníků. Nechybí doplnění souhrnu testovacích funkcí CEC benchmark setu z konkrétního roku i ukázky kódu testovacích funkcí.

Stěžejní část práce je praktická část, která je tvořena jednou kapitolou, která je rozdělena na podkapitoly. Praktická část je věnována testování CEC benchmark setu, konkrétně dvěma ročníkům, a to 2022 a 2017.

Cílem práce bylo zjistit, jak moc se se může pozměněná (vypočítaná) hodnota lišit od pozice optima v každé dimenzi, tak aby byl výstup z funkce stále optimem nebo byl v přípustné odchylce. Současně bylo nutné otestovat všechny implementace a vzájemně je mezi sebou porovnat.

K vyhodnocení výsledků testování byly použity grafy, v nichž byly odlišnými barvami znázorněny jednotlivé jazyky. Kromě grafů jsou součástí praktické části také tabulky, ve kterých jsou srovnány vyžadované přesnosti jednotlivých jazyků a další tabulka je výpisem vypočítaných odchylek hybridních funkcí.

V rámci testovací sady funkcí CEC 2022 bylo 12 funkcí. Použity byly tři jazyky na jejich implementaci, a to MATLAB, C a Python. Z testování vyplývá, že se přesnosti pro různé implementační jazyky odlišují. Z výsledků dále vyplývá, že znevýhodněn by byl soutěžící, který si zvolí jako implementační jazyk testovacích funkcí C, konkrétně na funkci F12, oproti soutěžícím, kteří by si vybrali MATLAB nebo Python.

V rámci testovací sady funkcí CEC 2017 bylo 29 funkcí. Použity byly dva jazyky na jejich implementaci, a to MATLAB a C. Z testování rovněž vyplývá, že se přesnosti pro různé implementační jazyky odlišují. I v tomto případě by byl znevýhodněn soutěžící, který by si zvolil jako implementační jazyk testovacích funkcí C, a to na některých hybridních funkcích a některých kompozitních funkcích. Složitost přiblížení se optimu v C oproti MATLABu se zvyšováním počtu dimenzí netriviálně roste.

Z výsledků testování ročníků CEC 2022 a CEC 2017 vyplývá, že v rámci soutěže nejsou nastaveny stejné podmínky pro všechny, což by mělo být základním kritériem jakékoli

soutěže. Kromě toto nevědomost účastníků o zvýhodnění a znevýhodnění během soutěže může vést k znehodnocení výsledků soutěže. Optimální by bylo, aby si účastník soutěže vybral z identických implementací, které pouze představují jiný programovací jazyk.

Na závěr je však potřeba dodat, že s ohledem na poslední ročník byly podmínky soutěže přinejmenším zdokonaleny. Už jen možnost výběru ze tří jazyků, které jsou si velmi podobné, co se týče přesnosti, je určitě správná úprava podmínek. Rovněž přispělo, že se v CEC 2022 snížil počet dimenzí a funkcí oproti CEC 2017, což logicky vede ke snížení počtu chyb.

SEZNAM POUŽITÉ LITERATURY

[1] GITHUB, P-N-SUGATHAN. *CEC2017-BoundConstrained*. Online. 2017. Dostupné z:https://github.com/P-N-Suganthan/CEC2017-BoundContrained/blob/master/Definitions%20of%20%20CEC2017%20benchmark%20suite%20final%20version%20updated.pdf. [cit. 2024-03-12].

[2] GITHUB, P-N-SUGATHAN. *CEC2017-BoundConstrained*. Online. 2017. Dostupné z: https://github.com/P-N-Suganthan/CEC2017-BoundContrained/blob/master/codes.rar. [cit. 2024-03-11].

[3] GITHUB, P-N-SUGATHAN. *CEC2019*. Online. 2019. Dostupné z: https://github.com/P-N-Suganthan/CEC2019/blob/master/Definitions%20of%20%20CEC2019%20Benchmark%20Sute.pdf. [cit. 2024-03-15].

[4] GITHUB, P-N-SUGATHAN. *CEC2019*. Online. 2019. Dostupné z: https://github.com/P-N-Suganthan/CEC2019/blob/master/100-Digit%20Challenge%20Analysis%20of%20Results.pdf. [cit. 2024-05-12].

[5] GITHUB, P-N-SUGATHAN. *2020-Bound-Constrained-Opt-Benchmark*. Online. 2020. Dostupné z: https://github.com/P-N-Suganthan/2020-Bound-Constrained-Opt-Benchmark/blob/master/Definitions%20of%20%20CEC2020%20benchmark%20suite%20Bound%20Constrained.pdf. [cit. 2024-03-10].

[6] GITHUB, P-N-SUGATHAN. *2020-Bound-Constrained-Opt-Benchmark*. Online. 2020. Dostupné z: https://github.com/P-N-Suganthan/2020-Bound-Constrained-Opt-Benchmark/blob/master/CEC2020%20BCC%20Results%20Analysis_R_Aug.16.pdf. [cit. 2024-03-12].

[7] GITHUB, P-N-SUGATHAN. *2021-SO-BCO*. Online. 2021. Dostupné z: https://github.com/P-N-Suganthan/2021-SO-BCO/blob/main/CEC2021%20TR_final%20(1).pdf. [cit. 2024-03-12].

[8] GITHUB, P-N-SUGATHAN. *2021-SO-BCO*. Online. 2021. Dostupné z: https://github.com/P-N-Suganthan/2021-SO-BCO/blob/main/CEC2021%20Bound-Constrained%20Optimization%20Results%20Analysis.pdf. [cit. 2024-03-22].

[9] GITHUB, P-N-SUGATHAN. *2022-SO-BO*. Online. 2022. Dostupné z: https://github.com/P-N-Suganthan/2022-SO-BO/blob/main/CEC2022%20TR.pdf. [cit. 2024-03-05].

[10] GITHUB, P-N-SUGATHAN. *2022-SO-BO*. Online. 2022. Dostupné z: https://github.com/P-N-Suganthan/2022-SO-BO/blob/main/2022-Full-Download.zip. [cit. 2024-03-05].

[11] GITHUB, P-N-SUGANTHAN. *CEC2014*. Online. 2014. Dostupné z: https://github.com/P-N-Sugan-than/CEC2014/blob/master/Definitions%20of%20%20CEC2014%20benchmark%20suite%20Part%20A.pdf. [cit. 2024-03-03].

[12] GITHUB, P-N-SUGANTHAN. *CEC2005*. Online. 2005. Dostupné z: https://github.com/P-N-Suganthan/CEC2005/blob/master/Tech-Report-May-30-05.pdf. [cit. 2024-02-28].

[13] SUGANTHAN, P.N.; LIANG, J.J.; QU, B.Y. a HERNÁNDEZ-DÍAZ, Alfredo G. *Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization*. Online. 2013. Dostupné z: https://al-roomi.org/multime-dia/CEC_Database/CEC2013/RealParameterOptimization/CEC2013_RealParameterOptimization_TechnicalReport.pdf#:~:text=URL%3A%20https%3A%2F%2Fal. [cit. 2024-03-02].

[14] CARDOSO, João M.P.; COUTINHO, José Gabriel F. a DINIZ, Pedro C. *Embedded Computing for High Performance*. Online. Copyright © 2017 Elsevier Inc. All rights reserved., 2017. ISBN 978-0-12-804189-5. Dostupné z: https://doi.org/https://doi.org/10.1016/C2015-0-00283-0. [cit. 2024-02-25].

[15] *The 2003 Congress on Evolutionary Computation CEC 2003*. Online. 2003. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1299366&tag=1. [cit. 2024-02-24].

[16] MOLINA, D.; LATORRE, A. a HERRERA, F. *An Insight into Bio-inspired and Evolutionary Algorithms for Global Optimization: Review, Analysis, and Lessons Learnt over a Decade of Competitions*. Online. 2018, roč. 2018, č. 4. 2018. ISSN 4-

1866-9964. Dostupné z: https://link.springer.com/article/10.1007/s12559-018-9554-0. [cit. 2024-02-25].

[17] *DETAILED PROGRAM*. Online. 2011. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5949583. [cit. 2024-02-12].

[18] GITHUB, P-N-SUGANTHAN. CEC2005. Online. 2005. Dostupné z: https://github.com/P-N-Suganthan/CEC2005/blob/master/C-windows-17-Mar-05.zip. [cit. 2024-03-09].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

CEC	Congress on Evolutionary Computation
D	Dimension(s) - počet dimenzí
MaxFES	Maximal functional evaluation
SIAM	Society for Industrial and Applied Mathematics

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

P1: Seznam s odkazy na repozitáře

PŘÍLOHA P I: SEZNAM S ODKAZY NA REPOZITÁŘE

Testování:

CEC 2017 – jazyk C (https://github.com/FrostNiles/CEC-benchmark-testing-2017-C)

CEC 2017 – jazyk MATLAB (https://github.com/FrostNiles/cec-2017-mat)

CEC 2022 – jazyk C (https://github.com/FrostNiles/cec-2022-testing-C)

CEC 2022 – jazyk MATLAB (https://github.com/FrostNiles/benchmark-mat-2022)

CEC 2022 – jazyk Python (https://github.com/FrostNiles/cec-2022-testing-python)

Výsledky (tabulky + grafy):

CEC 2017 – (https://github.com/FrostNiles/CEC-benchmark-testing-2017-C/tree/main/test_data/graph-results/

CEC 2017 – (https://github.com/FrostNiles/CEC-benchmark-testing-2017-C/tree/main/test_data/table-results/)

CEC 2022 - (https://github.com/FrostNiles/cec-2022-testing-C/test_data/graph-results/)

CEC 2022 - (https://github.com/FrostNiles/cec-2022-testing-C/test_data/table-results/)

Výsledky:

V jednotlivých repozitářích ve složce test_data/result/