

Experimentální vyhodnocení variant binární reprezentace v genetickém algoritmu

Bc. Jan Nedbal



ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Bc. Jan Nedbal
Osobní číslo: A23324
Studijní program: N0613A140022 Informační technologie
Specializace: Softwarové inženýrství
Forma studia: Prezenční
Téma práce: Experimentální vyhodnocení variant binární reprezentace v genetickém algoritmu
Téma práce anglicky: Experimental Evaluation of Binary Representation Variants in Genetic Algorithm

Zásady pro vypracování

- Vytvořte přehled metod binární reprezentace reálných hodnot v genetických algoritmech
- Zvolte vhodné testovací funkce pro experiment
- Otestujte a statisticky vyhodnoťte efektivitu různých binárních reprezentací
- Analyzujte výsledky a formulujte závěry
- Formulujte doporučení pro optimalizaci problémů v reálné doméně pomocí genetických algoritmů

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

Seznam doporučené literatury:

1. KRAMER, Oliver. Genetic algorithm essentials. Studies in computational intelligence. Cham, Switzerland: Springer, 2017. Dostupné z: <https://doi.org/9783319521565>.
2. DAVIS, Lawrence (ed.). Handbook of Genetic Algorithms. London: ITP, 1991. ISBN 1850328250.
3. HAUPT, Randy L. a HAUPT, S. E. Practical genetic algorithms. 2nd ed. Hoboken, N.J: John Wiley, c2004. ISBN 0471455652.
4. BÄCK, Thomas. Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. New York: Oxford University Press, 1996. ISBN 0195099710.

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

Datum zadání diplomové práce: **27. října 2024**

Termín odevzdání diplomové práce: **2. června 2025**



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 17. listopadu 2024

PROHLÁŠENÍ AUTORA DIPLOMOVÉ PRÁCE

Beru na vědomí, že

- odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení diplomové práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji diplomovou 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;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – diplomovou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- pokud je výstupem diplomové 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 diplomové práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato diplomová práce se zabývá problematikou reprezentace reálných čísel v genetických algoritmech. Hlavním cílem je experimentální srovnání výkonnosti různých přístupů k binárnímu kódování reálných čísel: standardního binárního kódu s pevným bodem a reprezentace dle standardu IEEE 754. Dále je zkoumán vliv následné aplikace Grayova kódování na binární reprezentaci s pevným bodem – techniky, která upravuje binární řetězec tak, aby kódy sousedních diskrétních hodnot měly Hammingovu vzdálenost 1. Tyto binární varianty jsou porovnávány mezi sebou a s referenčním genetickým algoritmem pracujícím přímo s reálnými hodnotami. Experimenty jsou provedeny na sadě komplexních testovacích funkcí na instanci benchmarku generovanou pomocí GNBG. Hodnotí se především dosažená přesnost nalezených řešení, rychlost konvergence a robustnost jednotlivých přístupů. Výsledky poskytují vhled do vlivu volby reprezentace a případných následných kódovacích úprav na efektivitu genetického algoritmu při řešení numerických optimalizačních úloh.

Klíčová slova: genetické algoritmy, binární reprezentace, optimalizace reálných hodnot, Grayův kód, IEEE 754, experimentální srovnání

ABSTRACT

This Master's thesis deals with the issue of real number representation in genetic algorithms. The main goal is an experimental comparison of the performance of different approaches to the binary encoding of real numbers: standard fixed-point binary code and representation according to the IEEE 754 standard. Furthermore, the influence of the subsequent application of Gray coding to the fixed-point binary representation is examined – a technique that modifies the binary string so that the codes of adjacent discrete values have a Hamming distance of 1. These binary variants are compared with each other and with a reference genetic algorithm working directly with real values. Experiments are conducted on a set of complex test functions on a benchmark instance generated using GNBG. The evaluation primarily focuses on the achieved accuracy of the found solutions, convergence speed, and the robustness of the individual approaches. The results provide insight into the influence of the choice of representation and potential subsequent coding modifications on the efficiency of the genetic

algorithm when solving numerical optimization problems.

Keywords: genetic algorithms, binary representation, real-value optimization, Gray code, IEEE 754, experimental comparison

Chtěl bych poděkovat mému vedoucímu panu Ing. Adamovi Viktorínovi, Ph.D. za ochotu mi vždy pomoci a spolupracovat. Také bych chtěl poděkovat své rodině za klidné prostředí. Bez vaší podpory by tato práce nevznikla.

OBSAH

ÚVOD	15
I TEORETICKÁ ČÁST	17
1 GENETICKÉ ALGORITMY	18
1.1 ZÁKLADNÍ PRINCIPY GENETICKÝCH ALGORITMŮ	18
1.1.1 Populace kandidátních řešení a dynamika evoluce	18
1.1.2 Hodnocení kvality řešení – Funkce přizpůsobení	19
1.1.3 Mechanismy selekce, křížení a mutace	19
1.1.4 Evoluční inspirace a adaptivní povaha	26
1.2 APLIKACE GENETICKÝCH ALGORITMŮ	26
1.2.1 Optimalizace v inženýrských disciplínách	27
1.2.2 Strojové učení a analýza dat	27
1.2.3 Návrh a optimalizace sítí	28
1.2.4 Finanční modelování a správa portfolia	28
1.3 IMPLEMENTAČNÍ ASPEKTY A STRATEGICKÁ DOPORUČENÍ	29
1.3.1 Architektura a softwarová implementace GA	29
1.3.2 Konfigurace parametrů a ladění optimalizačních strategií	29
2 BINÁRNÍ REPREZENTACE REÁLNÝCH HODNOT	31
2.1 STANDARDNÍ BINÁRNÍ KÓD	31
2.1.1 Převod s pevným bodem (Fixed-Point Conversion)	31
2.1.2 Binary-Coded Decimal	33
2.2 GRAYŮV KÓD	33
2.3 REPREZENTACE FLOATING-POINT DLE IEEE 754.....	34
2.3.1 Formát IEEE 754.....	34
2.4 DALŠÍ METODY REPREZENTACE.....	35
2.4.1 Real-Valued GA	35
2.4.2 Hybridní metody	36
3 OPERÁTORY GENETICKÉHO ALGORITMU	37
3.1 MUTACE V BINÁRNÍ REPREZENTACI	37

3.1.1	Bit-flip mutace	37
3.1.2	Dopad na vyhledávání řešení	37
3.2	KŘÍŽENÍ V BINÁRNÍ REPREZENTACI	38
3.2.1	Jednobodové křížení.....	38
3.2.2	Dvoubodové a uniformní křížení	38
3.3	MUTACE V REÁLNÉ REPREZENTACI	38
3.3.1	Aditivní mutace.....	38
3.3.2	Gaussovská mutace	39
3.4	KŘÍŽENÍ V REÁLNÉ REPREZENTACI	39
3.4.1	Aritmetické křížení	39
3.4.2	BLX- α křížení.....	39
4	TESTOVACÍ FUNKCE PRO GENETICKÉ ALGORITMY	41
4.1	KRITÉRIA PRO VÝBĚR TESTOVACÍCH FUNKCÍ.....	41
4.1.1	Multimodalita	41
4.1.2	Separabilita	41
4.1.3	Škálovatelnost.....	42
4.2	PŘEHLED A VLASTNOSTI TESTOVACÍCH FUNKCÍ.....	42
4.2.1	Unimodální funkce	42
4.2.2	Multimodální funkce	42
4.2.3	Funkce s neseparabilními proměnnými	43
4.3	PŘÍKLADY GENEROVANÝCH TESTOVACÍCH SAD A JEJICH VYUŽITÍ.....	43
4.3.1	Vlastnosti testovacích funkcí	43
4.3.2	Multimodalita, separabilita a škálovatelnost v praxi	43
5	METODY VYHODNOCENÍ EFEKTIVITY BINÁRNÍ REPREZENTACE	45
5.1	HAMMINGOVA VZDÁLENOST MEZI SOUSEDNÍMI HODNOTAMI	45
5.1.1	Definice a význam	45
5.1.2	Vliv na průběh optimalizace	46
5.2	ANALÝZA VLIVU MUTACE NA FENOTYP	47
5.2.1	Role mutace v genetických algoritmech	47
5.2.2	Vliv mutace u standardního binárního kódování	47

5.2.3	Vliv mutace u BCD reprezentace	47
5.2.4	Zacházení s neplatnými BCD kódy	48
5.3	REDUNDANCE KÓDOVÁNÍ	49
5.3.1	Definice a výskyt redundance	49
5.3.2	Redundance v BCD kódování	49
5.4	LOKALITA REPREZENTACE	50
5.4.1	Význam lokality v genetických algoritmech	50
5.4.2	Lokalita standardního binárního kódování	50
5.4.3	Lokalita BCD reprezentace	50
5.4.4	Důsledky lokality pro genetické operátory	51
5.5	METRIKY ÚSPĚŠNOSTI OPTIMALIZACE	51
5.6	STATISTICKÉ TESTY POUŽITÉ PRO ANALÝZU	52
II	PRAKTICKÁ ČÁST	54
6	IMPLEMENTACE EXPERIMENTU	55
6.1	POPIS SOFTWARE A POUŽITÝCH NÁSTROJŮ	55
6.2	NASTAVENÍ PARAMETRŮ EXPERIMENTU	57
6.3	METODIKA NASTAVENÍ DÉLKY CHROMOZOMU	58
6.4	IMPLEMENTACE RŮZNÝCH BINÁRNÍCH REPREZENTACÍ	58
6.4.1	Fixed-Point	58
6.4.2	Grayův kód	59
6.4.3	IEEE-754	59
6.4.4	Real Value	60
7	PROVEDENÍ EXPERIMENTU	61
7.1	PROCES TESTOVÁNÍ JEDNOTLIVÝCH VARIANT	61
8	VYHODNOCENÍ VÝSLEDKŮ	66
8.1	KONVERGENČNÍ CHARAKTERISTIKY PRO BĚH NA TŘECH FUNKCÍCH	66
8.1.1	Srovnání pro jednotlivé běhy	66
8.1.2	Srovnání variant mezi sebou	69
8.2	VÝPOČETNÍ NÁROČNOST RŮZNÝCH REPREZENTACÍ	72

8.3	GRAFICKÉ ZOBRAZENÍ FINAL ERROR PRO VŠECHNY FUNKCE.....	73
8.4	STATICKÁ ANALÝZA	78
8.5	VÝSLEDKY STATISTICKÉHO POROVNÁNÍ (FRIEDMANŮV TEST A NEMENYIHO POST-HOC)	84
9	DISKUZE A DOPORUČENÍ	87
9.1	ZHODNOCENÍ METOD.....	87
9.1.1	Zhodnocení experimentu s konvergencemi	87
9.1.2	Zhodnocení hlavního experimentu	88
9.2	DOPORUČENÍ PRO PRAKTICKÉ POUŽITÍ.....	89
9.3	NÁVRHY NA BUDOUCÍ VÝZKUM.....	90
	ZÁVĚR.....	92
	SEZNAM POUŽITÉ LITERATURY	95
	SEZNAM OBRÁZKŮ	97
	SEZNAM TABULEK	99
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	100
	SEZNAM PŘÍLOH.....	101

ÚVOD

Jedním z klíčových aspektů návrhu genetického algoritmu pro řešení úloh s reálnými proměnnými je volba vhodné reprezentace těchto proměnných v rámci genetického kódu jedinců populace. Přestože existují varianty GA pracující přímo s reálnými hodnotami (real-valued GA), tradičně se často využívají binární reprezentace. Způsob, jakým jsou reálná čísla zakódována do binárních řetězců, však může zásadně ovlivnit chování algoritmu, zejména efektivitu genetických operátorů, jako je křížení a mutace, a celkovou schopnost algoritmu konvergovat ke kvalitnímu řešení. Nevhodná reprezentace může vést k problémům, jako je například velká Hammingova vzdálenost mezi kódy číselně blízkých hodnot, což narušuje princip lokálního prohledávání.

Tato diplomová práce se proto zaměřuje na experimentální vyhodnocení a srovnání různých variant binárního kódování reálných čísel v kontextu genetických algoritmů. Konkrétně jsou zkoumány a porovnávány následující přístupy: standardní binární kódování s pevným bodem (fixed-point), Grayův kód, který je navržen tak, aby minimalizoval Hammingovu vzdálenost mezi sousedními hodnotami, a přímé kódování pomocí standardu IEEE 754 pro plovoucí řádovou čárku. Jako referenční metoda pro srovnání slouží genetický algoritmus pracující přímo s reálnými hodnotami.

Cílem práce je systematicky prozkoumat, jak volba binární reprezentace ovlivňuje výkonnost genetického algoritmu při řešení standardizovaných testovacích úloh. Zaměřujeme se především na rychlost konvergence, dosaženou přesnost nalezeného řešení a robustnost algoritmu. Pro experimentální vyhodnocení byla použita sada náročných testovacích funkcí generovaných pomocí nástroje GNBG, který umožňuje vytvářet funkce s definovatelnými vlastnostmi, jako je multimodalita, separabilita a škálovatelnost.

Práce je strukturována následovně: Teoretická část nejprve představuje základní principy genetických algoritmů, jejich klíčové komponenty (populace, fitness funkce, selekce, křížení, mutace) a oblasti jejich aplikací. Následně se podrobně věnuje problematice reprezentace reálných hodnot, kde jsou popsány a analyzovány jednotlivé zkoumané binární metody i principy relevantních genetických operátorů pro binární a reálnou reprezentaci. Závěr teoretické části je věnován testovacím funkcím a metodám hodnocení efektivit reprezentací. Praktická část detailně popisuje návrh a implementaci experimentálního prostředí, použité

nástroje, nastavení parametrů, metodiku provedení experimentů a způsob sběru a analýzy dat, včetně statistického vyhodnocení. V závěru práce jsou shrnuty dosažené výsledky, diskutovány jejich implikace a navrženy směry pro další výzkum.

I TEORETICKÁ ČÁST

1 GENETICKÉ ALGORITMY

Tato kapitola se věnuje komplexnímu přehledu evolučních výpočetních technik, přičemž klade zvláštní důraz na GA. Tyto algoritmy, inspirované principy biologické evoluce [1], představují výkonnou třídu stochastických optimalizačních heuristik, vhodných pro řešení širokého spektra složitých problémů, zejména těch, kde tradiční deterministické metody selhávají nebo jsou neefektivní [2]. Následující text detailně rozebírá teoretické základy GA, jejich fundamentální komponenty a mechanismy, jakož i jejich rozmanité praktické aplikace.

1.1 Základní principy genetických algoritmů

Tato podkapitola se soustředí na detailní analýzu základních stavebních prvků genetických algoritmů a jejich synergického působení v rámci evolučního procesu. Struktura výkladu je členěna do podsekcí, které systematicky pokrývají jednotlivé aspekty fungování GA.

1.1.1 Populace kandidátních řešení a dynamika evoluce

Populace řešení: Jádrem genetického algoritmu je *populace* – množina *jedinců*, kde každý jedinec kóduje jedno potenciální (kandidátní) řešení zkoumaného optimalizačního problému [2]. Genetická diverzita v rámci populace je esenciálním faktorem, neboť umožňuje algoritmu prohledávat rozsáhlý prostor řešení a snižuje riziko předčasné konvergence k suboptimálním řešením [3]. Evoluční proces je typicky iniciován vytvořením počáteční populace, což může být provedeno náhodně, pomocí specifických heuristik, nebo kombinací obou přístupů. Udržení dostatečné variability populace napříč generacemi je klíčové pro úspěšné nalezení globálního optima [4].

Evoluční dynamika: Principy řídící evoluci populace v GA jsou přímou analogií mechanismů přírodního výběru a genetiky, jak je popsal Darwin a jak jsou formalizovány v evoluční biologii. Populace prochází iterativním procesem generační obměny. V každé generaci jsou jedinci hodnoceni pomocí účelové funkce, následně probíhá selekce jedinců pro reprodukci a aplikace genetických operátorů – typicky *křížení* (*crossover*) a *mutace* (*mutation*) – pro generování nových jedinců (potomků) [2, 5]. Tento cyklický proces vede k postupnému zlepšování průměrné i nejlepší kvality řešení v populaci. Pro vizualizaci této dynamiky je vhodné použít grafy zobrazující vývoj fitness v závislosti na generaci (např. “Generation vs. Average/Best Fitness”).

1.1.2 Hodnocení kvality řešení – Funkce přizpůsobení

Funkce přizpůsobení (fitness function): Centrální komponentou, která řídí směr evoluce, je *funkce přizpůsobení* (často označovaná jako fitness funkce). Jejím úkolem je kvantifikovat kvalitu (vhodnost) každého jedince v populaci ve vztahu k cíli optimalizačního problému [3]. Návrh této funkce je kritický a musí přesně odrážet specifika dané úlohy. Kvalitně navržená fitness funkce umožňuje efektivní diferenciaci mezi kvalitnějšími a méně kvalitními řešeními, čímž poskytuje selekční tlak potřebný pro evoluční proces. Je žádoucí, aby byla funkce výpočetně efektivní a poskytovala dostatečný gradient pro vedení prohledávání.

1.1.3 Mechanismy selekce, křížení a mutace

Mechanismus výběru (selection): Selektce je proces, jehož prostřednictvím jsou jedinci vybíráni z aktuální populace, aby se stali rodiči pro následující generaci. Tento mechanismus napodobuje princip "přežití nejschopnějších" známý z přírody. Jedinci s vyšší hodnotou fitness mají vyšší pravděpodobnost, že budou vybráni a předají svůj genetický materiál dále [2]. Existuje řada selekčních strategií, jako je selektce ruletovým kolem (roulette wheel selection), turnajová selektce (tournament selection) nebo selektce založená na pořadí (rank-based selection). Volba konkrétní metody ovlivňuje selekční tlak a balancuje mezi *explorací* (prohledáváním nových oblastí prostoru řešení) a *exploatací* (zdokonalováním již nalezených dobrých řešení) [3].

Mějme náhodnou populaci složenou z těchto jedinců:

```
Populace:  
Jedinec 1: [1, 0, 1, 1, 0, 0, 1, 0], Fitness: 4  
Jedinec 2: [0, 1, 0, 0, 1, 1, 0, 1], Fitness: 4  
Jedinec 3: [1, 1, 1, 0, 1, 0, 1, 1], Fitness: 6  
Jedinec 4: [0, 0, 1, 1, 0, 1, 1, 0], Fitness: 4
```

Obrázek 1.1 Náhodná populace pro demonstraci mechanismů výběru

(zdroj: vlastní program)

Pro následující příklad použijeme fitness funkci, která bude počítat počet 1.

Selektce ruletovým kolem (Roulette Wheel Selection): Princip: Tato metoda je inspirována ruletou, kde každému jedinci v populaci je přiřazena "výseč" na ruletě. Velikost této výseče je úměrná jeho fitness hodnotě. Čím vyšší má jedinec fitness, tím větší má výseč a tím pádem

i větší pravděpodobnost, že bude vybrán.

Postup:

- Vypočítá se fitness hodnota pro každého jedince v populaci.
- Vypočítá se celková fitness populace (součet fitness hodnot všech jedinců).
- Pro každého jedince se vypočítá relativní fitness, což je jeho fitness hodnota dělená celkovou fitness. Relativní fitness představuje pravděpodobnost výběru daného jedince.
- Simuluje se ruleta: vygeneruje se náhodné číslo z intervalu $[0, 1]$.
- Postupně se procházejí jedinci a sčítají se jejich relativní fitness hodnoty. Jakmile akumulovaná hodnota překročí vygenerované náhodné číslo, jedinec, u kterého k tomu došlo, je vybrán.

Výhody: Jednoduchá implementace, silnější jedinci mají větší šanci na výběr.

Nevýhody: Slabší jedinci mohou být úplně vyřazeni, což může vést ke ztrátě genetické diverzity.

```
Selekce ruletou:  
Pravděpodobnosti výběru (ruleta):  
- Jedinec 1: 0.222  
- Jedinec 2: 0.222  
- Jedinec 3: 0.333  
- Jedinec 4: 0.222  
Náhodné číslo: 0.975  
Vybraný jedinec: Jedinec 4, Fitness: 4  
Vybraný jedinec: [0, 0, 1, 1, 0, 1, 1, 0]
```

Obrázek 1.2 Demonstrace ruletovým výběrem z populací z obrázku 1.1

(zdroj: vlastní program)

Turnajová selekce (Tournament Selection): Princip: V této metodě se z populace náhodně vybere určitý počet jedinců (tzv. velikost turnaje). Z této skupiny se pak vybere jedinec s nejvyšší fitness hodnotou, který se stane rodičem pro další generaci. Tento proces se opakuje, dokud není vybrán dostatečný počet rodičů.

Postup:

- Z populace se náhodně vybere k jedinců (kde k je velikost turnaje).
- Z vybraných k jedinců se vybere ten s nejvyšší fitness hodnotou.
- Tento jedinec se stane rodičem.
- Kroky 1-3 se opakují, dokud není vybrán dostatečný počet rodičů pro vytvoření nové populace.

Výhody: Jednoduchá implementace, snadné nastavení selekčního tlaku (čím větší turnaj, tím silnější selekční tlak).

Nevýhody: Může být výpočetně náročnější než ruleta, pokud je velikost turnaje velká.

```
Turnajová selekce (velikost turnaje=3):  
Účastníci turnaje:  
  - Jedinec 3, Fitness: 6  
  - Jedinec 2, Fitness: 4  
  - Jedinec 1, Fitness: 4  
Vítěz turnaje: Jedinec 3, Fitness: 6  
Vybraný jedinec: [1, 1, 1, 0, 1, 0, 1, 1]
```

Obrázek 1.3 Demonstrace
turnajovým výběrem z populací z
obrázku 1.1

(zdroj: vlastní program)

Selekce založená na pořadí (Rank-Based Selection): Princip: V této metodě se jedinci v populaci seřadí podle jejich fitness hodnoty. Poté se každému jedinci přiřadí pravděpodobnost výběru na základě jeho pořadí v tomto seřazeném seznamu. Jedinci s lepším pořadím (vyšší fitness) mají vyšší pravděpodobnost výběru.

Postup:

- Populace se seřadí podle fitness hodnot.
- Každému jedinci se přiřadí pořadí (např. od 1 pro nejlepšího po N pro nejhoršího, kde N je velikost populace).
- Vypočítají se pravděpodobnosti výběru na základě pořadí. Existuje několik způsobů, jak to udělat, např.:
 - Lineární pořadí: Pravděpodobnost se lineárně zvyšuje s pořadím.

– Exponenciální pořadí: Pravděpodobnost se exponenciálně zvyšuje s pořadím.

- Jedinci se vybírají na základě těchto pravděpodobností (např. pomocí rulety).

Výhody: Zabraňuje předčasné konvergenci, protože i slabší jedinci mají šanci být vybráni. Selekční tlak se dá snadno nastavit.

Nevýhody: Může být výpočetně náročnější než ruleta kvůli nutnosti řazení.

```
Selekce podle pořadí:
Pořadí jedinců (od nejlepšího):
- Jedinec 1: Fitness: 4, Pravděpodobnost: 0.250
- Jedinec 2: Fitness: 4, Pravděpodobnost: 0.500
- Jedinec 4: Fitness: 4, Pravděpodobnost: 0.750
- Jedinec 3: Fitness: 6, Pravděpodobnost: 1.000
Náhodné číslo: 0.114
Vybraný jedinec: Jedinec 1, Fitness: 4
Vybraný jedinec: [1, 0, 1, 1, 0, 0, 1, 0]
```

Obrázek 1.4 Demonstrace výběrem
podle pořadí z populací z obrázku
1.1

(zdroj: vlastní program)

Operátor křížení (crossover): Křížení je hlavním operátorem zodpovědným za exploraci. Simuluje biologickou rekombinaci genetického materiálu mezi dvěma (nebo více) rodičovskými jedinci za účelem vytvoření jednoho či více potomků. Cílem je kombinovat potenciálně přínosné části rodičovských řešení a generovat tak nové, potenciálně lepší, kandidáty [2, 6]. Běžně používané metody zahrnují jednobodové, dvoubodové, vícebodové či uniformní křížení, jejichž volba závisí na reprezentaci jedinců a povaze problému. Schématické diagramy ilustrující výměnu segmentů chromozomů mezi rodiči mohou významně přispět k pochopení tohoto procesu.

Pro první generaci si dovolím tento jednoduchý princip vysvětlit: Generace 1: Obrázek ukazuje výpočet v první generaci evolučního algoritmu. Nejlepší fitness: 6: V této generaci má nejlepší jedinec (chromozom) hodnotu fitness 6. Fitness je míra kvality řešení; v tomto případě odpovídá počtu jedniček v chromozomu. Rodič 1: 11110101, Rodič 2: 11000101: První pár rodičů má chromozomy "11110101" a "11000101". (Křížení se neprovedlo): Pro tento pár rodičů se operace křížení neprovedla. To je typické, protože křížení se aplikuje s určitou pravděpodobností (ne vždy). Jelikož je výstup nastavený pro výpis hodnot, kde křížení proběhlo, tak se neukazuje jeho výstup. Rodič 1: 01101001, Rodič 2: 10110001:

```
Generace 1
  Nejlepší fitness: 6
  Rodič 1:  11110101
  Rodič 2:  11000101
  (Křížení se neprovedlo)
-----
  Rodič 1:  01101001
  Rodič 2:  10110001
              ^
  Potomek 1: 00110001
  Potomek 2: 11101001
-----
```

Obrázek 1.5 Výstup z jednoduchého genetického algoritmu, který konkrétně ilustruje proces křížení (crossover) binárních chromozomů v první generaci

(zdroj: vlastní program)

Druhý pár rodičů má chromozomy "01101001" a "10110001". Symbol stříšky ukazuje místo (bod) křížení na chromozomech rodičů. V tomto případě došlo ke křížení po druhém bitu. Potomek 1: 00110001, Potomek 2: 11101001: Výsledné chromozomy potomků po křížení. Potomek 1 získal první dva bity od Rodiče 1 a zbytek od Rodiče 2. Potomek 2 získal první dva bity od Rodiče 2 a zbytek od Rodiče 1.

Generace 2	Generace 3
Nejlepší fitness: 6	Nejlepší fitness: 6
Rodič 1: 11101001	Rodič 1: 11110001
Rodič 2: 11110101	Rodič 2: 00101001
^	^
Potomek 1: 11101001	Potomek 1: 11110001
Potomek 2: 11110101	Potomek 2: 00101001
-----	-----
Rodič 1: 11101001	Rodič 1: 11110101
Rodič 2: 00110001	Rodič 2: 11110101
^	^
Potomek 1: 11110001	Potomek 1: 11110101
Potomek 2: 00101001	Potomek 2: 11110101
-----	-----

Obrázek 1.6 Výstup z jednoduchého genetického algoritmu, který konkrétně ilustruje proces křížení (crossover) binárních chromozomů v druhé generaci

(zdroj: vlastní program)

Obrázek 1.7 Výstup z jednoduchého genetického algoritmu, který konkrétně ilustruje proces křížení (crossover) binárních chromozomů v třetí generaci

(zdroj: vlastní program)

Operátor mutace (mutation): Mutace zavádí do genetického materiálu jedince malé, náhodné změny. Tento operátor hraje klíčovou roli v udržování genetické diverzity populace a zabraňuje předčasné konvergenci algoritmu [2]. Mutace umožňuje algoritmu opustit lokální optima a prozkoumat nové oblasti prostoru řešení. Forma mutace závisí na kódování řešení; příklady zahrnují bit-flip mutaci pro binární řetězce nebo přidání malého náhodného šumu k reálným parametrům. Pravděpodobnost mutace je obvykle nízká, aby nedošlo k narušení již nalezených kvalitních řešení. Grafické znázornění účinku mutace na jedince může pomoci ilustrovat její roli při obnovování ztracené genetické informace.

```
Generace 1
Nejlepší fitness: 7
Původní: 10110011
| | | | | ^
Mutovaný: 10110001
-----
Původní: 11111011
| | | | | ^ ^
Mutovaný: 01111010
-----
Původní: 11111011
(Žádná mutace)
-----
Původní: 11111011
| | | | | ^
Mutovaný: 10111011
-----
```

Obrázek 1.8 Výstup z jednoduchého genetického algoritmu, který konkrétně ilustruje proces mutace (mutation) binárních chromozomů v první generaci

(zdroj: vlastní program)

Původní: Chromozom před mutací. ^ Symbol stříšky označuje pozici (bod) mutace. Mutovaný: Chromozom po mutaci.

Popis mutací u jednotlivých jedinců: Jedinec 1: Původní: 10110011 Mutovaný: 10110001 (Mutace na 7. pozici) Jedinec 2: Původní: 11111011 Mutovaný: 01111010 (Mutace na 0. a 7. pozici) Jedinec 3: Původní: 11111011 (Žádná mutace): V tomto případě k mutaci nedošlo. Jedinec 4: Původní: 11111011 Mutovaný: 10111011 (Mutace na 0. pozici)

Generace 2	Generace 3
Nejlepší fitness: 7	Nejlepší fitness: 7
Původní: 11111011	Původní: 11111011
(Žádná mutace)	^ ^
-----	Mutovaný: 11010011
Původní: 11111011	-----
^	Původní: 11110011
Mutovaný: 11110011	(Žádná mutace)
-----	-----
Původní: 01111010	Původní: 01111011
^	^
Mutovaný: 01111011	Mutovaný: 01101011
-----	-----
Původní: 10111011	Původní: 00101011
^ ^	^
Mutovaný: 00101011	Mutovaný: 00001011
-----	-----

Obrázek 1.9 Výstup z jednoduchého genetického algoritmu, který konkrétně ilustruje proces mutace (mutation) binárních chromozomů v druhé generaci

(zdroj: vlastní program)

Obrázek 1.10 Výstup z jednoduchého genetického algoritmu, který konkrétně ilustruje proces mutace (mutation) binárních chromozomů v třetí generaci

(zdroj: vlastní program)

1.1.4 Evoluční inspirace a adaptivní povaha

Celý koncept genetických algoritmů je hluboce zakořeněn v principech biologické evoluce [1]. Algoritmus napodobuje proces adaptace populací na prostředí prostřednictvím iterativního cyklu variace (křížení, mutace) a selekce. Úspěšná řešení ("jedinci") se "rozmnožují" a jejich vlastnosti jsou předávány a potenciálně vylepšovány v dalších generacích. Klíčovou silou tohoto přístupu je jeho inherentní adaptabilita, která umožňuje GA efektivně řešit komplexní, multimodální, nelineární a špatně definované optimalizační problémy, kde jiné metody mohou selhávat [4].

1.2 Aplikace genetických algoritmů

Tato sekce demonstruje široké spektrum aplikačních domén, ve kterých genetické algoritmy prokázaly svou užitečnost a efektivitu. Následující příklady ilustrují jejich univerzálnost.

1.2.1 Optimalizace v inženýrských disciplínách

V inženýrství nacházejí GA uplatnění při návrhu a optimalizaci mechanických konstrukcí (např. minimalizace hmotnosti při zachování pevnosti), optimalizaci výrobních a logistických procesů, či v návrhu řídicích systémů [3]. Příkladem může být optimalizace tvaru aerodynamických profilů, návrh topologie nosných struktur nebo třeba plánování trajektorií robotických ramen v komplexním prostředí.

Optimalizace tvaru aerodynamických profilů: Genetické algoritmy se s úspěchem používají k nalezení ideálního tvaru křídla letadla nebo lopatek turbíny. Cílem je obvykle minimalizovat odpor vzduchu, maximalizovat vztlak nebo dosáhnout optimálního proudění, což vede k vyšší efektivitě a nižší spotřebě paliva. [7]

Návrh topologie nosných struktur: GA mohou být využity k nalezení optimálního rozmístění materiálu v konstrukci, jako je například most nebo rám budovy. Algoritmus iterativně odstraňuje a přidává materiál v různých částech konstrukce s cílem minimalizovat její hmotnost při zachování požadované pevnosti a stability. [8]

1.2.2 Strojové učení a analýza dat

V oblasti strojového učení jsou GA často využívány pro optimalizaci hyperparametrů modelů (např. neuronových sítí, podpůrných vektorových strojů), kde nalezení optimální konfigurace je klíčové pro výkon modelu. Další významnou aplikací je výběr příznaků (feature selection), kde GA pomáhají identifikovat nejrelevantnější podmnožinu vstupních proměnných, což vede ke zjednodušení modelu a zlepšení jeho generalizační schopnosti [2].

Optimalizace hyperparametrů neuronových sítí: Genetické algoritmy se používají k automatickému ladění klíčových parametrů neuronových sítí, jako je počet vrstev, počet neuronů v každé vrstvě, rychlost učení nebo koeficient regularizace. Nalezení optimální kombinace těchto hyperparametrů může výrazně zlepšit přesnost a robustnost trénovaného modelu. [5]

Výběr relevantních příznaků pro klasifikaci: V úlohách klasifikace s vysokým počtem vstupních proměnných mohou GA pomoci identifikovat podmnožinu nejdůležitějších příznaků. Tím se nejen zjednodušuje model a snižuje výpočetní náročnost, ale často se i zlepšuje jeho schopnost zobecnění na nová data odstraněním šumu a redundantních informací. [9]

1.2.3 Návrh a optimalizace sítí

V telekomunikacích a počítačových sítích se GA uplatňují při řešení problémů, jako je optimalizace topologie sítě, směrování provozu (routing) nebo alokace zdrojů. Cílem může být například minimalizace latence, maximalizace propustnosti nebo snížení nákladů na infrastrukturu.

Optimalizace směrování provozu (routing): Genetické algoritmy mohou být použity k nalezení nejefektivnějších cest pro přenos dat v rozsáhlých počítačových sítích. Cílem je minimalizovat zpoždění, ztrátu paketů a maximalizovat propustnost sítě dynamickým přizpůsobováním směrovacích tabulek v reakci na změny v síťovém provozu. [10]

Alokace frekvenčního spektra v bezdrátových sítích: GA mohou pomoci s optimálním rozdělením omezeného frekvenčního spektra mezi různé uživatele a buňky v mobilních sítích. Cílem je minimalizovat interference, maximalizovat počet obslužených uživatelů a zajistit spravedlivé přidělování zdrojů. [11]

1.2.4 Finanční modelování a správa portfolia

V ekonomii a financích slouží GA jako nástroj pro optimalizaci investičních strategií, například při sestavování portfolia cenných papírů s cílem maximalizovat očekávaný výnos při dané úrovni rizika (nebo naopak). Mohou být také použity pro kalibraci komplexních finančních modelů nebo pro detekci arbitrážních příležitostí [3].

Optimalizace složení investičního portfolia: Genetické algoritmy se využívají k nalezení optimální alokace aktiv (například akcií, dluhopisů, komodit) v portfoliu investora. Cílem je obvykle maximalizovat očekávaný výnos pro danou úroveň rizika nebo minimalizovat riziko při požadovaném výnosu, přičemž se zohledňují různá omezení a investiční cíle.

Kalibrace finančních modelů oceňování opcí: Komplexní finanční modely, jako jsou modely pro oceňování opcí, mají často parametry, které je obtížné odhadnout. Genetické algoritmy mohou být použity k nalezení optimálních hodnot těchto parametrů tak, aby model co nejlépe odpovídal pozorovaným tržním cenám. [12] Hollandova práce je raným základem GA, ale principy se aplikují i na kalibraci finančních modelů.

1.3 Implementační aspekty a strategická doporučení

1.3.1 Architektura a softwarová implementace GA

Tato část se zaměřuje na praktické aspekty implementace genetického algoritmu. Z hlediska softwarové architektury je klíčová modularita, která umožňuje snadnou výměnu nebo modifikaci jednotlivých komponent (reprezentace jedinců, fitness funkce, operátory selekce, křížení, mutace, strategie řízení populace).

Flexibilní návrh fitness funkce: Při implementaci GA je klíčové navrhnout fitness funkci, která přesně kvantifikuje kvalitu řešení pro daný problém. Měla by být modulární, aby se dala snadno upravovat při změně požadavků nebo cílů optimalizace. Například, při optimalizaci trasy vozidel může fitness funkce zohledňovat ujetou vzdálenost, čas doručení a náklady.

Volba vhodné reprezentace jedinců: Způsob, jakým jsou potenciální řešení (jedinci) reprezentována v algoritmu (například jako binární řetězce, celá čísla, reálná čísla nebo složitější datové struktury), má zásadní vliv na efektivitu GA. Problémy s diskrétními proměnnými mohou vyžadovat odlišnou reprezentaci než problémy s kontinuálními proměnnými. Například pro problém obchodního cestujícího se často používá reprezentace pořadí měst.

Implementace sady genetických operátorů: Účinnost GA závisí na správné implementaci operátorů selekce (výběr lepších jedinců pro reprodukci), křížení (kombinace vlastností rodičů) a mutace (zavedení náhodných změn). Je důležité zvolit operátory, které jsou vhodné pro danou reprezentaci jedinců a prohledávaný prostor řešení. Například, pro binární reprezentaci se často používá jednobodové křížení a bitová mutace.

1.3.2 Konfigurace parametrů a ladění optimalizačních strategií

Výkon genetického algoritmu je významně ovlivněn nastavením jeho řídicích parametrů, jako jsou velikost populace, pravděpodobnosti aplikace genetických operátorů (křížení, mutace) a kritérium ukončení [2].

Empirická doporučení pro nastavení parametrů: V praxi se často vychází z obecných doporučení pro nastavení klíčových parametrů GA. Například, velikost populace se typicky pohybuje v řádu desítek až stovek jedinců, pravděpodobnost křížení bývá poměrně vysoká (např. 0.6-0.9) a pravděpodobnost mutace nízká (např. 0.001-0.1), aby se zajistila diverzita populace a zabránilo se předčasné konvergenci k suboptimálnímu řešení. [13]

Techniky adaptivního řízení parametrů: Pro zlepšení robustnosti a výkonu GA se použí-

vají strategie, které dynamicky upravují hodnoty parametrů během běhu algoritmu. Například, pravděpodobnost mutace se může zvyšovat, pokud se populace příliš homogenizuje, aby se podpořilo prozkoumávání nových oblastí řešení. Podobně se může měnit i pravděpodobnost křížení v závislosti na fázi běhu algoritmu nebo na zlepšování nejlepšího řešení. [14]

Experimentální ladění parametrů: Systematické testování různých kombinací parametrů na reprezentativních benchmarkových problémech je často nezbytné pro nalezení optimálního nastavení pro konkrétní aplikaci. Lze využít metody jako grid search nebo sofistikovanější přístupy založené na metamodelování pro efektivnější prohledávání prostoru parametrů. Výsledky těchto experimentů pak mohou poskytnout cenné informace o citlivosti algoritmu na různé parametry.

2 BINÁRNÍ REPREZENTACE REÁLNÝCH HODNOT

Tato kapitola se podrobně zabývá metodami převodu reálných čísel na diskretní binární řetězce, které se následně používají v genetických algoritmech (GA). Přesné zakódování spojitých hodnot je v evoluční optimalizaci klíčové. Proto se zde rozebírají následující metody: standardní binární kódování, Grayův kód, reprezentace čísel v plovoucí řádové čárce podle standardu IEEE 754 a některé hybridní přístupy, včetně BCD a real-valued GA. U všech přístupů je popsán převodní mechanismus, výhody a nevýhody, a to jak z pohledu implementace, tak i vliv na chování celého algoritmu.

2.1 Standardní binární kód

Standardní binární kódování transformuje reálná čísla na binární řetězce s pevnou délkou. Tento přístup, využívající metodu `binary_to_real_fixed_point`, je založen na normalizaci hodnot z definovaného intervalu. Pro ilustraci principu si v následujících příkladech představme, že chceme zakódovat reálná čísla z intervalu $[0,10]$ pomocí binárního řetězce délky $L=4$.

2.1.1 Převod s pevným bodem (Fixed-Point Conversion)

Nechť x je prvek intervalu $[a,b]$. Normalizace se provádí pomocí vzorce:

$$x_{\text{norm}} = \frac{x - a}{b - a}, \quad (2.1)$$

kde výsledná hodnota je mapována na diskretní množinu reprezentovanou celým číslem X podle vzorce:

$$X = \lfloor x_{\text{norm}} \cdot (2^L - 1) \rfloor, \quad (2.2)$$

kde L je délka binárního řetězce. Takto získaný celočíselný kód se následně převede do binární podoby. Uvažujme například hodnotu $x=6.5$ z intervalu $[a,b]=[0,10]$ a délku binárního řetězce $L=4$.

1. Normalizace:

$$x_{\text{norm}} = \frac{6.5 - 0}{10 - 0} = 0.65. \quad (2.3)$$

2. Mapování na celé číslo:

$$X = \lfloor 0.65 \cdot (2^4 - 1) \rfloor = \lfloor 0.65 \cdot 15 \rfloor = \lfloor 9.75 \rfloor = 9. \quad (2.4)$$

3. Převod na binární kód:

$$9_{10} = 1001_2. \quad (2.5)$$

Reálná hodnota 6.5 je tedy v tomto případě reprezentována binárním řetězcem 1001. Tento přístup je jednoduchý a výpočetně efektivní, ale jeho přesnost je omezena délkou řetězce L. Pokud bychom chtěli dekódovat řetězec 1001 zpět na reálné číslo, nejprve převedeme

$$1001_2 = 9_{10} \quad (2.6)$$

Poté aplikujeme inverzní transformaci (pro zjednodušení zde neuvažujeme zaokrouhlovací chybu ·):

$$x' = \frac{X}{2^L - 1}(b - a) + a = \frac{9}{15}(10 - 0) + 0 = \frac{90}{15} = 6.0. \quad (2.7)$$

Vidíme, že kvůli diskretizaci a omezené délce řetězce došlo ke ztrátě přesnosti, neboť původní hodnota byla 6.5, zatímco dekódovaná hodnota je 6.0. Velikost kroku (granularita) této reprezentace je

$$\Delta x = \frac{b - a}{2^L - 1} = \frac{10 - 0}{15} \approx 0.667. \quad (2.8)$$

Každé reálné číslo je tedy mapováno na nejbližší reprezentovatelný diskretní bod. Problémem tohoto kódování může být také tzv. Hammingova stěna (Hamming cliff), kdy dvě číselně blízké hodnoty mají binární kódy s velkou Hammingovou vzdáleností. Uvažujme pro jednoduchost interval [0,7] a délku řetězce L=3. V tomto případě je $2^L - 1 = 7$, takže mapování celočíselné části $X = x_{\text{norm}} * 7$

na intervalu [0,7] (pokud a=0,b=7) přímo odpovídá celočíselné hodnotě x (pokud je x celé).

- Číslo 3 (x=3) je reprezentováno jako X=3, což je binárně 011_2 .

- Číslo 4 ($x=4$) je reprezentováno jako $X=4$, což je binárně 100_2 .

Ačkoliv jsou celá čísla 3 a 4 v dekadické soustavě sousední, jejich standardní binární reprezentace 011_2 a 100_2 se liší ve všech třech bitech (Hammingova vzdálenost je 3). Malá změna reálné hodnoty, která překročí hranici mezi těmito reprezentacemi, tak může vyžadovat velkou změnu v binárním řetězci, což může komplikovat práci genetických operátorů, zejména mutace.

2.1.2 Binary-Coded Decimal

Další možností je metoda BCD, která kóduje každou číslici desetinného čísla samostatně v binárním tvaru. Tato reprezentace usnadňuje převody mezi číselnými systémy, ale je méně kompaktní než standardní binární kódování. To může vést ke zvýšení výpočetní náročnosti při aplikaci genetických operátorů.

2.2 Grayův kód

Grayův kód je modifikace binárního kódování, kde se sousední čísla liší pouze v jednom bitu. To snižuje riziko chyb při malých změnách.

Převod ze standardního binárního kódu B na Grayův kód G se provádí bit po bitu. Nejvýznamnější bit Grayova kódu G_0 je stejný jako nejvýznamnější bit binárního kódu B_0 . Ostatní bity Grayova kódu G_i (pro $i>0$) se získají pomocí operace XOR mezi i -tým bitem binárního kódu B_i a jeho předchozím bitem B_{i-1} :

$$G_i = B_i \oplus B_{i-1}, \quad \text{pro } i = 1, 2, \dots, L-1, \quad (2.9)$$

a

$$G_0 = B_0. \quad (2.10)$$

Tato vlastnost zajišťuje, že změna jednoho bitu v Grayově kódu způsobí změnu pouze o $+1$ nebo -1 v dekadické hodnotě (v případě celočíselné reprezentace), což podporuje plynulejší evoluci populace v GA.

Ukázka převodu:

Uvažujme standardní binární kód $B = 1011_2$ (dekadicky 11). Pro 4 bity, kde indexujeme od 0 (MSB) do 3 (LSB), tedy $B = B_0B_1B_2B_3 = 1011$, provedeme převod na Grayův kód $G = G_0G_1G_2G_3$ následovně: $G_0 = B_0 = 1$ $G_1 = B_0 \oplus B_1 = 1 \oplus 0 = 1$ $G_2 = B_1 \oplus B_2 = 0 \oplus 1 = 1$ $G_3 = B_2 \oplus B_3 = 1 \oplus 1 = 0$

Standardní binární kód 1011_2 (11) tedy odpovídá Grayovu kódu 1110_G .

Nyní se podívejme na sousední dekadické hodnoty a jejich kódy (pro 3 bity):

Tabulka 2.1 Převod mezi binárním a Grayovým kódem (pro 3 bity)

Dekadicky	Binárně (B)	Grayův kód (G)
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

(zdroj: vlastní program)

2.3 Reprezentace Floating-Point dle IEEE 754

Přímá reprezentace reálných čísel se provádí podle standardu IEEE 754, také označovaného jako metoda `binary_to_real_ieee754`. Tento standard rozkládá číslo na znaménko, exponent a mantisu a umožňuje reprezentovat velmi široký rozsah hodnot.

2.3.1 Formát IEEE 754

Formální zápis čísla podle IEEE 754 je:

$$x = (-1)^S \cdot 1.M \cdot 2^{E-B}, \quad (2.11)$$

kde (S) je bit znaménka, (M) je mantisa, (E) je exponent a (B) je bias (posun exponentu). Tento přístup nabízí vysokou přesnost a flexibilitu, ale implementace vyžaduje speciální úpravy genetických operátorů, aby byla zajištěna korektnost reprezentace při křížení a mutaci.

Uvažujme reprezentaci čísla -0.75_{10} ve formátu IEEE 754 single precision (32 bitů: 1 bit znaménko, 8 bitů exponent, 23 bitů mantisa).

- **Znaménko (S):** Číslo je záporné, tedy $S = 1$.
- **Binární podoba absolutní hodnoty:** $|-0.75|_{10} = 0.11_2$.

- **Normalizace:** $0.11_2 = 1.1_2 \cdot 2^{-1}$.
- **Mantisa (M):** Z 1.1_2 je mantisa část za desetinnou čárkou, tj. $M = 1_2$ (doplněno nulami na 23 bitů). Tedy 10000000000000000000000.
- **Exponent (E):** Skutečný exponent je -1 . Bias pro single precision je $B = 127$. Kódovaný exponent $E = \text{skutečný exponent} + B = -1 + 127 = 126$.
- **Binární podoba exponentu E :** $126_{10} = 01111110_2$.

Výsledný 32bitový řetězec je:

S EEEEEEE MMMMMMMMMMMMMMMMMMMMMMM
 1 01111110 10000000000000000000000

Tato reprezentace umožňuje kódovat široký rozsah čísel s proměnlivou přesností, ale přímá aplikace standardních genetických operátorů (jako jednobodové křížení) na tento binární řetězec může vést k neplatným nebo velmi odlišným číslům, protože struktura znaménka, exponentu a mantisy by byla narušena.

2.4 Další metody reprezentace

Kromě tradičních metod existují i alternativní a hybridní přístupy, které kombinují výhody pevné binární reprezentace s přímým kódováním reálných hodnot.

2.4.1 Real-Valued GA

RGA, neboli spojitě genetické algoritmy, eliminují nutnost převodu reálných čísel na binární řetězce. V těchto algoritmech jsou hodnoty reprezentovány přímo jako reálná čísla a genetické operátory, jako je aritmetické křížení a aditivní mutace, s nimi přímo pracují. Tento přístup snižuje chyby vznikající aproximací a zajišťuje hladší průběh evoluce, ale může být výpočetně náročnější.

Například jedinec reprezentující řešení problému se dvěma proměnnými $x_1 \in [-5, 5]$ a $x_2 \in [0, 10]$ by mohl být jednoduše uložen jako pole nebo vektor reálných čísel, např. $[3.14159, 7.89]$.

2.4.2 Hybridní metody

Hybridní přístupy zahrnují kombinace standardního binárního kódování a reálné reprezentace, například s využitím BCD nebo modifikovaných technik, které propojují reprezentaci s pevnou a plovoucí řádovou čárkou. Tyto metody jsou vhodné zejména pro složité optimalizační úlohy, kde je potřeba skloubit vysokou přesnost s efektivním prohledáváním rozsáhlého prostoru řešení.

Příkladem hybridního přístupu by mohlo být kódování, kde část genomu reprezentuje celé číslo standardním binárním kódem (např. pro výběr diskrétního režimu) a jiná část kóduje reálné parametry pomocí IEEE 754 nebo scaled fixed-point pro daný režim.

Volba konkrétní reprezentace reálných hodnot (např. Fixed-Point, IEEE 754, nebo přímé Real-Valued) a kódovacího schématu (jako je standardní binární kódování nebo Grayův kód) má zásadní vliv na chování a výkon genetického algoritmu. Zatímco standardní binární kódování a Grayův kód nabízejí jednoduchost a stabilitu pro diskrétní reprezentace, reprezentace floating-point podle IEEE 754 a RGA poskytují vysokou flexibilitu a přesnost pro spojitě prostory řešení. Hybridní metody umožňují kombinovat výhody různých přístupů, což je klíčové hlavně v aplikacích, které vyžadují jak vysokou přesnost, tak robustní hledání optimálních řešení.

3 OPERÁTORÝ GENETICKÉHO ALGORITMU

Tato kapitola se věnuje detailnímu popisu jednotlivých operátorů genetických algoritmů a jejich vlivu na optimalizační proces. Vysvětlení je podpořeno matematickými vzorci a diagramy (doporučeno zahrnout vizuální materiály s anglickými popisky), které ilustrují principy a dynamiku jednotlivých operátorů. Níže jsou rozděleny klíčové aspekty do čtyř hlavních sekcí.

3.1 Mutace v binární reprezentaci

Mutace je proces, který zajišťuje diverzitu populace a zabraňuje předčasné konvergenci algoritmu k lokálním extrémům. V binární reprezentaci se nejčastěji používá *bit-flip mutace*, kdy se jednotlivé bity náhodně invertují s předem stanovenou pravděpodobností p_m .

3.1.1 Bit-flip mutace

Necht' $C = \{b_1, b_2, \dots, b_L\}$ představuje binární řetězec (chromozom) jedince s délkou L . Pravděpodobnost, že se u daného bitu b_i dojde k jeho inverzi, je p_m . Matematicky můžeme operaci vyjádřit jako:

$$[b_i^{\text{new}} = \begin{cases} 1 - b_i, & \text{if } r < p_m, \\ b_i, & \text{if } r \geq p_m, \end{cases} \quad (3.1)$$

] kde $r \sim U(0, 1)$ je náhodná proměnná s uniformním rozdělením. Celkový počet mutací k v chromozomu je pak dán binomickým rozdělením:

$$[P(k) = \binom{L}{k} p_m^k (1 - p_m)^{L-k}. \quad (3.2)$$

] Tato metoda umožňuje jemné dolad'ování řešení, kde i malá změna bitu může vést ke značné změně fitness jedince, což je přínosné zejména při prohledávání složitých optimalizačních prostorů.

3.1.2 Dopad na vyhledávání řešení

Bit-flip mutace zavádí náhodnost, která podporuje explorační prostor řešení. Příliš vysoká hodnota p_m však může vést k příliš radikálním změnám, zatímco příliš nízká hodnota může způsobit stagnaci. Je proto nezbytné vhodně nastavit p_m tak, aby se dosáhlo rovnováhy mezi exploitací (vylepšováním dobrých řešení) a explorační (průzkumem nových oblastí).

3.2 Křížení v binární reprezentaci

Křížení (crossover) je proces, při kterém dochází k výměně genetické informace mezi dvěma rodiči s cílem vytvořit potomky, kteří kombinují jejich vlastnosti. V binární reprezentaci se využívá několik metod, mezi něž patří jednobodové, dvoubodové a uniformní křížení.

3.2.1 Jednobodové křížení

V jednobodovém křížení se náhodně vybere jeden index i v rozsahu $\{1, 2, \dots, L-1\}$, a potomci jsou vytvořeni spojením úseků rodičovských řetězců:

$$\begin{cases} C_{\text{child}}^{(1)} = \{b_1^{(1)}, b_2^{(1)}, \dots, b_i^{(1)}, b_{i+1}^{(2)}, \dots, b_L^{(2)}\}, \\ C_{\text{child}}^{(2)} = \{b_1^{(2)}, b_2^{(2)}, \dots, b_i^{(2)}, b_{i+1}^{(1)}, \dots, b_L^{(1)}\}. \end{cases} \quad (3.3)$$

] Tento způsob křížení efektivně kombinuje vlastnosti obou rodičů a podporuje rychlý průběh evolučního procesu.

3.2.2 Dvoubodové a uniformní křížení

Dvoubodové křížení vybírá dva náhodné indexy i a j ($i < j$) a vytváří potomky tak, že úseky mezi těmito body jsou mezi rodiči prohozeny:

$$[C_{\text{child}}^{(1)} = \{b_1^{(1)}, \dots, b_i^{(1)}, b_{i+1}^{(2)}, \dots, b_j^{(2)}, b_{j+1}^{(1)}, \dots, b_L^{(1)}\}. \quad (3.4)$$

] Uniformní křížení naopak pracuje s individuálními bity, kde pro každý bit rodičovského řetězce je s určitou pravděpodobností vybrán bit z jednoho z rodičů. Matematicky lze uniformní křížení vyjádřit pomocí masky $M = \{m_1, m_2, \dots, m_L\}$, kde:

$$[m_i \sim U\{0, 1\}, \quad C_{\text{child}}^{(1)}[i] = \begin{cases} b_i^{(1)}, & \text{if } m_i = 1, \\ b_i^{(2)}, & \text{if } m_i = 0. \end{cases} \quad (3.5)$$

] Tato metoda podporuje vysokou míru variability, což je výhodné při prohledávání velmi rozsáhlých prostorů řešení.

3.3 Mutace v reálné reprezentaci

Při přímém zpracování reálných hodnot se mutační operace realizují jiným způsobem než v případě binárního kódování. Zde se často využívá aditivní nebo gaussovská mutace.

3.3.1 Aditivní mutace

Při aditivní mutaci je každá hodnota x modifikována následujícím způsobem:

$$[x_{\text{new}} = x_{\text{old}} + \delta, \quad (3.6)$$

] kde $\delta \sim U(-\varepsilon, \varepsilon)$ je náhodně generovaná hodnota z uniformního rozdělení s parametrem ε určujícím maximální možný odchylkový posun. Tento způsob zajišťuje kontrolovanou modifikaci hodnoty s ohledem na rozsah optimalizované funkce.

3.3.2 Gaussovská mutace

Alternativně lze využít gaussovskou mutaci, kde se náhodná odchylka generuje na základě normálního rozdělení:

$$[x_{\text{new}} = x_{\text{old}} + \sigma \cdot N(0, 1), \quad (3.7)$$

] kde $N(0, 1)$ reprezentuje standardní normální rozdělení a σ je parametr určující intenzitu mutace. Tento přístup umožňuje jemnější úpravy s vyšší pravděpodobností malých odchylek a zároveň občasnou velkou změnu, což podporuje exploraci řešení.

3.4 Křížení v reálné reprezentaci

V reálné reprezentaci genetických algoritmů se křížení provádí přímo nad reálnými čísly. Mezi nejčastěji využívané metody patří aritmetické křížení a BLX- α crossover.

3.4.1 Aritmetické křížení

Aritmetické křížení kombinuje hodnoty dvou rodičovských řešení pomocí váženého průměru. Pro rodiče s hodnotami x_1 a x_2 lze potomka vypočítat jako:

$$[x_{\text{child}} = \alpha x_1 + (1 - \alpha)x_2, \quad (3.8)$$

] kde $\alpha \in [0, 1]$ je náhodně zvolená váha. Tato metoda umožňuje plynulé interpolace mezi hodnotami rodičů a podporuje kontinuitu prostoru řešení.

3.4.2 BLX- α křížení

BLX- α (Blend Crossover) rozšiřuje interval mezi hodnotami rodičů. Necht' x_1 a x_2 jsou dva rodičovské geny a definujme:

$$[I = |x_1 - x_2|. \quad (3.9)$$

] Potom se hodnota potomka vybírá náhodně z intervalu:

$$[x_{\text{child}} \in [\min(x_1, x_2) - \alpha I, \max(x_1, x_2) + \alpha I], \quad (3.10)$$

] kde α je parametr určující míru rozšíření intervalu. Tato metoda umožňuje průzkum širšího prostoru řešení a pomáhá předcházet předčasné konvergenci do lokálních extrémů.

Všechny výše uvedené operátory, ať již v binární či reálné reprezentaci, hrají klíčovou roli v efektivitě genetických algoritmů. Vhodná volba a parametrizace mutačních a crossover operací má zásadní vliv na schopnost algoritmu najít globální optimum v komplexních optimalizačních úlohách.

4 TESTOVACÍ FUNKCE PRO GENETICKÉ ALGORITMY

Testovací funkce jsou klíčovým prvkem pro ověření účinnosti a robustnosti GA. Výběr vhodné testovací funkce nejen umožňuje zhodnotit vlastnosti optimalizačního algoritmu, ale také poskytuje cenné informace o chování algoritmu ve více komplexních, reálných situacích. V této kapitole jsou diskutována kritéria výběru, vlastnosti a srovnání testovacích funkcí, přičemž text zahrnuje teoretické popisy, matematické modely i praktické příklady na základě standardních i moderních benchmarkových sad, jako je například GNBG [15, 16].

4.1 Kritéria pro výběr testovacích funkcí

Výběr vhodné testovací funkce podléhá několika zásadním kritériím, která ovlivňují chování a vyhodnocení GA. Mezi hlavní kritéria patří:

4.1.1 Multimodalita

Multimodální testovací funkce obsahují více než jedno lokální minimum (resp. maximum), což znamená, že vyhledávací prostor je velmi komplexní a algoritmus se musí vyhnout uvěznění v lokálních extrémech. Formálně lze multimodalitu ilustrovat jako funkci

$$[f : \mathbb{R}^n \rightarrow \mathbb{R}, \quad \exists x_1, x_2 \in \Omega \text{ tak, že } f(x_1) \neq f(x_2), \quad (4.1)$$

] kde $\Omega \subset \mathbb{R}^n$ označuje definiční obor. Efektivní GA musí disponovat mechanismy, například rozumnou mírou mutace nebo adaptivním crossover operátorem, aby byla schopna najít globální optimum přes množství lokálních minimum.

4.1.2 Separabilita

Separabilita testovací funkce popisuje, zda lze optimalizační úlohu rozdělit na menší, nezávislé podúlohy. Formálně vyjádřeno, funkce $f(x_1, x_2, \dots, x_n)$ je separabilní, pokud existují funkce g_i tak, že

$$[f(x_1, x_2, \dots, x_n) = \sum_{i=1}^n g_i(x_i). \quad (4.2)$$

] V případě separabilních úloh může být optimalizace jednotlivých proměnných prováděna nezávisle, což značně zjednodušuje hledání globálního optima. Naopak u neseparabilních funkcí je interakce proměnných tak silná, že konvergence GA je obtížnější.

4.1.3 Škálovatelnost

Škálovatelnost se týká schopnosti testovací funkce odpovídat změnám v dimenzionalitě úlohy. Pro zvýšení škálovatelnosti je třeba, aby funkce zachovala své charakteristické vlastnosti (např. počet lokálních extrémů) i při nárůstu rozměru n . Škálovatelnost je často testována pomocí přímých funkcí

$$[f(x_1, x_2, \dots, x_n) = \sum_{i=1}^n \phi(x_i), \quad (4.3)$$

] kde ϕ je jednoznačně definovaná funkce s proměnlivými složitostmi. Vyhodnocení efektivity GA v souvislosti se škálovatelností pomáhá určit, jak algoritmus zvládá postupné rozšiřování vyhledávacího prostoru.

4.2 Přehled a vlastnosti testovacích funkcí

Testovací sady jsou v praxi vybírány tak, aby pokryly širokou škálu vlastností reálných optimalizačních problémů. Mezi základní testovací funkce patří:

4.2.1 Unimodální funkce

Tyto funkce mají jedno globální minimum nebo maximum. Jsou vhodné pro hodnocení rychlosti a efektivity konvergence algoritmu, aniž by se algoritmus musel potýkat s problémy lokálních extrémů. Typickým příkladem je jednoduchá parabolická funkce

$$[f(x) = \sum_{i=1}^n x_i^2, \quad (4.4)$$

] kde globální minimum je dosaženo v $x_i = 0$ pro všechna i .

4.2.2 Multimodální funkce

Jak již bylo zmíněno, multimodální funkce (např. Rastriginova funkce, Rosenbrockova funkce či Ackleyho funkce) obsahují několik lokálních extrémů. Například Rastriginova funkce je definována jako:

$$[f(x) = 10n + \sum_{i=1}^n [x_i^2 - 10 \cos(2\pi x_i)], \quad (4.5)$$

] a představuje náročný test na schopnost GA překonat mnohonásobné pasti lokálních optima.

4.2.3 Funkce s neseparabilními proměnnými

Tyto funkce jsou navrženy tak, aby testovaly interakci mezi proměnnými. Příkladem může být Rosenbrockova funkce:

$$[f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2], \quad (4.6)$$

] která je známá svým úzkým pásem globálního optima a vysokou závislostí mezi jednotlivými proměnnými.

4.3 Příklady generovaných testovacích sad a jejich využití

V nedávné době byly vyvinuty pokročilé nástroje, jako je Generalized Numerical Benchmark Generator (GNBG) [15, 16], které umožňují konfigurovat testovací sady pro kontinuální optimalizační úlohy. GNBG umožňuje dynamickou změnu parametrů, což je výhodné pro systematické testování GA, a poskytuje následující vlastnosti:

4.3.1 Vlastnosti testovacích funkcí

GNBG testovací sady jsou navrženy tak, aby obsahovaly různé stupně multimodalit, separability a škálovatelnosti. Uživatel může definovat počet komponent, jejich váhové koeficienty a transformační parametry, což umožňuje generovat funkce s různou mírou složitosti. Tyto vlastnosti jsou často charakterizovány pomocí následujícího modelu:

$$[f(x) = \min_{k \in \{1, \dots, K\}} \left\{ \sigma_k + [(x - p_k)^T H_k (x - p_k)]^{\lambda_k} \right\}, \quad (4.7)$$

] kde p_k představuje pozici k -té komponenty, H_k je matice určující tvar řešení, σ_k je posun a λ_k parametr ovlivňující strmost. Takový model umožňuje generovat testovací funkce s velmi různorodými vlastnostmi.

4.3.2 Multimodalita, separabilita a škálovatelnost v praxi

- **Multimodalita:** Zvyšující se počet lokálních extrémů v testovací funkci zvyšuje náročnost optimalizace. GA musí mít dostatečně robustní mutační a crossover operátory, aby dokázaly najít globální optimum i přes přítomnost mnoha lokálních optim. - **Separabilita:** U neseparabilních funkcí se interakce mezi jednotlivými proměnnými stávají kritickým faktorem. GA musí být schopna zachytit tyto vzájemné závislosti, což se často projevuje vyšší složitostí konvergence. - **Škálovatelnost:** Testovací funkce by měly být schopny adaptace na různé dimenze vyhledávacího prostoru. Zdravým indikátorem je, pokud algoritmus zachovává svou

výkonnost při zvýšení počtu proměnných, což je často doloženo srovnávacími grafy, kde osa x reprezentuje dimenzionalitu a osa y dosaženou přesnost.

Výše uvedená kritéria a vlastnosti testovacích funkcí lze využít k vytvoření komplexních benchmarkových sad, které poskytují detailní přehled o chování genetických algoritmů v různých optimalizačních podmínkách. Při implementaci testů je vhodné využít především dynamické generátory, jako je GNBG, který umožňuje detailní konfigurovatelnost jednotlivých parametrů, a tím ještě lépe simulovat reálné optimalizační problémy. Pro náš příklad jsme zvolili před danou sadu, kvůli férovosti.

5 METODY VYHODNOCENÍ EFEKTIVITY BINÁRNÍ REPREZENTACE

Tato kapitola se zabývá podrobným rozбором metod a metrik, které umožňují objektivně posoudit kvalitu a efektivitu různých variant binárních reprezentací v kontextu genetických algoritmů. Primárním cílem je nejen kvantifikovat přesnost, s jakou lze reálné hodnoty převést do binární formy, ale také hlouběji porozumět tomu, jak tyto reprezentace ovlivňují chování samotného optimalizačního algoritmu při hledání globálního optima. Následující podsekcce se věnují teoretické analýze klíčových vlastností binárních reprezentací, experimentální validaci těchto vlastností pomocí standardních testovacích funkcí a statistickému vyhodnocení získaných výsledků.

5.1 Hammingova vzdálenost mezi sousedními hodnotami

5.1.1 Definice a význam

Hammingova vzdálenost, značená jako d_H , představuje fundamentální metriku v teorii informace, která kvantifikuje míru odlišnosti mezi dvěma řetězcí nebo vektory stejné délky. Formálně je definována jako počet pozic, na kterých se odpovídající symboly v obou řetězcích liší. V kontextu binárních řetězců $A = (a_1, a_2, \dots, a_L)$ a $B = (b_1, b_2, \dots, b_L)$ o délce L je Hammingova vzdálenost vypočítána součtem rozdílů na každé pozici, kde rozdíl je indikován funkcí $\delta(a_i, b_i)$. Tato funkce vrací 1, pokud se bity na dané pozici liší, a 0, pokud jsou stejné. [17]

$$\delta(a_i, b_i) = \begin{cases} 1, & \text{jestliže } a_i \neq b_i, \\ 0, & \text{jestliže } a_i = b_i. \end{cases} \quad (5.1)$$

V genetických algoritmech, kde jsou kandidátní řešení často reprezentována jako binární chromozomy, hraje Hammingova vzdálenost klíčovou roli. Udává minimální počet bitových změn (mutací), které jsou nutné k transformaci jednoho řešení na jiné. Z tohoto pohledu lze Hammingovu vzdálenost interpretovat jako míru "vzdálenosti" mezi dvěma body v prohledávacím prostoru. Důležitým aspektem je, že i malá Hammingova vzdálenost, například rovna jedné, může znamenat přechod mezi dvěma fenotypově velmi odlišnými řešeními, což se může projevit jako výrazný skok v hodnotě fitness funkce. Tato vlastnost má zásadní implikace pro efektivitu prohledávání řešení v genetickém algoritmu.

5.1.2 Vliv na průběh optimalizace

Vysoká Hammingova vzdálenost mezi genotypy, které reprezentují fenotypově blízké hodnoty, může negativně ovlivnit lokální kontinuitu genetických operátorů, zejména mutace a křížení. Například u standardního binárního kódování s pevnou délkou může změna jediného bitu (operace bit-flip mutace) vést k přechodu z hodnoty x na hodnotu x' , přičemž absolutní rozdíl $|x' - x|$ může být relativně velký. Tento jev lze přiblížit pomocí diferenciální aproximace, kde malá změna v Hammingově vzdálenosti ($\Delta d_H = 1$) způsobí odchylku ve fenotypu:

$$\Delta \text{fenotyp} \approx f'(x) \cdot \Delta d_H, \quad (5.2)$$

kde $f'(x)$ představuje citlivost fenotypu na změny v genotypu a může nabývat značně velkých hodnot v závislosti na charakteru kódování. Tato problematika je zvláště patrná při použití binárně kódované desítkové (BCD) reprezentace, kde redundance kódování může ještě zesílit dopad jednotlivých bitových změn.

V kontextu BCD kódování je situace specifická, neboť Hammingova vzdálenost mezi sousedními desítkovými čísly reprezentovanými v BCD nemusí být konstantní. Například přechod od desítkové 0 (binárně 0000) k 1 (binárně 0001) vyžaduje změnu jediného bitu, a tedy má Hammingovu vzdálenost 1. Naopak, přechod od desítkové 7 (binárně 0111) k 8 (binárně 1000) vyžaduje změnu všech čtyř bitů, což odpovídá Hammingově vzdálenosti 4. Tento proměnlivý charakter Hammingovy vzdálenosti mezi fenotypově sousedními hodnotami může významně ovlivnit efektivitu lokálního vyhledávání prováděného genetickým algoritmem. Změna jediného bitu v BCD reprezentaci nemusí vždy vést k platnému BCD kódu (číslíce 0-9), což zavádí redundanci, která bude podrobněji diskutována v sekci 5.3.

Teoreticky, pokud má sousední dvojice řešení ve vyhledávacím prostoru velkou Hammingovu vzdálenost, může mít genetický algoritmus potíže s postupným zlepšováním řešení pomocí genetických operátorů, jako je mutace a křížení, které typicky provádějí malé změny v genotypu. To může vést k pomalejší konvergenci k optimálnímu řešení nebo k uvíznutí v lokálních maximech či minimech fitness funkce. Efektivní genetický algoritmus by ideálně měl dosahovat stavu, kdy malé změny v genotypu vedou k malým změnám ve fenotypu, což umožňuje jemné doladování řešení a efektivní prozkoumávání okolí slibných kandidátů.

5.2 Analýza vlivu mutace na fenotyp

5.2.1 Role mutace v genetických algoritmech

Mutace je klíčový genetický operátor, který vnáší do genotypu jedinců náhodné změny. V kontextu binární reprezentace se nejčastěji používá bit-flip mutace, při níž se s určitou pravděpodobností změní hodnota jednotlivých bitů (z 0 na 1 nebo z 1 na 0). Hlavním cílem mutace je udržet genetickou diverzitu v populaci a zabránit předčasné konvergenci algoritmu k suboptimálnímu řešení. Mutace umožňuje algoritmu prozkoumávat nové oblasti prohledávacího prostoru, které by jinak nemusely být dosažitelné pomocí samotného křížení.[18]

5.2.2 Vliv mutace u standardního binárního kódování

Vliv mutace na fenotyp je silně závislý na použitém typu binární reprezentace. U standardního binárního kódování, jak již bylo zmíněno v předchozí sekci, může změna bitu na vyšší pozici (více vlevo, s vyšší váhou) způsobit mnohem větší změnu v reprezentované číselné hodnotě (fenotypu) než změna bitu na nižší pozici. To znamená, že mutace může vést k prozkoumávání vzdálenějších oblastí vyhledávacího prostoru, což je zásadní pro únik z lokálních optim. Nicméně, příliš velké skoky ve fenotypu způsobené mutací mohou také narušit proces jemného doladování řešení v okolí potenciálních optim. Optimální míra mutace je proto klíčová pro vyvážení explorační a exploatační prohledávacího prostoru.

5.2.3 Vliv mutace u BCD reprezentace

U BCD reprezentace je vliv mutace na fenotyp mnohem komplexnější. Změna jediného bitu v 4bitovém bloku reprezentujícím jednu desítkovou číslici může vést k několika různým výsledkům:

- Změna o jednotku v dané desítkové číslici: Pokud výsledný 4bitový kód po mutaci stále odpovídá platné desítkové číslici (0-9) a liší se od původní číslice o jedničku.
- Změna o více než jednotku v dané desítkové číslici: Například změna bitu, která způsobí přechod z 6 (0110) na 4 (0100) nebo podobně.
- Vytvoření neplatného BCD kódu: Pokud se po změně bitu objeví 4bitový kód, který neodpovídá žádné desítkové číslici (tj. hodnota větší než 9, binárně 1010 až 1111).

Třetí případ představuje mutaci, která nemá přímý vliv na fenotyp, pokud je neplatný kód následně nějakým způsobem ošetřen (např. ignorován, opraven). Poměr těchto tří typů mutací závisí na konkrétní bitové pozici, aktuální hodnotě dané desítkové číslice a pravděpodobnosti mutace. Analýza tohoto vlivu je klíčová pro pochopení, jak efektivně mutace prohledává prostor řešení při použití BCD.

Experimentálně lze analyzovat vliv mutace na fenotyp sledováním průměrné změny hodnoty fitness funkce po provedení mutace na reprezentativním vzorku jedinců z populace. Porovnání této metriky mezi různými typy binárních reprezentací může poskytnout cenné informace o jejich prohledávacích schopnostech.

5.2.4 Zacházení s neplatnými BCD kódy

Při implementaci genetického algoritmu s BCD reprezentací je nutné zvážit, jakým způsobem se budou zpracovávat neplatné BCD kódy. Existuje několik přístupů:

- Ignorování neplatných kódů: Pokud mutace vytvoří neplatný BCD kód, může být tento jedinec v následující generaci penalizován nízkou hodnotou fitness nebo může být rovnou nahrazen jiným, platným jedincem. Tento přístup je jednoduchý na implementaci, ale může být neefektivní, pokud mutace často generují neplatné kódy, čímž se ztrácí čas výpočtu.
- Oprava neplatných kódů: Další možností je pokusit se neplatný BCD kód opravit na nejbližší platný kód. Například pokud vznikne kód 1010 (desítkově 10), může být opraven na 1001 (desítkově 9) nebo 0001 0000 (desítkově 10, pokud se jedná o více-ciferné číslo). Tato metoda zajišťuje, že všichni jedinci v populaci budou mít platnou reprezentaci, ale může zavést zkreslení do prohledávacího procesu.
- Penalizace neplatných kódů: Funkce fitness může být navržena tak, aby jedincům s neplatnými BCD kódy přiřazovala velmi nízkou fitness hodnotu. Tím se sníží pravděpodobnost jejich výběru pro reprodukci a postupně se z populace eliminují. Tento přístup umožňuje algoritmu přirozeně preferovat platná řešení.

Volba strategie pro zacházení s neplatnými BCD kódy může významně ovlivnit výkon a efektivitu genetického algoritmu. Je důležité zvolit přístup, který nejlépe odpovídá řešenému problému a charakteristikám prohledávacího prostoru.

5.3 Redundance kódování

5.3.1 Definice a výskyt redundance

Redundance v kódování nastává, když existuje více než jedna reprezentace stejné informace. V kontextu binární reprezentace reálných čísel s pevnou délkou obvykle redundance nevzniká – každý binární řetězec odpovídá unikátnímu číslu v daném rozsahu (pokud nepočítáme případy zaokrouhlení). Nicméně, u BCD kódování se redundance projevuje přirozeně.

5.3.2 Redundance v BCD kódování

V BCD reprezentaci se pro kódování jedné desítkové číslice používají 4 bity, což teoreticky umožňuje reprezentovat 16 různých stavů (0000 až 1111). Pouze 10 z těchto stavů (0000 až 1001) však odpovídá platným desítkovým číslicím (0 až 9). Zbývajících 6 stavů (1010 až 1111) je redundantních nebo neplatných.

Tato redundance má několik důsledků pro genetický algoritmus:

- Neefektivní mutace: Jak již bylo zmíněno, některé bitové mutace v BCD mohou vést k neplatným kódům, které nerepresentují žádnou platnou desítkovou číslici. Tyto mutace lze považovat za redundantní v tom smyslu, že nepřispívají k prozkoumávání platného prostoru řešení.
- Prostor pro omezení: Redundance může být také výhodná v případě, že je potřeba reprezentovat omezený rozsah hodnot. Neplatné BCD kódy mohou být interpretovány jako mimo povolený rozsah, což může usnadnit implementaci omezení přímo na úrovni reprezentace.

Pro kvantifikaci redundance v BCD lze spočítat poměr redundantních mutací k neredundantním mutacím. Neredundantní mutace jsou ty, které vedou k platnému BCD kódu a zároveň změni reprezentovanou desítkovou číslici. Experimentální analýza tohoto poměru pro různé délky BCD reprezentace a různé mutační operátory může poskytnout vhled do efektivity prohledávání prostoru řešení.

Je důležité poznamenat, že i když redundance může na první pohled vypadat jako nevýhoda, v některých případech může přispět k robustnosti algoritmu tím, že poskytuje "záložní" reprezentace nebo umožňuje snadnější implementaci určitých omezení. Nicméně, v

kontextu efektivního prohledávání spojitého prostoru se zdá, že vysoká míra redundance spíše snižuje efektivitu genetického algoritmu tím, že zmenšuje podíl užitečných mutací.

5.4 Lokalita reprezentace

5.4.1 Význam lokality v genetických algoritmech

Lokalita reprezentace se týká vztahu mezi blízkostí genotypů a blízkostí odpovídajících fenotypů. Reprezentace s dobrou lokalitou zajišťuje, že malé změny v genotypu (např. malá Hammingova vzdálenost) vedou k malým změnám ve fenotypu. Dobrá lokalita je v genetických algoritmech vysoce žádoucí, protože umožňuje efektivní prohledávání okolí slibných řešení a jejich postupné zlepšování pomocí operátorů, jako je mutace a křížení. Pokud má reprezentace špatnou lokalitu, může být pro algoritmus obtížné dosáhnout konvergence k optimálnímu řešení, protože malé genetické změny mohou vést k náhlým a nepředvídatelným skokům ve fenotypovém prostoru.[19]

5.4.2 Lokalita standardního binárního kódování

Standardní binární kódování reálných čísel má obecně relativně dobrou lokalitu, zejména pro malé změny v méně významných bitech. Změna jediného bitu na pozici i (počítáno zprava od 0) způsobí změnu ve fenotypu o

$$\frac{b-a}{2^L-1} \cdot 2^i, \quad (5.3)$$

kde L je délka bitového řetězce a $[a, b]$ je reprezentovaný rozsah. Pro malé hodnoty i je tato změna malá. Nicméně, jak již bylo diskutováno, změna nejvýznamnějšího bitu může vést k velké změně, což představuje určitý nedostatek lokality pro velké skoky v genotypu.

5.4.3 Lokalita BCD reprezentace

BCD reprezentace může mít horší lokalitu ve srovnání se standardním binárním kódováním. Například přechod z desítkové 9 (1001) na 10 (0001 0000) vyžaduje změnu více bitů a může vést k relativně velké změně ve fenotypu, i když se jedná o sousední celá čísla. Podobně, změna jediného bitu v BCD může někdy vést k neplatnému kódu, což narušuje plynulý přechod mezi sousedními platnými fenotypy. Navíc, jak ukazuje Tabulka 5.1, Hammingova vzdálenost mezi sousedními desítkovými čísly v BCD může být značně proměnlivá, což indikuje nekonzistentní mapování malých změn v genotypu na malé změny ve fenotypu.

Experimentální vyhodnocení lokality lze provést měřením průměrné změny ve fenotypu

v důsledku jednobitové mutace pro různé binární reprezentace. Reprezentace s menší průměrnou změnou fenotypu pro jednobitovou mutaci lze považovat za mající lepší lokalitu. Dále lze analyzovat distribuci změn ve fenotypu pro různé Hammingovy vzdálenosti mezi genotypy.

5.4.4 Důsledky lokality pro genetické operátory

Lokalita reprezentace úzce souvisí s efektivitou genetických operátorů. Pokud má reprezentace dobrou lokalitu, pak operátory jako mutace a křížení, které typicky provádějí malé změny v genotypu, mají větší šanci na postupné zlepšování řešení. Naopak, u reprezentací se špatnou lokalitou mohou malé genetické změny vést k náhlým a potenciálně škodlivým změnám ve fenotypu, což ztěžuje konvergenci k optimu.

5.5 Metriky úspěšnosti optimalizace

Pro objektivní porovnání efektivity různých binárních reprezentací v genetickém algoritmu je nezbytné definovat vhodné metriky úspěšnosti optimalizace. Mezi běžně používané metriky patří:

- Nejlepší dosažená fitness: Hodnota fitness funkce nejlepšího jedince nalezeného během běhu algoritmu. Tato metrika udává, jak blízko se algoritmus dostal k optimálnímu řešení.
- Průměrná fitness populace: Průměrná hodnota fitness funkce všech jedinců v populaci na konci běhu algoritmu nebo v určitém počtu generací. Tato metrika indikuje celkovou kvalitu řešení v populaci.
- Úspěšnost: Procento běhů algoritmu, ve kterých bylo dosaženo předem definovaného cílového fitness (např. globálního optima s určitou tolerancí). Tato metrika je důležitá pro posouzení spolehlivosti algoritmu.
- Počet generací do konvergence: Počet generací, které algoritmus potřeboval k dosažení určité úrovně fitness nebo k ustálení populace (např. když se průměrná fitness přestane výrazně měnit). Tato metrika měří rychlost konvergence algoritmu.
- Výpočetní čas: Celkový čas potřebný k provedení algoritmu. Tato metrika je důležitá pro praktické aplikace, kde je časové omezení.

- Diverzita populace: Míra rozmanitosti genotypů v populaci. Vysoká diverzita může pomoci zabránit předčasné konvergenci, zatímco nízká diverzita může indikovat, že algoritmus uvízl v lokálním optimu. Diverzitu lze měřit různými způsoby, např. průměrnou Hammingovou vzdáleností mezi jedinci v populaci.

Při experimentálním vyhodnocování různých binárních reprezentací je důležité použít sadu těchto metrik, aby se získal komplexní obraz o jejich výkonnosti. Jedna reprezentace může dosahovat lepších výsledků v jedné metrice, zatímco jiná v jiné. Proto je nutné analyzovat výsledky z více perspektiv.

5.6 Statistické testy použité pro analýzu

Pro objektivní porovnání výkonnosti genetického algoritmu s různými binárními reprezentacemi na sadě složitých benchmarkových funkcí GNBG je nezbytné použít statistické testy k vyhodnocení, zda jsou pozorované rozdíly ve výsledcích statisticky významné, a nikoli pouze náhodné fluktuace. V této práci budou použity následující statistické testy:

- T-test pro dva nezávislé výběry: Tento test se použije k porovnání průměrných hodnot metriky úspěšnosti (např. nejlepší dosažená fitness, počet generací do konvergence) mezi dvěma různými binárními reprezentacemi. Předpokladem pro použití t-testu je normální rozdělení dat a rovnost rozptylů (kterou lze ověřit např. pomocí Levenova testu). Pokud tyto předpoklady nejsou splněny, použije se neparametrická alternativa, Mann-Whitney U test.
- ANOVA (Analýza rozptylu): Pokud se bude porovnávat výkonnost více než dvou variant binární reprezentace, použije se ANOVA. Ta testuje, zda existuje statisticky významný rozdíl mezi průměry skupin. Stejně jako u t-testu, i zde platí předpoklady o normalitě a rovnosti rozptylů. Pokud tyto předpoklady nejsou splněny, použije se neparametrická alternativa, Kruskal-Wallisův test.
- Chí-kvadrát test nezávislosti: Tento test se může použít k porovnání úspěšnosti (binární výsledek: úspěch/neúspěch) mezi různými reprezentacemi. Testuje, zda existuje statisticky významná závislost mezi typem reprezentace a dosažením cílového fitness.

Před provedením statistických testů bude pro každou variantu binární reprezentace provedeno více nezávislých běhů genetického algoritmu na sadě instancí benchmarkových funkcí

GNBG (Generalized Noisy Bi-Objective Generator), dostupných na [15]. Získané výsledky pro každou metriku budou následně statisticky analyzovány. Pro všechny statistické testy bude jako hladina významnosti (α) zvolena hodnota 0.05. To znamená, že rozdíl mezi výsledky bude považován za statisticky významný, pokud bude pravděpodobnost náhodného výskytu takového rozdílu menší než 5.

II PRAKTICKÁ ČÁST

6 IMPLEMENTACE EXPERIMENTU

V této sekci je podrobně popsána implementace experimentálního prostředí, ve kterém byla provedena srovnávací analýza různých reprezentací genetických algoritmů. Cílem bylo objektivně zhodnocení vlivu těchto reprezentací na výkon GA při řešení optimalizačních problémů definovaných sadou benchmarkových funkcí GNBG. Následující subsekcce rozebírají klíčové aspekty implementace, včetně použitého softwaru a nástrojů, konfigurace parametrů algoritmu, postupu analýzy citlivosti, metodiky stanovení délky chromozomu a technických detailů implementace jednotlivých binárních reprezentací.

6.1 Popis softwaru a použitých nástrojů

Klíčové knihovny a nástroje, které tvořily základ experimentálního prostředí, jsou popsány níže:

- **NumPy:** Tato knihovna byla nezbytná pro efektivní numerické výpočty a operace lineární algebry. NumPy nám umožnila pracovat s vícerozměrnými poli a provádět matematické operace potřebné pro implementaci genetických algoritmů a vyhodnocování benchmarkových funkcí.
- **SciPy:** Knihovna SciPy byla použita především pro dvě klíčové funkce:
 - Načítání dat z .mat souborů, ve kterých byly uloženy instance benchmarkových funkcí GNBG.
 - Provádění statistických testů, konkrétně Mann-Whitney U testu, který byl použit jako neparametrická alternativa k t-testu v případech, kdy nebyly splněny předpoklady normality dat.
- **Matplotlib:** Pro vizualizaci výsledků experimentů a usnadnění jejich interpretace jsme použili knihovnu Matplotlib. Tato knihovna nám umožnila vytvářet konvergenční grafy, grafy citlivosti parametrů a další typy grafů, které přehledně zobrazovaly výkonnost různých reprezentací.
- **pickle:** Pro ukládání a načítání výsledků experimentů jsme použili knihovnu pickle. Tato knihovna nám umožnila serializovat a deserializovat objekty Pythonu, což bylo užitečné pro ukládání průběžných i finálních výsledků a jejich následné zpracování.

- **Vlastní třídy:** Základem našeho experimentálního frameworku byly dvě klíčové třídy, které jsme sami implementovali:
 - GNBG: Tato třída sloužila k inicializaci a vyhodnocování benchmarkových funkcí GNBG. Zapouzdřovala veškerou logiku potřebnou pro výpočet fitness hodnoty pro dané řešení.
 - BinaryGeneticAlgorithm a RealValueGeneticAlgorithm: Tyto třídy implementovaly samotné genetické algoritmy. BinaryGeneticAlgorithm byla navržena pro práci s binárními reprezentacemi, zatímco RealValueGeneticAlgorithm byla použita pro reprezentaci reálných čísel. (Viz Výpis 6.1 pro příklad inicializace)

Následující výpis kódu ilustruje inicializaci genetického algoritmu s reprezentací fixed-point:

```
1 # Inicializace GA s fixed-point reprezentací
2 ga = BinaryGeneticAlgorithm(
3     representation='fixed-point', bits_per_dim=32,
4     population_size=100, dimension=10,
5     value_range=(-5.12, 5.12),
6     mutation_rate=0.01,
7     crossover_rate=0.8,
8     selection_pressure=2
9 )
10 ga.set_fitness_function(gnbg_instance)
11
12 genome, fitness = ga.evolve()
```

Výpis 6.1 Příklad inicializace třídy GA (zdroj: nedbal_BGA.py)

Výše uvedený výpis kódu ukazuje, jak se při inicializaci třídy BinaryGeneticAlgorithm nastavuje konkrétní reprezentace ('fixed-point') a další parametry, jako je počet bitů na dimenzi (bits_per_dim=32), a jak je algoritmus připraven k optimalizaci dané fitness funkce.

Obrázek 6.1 ukazuje typický průběh konvergence genetického algoritmu. Tento graf znázorňuje, jak se hodnota fitness nejlepšího jedince v populaci zlepšuje v průběhu generací, což ilustruje optimalizační proces.

Obrázek 6.1 Konvergence nejlepší fitness během počtu vyhodnocení.

6.2 Nastavení parametrů experimentu

Pro zajištění férového srovnání mezi různými binárními reprezentacemi jsme pro všechny experimenty použili jednotné nastavení klíčových parametrů genetického algoritmu. Toto jednotné nastavení nám umožnilo izolovat vliv samotné reprezentace na výkon algoritmu a minimalizovat vliv ostatních faktorů. Níže uvádíme hodnoty těchto parametrů:

- **Velikost populace (N):** 100 jedinců. Tato hodnota byla zvolena jako kompromis mezi prohledávací schopností algoritmu a výpočetní náročností.
- **Míra mutace (p_m):** 0,01. Pravděpodobnost, že se bit v chromozomu jedince změní. Tato hodnota byla zvolena na základě doporučení z literatury a předběžných experimentů.
- **Míra křížení (p_c):** 0,80. Pravděpodobnost, že dojde ke křížení mezi dvěma vybranými jedinci. Tato poměrně vysoká hodnota podporuje výměnu genetické informace a prozkoumávání prohledávacího prostoru.
- **Selektivní tlak (s):** 2. Tento parametr určuje, jak moc jsou preferováni jedinci s vyšší fitness. Hodnota 2 odpovídá lineárnímu škálování a poskytuje mírný selektivní tlak.
- **Počet opakování (R):** 31. Každý experiment byl proveden 31krát s různými počátečními populacemi, abychom získali robustnější statistické výsledky a minimalizovali vliv náhodných fluktuací.

Celkový počet vyhodnocení fitness funkce (FES) je klíčovým ukazatelem výpočetní náročnosti. V našem experimentu platí následující vztahy:

- Počet vyhodnocení na generaci (FES_{gen}) se rovná velikosti populace (N):

$$\text{FES}_{\text{gen}} = N \quad (6.1)$$

- Celkový počet vyhodnocení (FES_{total}) se vypočítá jako součin počtu opakování (R), počtu generací (G) a velikosti populace (N):

$$\text{FES}_{\text{total}} = R \times G \times N \quad (6.2)$$

Kde počet generací (G) je určen jako celočíselná část podílu maximálního počtu vyhodnocení (MaxEvals) a velikosti populace (N):

$$G = \lfloor \text{MaxEvals}/N \rfloor \quad (6.3)$$

6.3 Metodika nastavení délky chromozomu

V tomto experimentu byla délka chromozomu (L) určena na základě dimenze řešeného problému (D) a počtu bitů (b) použitých na reprezentaci jedné proměnné:

$$L = D \times b \quad (6.4)$$

Pro všechny testované číselné reprezentace (fixed-point a IEEE-754) a kódovací schéma (Grayův kód aplikovaný na binární řetězec) byl počet bitů na dimenzi pevně nastaven na hodnotu $b = 32$. Toto jednotné nastavení umožnilo konzistentní srovnání vlivu různých přístupů na výkon genetického algoritmu.

6.4 Implementace různých binárních reprezentací

V rámci experimentu jsme implementovali a porovnávali čtyři různé reprezentace chromozomu: fixed-point, Grayův kód, IEEE-754 a reprezentaci reálnými čísly. Každá z těchto reprezentací má své specifické vlastnosti, které ovlivňují způsob kódování a dekódování řešení optimalizačního problému.

6.4.1 Fixed-Point

Reprezentace fixed-point (s pevnou desetinnou čárkou) kóduje reálné proměnné pomocí celočíselné reprezentace. Pro převod reálné hodnoty na binární řetězec se hodnota nejprve normalizuje do určeného rozsahu, následně škáluje a nakonec převede na bitový řetězec. Přesnost této reprezentace je dána počtem bitů vyhrazených pro reprezentaci proměnné.

```

1 """
2 Převode reálnou hodnotu na binární řetězec pomocí fixed-point
  reprezentace.
3
4 Args:
5     value (float): Reálná hodnota k zakódování.
6     value_range (tuple): Rozsah hodnot (min, max).
7     bits_per_dim (int): Počet bitů na dimenzi.
8
9 Returns:
10    str: Binární řetězec reprezentující hodnotu.
```

11 """

Výpis 6.2 Převod na fixed-point (část 1 - definice)

```

1 def real_to_binary_fixed_point(value, value_range, bits_per_dim):
2     x_min, x_max = value_range
3     normalized_value = (value - x_min) / (x_max - x_min) #
4         Normalizace do [0, 1]
5     scaled_value = int(normalized_value * (2**bits_per_dim - 1)) #
6         Skalovani na rozsah celych cisel
7     return format(scaled_value, '0{}b'.format(bits_per_dim)) # Prevod
8         na bitovy retezec

```

Výpis 6.3 Převod na fixed-point (část 2 - implementace)

6.4.2 Grayův kód

Grayův kód je binární kódování, kde se sousední hodnoty liší pouze v jednom bitu. Tato vlastnost, známá jako minimální Hammingova vzdálenost, může být výhodná v genetických algoritmech, protože malé změny v chromozomu vedou k malým změnám v reprezentované hodnotě. Převod z binárního kódu (B) na Grayův kód (G) se provádí bitovou operací XOR s posunutou verzí binárního kódu:

$$G = B \oplus (B \gg 1) \quad (6.5)$$

6.4.3 IEEE-754

IEEE-754 je standard pro reprezentaci čísel s plovoucí desetinnou čárkou v počítačích. Tato reprezentace umožňuje reprezentovat velmi široký rozsah čísel s proměnlivou přesností. Pro převod mezi reálnými čísly a jejich binární reprezentací ve formátu IEEE-754 se v Pythonu používá modul 'struct'.

```

1 """
2 Prevede float na binarni retezec IEEE-754.
3 Args:
4     value (float): cislo s plovouci desetinou carkou.
5 Returns:
6     str: Binarni retezec IEEE-754 reprezentace.
7 """
8
9
10

```

Výpis 6.4 Převod na IEEE-754 (část 1 - definice)

```
1 import struct
2
3 def float_to_binary_ieee754(value):
4
5     return ''.join(bin(c).replace('0b', '').zfill(8) for c in
                      struct.pack('!f', value))
```

Výpis 6.5 Převod na IEEE-754 (část 2 - implementace)

6.4.4 Real Value

Kromě binárních reprezentací byla jako srovnávací základ použita i reprezentace reálnými čísly. V této reprezentaci je každá proměnná problému přímo reprezentována svou reálnou hodnotou, bez nutnosti kódování do binárního řetězce. Tento přístup je intuitivní, ale v tradičních implementacích genetických algoritmů se používá méně často.

7 PROVEDENÍ EXPERIMENTU

Tato sekce popisuje postup provedení experimentů, včetně detailů testování jednotlivých variant genetického algoritmu a metod sběru a analýzy dat. Cílem bylo systematické porovnání výkonnosti různých reprezentací chromozomu při řešení optimalizačních problémů.

Hlavní experiment jsem provedl pro čtyři přístupy k reprezentaci: Fixed-Point, IEEE 754, Reálná Hodnota a Grayův kód, což je specifické kódovací schéma pro binární reprezentace. Tyto metody jsem aplikoval na 24 benchmarkových funkcí, přičemž každou funkci jsem testoval v 31 bžích. Následně jsem přidal dodatečný experiment na třech vybraných funkcích benchmarku (f1, f7, f16) s cílem demonstrovat jejich konvergenci na grafech.

Experimenty byly prováděny primárně v lokálním prostředí na osobním počítači a jejich realizace trvala přibližně dva měsíce. Zpočátku se vyskytly obtíže spojené s persistencí dat. Původní implementace experimentálního kódu nezahrnovala průběžné ukládání výsledků, což představovalo riziko ztráty dat v případě předčasného přerušení běhu před dokončením všech 31 opakování.

Vzhledem k značné časové náročnosti experimentů a potenciální ztrátě dat, například v důsledku neočekávaného vypnutí počítače, byla tato funkcionality doplněna. Následně bylo zajištěno automatické ukládání všech relevantních výsledků do souborů ve formátu .pk1 po každém dokončeném běhu. Krátce poté bylo při kontrole zjištěno, že časy délky jednotlivých běhů byly vypisovány pouze do konzole. Tato opomenutá data byla následně rovněž implementována do .pk1 souborů pro kompletní záznam.

Původně bylo zamýšleno ukládat výsledky ve dvou redundantních lokacích: lokálně na osobním počítači a na školním serveru. Bohužel, školní server prošel restartem, což vedlo ke ztrátě všech experimentálních dat, která na něm byla uložena. Z tohoto důvodu veškeré prezentované výsledky v této práci pocházejí výhradně z experimentů provedených v lokálním prostředí.

7.1 Proces testování jednotlivých variant

Pro analýzu byly shromážděny výsledky experimentů uložené ve čtyřech samostatných souborech ve formátu .pk1, přičemž každý soubor odpovídal jedné testované metodě (reprezentaci chromozomu). Každý z těchto čtyř souborů obsahuje 744 jednotlivých záznamů, což

po jejich sloučení dohromady poskytuje celkem 2976 záznamů. Každý záznam reprezentuje jeden dokončený běh genetického algoritmu pro specifickou kombinaci funkce a reprezentace.

Příklad struktury jednoho záznamu je následující:

```
Záznam 0: {'run_id': 0, 'instance': 'f1', 'representation': 'real-value',  
          'dimension': 30, 'final_error': 0.0004195789117602544,  
          'fes_to_threshold': 500000, 'success': False,  
          'best_fitness_found': -1081.9833798214281,  
          'max_fes_used': 500000}
```

Zde `run_id` identifikuje konkrétní běh, `instance` odkazuje na testovanou optimalizační funkci (např. 'f1'), `representation` specifikuje použitou reprezentaci chromozomu (např. 'real-value'), `dimension` udává dimenzi problému. `final_error` představuje konečnou chybu dosaženou algoritmem, `fes_to_threshold` uvádí počet vyhodnocení fitness funkce (Fitness Evaluations) potřebných k dosažení definované prahové hodnoty (pokud byla dosažena), `success` indikuje, zda byl běh považován za úspěšný (splnil kritéria konvergence), `best_fitness_found` je nejlepší nalezená hodnota fitness funkce a `max_fes_used` udává maximální počet vyhodnocení fitness funkce, který byl během běhu použit. Sloučením těchto záznamů je získán komplexní dataset pro podrobnou statistickou analýzu a porovnání výkonnosti jednotlivých variant genetického algoritmu. Test probíhal na funkcích s následujícími vlastnostmi:

Characteristic	Problem instances																							
	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}	f_{21}	f_{22}	f_{23}	f_{24}
Modality	Unimodal												Multimodal with multiple competing components											
Basin local optima*	X	X	X	X	X	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	X	X	✓	✓	✓	✓	✓	✓	✓
Separability	$\mathcal{F}^\dagger$	$\mathcal{F}$	$\mathcal{F}$	$\mathcal{N}^\dagger$	$\mathcal{N}$	$\mathcal{N}$	$\mathcal{F}$	$\mathcal{F}$	$\mathcal{F}$	$\mathcal{F}$	$\mathcal{N}$	$\mathcal{P}^\dagger$	$\mathcal{N}$	$\mathcal{N}$	$\mathcal{N}$	$\mathcal{N}$	$\mathcal{N}$	$\mathcal{N}$	$\mathcal{N}$	$\mathcal{N}$	$\mathcal{N}$	$\mathcal{N}$	$\mathcal{N}$	$\mathcal{N}$
Varying variable interactions	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	✓	✓	✓	✓	✓	✓	✓	✓
Symmetry	$\mathcal{S}^\dagger$	$\mathcal{S}$	$\mathcal{S}$	$\mathcal{S}$	$\mathcal{S}$	$\mathcal{S}$	$\mathcal{S}$	$\mathcal{S}$	$\mathcal{S}$	$\mathcal{A}^\dagger$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{S}$	$\mathcal{A}$	$\mathcal{S}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$
Ill-conditioning	X	X	✓	X	✓	✓	X	X	X	X	X	X	X	✓	✓	X	✓	X	X	X	X	X	X	X
Basin linearity	$\mathcal{E}^*$	$\mathcal{L}^*$	$\mathcal{E}$	$\mathcal{E}$	$\mathcal{L}$	$\mathcal{L}$	$\mathcal{E}$	$\mathcal{E}$	$\mathcal{E}$	$\mathcal{E}$	$\mathcal{E}$	$\mathcal{E}$	$\mathcal{E}$	$\mathcal{E}$	$\mathcal{L}$	$\mathcal{E}$	$\mathcal{E}$	$\mathcal{E}$	$\mathcal{E}$	$\mathcal{L}$	$\mathcal{L}^*$	$\mathcal{E}$	$\mathcal{L}$	$\mathcal{L}$
Deceptive	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	✓	✓	✓	✓	✓	✓	✓	X

* Existence of local optima within the basin of each component.

† $\mathcal{F}$, $\mathcal{N}$ and $\mathcal{P}$ stand for fully separable, non-separable, and partially separable, respectively.

‡ $\mathcal{S}$ and $\mathcal{A}$ stand for symmetric and asymmetric, respectively.

* $\mathcal{E}$, $\mathcal{L}$, and $\mathcal{L}$ stand for super-linear, sub-linear, and linear, respectively.

Obrázek 7.1 Tabulka vlastností funkcí pro benchmark

(zdroj: [16])

Kromě primárního cíle porovnání výkonnosti různých reprezentací bylo rovněž cílem demonstrovat konvergenci genetického algoritmu v průběhu času a analyzovat časovou náročnost jednotlivých metod. Pro tento účel byly vybrány tři reprezentativní funkce: f_1 (unimodální), f_7 (multimodální s jednou komponentou) a f_{16} (multimodální s více komponentami). Tyto funkce byly spuštěny znovu, tentokrát s detailním záznamem průběhu konvergence a měřením času.

Záznamy z těchto dodatečných běhů obsahovaly rozšířené informace, které umožňovaly vizualizaci zlepšování fitness funkce v závislosti na počtu vyhodnocení fitness funkce (FES) a zároveň sledování časového průběhu. Příklad jednoho takového záznamu je následující:

```
Záznam 0: {'run_id': 0, 'instance': 'f1', 'representation': 'gray',
           'dimension': 30, 'final_error': 38.8499219433927,
           'fes_to_threshold': 500000, 'success': False,
           'best_fitness_found': -1043.1338774569472, 'max_fes_used':
           500000, 'best_fitness_history': [70508.19648392842,
           70508.19648392842, ...], 'optimal_value':
           -1081.9837994003399, 'status': 'Completed',
           'improvement_fes_points': [100, 300, 400, ...],
           'improvement_fitness_values': [70508.19648392842,
           59362.04204535508, ...]}
```

Klíčovými novými položkami v těchto záznamech jsou:

- `best_fitness_history`: Seznam hodnot nejlepší fitness nalezené v každé generaci, což umožňuje vizualizaci konvergenční křivky.
- `optimal_value`: Předem známá optimální hodnota pro danou testovací funkci, sloužící jako referenční bod pro posouzení kvality řešení.
- `status`: Indikuje stav dokončení běhu (např. 'Completed').
- `improvement_fes_points`: Seznam počtů FES, při kterých došlo ke zlepšení nalezené fitness hodnoty.
- `improvement_fitness_values`: Seznam nejlepších fitness hodnot odpovídajících bodům v `improvement_fes_points`.

Tyto rozšířené záznamy umožňují hlubší analýzu dynamiky konvergence algoritmu a poskytují data pro vizuální porovnání rychlosti a efektivity různých reprezentací při řešení optimalizačních problémů na vybraných funkcích. Zároveň je na těchto datech demonstrována i rozdílná časová náročnost jednotlivých metod, jelikož záznamy `improvement_fes_points`

a `improvement_fitness_values` nepřímo reflektují, kolik výpočetního úsilí bylo vynaloženo na dosažení konkrétního zlepšení.

8 VYHODNOCENÍ VÝSLEDKŮ

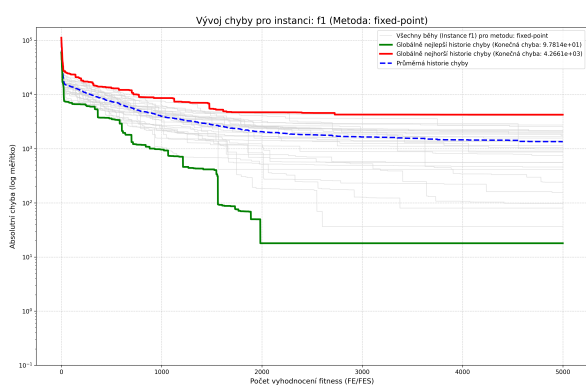
Tato kapitola je věnována prezentaci výsledků experimentálního šetření. Prezentace bude zahrnovat grafické vizualizace konvergenčních charakteristik a tabulkové souhrny klíčových metrik, které umožní komplexní posouzení výkonnosti a chování testovaných algoritmů s různými reprezentacemi.

8.1 Konvergenční charakteristiky pro běh na třech funkcích

Primárním cílem této sekce je grafická prezentace konvergenčních charakteristik algoritmů, která slouží k vizuálnímu posouzení dynamiky optimalizačního procesu a vhodnosti jednotlivých reprezentací. Tyto grafy poskytují náhled na to, jak efektivně algoritmy konvergují k optimálním řešením a jak se jejich výkonnost vyvíjí v průběhu přiděleného počtu vyhodnocení fitness funkcí.

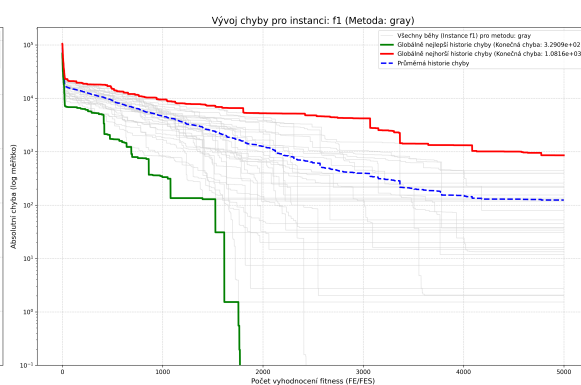
8.1.1 Srovnání pro jednotlivé běhy

V úvodní fázi byla provedena analýza sady 31 nezávislých běhů na zvolené sadě benchmarkových funkcí. Pro vizualizaci konvergenčních charakteristik byly graficky prezentovány trajektorie všech individuálních běhů, doplněné o křivky reprezentující globálně nejlepší dosaženou historii chyby, globálně nejhorší historii chyby a průměrnou historii chyby napříč všemi běhy.



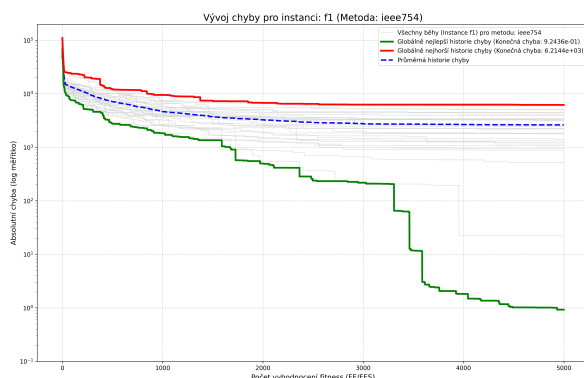
Obrázek 8.1 Grafické zobrazení vývoje chyby pro f1 u reprezentace fixed-point

(zdroj: vlastní program)



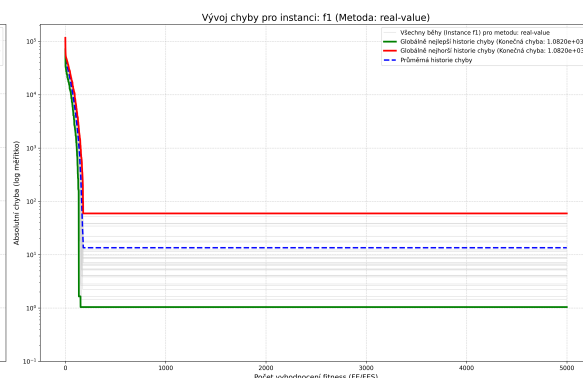
Obrázek 8.2 Grafické zobrazení vývoje chyby pro f1 u reprezentace gray

(zdroj: vlastní program)



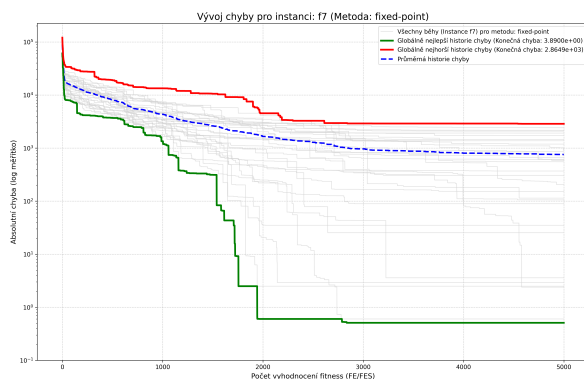
Obrázek 8.3 Grafické zobrazení vývoje chyby pro f1 u reprezentace ieee754

(zdroj: vlastní program)



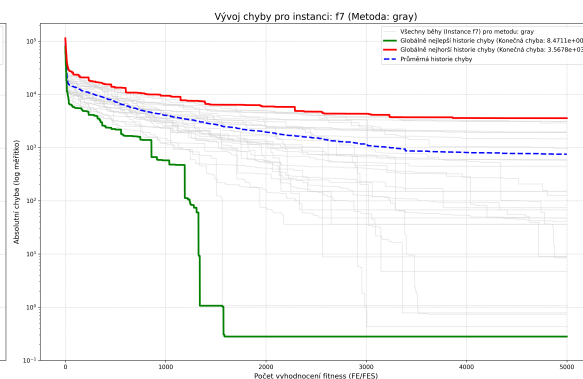
Obrázek 8.4 Grafické zobrazení vývoje chyby pro f1 u reprezentace real-value

(zdroj: vlastní program)



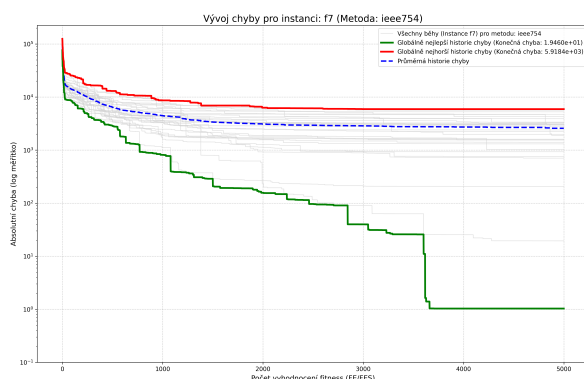
Obrázek 8.5 Grafické zobrazení vývoje chyby pro f7 u reprezentace fixed-point

(zdroj: vlastní program)



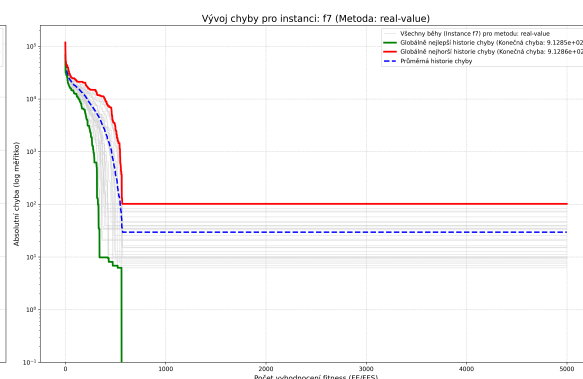
Obrázek 8.6 Grafické zobrazení vývoje chyby pro f7 u reprezentace gray

(zdroj: vlastní program)



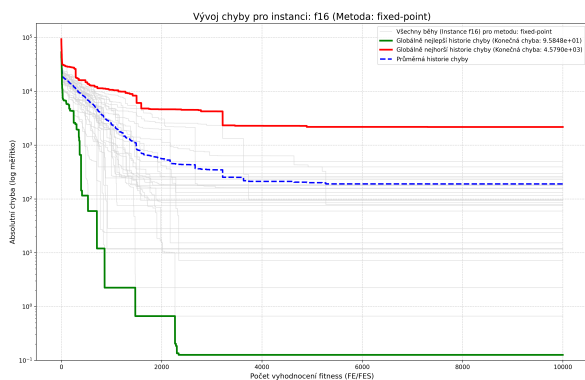
Obrázek 8.7 Grafické zobrazení vývoje chyby pro f7 u reprezentace ieee754

(zdroj: vlastní program)



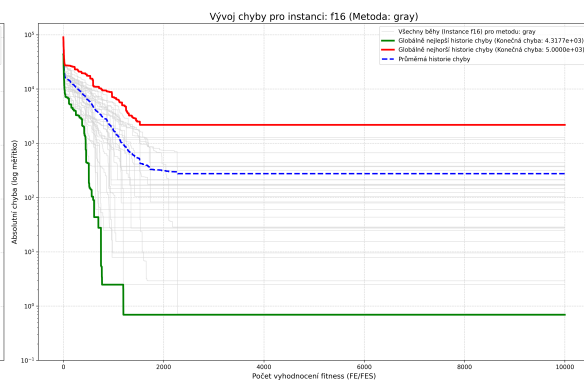
Obrázek 8.8 Grafické zobrazení vývoje chyby pro f7 u reprezentace real-value

(zdroj: vlastní program)



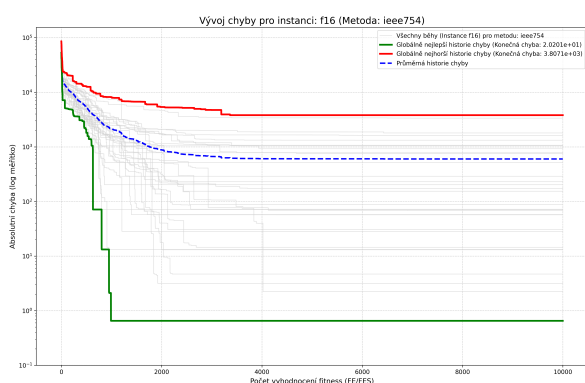
Obrázek 8.9 Grafické zobrazení vývoje chyby pro f16 u reprezentace fixed-point

(zdroj: vlastní program)



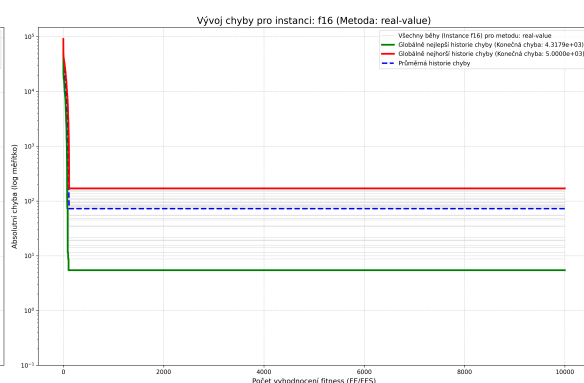
Obrázek 8.10 Grafické zobrazení vývoje chyby pro f16 u reprezentace gray

(zdroj: vlastní program)



Obrázek 8.11 Grafické zobrazení vývoje chyby pro f16 u reprezentace ieee754

(zdroj: vlastní program)

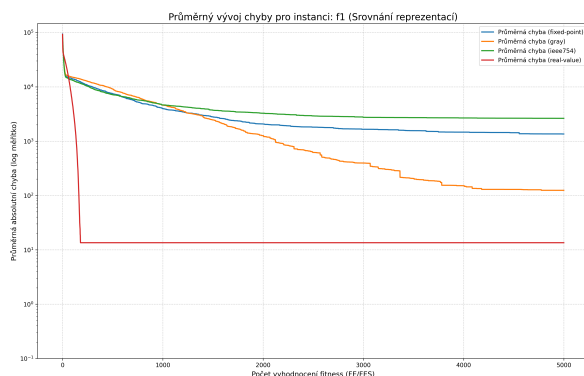


Obrázek 8.12 Grafické zobrazení vývoje chyby pro f16 u reprezentace real-value

(zdroj: vlastní program)

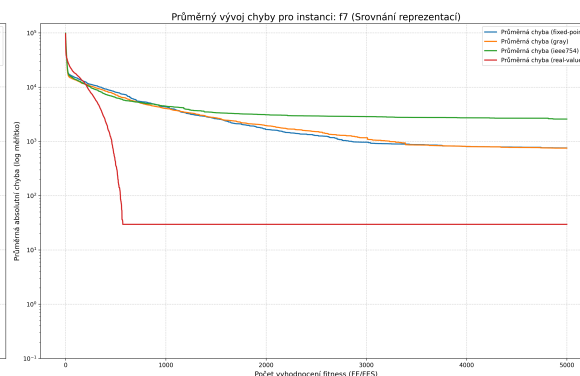
8.1.2 Srovnání variant mezi sebou

Tato subsekce se zaměřuje na komparativní analýzu výkonnosti jednotlivých variant algoritmu. Srovnání je provedeno jak na základě průměrných konvergenčních charakteristik každé varianty, tak i pomocí vzájemného porovnání jejich nejlepších dosažených výsledků (konečných chyb) v rámci testovací sady funkcí.



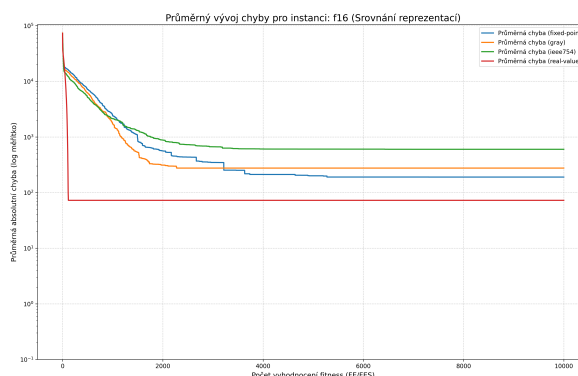
Obrázek 8.13 Grafické zobrazení průměrného vývoje chyby pro f1 (srovnání reprezentací)

(zdroj: vlastní program)



Obrázek 8.14 Grafické zobrazení průměrného vývoje chyby pro f7 (srovnání reprezentací)

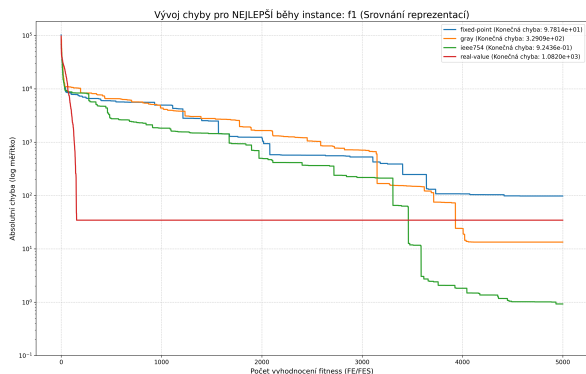
(zdroj: vlastní program)



Obrázek 8.15 Grafické zobrazení průměrného vývoje chyby pro f16 (srovnání reprezentací)

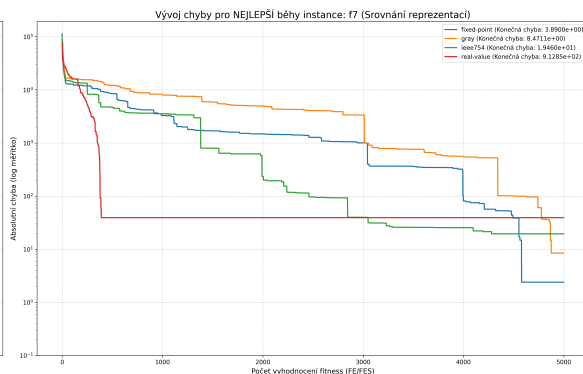
(zdroj: vlastní program)

Experimentální nastavení zahrnovalo provedení 31 nezávislých běhů pro každou testovanou funkci a reprezentaci. Optimalizace probíhala ve 30 dimenzích. Je důležité poznamenat, že žádná z testovaných funkcí nedosáhla optimální hodnoty během přiděleného počtu vyhodnocení fitness. Nastavení FES bylo převzato z benchmarku pro f1 a f7 je 500000 a pro f16 je 1000000.



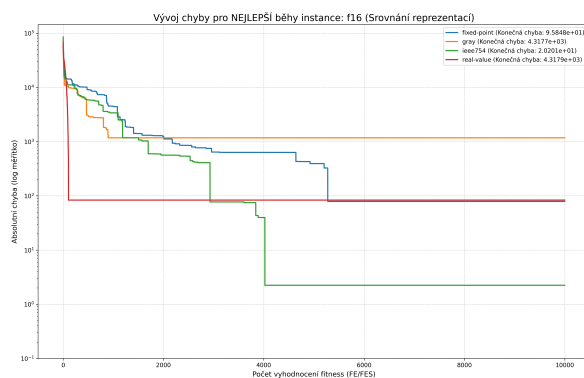
Obrázek 8.16 Grafické zobrazení vývoje chyby pro nejlepší běhy variant u f1 (srovnání reprezentací)

(zdroj: vlastní program)



Obrázek 8.17 Grafické zobrazení vývoje chyby pro nejlepší běhy variant u f7 (srovnání reprezentací)

(zdroj: vlastní program)



Obrázek 8.18 Grafické zobrazení vývoje chyby pro nejlepší běhy variant u f16 (srovnání reprezentací)

(zdroj: vlastní program)

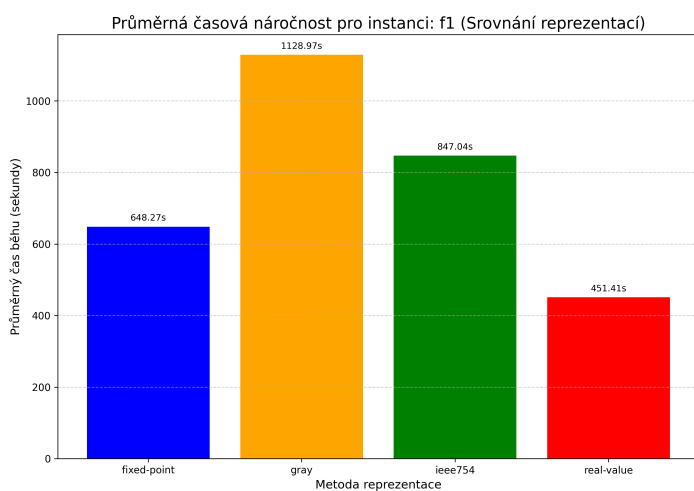
Tabulka 8.1 Statistiky Konečné Chyby (Min, Max, Průměr, Směrodatná odchylka)

instance	representation	min_error	max_error	mean_error	std_error
f1	fixed-point	9.7814×10^1	4.2661×10^3	1.3932×10^3	1.0593×10^3
f1	gray	3.2909×10^2	1.0816×10^3	9.6650×10^2	1.8953×10^2
f1	ieee754	9.2436×10^{-1}	6.2144×10^3	2.6288×10^3	1.6860×10^3
f1	real-value	1.0820×10^3	1.0820×10^3	1.0820×10^3	1.3971×10^{-4}
f7	fixed-point	3.8900×10^0	2.8649×10^3	8.0463×10^2	8.0040×10^2
f7	gray	8.4711×10^0	3.5678×10^3	9.4340×10^2	1.0237×10^3
f7	ieee754	1.9460×10^1	5.9184×10^3	2.5826×10^3	1.7006×10^3
f7	real-value	9.1285×10^2	9.1286×10^2	9.1286×10^2	6.6926×10^{-4}
f16	fixed-point	9.5848×10^1	4.5790×10^3	2.0642×10^3	1.1953×10^3
f16	gray	4.3177×10^3	5.0000×10^3	4.4576×10^3	2.1815×10^2
f16	ieee754	2.0201×10^1	3.8071×10^3	1.4280×10^3	1.2415×10^3
f16	real-value	4.3179×10^3	5.0000×10^3	4.3571×10^3	1.2455×10^2

(zdroj: vlastní program)

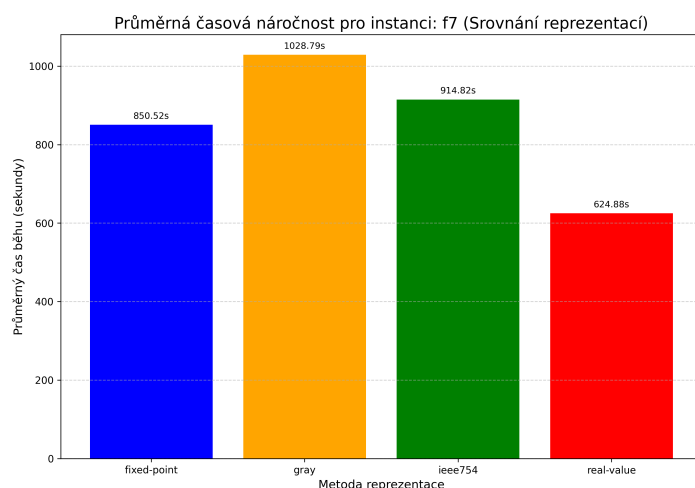
8.2 Výpočetní náročnost různých reprezentací

Tato subsekce se zaměřuje na analýzu výpočetní náročnosti testovaných algoritmů s různými reprezentacemi. Bude provedeno srovnání průměrných časových náročností jednotlivých běhů. Je však nezbytné zdůraznit, že prezentované časové údaje mohou být ovlivněny režijními náklady spojenými s persistentním zaznamenáváním dat pro účely vizualizace konvergenčních grafů, což by mělo být vzato v úvahu při interpretaci absolutních hodnot.



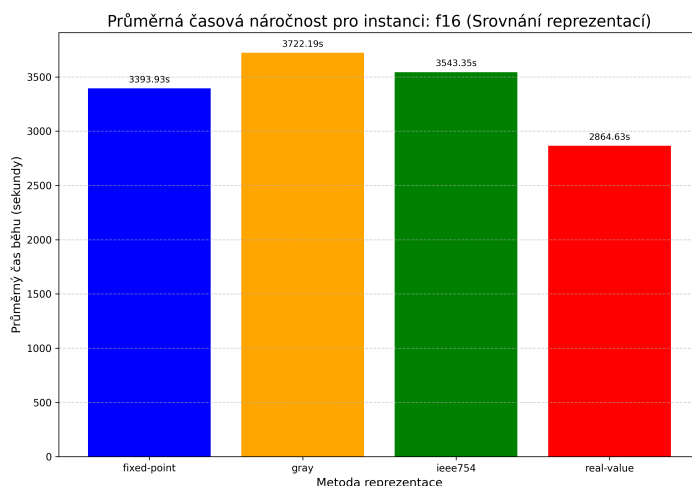
Obrázek 8.19 Grafické zobrazení průměrné časové náročnosti pro varianty u f1

(zdroj: vlastní program)



Obrázek 8.20 Grafické zobrazení průměrné časové náročnosti pro varianty u f7

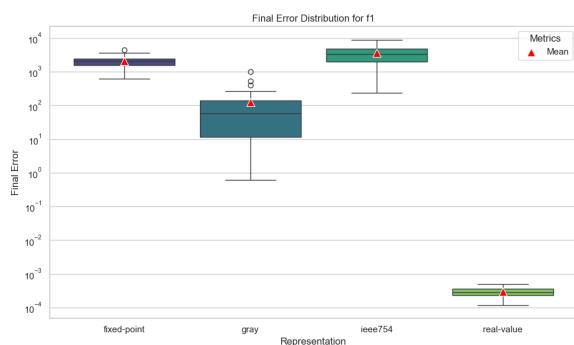
(zdroj: vlastní program)



Obrázek 8.21 Grafické zobrazení průměrné časové náročnosti pro varianty u f16

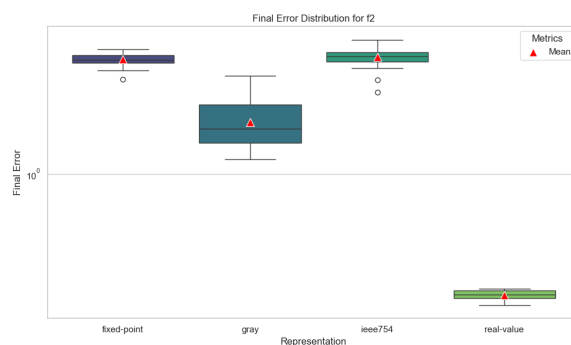
(zdroj: vlastní program)

8.3 Grafické zobrazení Final Error pro všechny funkce



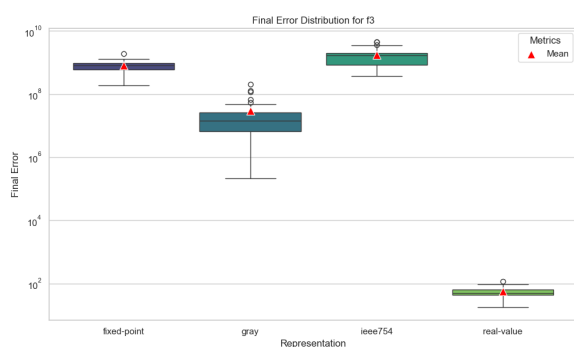
Obrázek 8.22 Grafické zobrazení final error pro f1

(zdroj: vlastní program)



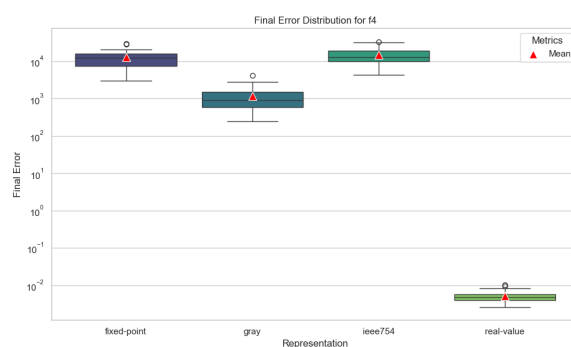
Obrázek 8.23 Grafické zobrazení final error pro f2

(zdroj: vlastní program)



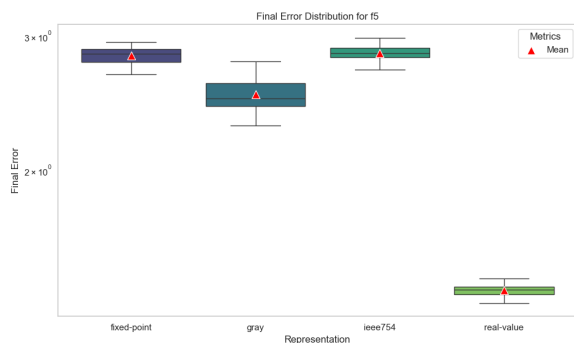
Obrázek 8.24 Grafické zobrazení final error pro f3

(zdroj: vlastní program)



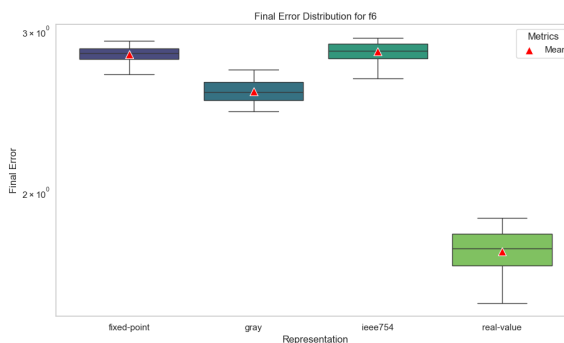
Obrázek 8.25 Grafické zobrazení final error pro f4

(zdroj: vlastní program)



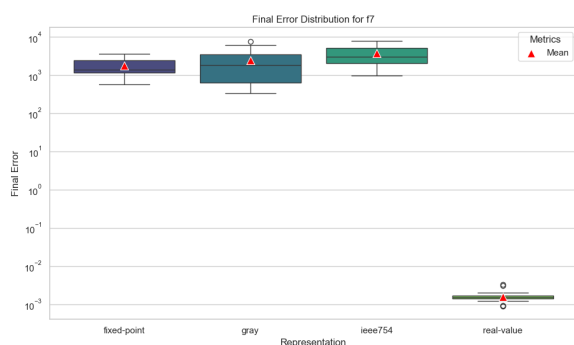
Obrázek 8.26 Grafické zobrazení final error pro f5

(zdroj: vlastní program)



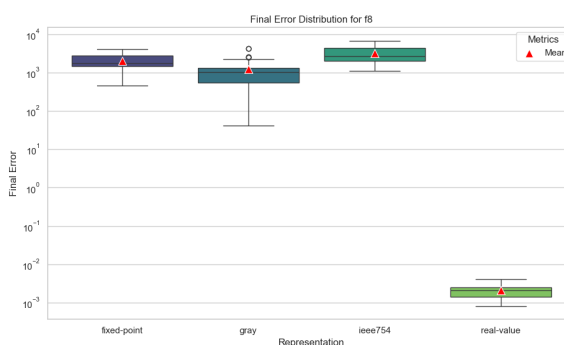
Obrázek 8.27 Grafické zobrazení final error pro f6

(zdroj: vlastní program)



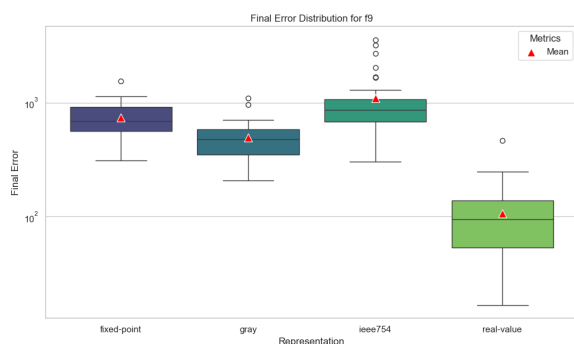
Obrázek 8.28 Grafické zobrazení final error pro f7

(zdroj: vlastní program)



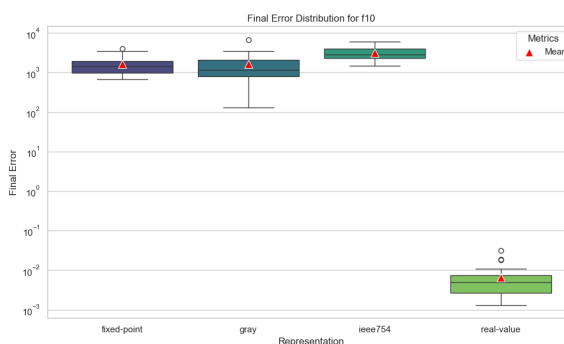
Obrázek 8.29 Grafické zobrazení final error pro f8

(zdroj: vlastní program)



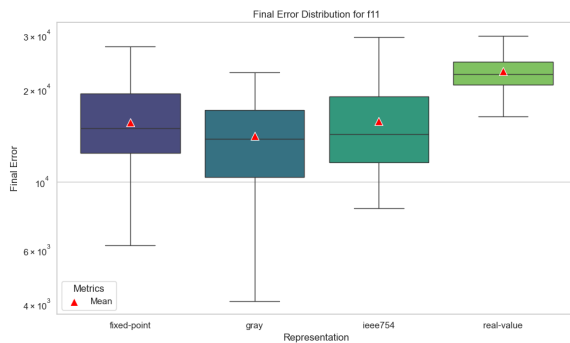
Obrázek 8.30 Grafické zobrazení final error pro f9

(zdroj: vlastní program)



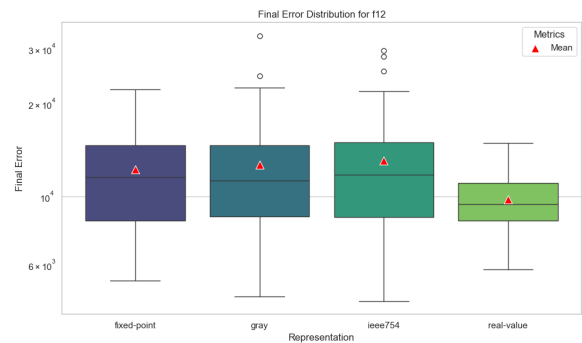
Obrázek 8.31 Grafické zobrazení final error pro f10

(zdroj: vlastní program)



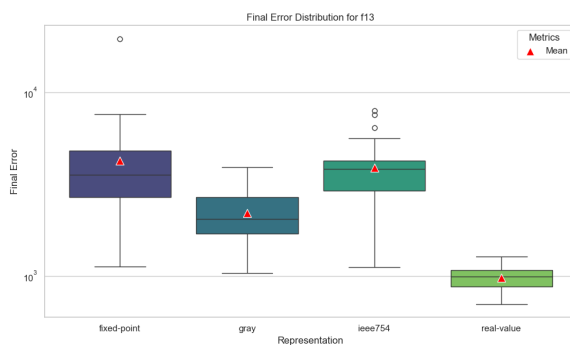
Obrázek 8.32 Grafické zobrazení final error pro f11

(zdroj: vlastní program)



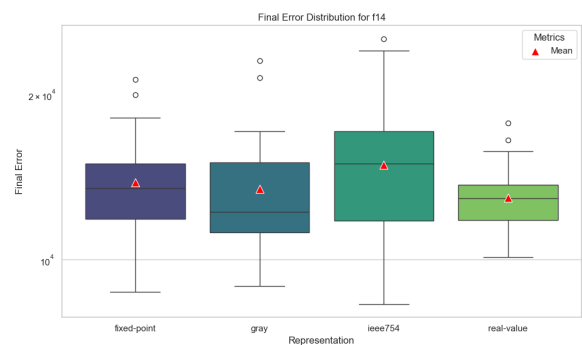
Obrázek 8.33 Grafické zobrazení final error pro f12

(zdroj: vlastní program)



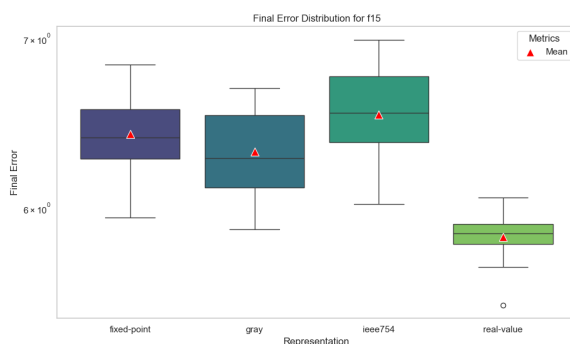
Obrázek 8.34 Grafické zobrazení final error pro f13

(zdroj: vlastní program)



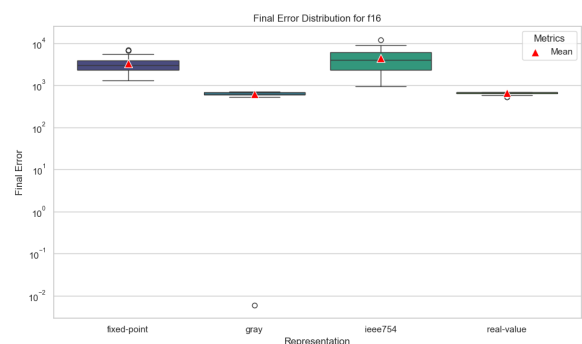
Obrázek 8.35 Grafické zobrazení final error pro f14

(zdroj: vlastní program)



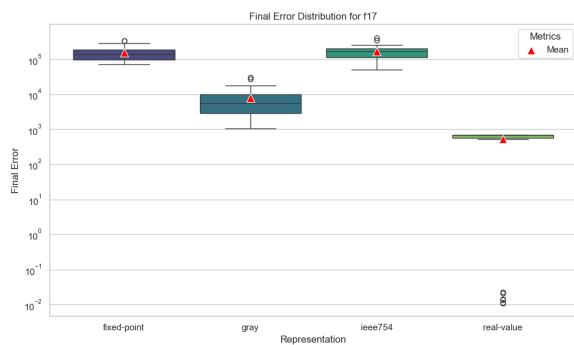
Obrázek 8.36 Grafické zobrazení final error pro f15

(zdroj: vlastní program)



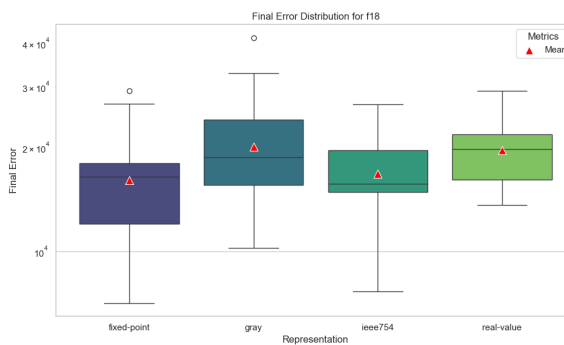
Obrázek 8.37 Grafické zobrazení final error pro f16

(zdroj: vlastní program)



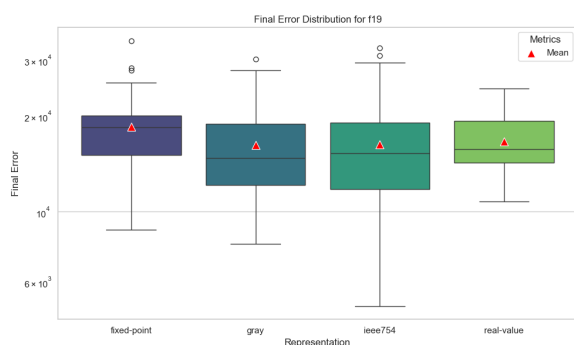
Obrázek 8.38 Grafické zobrazení final error pro f17

(zdroj: vlastní program)



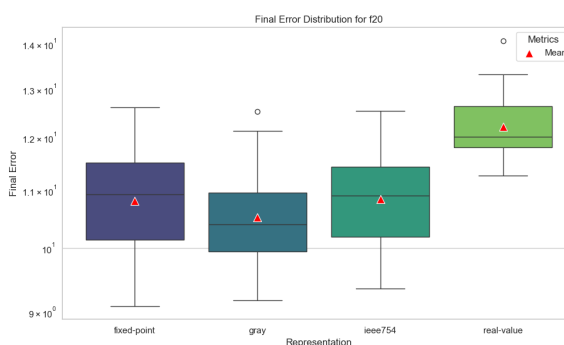
Obrázek 8.39 Grafické zobrazení final error pro f18

(zdroj: vlastní program)



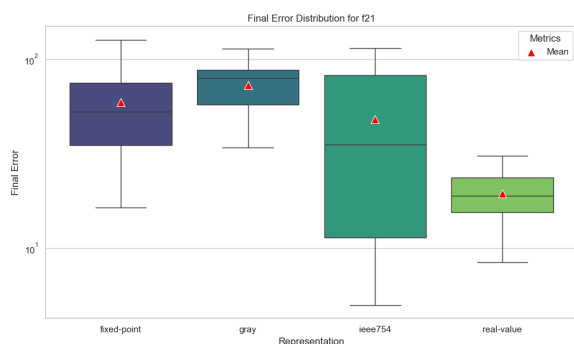
Obrázek 8.40 Grafické zobrazení final error pro f19

(zdroj: vlastní program)



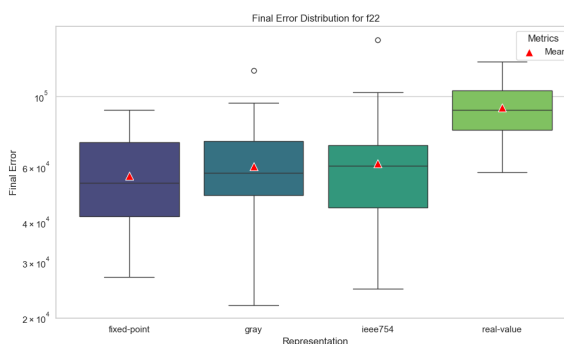
Obrázek 8.41 Grafické zobrazení final error pro f20

(zdroj: vlastní program)



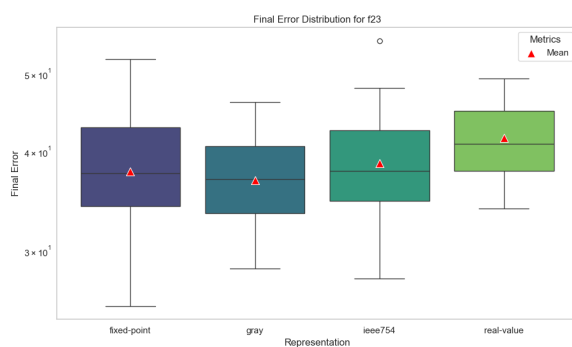
Obrázek 8.42 Grafické zobrazení final error pro f21

(zdroj: vlastní program)



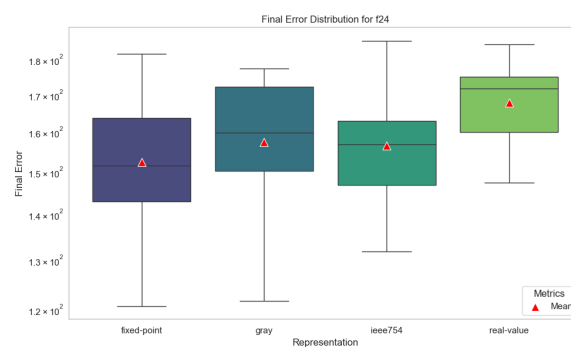
Obrázek 8.43 Grafické zobrazení final error pro f22

(zdroj: vlastní program)



Obrázek 8.44 Grafické zobrazení final error pro f23

(zdroj: vlastní program)



Obrázek 8.45 Grafické zobrazení final error pro f24

(zdroj: vlastní program)

8.4 Statická analýza

Bohužel se žádnému běhu nepodařilo dosáhnout úspěchu. Všechny běhy běžely do maximálního FES, což bylo pro f1 až f15 500000 a pro f16 až f24 1000000.

Tabulka 8.2 Statistika Final Error (Min, Max, Průměr, Směrodatná odchylka) pro různé reprezentace pro unimodální funkce

Inst	Reprezentace	Min Error	Max Error	Mean Error	Std Error	Count
f1	fixed-point	6.2990×10^2	4.4375×10^3	2.0655×10^3	9.2617×10^2	31
f1	gray	6.0790×10^{-1}	9.9574×10^2	1.2300×10^2	2.0103×10^2	31
f1	ieee754	2.3378×10^2	8.8610×10^3	3.6241×10^3	2.0776×10^3	31
f1	real-value	1.1812×10^{-4}	4.9758×10^{-4}	2.8860×10^{-4}	8.3847×10^{-5}	31
f2	fixed-point	1.3794×10^0	1.5258×10^0	1.4749×10^0	3.2449×10^{-2}	31
f2	gray	1.0506×10^0	1.3943×10^0	1.1908×10^0	9.7153×10^{-2}	31
f2	ieee754	1.3195×10^0	1.5770×10^0	1.4872×10^0	5.3939×10^{-2}	31
f2	real-value	6.4015×10^{-1}	6.7655×10^{-1}	6.6208×10^{-1}	1.0278×10^{-2}	31
f3	fixed-point	1.9010×10^8	1.8715×10^9	8.0993×10^8	3.4881×10^8	31
f3	gray	2.1741×10^5	2.0448×10^8	3.0020×10^7	4.4415×10^7	31
f3	ieee754	3.7758×10^8	4.5011×10^9	1.7353×10^9	1.1535×10^9	31
f3	real-value	1.8149×10^1	1.1925×10^2	5.6771×10^1	2.0397×10^1	31
f4	fixed-point	3.0055×10^3	2.9991×10^4	1.2854×10^4	6.5875×10^3	31
f4	gray	2.4700×10^2	4.2077×10^3	1.1548×10^3	8.8527×10^2	31
f4	ieee754	4.3082×10^3	3.3104×10^4	1.5092×10^4	7.8301×10^3	31
f4	real-value	2.6162×10^{-3}	1.0394×10^{-2}	5.2236×10^{-3}	1.9036×10^{-3}	31
f5	fixed-point	2.6695×10^0	2.9452×10^0	2.8266×10^0	7.8276×10^{-2}	31
f5	gray	2.2884×10^0	2.7769×10^0	2.5146×10^0	1.3126×10^{-1}	31
f5	ieee754	2.7103×10^0	2.9847×10^0	2.8509×10^0	6.9071×10^{-2}	31
f5	real-value	1.3362×10^0	1.4383×10^0	1.3880×10^0	2.3470×10^{-2}	31
f6	fixed-point	2.6883×10^0	2.9270×10^0	2.8275×10^0	5.9886×10^{-2}	31
f6	gray	2.4492×10^0	2.7206×10^0	2.5779×10^0	7.5358×10^{-2}	31
f6	ieee754	2.6607×10^0	2.9492×10^0	2.8487×10^0	7.1768×10^{-2}	31
f6	real-value	1.5137×10^0	1.8744×10^0	1.7221×10^0	9.6931×10^{-2}	31

(zdroj: vlastní program)

Tabulka 8.3 Statistika Final Error (Min, Max, Průměr, Směrodatná odchylka) pro různé reprezentace pro multimodální funkce s jedním komponentem

Inst	Reprezentace	Min Error	Max Error	Mean Error	Std Error	Count
f7	fixed-point	5.7878×10^2	3.6083×10^3	1.7651×10^3	9.4732×10^2	31
f7	gray	3.3120×10^2	7.7296×10^3	2.4157×10^3	1.9178×10^3	31
f7	ieee754	9.7195×10^2	7.9323×10^3	3.6868×10^3	2.0685×10^3	31
f7	real-value	8.9565×10^{-4}	3.2668×10^{-3}	1.5843×10^{-3}	4.9672×10^{-4}	31
f8	fixed-point	4.5258×10^2	4.0858×10^3	2.0095×10^3	8.7803×10^2	31
f8	gray	4.1810×10^1	4.2852×10^3	1.2524×10^3	1.0479×10^3	31
f8	ieee754	1.0980×10^3	6.7596×10^3	3.2205×10^3	1.5423×10^3	31
f8	real-value	8.1380×10^{-4}	4.1326×10^{-3}	2.0849×10^{-3}	8.3289×10^{-4}	31
f9	fixed-point	3.1161×10^2	1.5511×10^3	7.4345×10^2	2.5630×10^2	31
f9	gray	2.0792×10^2	1.0999×10^3	4.9252×10^2	2.0307×10^2	31
f9	ieee754	3.0341×10^2	3.5674×10^3	1.1007×10^3	7.8857×10^2	31
f9	real-value	1.6769×10^1	4.6884×10^2	1.0724×10^2	8.6802×10^1	31
f10	fixed-point	6.8009×10^2	4.0162×10^3	1.6427×10^3	7.8875×10^2	31
f10	gray	1.3035×10^2	6.7310×10^3	1.6389×10^3	1.3117×10^3	31
f10	ieee754	1.5038×10^3	6.0150×10^3	3.1721×10^3	1.2317×10^3	31
f10	real-value	1.3107×10^{-3}	3.0934×10^{-2}	6.5058×10^{-3}	6.2670×10^{-3}	31
f11	fixed-point	6.1995×10^3	2.7563×10^4	1.5572×10^4	5.5717×10^3	31
f11	gray	4.0940×10^3	2.2687×10^4	1.4110×10^4	4.7175×10^3	31
f11	ieee754	8.1891×10^3	2.9524×10^4	1.5744×10^4	5.5772×10^3	31
f11	real-value	1.6279×10^4	2.9815×10^4	2.2847×10^4	3.2982×10^3	31
f12	fixed-point	5.3406×10^3	2.2159×10^4	1.2206×10^4	4.9468×10^3	31
f12	gray	4.7449×10^3	3.3017×10^4	1.2625×10^4	6.1429×10^3	31
f12	ieee754	4.5826×10^3	2.9592×10^4	1.3016×10^4	6.4053×10^3	31
f12	real-value	5.8075×10^3	1.4846×10^4	9.7542×10^3	2.2410×10^3	31
f13	fixed-point	1.1282×10^3	1.9580×10^4	4.2522×10^3	3.2833×10^3	31
f13	gray	1.0360×10^3	3.9195×10^3	2.2103×10^3	7.6472×10^2	31
f13	ieee754	1.1169×10^3	7.9507×10^3	3.8825×10^3	1.5544×10^3	31
f13	real-value	7.0445×10^2	1.2750×10^3	9.7957×10^2	1.3658×10^2	31
f14	fixed-point	8.7429×10^3	2.1290×10^4	1.3825×10^4	2.9906×10^3	31
f14	gray	8.9638×10^3	2.3069×10^4	1.3459×10^4	3.1956×10^3	31
f14	ieee754	8.3047×10^3	2.5268×10^4	1.4893×10^4	3.8228×10^3	31
f14	real-value	1.0118×10^4	1.7727×10^4	1.2963×10^4	1.7207×10^3	31
f15	fixed-point	5.9491×10^0	6.8372×10^0	6.4163×10^0	2.2169×10^{-1}	31
f15	gray	5.8867×10^0	6.6917×10^0	6.3159×10^0	2.5003×10^{-1}	31
f15	ieee754	6.0215×10^0	6.9911×10^0	6.5332×10^0	2.8071×10^{-1}	31
f15	real-value	5.4941×10^0	6.0588×10^0	5.8440×10^0	1.0776×10^{-1}	31

(zdroj: vlastní program)

Tabulka 8.4 Statistika Final Error (Min, Max, Průměr, Směrodatná odchylka) pro různé reprezentace pro multimodální funkce s více komponenty

Inst	Reprezentace	Min Error	Max Error	Mean Error	Std Error	Count
f16	fixed-point	1.2864×10^3	6.9995×10^3	3.3363×10^3	1.5244×10^3	31
f16	gray	5.9701×10^{-3}	7.0296×10^2	6.1020×10^2	1.2688×10^2	31
f16	ieee754	9.2790×10^2	1.2022×10^4	4.4248×10^3	2.4941×10^3	31
f16	real-value	5.2960×10^2	6.8210×10^2	6.5510×10^2	4.8859×10^1	31
f17	fixed-point	7.2011×10^4	3.3692×10^5	1.5601×10^5	7.3755×10^4	31
f17	gray	1.0659×10^3	3.0319×10^4	7.7803×10^3	7.0416×10^3	31
f17	ieee754	4.9895×10^4	4.0761×10^5	1.6765×10^5	7.9758×10^4	31
f17	real-value	1.1071×10^{-2}	6.8221×10^2	5.3586×10^2	2.6976×10^2	31
f18	fixed-point	7.0926×10^3	2.9067×10^4	1.6039×10^4	5.3792×10^3	31
f18	gray	1.0222×10^4	4.1437×10^4	2.0084×10^4	7.1201×10^3	31
f18	ieee754	7.6738×10^3	2.6644×10^4	1.6746×10^4	4.4936×10^3	31
f18	real-value	1.3615×10^4	2.9106×10^4	1.9637×10^4	3.9429×10^3	31
f19	fixed-point	8.7639×10^3	3.4551×10^4	1.8485×10^4	5.5272×10^3	31
f19	gray	7.9339×10^3	3.0297×10^4	1.6192×10^4	5.7387×10^3	31
f19	ieee754	5.0461×10^3	3.2799×10^4	1.6310×10^4	6.5081×10^3	31
f19	real-value	1.0787×10^4	2.4483×10^4	1.6610×10^4	3.6677×10^3	31
f20	fixed-point	9.0886×10^0	1.2605×10^1	1.0804×10^1	8.9299×10^{-1}	31
f20	gray	9.1731×10^0	1.2516×10^1	1.0515×10^1	8.8911×10^{-1}	31
f20	ieee754	9.3494×10^0	1.2533×10^1	1.0834×10^1	8.4210×10^{-1}	31
f20	real-value	1.1268×10^1	1.4066×10^1	1.2213×10^1	6.3820×10^{-1}	31
f21	fixed-point	1.6465×10^1	1.2698×10^2	5.9289×10^1	3.0107×10^1	31
f21	gray	3.4026×10^1	1.1366×10^2	7.2936×10^1	2.0315×10^1	31
f21	ieee754	5.0000×10^0	1.1491×10^2	4.8058×10^1	3.8652×10^1	31
f21	real-value	8.4310×10^0	3.0894×10^1	1.9542×10^1	5.8432×10^0	31
f22	fixed-point	2.6866×10^4	9.0825×10^4	5.6192×10^4	1.7838×10^4	31
f22	gray	2.1919×10^4	1.2075×10^5	6.0060×10^4	2.0283×10^4	31
f22	ieee754	2.4647×10^4	1.5096×10^5	6.1346×10^4	2.5549×10^4	31
f22	real-value	5.7692×10^4	1.2875×10^5	9.2227×10^4	1.8427×10^4	31
f23	fixed-point	2.5597×10^1	5.2114×10^1	3.7712×10^1	6.2912×10^0	31
f23	gray	2.8490×10^1	4.6045×10^1	3.6778×10^1	4.5248×10^0	31
f23	ieee754	2.7700×10^1	5.4954×10^1	3.8602×10^1	6.2286×10^0	31
f23	real-value	3.3865×10^1	4.9276×10^1	4.1526×10^1	4.4719×10^0	31
f24	fixed-point	1.2067×10^2	1.8167×10^2	1.5237×10^2	1.5908×10^1	31
f24	gray	1.2172×10^2	1.7732×10^2	1.5741×10^2	1.6788×10^1	31
f24	ieee754	1.3183×10^2	1.8548×10^2	1.5648×10^2	1.3857×10^1	31
f24	real-value	1.4738×10^2	1.8435×10^2	1.6777×10^2	1.0791×10^1	31

(zdroj: vlastní program)

Tabulka 8.5 Statistické porovnání mediánů final error pro různé reprezentace pro unimodální funkce

Instance	Porovnání rep.	n	p-hodnota	Významné	Mediány (nižší → vyšší)	Lepší
f1	fixed-point vs gray	31 vs 31	0.000	Ano	5.672×10^1 vs 2.041×10^3	gray
	fixed-point vs ieee754	31 vs 31	0.001	Ano	2.041×10^3 vs 3.312×10^3	fixed-point
	fixed-point vs real-value	31 vs 31	0.000	Ano	2.766×10^{-4} vs 2.041×10^3	real-value
	gray vs ieee754	31 vs 31	0.000	Ano	5.672×10^1 vs 3.312×10^3	gray
	gray vs real-value	31 vs 31	0.000	Ano	2.766×10^{-4} vs 5.672×10^1	real-value
	ieee754 vs real-value	31 vs 31	0.000	Ano	2.766×10^{-4} vs 3.312×10^3	real-value
f2	fixed-point vs gray	31 vs 31	0.000	Ano	1.166×10^0 vs 1.473×10^0	gray
	fixed-point vs ieee754	31 vs 31	0.121	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	6.625×10^{-1} vs 1.473×10^0	real-value
	gray vs ieee754	31 vs 31	0.000	Ano	1.166×10^0 vs 1.492×10^0	gray
	gray vs real-value	31 vs 31	0.000	Ano	6.625×10^{-1} vs 1.166×10^0	real-value
	ieee754 vs real-value	31 vs 31	0.000	Ano	6.625×10^{-1} vs 1.492×10^0	real-value
f3	fixed-point vs gray	31 vs 31	0.000	Ano	1.429×10^7 vs 8.213×10^8	gray
	fixed-point vs ieee754	31 vs 31	0.000	Ano	8.213×10^8 vs 1.664×10^9	fixed-point
	fixed-point vs real-value	31 vs 31	0.000	Ano	5.108×10^1 vs 8.213×10^8	real-value
	gray vs ieee754	31 vs 31	0.000	Ano	1.429×10^7 vs 1.664×10^9	gray
	gray vs real-value	31 vs 31	0.000	Ano	5.108×10^1 vs 1.429×10^7	real-value
	ieee754 vs real-value	31 vs 31	0.000	Ano	5.108×10^1 vs 1.664×10^9	real-value
f4	fixed-point vs gray	31 vs 31	0.000	Ano	8.934×10^2 vs 1.249×10^4	gray
	fixed-point vs ieee754	31 vs 31	0.456	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	4.791×10^{-3} vs 1.249×10^4	real-value
	gray vs ieee754	31 vs 31	0.000	Ano	8.934×10^2 vs 1.291×10^4	gray
	gray vs real-value	31 vs 31	0.000	Ano	4.791×10^{-3} vs 8.934×10^2	real-value
	ieee754 vs real-value	31 vs 31	0.000	Ano	4.791×10^{-3} vs 1.291×10^4	real-value
f5	fixed-point vs gray	31 vs 31	0.000	Ano	2.484×10^0 vs 2.845×10^0	gray
	fixed-point vs ieee754	31 vs 31	0.439	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	1.390×10^0 vs 2.845×10^0	real-value
	gray vs ieee754	31 vs 31	0.000	Ano	2.484×10^0 vs 2.847×10^0	gray
	gray vs real-value	31 vs 31	0.000	Ano	1.390×10^0 vs 2.484×10^0	real-value
	ieee754 vs real-value	31 vs 31	0.000	Ano	1.390×10^0 vs 2.847×10^0	real-value
f6	fixed-point vs gray	31 vs 31	0.000	Ano	2.573×10^0 vs 2.839×10^0	gray
	fixed-point vs ieee754	31 vs 31	0.177	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	1.736×10^0 vs 2.839×10^0	real-value
	gray vs ieee754	31 vs 31	0.000	Ano	2.573×10^0 vs 2.856×10^0	gray
	gray vs real-value	31 vs 31	0.000	Ano	1.736×10^0 vs 2.573×10^0	real-value
	ieee754 vs real-value	31 vs 31	0.000	Ano	1.736×10^0 vs 2.856×10^0	real-value

Tabulka 8.6 Statistické porovnání mediánů final error pro multimodální funkce s jedním komponentem

Instance	Porovnání rep.	n	p-hodnota	Významné	Mediány (nižší → vyšší)	Lepší
f7	fixed-point vs gray	31 vs 31	0.536	Ne	-	-
	fixed-point vs ieee754	31 vs 31	0.000	Ano	1.394×10^3 vs 3.049×10^3	fixed-point
	fixed-point vs real-value	31 vs 31	0.000	Ano	1.506×10^{-3} vs 1.394×10^3	real-value
	gray vs ieee754	31 vs 31	0.011	Ano	1.862×10^3 vs 3.049×10^3	gray
	gray vs real-value	31 vs 31	0.000	Ano	1.506×10^{-3} vs 1.862×10^3	real-value
	ieee754 vs real-value	31 vs 31	0.000	Ano	1.506×10^{-3} vs 3.049×10^3	real-value
f8	fixed-point vs gray	31 vs 31	0.000	Ano	1.024×10^3 vs 1.734×10^3	gray
	fixed-point vs ieee754	31 vs 31	0.001	Ano	1.734×10^3 vs 2.679×10^3	fixed-point
	fixed-point vs real-value	31 vs 31	0.000	Ano	2.091×10^{-3} vs 1.734×10^3	real-value
	gray vs ieee754	31 vs 31	0.000	Ano	1.024×10^3 vs 2.679×10^3	gray
	gray vs real-value	31 vs 31	0.000	Ano	2.091×10^{-3} vs 1.024×10^3	real-value
	ieee754 vs real-value	31 vs 31	0.000	Ano	2.091×10^{-3} vs 2.679×10^3	real-value
f9	fixed-point vs gray	31 vs 31	0.000	Ano	4.772×10^2 vs 6.937×10^2	gray
	fixed-point vs ieee754	31 vs 31	0.057	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	9.459×10^1 vs 6.937×10^2	real-value
	gray vs ieee754	31 vs 31	0.000	Ano	4.772×10^2 vs 8.665×10^2	gray
	gray vs real-value	31 vs 31	0.000	Ano	9.459×10^1 vs 4.772×10^2	real-value
	ieee754 vs real-value	31 vs 31	0.000	Ano	9.459×10^1 vs 8.665×10^2	real-value
f10	fixed-point vs gray	31 vs 31	0.318	Ne	-	-
	fixed-point vs ieee754	31 vs 31	0.000	Ano	1.454×10^3 vs 2.872×10^3	fixed-point
	fixed-point vs real-value	31 vs 31	0.000	Ano	4.850×10^{-3} vs 1.454×10^3	real-value
	gray vs ieee754	31 vs 31	0.000	Ano	1.184×10^3 vs 2.872×10^3	gray
	gray vs real-value	31 vs 31	0.000	Ano	4.850×10^{-3} vs 1.184×10^3	real-value
	ieee754 vs real-value	31 vs 31	0.000	Ano	4.850×10^{-3} vs 2.872×10^3	real-value
f11	fixed-point vs gray	31 vs 31	0.439	Ne	-	-
	fixed-point vs ieee754	31 vs 31	0.866	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	1.494×10^4 vs 2.235×10^4	fixed-point
	gray vs ieee754	31 vs 31	0.375	Ne	-	-
	gray vs real-value	31 vs 31	0.000	Ano	1.378×10^4 vs 2.235×10^4	gray
	ieee754 vs real-value	31 vs 31	0.000	Ano	1.430×10^4 vs 2.235×10^4	ieee754
f12	fixed-point vs gray	31 vs 31	0.989	Ne	-	-
	fixed-point vs ieee754	31 vs 31	0.778	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.081	Ne	-	-
	gray vs ieee754	31 vs 31	0.877	Ne	-	-
	gray vs real-value	31 vs 31	0.052	Ne	-	-
	ieee754 vs real-value	31 vs 31	0.054	Ne	-	-
f13	fixed-point vs gray	31 vs 31	0.000	Ano	2.049×10^3 vs 3.569×10^3	gray
	fixed-point vs ieee754	31 vs 31	0.746	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	9.900×10^2 vs 3.569×10^3	real-value
	gray vs ieee754	31 vs 31	0.000	Ano	2.049×10^3 vs 3.832×10^3	gray
	gray vs real-value	31 vs 31	0.000	Ano	9.900×10^2 vs 2.049×10^3	real-value
	ieee754 vs real-value	31 vs 31	0.000	Ano	9.900×10^2 vs 3.832×10^3	real-value
f14	fixed-point vs gray	31 vs 31	0.481	Ne	-	-
	fixed-point vs ieee754	31 vs 31	0.266	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.231	Ne	-	-
	gray vs ieee754	31 vs 31	0.094	Ne	-	-
	gray vs real-value	31 vs 31	0.910	Ne	-	-
	ieee754 vs real-value	31 vs 31	0.022	Ano	1.293×10^4 vs 1.496×10^4	real-value
f15	fixed-point vs gray	31 vs 31	0.151	Ne	-	-
	fixed-point vs ieee754	31 vs 31	0.069	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	5.863×10^0 vs 6.396×10^0	real-value
	gray vs ieee754	31 vs 31	0.003	Ano	6.279×10^0 vs 6.542×10^0	gray
	gray vs real-value	31 vs 31	0.000	Ano	5.863×10^0 vs 6.279×10^0	real-value
	ieee754 vs real-value	31 vs 31	0.000	Ano	5.863×10^0 vs 6.542×10^0	real-value

Tabulka 8.7 Statistické porovnání mediánů final error pro multimodální funkce s více komponenty

Instance	Porovnání rep.	n	p-hodnota	Významné	Mediány (nižší → vyšší)	Lepší
f16	fixed-point vs gray	31 vs 31	0.000	Ano	6.016×10^2 vs 2.957×10^3	gray
	fixed-point vs ieee754	31 vs 31	0.105	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	6.821×10^2 vs 2.957×10^3	real-value
	gray vs ieee754	31 vs 31	0.000	Ano	6.016×10^2 vs 4.042×10^3	gray
	gray vs real-value	31 vs 31	0.368	Ne	-	-
	ieee754 vs real-value	31 vs 31	0.000	Ano	6.821×10^2 vs 4.042×10^3	real-value
f17	fixed-point vs gray	31 vs 31	0.000	Ano	5.640×10^3 vs 1.353×10^5	gray
	fixed-point vs ieee754	31 vs 31	0.447	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	6.821×10^2 vs 1.353×10^5	real-value
	gray vs ieee754	31 vs 31	0.000	Ano	5.640×10^3 vs 1.652×10^5	gray
	gray vs real-value	31 vs 31	0.000	Ano	6.821×10^2 vs 5.640×10^3	real-value
	ieee754 vs real-value	31 vs 31	0.000	Ano	6.821×10^2 vs 1.652×10^5	real-value
f18	fixed-point vs gray	31 vs 31	0.019	Ano	1.645×10^4 vs 1.872×10^4	fixed-point
	fixed-point vs ieee754	31 vs 31	0.422	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.004	Ano	1.645×10^4 vs 1.973×10^4	fixed-point
	gray vs ieee754	31 vs 31	0.081	Ne	-	-
	gray vs real-value	31 vs 31	0.767	Ne	-	-
	ieee754 vs real-value	31 vs 31	0.012	Ano	1.569×10^4 vs 1.973×10^4	ieee754
f19	fixed-point vs gray	31 vs 31	0.052	Ne	-	-
	fixed-point vs ieee754	31 vs 31	0.105	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.210	Ne	-	-
	gray vs ieee754	31 vs 31	0.933	Ne	-	-
	gray vs real-value	31 vs 31	0.318	Ne	-	-
	ieee754 vs real-value	31 vs 31	0.414	Ne	-	-
f20	fixed-point vs gray	31 vs 31	0.168	Ne	-	-
	fixed-point vs ieee754	31 vs 31	0.944	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	1.093×10^1 vs 1.202×10^1	fixed-point
	gray vs ieee754	31 vs 31	0.159	Ne	-	-
	gray vs real-value	31 vs 31	0.000	Ano	1.039×10^1 vs 1.202×10^1	gray
	ieee754 vs real-value	31 vs 31	0.000	Ano	1.090×10^1 vs 1.202×10^1	ieee754
f21	fixed-point vs gray	31 vs 31	0.023	Ano	5.304×10^1 vs 7.977×10^1	fixed-point
	fixed-point vs ieee754	31 vs 31	0.147	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	1.897×10^1 vs 5.304×10^1	real-value
	gray vs ieee754	31 vs 31	0.014	Ano	3.549×10^1 vs 7.977×10^1	ieee754
	gray vs real-value	31 vs 31	0.000	Ano	1.897×10^1 vs 7.977×10^1	real-value
	ieee754 vs real-value	31 vs 31	0.045	Ano	1.897×10^1 vs 3.549×10^1	real-value
f22	fixed-point vs gray	31 vs 31	0.398	Ne	-	-
	fixed-point vs ieee754	31 vs 31	0.508	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	5.340×10^4 vs 9.085×10^4	fixed-point
	gray vs ieee754	31 vs 31	0.921	Ne	-	-
	gray vs real-value	31 vs 31	0.000	Ano	5.735×10^4 vs 9.085×10^4	gray
	ieee754 vs real-value	31 vs 31	0.000	Ano	6.045×10^4 vs 9.085×10^4	ieee754
f23	fixed-point vs gray	31 vs 31	0.545	Ne	-	-
	fixed-point vs ieee754	31 vs 31	0.683	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.005	Ano	3.754×10^1 vs 4.086×10^1	fixed-point
	gray vs ieee754	31 vs 31	0.248	Ne	-	-
	gray vs real-value	31 vs 31	0.001	Ano	3.688×10^1 vs 4.086×10^1	gray
	ieee754 vs real-value	31 vs 31	0.035	Ano	3.778×10^1 vs 4.086×10^1	ieee754
f24	fixed-point vs gray	31 vs 31	0.159	Ne	-	-
	fixed-point vs ieee754	31 vs 31	0.414	Ne	-	-
	fixed-point vs real-value	31 vs 31	0.000	Ano	1.515×10^2 vs 1.717×10^2	fixed-point
	gray vs ieee754	31 vs 31	0.526	Ne	-	-
	gray vs real-value	31 vs 31	0.014	Ano	1.599×10^2 vs 1.717×10^2	gray
	ieee754 vs real-value	31 vs 31	0.001	Ano	1.568×10^2 vs 1.717×10^2	ieee754

8.5 Výsledky statistického porovnání (Friedmanův test a Nemenyiho post-hoc)

Následující tabulky prezentují výsledky Friedmanova Rank Testu pro porovnání čtyř reprezentací (fixed-point, gray, ieee754, real-value) pro každou instanci a, kde je to statisticky významné, výsledky Post-hoc Nemenyiho testu.

Tabulka 8.8 Výsledky Friedmanova a Nemenyiho testu pro unimodální funkce

Instance	χ^2	p-hodnota	df	Závěr Friedmanova testu / Prům. ranky / Post-hoc Nemenyi
f1	83.3226	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=3.2903, Gray=2.0323, IEEE=3.6774, RV=1.0000 FP vs Gray: 1.2581 (>CD), FP vs IEEE: 0.3871 (<CD), FP vs RV: 2.2903 (>CD) Gray vs IEEE: 1.6452 (>CD), Gray vs RV: 1.0323 (>CD), IEEE vs RV: 2.6774 (>CD)
f2	84.8710	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=3.3226, Gray=2.0000, IEEE=3.6774, RV=1.0000 FP vs Gray: 1.3226 (>CD), FP vs IEEE: 0.3548 (<CD), FP vs RV: 2.3226 (>CD) Gray vs IEEE: 1.6774 (>CD), Gray vs RV: 1.0000 (>CD), IEEE vs RV: 2.6774 (>CD)
f3	85.8774	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=3.2581, Gray=2.0000, IEEE=3.7419, RV=1.0000 FP vs Gray: 1.2581 (>CD), FP vs IEEE: 0.4839 (<CD), FP vs RV: 2.2581 (>CD) Gray vs IEEE: 1.7419 (>CD), Gray vs RV: 1.0000 (>CD), IEEE vs RV: 2.7419 (>CD)
f4	83.7097	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=3.5161, Gray=2.0000, IEEE=3.4839, RV=1.0000 FP vs Gray: 1.5161 (>CD), FP vs IEEE: 0.0323 (<CD), FP vs RV: 2.5161 (>CD) Gray vs IEEE: 1.4839 (>CD), Gray vs RV: 1.0000 (>CD), IEEE vs RV: 2.4839 (>CD)
f5	81.9677	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=3.4516, Gray=2.0323, IEEE=3.5161, RV=1.0000 FP vs Gray: 1.4194 (>CD), FP vs IEEE: 0.0645 (<CD), FP vs RV: 2.4516 (>CD) Gray vs IEEE: 1.4839 (>CD), Gray vs RV: 1.0323 (>CD), IEEE vs RV: 2.5161 (>CD)
f6	81.9677	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=3.4516, Gray=2.0323, IEEE=3.5161, RV=1.0000 FP vs Gray: 1.4194 (>CD), FP vs IEEE: 0.0645 (<CD), FP vs RV: 2.4516 (>CD) Gray vs IEEE: 1.4839 (>CD), Gray vs RV: 1.0323 (>CD), IEEE vs RV: 2.5161 (>CD)

Tabulka 8.9 Výsledky Friedmanova a Nemenyiho testu pro multimodální funkce s jedním komponentem

Instance	χ^2	p-hodnota	df	Závěr Friedmanova testu / Prům. ranky / Post-hoc Nemenyi
f7	62.1097	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=2.6452, Gray=2.9032, IEEE=3.4516, RV=1.0000 FP vs Gray: 0.2581 (<CD), FP vs IEEE: 0.8065 (<CD), FP vs RV: 1.6452 (>CD) Gray vs IEEE: 0.5484 (<CD), Gray vs RV: 1.9032 (>CD), IEEE vs RV: 2.4516 (>CD)
f8	71.2839	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=3.0000, Gray=2.3548, IEEE=3.6452, RV=1.0000 FP vs Gray: 0.6452 (<CD), FP vs IEEE: 0.6452 (<CD), FP vs RV: 2.0000 (>CD) Gray vs IEEE: 1.2903 (>CD), Gray vs RV: 1.3548 (>CD), IEEE vs RV: 2.6452 (>CD)
f9	63.3871	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=3.0323, Gray=2.4516, IEEE=3.4839, RV=1.0323 FP vs Gray: 0.5806 (<CD), FP vs IEEE: 0.4516 (<CD), FP vs RV: 2.0000 (>CD) Gray vs IEEE: 1.0323 (>CD), Gray vs RV: 1.4194 (>CD), IEEE vs RV: 2.4516 (>CD)
f10	68.6903	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=2.7097, Gray=2.6129, IEEE=3.6774, RV=1.0000 FP vs Gray: 0.0968 (<CD), FP vs IEEE: 0.9677 (>CD), FP vs RV: 1.7097 (>CD) Gray vs IEEE: 1.0645 (>CD), Gray vs RV: 1.6129 (>CD), IEEE vs RV: 2.6774 (>CD)
f11	35.2452	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=2.2903, Gray=1.8065, IEEE=2.2581, RV=3.6452 FP vs Gray: 0.4839 (<CD), FP vs IEEE: 0.0323 (<CD), FP vs RV: 1.3548 (>CD) Gray vs IEEE: 0.4516 (<CD), Gray vs RV: 1.8387 (>CD), IEEE vs RV: 1.3871 (>CD)
f12	6.2903	0.0983	3	Není významný ($p \geq 0.05$). Post-hoc test není potřeba.
f13	68.6129	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=3.3226, Gray=2.3226, IEEE=3.3548, RV=1.0000 FP vs Gray: 1.0000 (>CD), FP vs IEEE: 0.0323 (<CD), FP vs RV: 2.3226 (>CD) Gray vs IEEE: 1.0323 (>CD), Gray vs RV: 1.3226 (>CD), IEEE vs RV: 2.3548 (>CD)
f14	7.3742	0.0609	3	Není významný ($p \geq 0.05$). Post-hoc test není potřeba.
f15	55.9161	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=2.9677, Gray=2.7419, IEEE=3.2581, RV=1.0323 FP vs Gray: 0.2258 (<CD), FP vs IEEE: 0.2903 (<CD), FP vs RV: 1.9355 (>CD) Gray vs IEEE: 0.5161 (<CD), Gray vs RV: 1.7097 (>CD), IEEE vs RV: 2.2258 (>CD)

Tabulka 8.10 Výsledky Friedmanova a Nemenyiho testu pro multimodální funkce s více komponenty

Instance	χ^2	p-hodnota	df	Závěr Friedmanova testu / Prům. ranky / Post-hoc Nemenyi
f16	74.4968	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=3.4516, Gray=1.5161, IEEE=3.5484, RV=1.4839 FP vs Gray: 1.9355 (>CD), FP vs IEEE: 0.0968 (<CD), FP vs RV: 1.9677 (>CD) Gray vs IEEE: 2.0323 (>CD), Gray vs RV: 0.0323 (<CD), IEEE vs RV: 2.0645 (>CD)
f17	83.7097	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=3.4839, Gray=2.0000, IEEE=3.5161, RV=1.0000 FP vs Gray: 1.4839 (>CD), FP vs IEEE: 0.0323 (<CD), FP vs RV: 2.4839 (>CD) Gray vs IEEE: 1.5161 (>CD), Gray vs RV: 1.0000 (>CD), IEEE vs RV: 2.5161 (>CD)
f18	10.4323	0.0152	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=2.0645, Gray=2.8387, IEEE=2.1935, RV=2.9032 FP vs Gray: 0.7742 (<CD), FP vs IEEE: 0.1290 (<CD), FP vs RV: 0.8387 (<CD) Gray vs IEEE: 0.6452 (<CD), Gray vs RV: 0.0645 (<CD), IEEE vs RV: 0.7097 (<CD)
f19	5.7871	0.1224	3	Není významný ($p \geq 0.05$). Post-hoc test není potřeba.
f20	40.9742	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=2.2258, Gray=1.7742, IEEE=2.2581, RV=3.7419 FP vs Gray: 0.4516 (<CD), FP vs IEEE: 0.0323 (<CD), FP vs RV: 1.5161 (>CD) Gray vs IEEE: 0.4839 (<CD), Gray vs RV: 1.9677 (>CD), IEEE vs RV: 1.4839 (>CD)
f21	43.2968	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=3.0323, Gray=3.2581, IEEE=2.4194, RV=1.2903 FP vs Gray: 0.2258 (<CD), FP vs IEEE: 0.6129 (<CD), FP vs RV: 1.7419 (>CD) Gray vs IEEE: 0.8387 (<CD), Gray vs RV: 1.9677 (>CD), IEEE vs RV: 1.1290 (>CD)
f22	29.1677	0.0000	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=2.0968, Gray=2.0968, IEEE=2.2258, RV=3.5806 FP vs Gray: 0.0000 (<CD), FP vs IEEE: 0.1290 (<CD), FP vs RV: 1.4839 (>CD) Gray vs IEEE: 0.1290 (<CD), Gray vs RV: 1.4839 (>CD), IEEE vs RV: 1.3548 (>CD)
f23	7.7613	0.0512	3	Není významný ($p \geq 0.05$). Post-hoc test není potřeba.
f24	15.8516	0.0012	3	Významný ($p < 0.05$). Existuje statisticky významný rozdíl. CD = 0.8424 Ranky: FP=2.1290, Gray=2.3226, IEEE=2.2581, RV=3.2903 FP vs Gray: 0.1935 (<CD), FP vs IEEE: 0.1290 (<CD), FP vs RV: 1.1613 (>CD) Gray vs IEEE: 0.0645 (<CD), Gray vs RV: 0.9677 (>CD), IEEE vs RV: 1.0323 (>CD)

9 DISKUZE A DOPORUČENÍ

V této kapitole se zaměřím na syntézu zjištění, která jsem prezentoval v předchozích částech práce, zejména v kapitole 8, kde jsem analyzoval výsledky experimentů. Na základě statistických analýz a porovnání jednotlivých reprezentací se pokusím formulovat závěry a doporučení pro praktické použití, stejně jako nastínit oblasti pro budoucí výzkum.

9.1 Zhodnocení metod

9.1.1 Zhodnocení experimentu s konvergencemi

Na základě tabulky 8.1 lze pozorovat, že volba číselné reprezentace a kódovacího schématu má významný vliv na výkon a stabilitu algoritmu napříč různými instancemi (f1, f7, f16). Reprezentace "ieee754" (standard pro čísla s plovoucí desetinnou čárkou) se jeví jako nejeftektivnější z hlediska průměrné fitness pro instance f1 a f7, dosahující nejvyšších hodnot. To však přichází s kompromisem v podobě nejvyšší směrodatné odchylky, což naznačuje nižší konzistenci výsledků. Naopak "real-value" reprezentace vykazuje extrémně nízkou směrodatnou odchylku napříč všemi instancemi, což svědčí o její výjimečné stabilitě a reprodukovatelnosti. Avšak často dosahuje nejnižších nebo velmi nízkých průměrných hodnot fitness, což naznačuje, že její přesnost nemusí vždy vést k optimálnímu řešení problému. "Fixed-point" reprezentace a "gray" kódovací schéma (pravděpodobně aplikované na celočíselné hodnoty) nabízejí smíšené výsledky. Konkrétně "gray" kód často vykazuje nejnižší průměrné fitness hodnoty, například u instance f16, což může naznačovat, že jeho vlastnosti (jednobitová změna mezi sousedními hodnotami) nemusí být vždy výhodné pro daný optimalizační problém. Pro dosažení optimálního výkonu je tak klíčové pečlivé zvážení jak samotné číselné reprezentace, tak i případného kódovacího schématu v závislosti na charakteru konkrétní úlohy.

Je důležité poznamenat, že prezentované statistiky fitness, zejména ty týkající se průměru a směrodatné odchylky, mohly být ovlivněny procesem ukládání dat v průběhu konvergence algoritmu. Agregace mezilehlých hodnot pro účely vizualizace konvergence může zkreslit celkové výsledky a nemusí přesně odrážet finální optimální řešení nebo chování algoritmu v jeho ustáleném stavu. Z tohoto důvodu nebudou tato data použita pro závěrečné hodnocení celkové efektivity a výkonu jednotlivých reprezentací, ale spíše pro ilustraci dynamiky jejich vývoje.

9.1.2 Zhodnocení hlavního experimentu

Experimentální výsledky, shrnuté v tabulkách v kapitole 8, mi poskytují cenný vhled do chování různých číselných reprezentací (Fixed-Point, IEEE 754 a Reálná Hodnota) a kódovacího schématu (Gray kód) v kontextu optimalizačních algoritmů. Celkově jsem pozoroval, že výkonnost jednotlivých přístupů se výrazně liší v závislosti na konkrétní testovací funkci.

U funkcí jako f_1 , f_2 , f_4 , f_5 , f_6 a f_{22} jsem zjistil statisticky významné rozdíly mezi reprezentacemi (p -hodnota < 0.05). Zejména u funkcí f_1 a f_4 jsem pozoroval, že reprezentace real-value dosahovala nejlepších průměrných ranků. Například pro f_1 jsem zaznamenal průměrný rank real-value = 1.0000, následovaný gray = 2.0323, fixed-point = 3.2903 a ieee754 = 3.6774. Pro f_4 byly průměrné ranky: real-value = 1.0000, gray = 2.0000, ieee754 = 3.4839 a fixed-point = 3.5161. To mi naznačuje, že v těchto případech byla reprezentace reálnou hodnotou nejefektivnější.

Naopak, u funkcí jako f_{12} , f_{14} , f_{19} a f_{23} jsem nezjistil statisticky významné rozdíly (p -hodnota ≥ 0.05). To mi naznačuje, že pro tyto funkce je volba reprezentace méně kritická a všechny zkoumané metody podávají srovnatelný výkon. Pro tyto funkce nebyl proveden post-hoc test, protože celkový test neprokázal významný rozdíl. U většiny ostatních testovaných funkcí (např. f_3 , f_7 , f_8 , f_9 , f_{10} , f_{11} , f_{13} , f_{15} , f_{16} , f_{17} , f_{18} , f_{20} , f_{21} , f_{24}) existuje statisticky významný rozdíl mezi reprezentacemi (p -hodnota < 0.05).

V případě funkce f_{11} výsledky ukazují statisticky významný rozdíl mezi reprezentacemi ($p < 0.05$), přičemž RV dosahuje nejhoršího průměrného ranku (3.6452) a je statisticky významně horší než Fixed-Point, Gray i IEEE 754. Naopak, Gray kódování se pro tuto funkci ukazuje jako nejefektivnější. Toto specifické chování je způsobeno extrémní multimodalností a vysokou dimenzionalitou Michalewiczovy funkce. V 30 dimenzích exponenciálně narůstající počet lokálních minim vytváří vysoce členitou optimalizační krajinu, kde se algoritmy s kontinuálními operátory reálné hodnoty snadno "zaseknou" a obtížně unikají z lokálních optim.

Binární reprezentace, zejména Gray kódování, zde prokazuje svou robustnost. Jejich diskrétní povaha umožňuje genetickému algoritmu efektivněji prohledávat tuto komplexní krajinu prostřednictvím "skoků" mezi vzdálenějšími body, což je pro únik z lokálních minim a nalezení globálního optima klíčové. Zatímco u jiných funkcí s hladšími fitness krajinami je výhodou jemné ladění pomocí reálných čísel, u vysoce multimodálních a dimenzionálně ná-

ročních problémů, jako je Michalewiczova funkce, se projeví limitace kontinuálních operátorů. Diskrétní povaha binárních kódování tak poskytuje implicitní regularizaci a efektivnější mechanismus pro globální průzkum, což vysvětluje jejich nadřazený výkon nad reprezentací reálnou hodnotou v tomto specifickém kontextu.

Specificky pro f22, kde jsem zjistil významný rozdíl (p -hodnota < 0.05), jsem pozoroval, že FP a Gray opět dosahují lepších ranků (FP=2.0968, Gray=2.0968) oproti IEEE (2.2258) a RV (3.5806). Rozdíl mezi IEEE a RV (1.3548) byl navíc větší než kritická difference (CD = 0.8424), což mi potvrzuje významný rozdíl mezi těmito dvěma reprezentacemi u této funkce.

Celkově mohu konstatovat, že reprezentace reálnou hodnotou (real-value) se ukázala jako neefektivnější pro značnou část testovaných funkcí, často s výrazným statisticky významným rozdílem oproti ostatním reprezentacím (např. f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f13, f15, f16, f17). Naopak, Fixed-Point a Grayův kód dosahovaly lepších výsledků než IEEE 754 u některých funkcí, nebo byly s real-value srovnatelné tam, kde celkový rozdíl nebyl statisticky významný (např. f12, f14, f19, f23), nebo kde byly ranky blízko u sebe (např. f11, f20, f22, f24). Reprezentace IEEE 754 často vykazovala horší výkon ve srovnání s real-value a v mnoha případech i s fixed-point a gray reprezentacemi.

9.2 Doporučení pro praktické použití

Na základě mnou provedených analýz mohu formulovat následující doporučení pro volbu číselné reprezentace v kontextu genetických algoritmů a podobných optimalizačních metod:

- **Pro jednoduché a unimodální funkce (např. f1, f2, f4, f5, f6):** Doporučuji preferovat reprezentaci **reálnou hodnotou (real-value)**. Tato reprezentace prokázala konzistentně nejlepší výsledky s průměrným rankem 1.0000 pro tyto funkce a často statisticky významně překonávala ostatní metody, včetně Fixed-Point a Gray kódování.
- **Pro složité a vysoce multimodální funkce (např. f22):** Zde jsem pozoroval, že **Fixed-Point a Gray kódování** dosahovaly lepších průměrných ranků (2.0968 pro obě) než IEEE 754 (2.2258) a zejména real-value (3.5806). Rozdíl mezi IEEE 754 a real-value byl navíc statisticky významný (rozdíl = 1.3548, CD = 0.8424). Při těchto problémech je vhodné zvážit i další aspekty, jako je přesnost požadovaného řešení a výpočetní náročnost.

- **Při absenci předchozích znalostí o optimalizované funkci:** Pokud typ funkce není znám, doporučuji provést pilotní studie s několika reprezentacemi. Na základě mých experimentů se **reprezentace reálnou hodnotou (RV)** často ukázala jako nejvýkonnější. Je však vhodné porovnat ji i s **Fixed-Point** a **Gray kódováním**, které mohou být konkurenceschopné pro specifické typy funkcí (např. f_{11} , f_{20} , f_{22} , f_{24}), a s **IEEE 754**, která je standardní reprezentací pro plovoucí desetinnou čárku.
- **Zohlednění kritické difference (CD):** Při interpretaci výsledků jsem si uvědomil, že je klíčové brát v úvahu kritickou diferenci (CD). Rozdíly ranků menší než CD by neměly být považovány za statisticky významné. Toto pravidlo mi pomáhá předcházet chybným závěrům a zajišťuje, že doporučení jsou založena na robustních statistických důkazech.
- **Implementační složitost:** Fixed-Point a Gray kódování mohou vyžadovat složitější implementaci převodů mezi binární a desítkovou formou. Na druhou stranu, IEEE 754 a RV jsou obvykle nativně podporovány programovacími jazyky, což zjednodušuje jejich použití. Tento aspekt bych měl také zvážit, zejména u menších projektů nebo při omezených zdrojích.

9.3 Návrhy na budoucí výzkum

Na základě mnou provedeného výzkumu a získaných poznatků se otevírá řada směrů pro budoucí bádání:

- **Rozšíření sady testovacích funkcí:** Provedené experimenty byly omezeny na sadu benchmarkových funkcí. Budoucí výzkum by mohl zahrnovat širší spektrum optimalizačních problémů, včetně reálných aplikací, které by lépe simulovaly praktické scénáře.
- **Vliv parametrů genetického algoritmu:** Zkoumání vlivu různých parametrů genetického algoritmu (např. velikost populace, míra mutace, míra křížení) na výkonnost jednotlivých reprezentací by mohlo poskytnout hlubší vhled do jejich chování.
- **Hybridní reprezentace:** Výzkum hybridních reprezentací genotypu představuje slibnou cestu pro zvýšení robustnosti a efektivnosti evolučních algoritmů. Integrace heterogenních reprezentací, jako je kombinace Fixed-Point pro diskrétní nebo celočíselné

komponenty proměnných a IEEE 754 pro kontinuitní aspekty, by mohla optimalizovat výkon algoritmů pro multimodální a komplexní optimalizační problémy. Klíčovým aspektem by zde byla granulární aplikace genetických operátorů. Konkrétně by crossover nebyl aplikován globálně na celý chromozom, nýbrž individuálně pro každou dimenzi nebo její specifický segment, reflektující inherentní reprezentaci. Tento přístup by sice teoreticky zachoval integritu a specifické algebraické vlastnosti každé lokální reprezentace. Zásadním problémem by byla kompatibilita mezi různými typy reprezentací a alokace důležitosti bitových polí v rámci jedné proměnné. Například u Fixed-Point reprezentace by neadekvátní operace mohla narušit distribuci bitů mezi celočíselnou a zlomkovou částí, vedoucí k významné ztrátě přesnosti nebo k úplnému narušení sémantiky proměnné.

- **Adaptivní volba reprezentace:** Vývoj algoritmů, které by dynamicky vybíraly nebo upravovaly číselnou reprezentaci v průběhu optimalizačního procesu na základě aktuálního stavu a charakteristik prohledávaného prostoru.
- **Porovnání s jinými optimalizačními algoritmy:** Aplikace zkoumaných číselných reprezentací na jiné typy optimalizačních algoritmů (např. Particle Swarm Optimization, Differential Evolution) by mohla odhalit širší použitelnost a zobecnitelnost zjištění.
- **Teoretická analýza a modelování:** Detailnější teoretická analýza dopadu diskretizace a kódování na topologii prohledávaného prostoru a dynamiku evoluce by mohla poskytnout hlubší porozumění pozorovaným jevům.
- **Optimalizace pro specifický hardware:** Zkoumání, jak různé číselné reprezentace ovlivňují výkonnost optimalizačních algoritmů na specifickém hardware (např. FPGA, GPU), by mohlo být relevantní pro aplikace vyžadující vysoký výkon.

Tyto návrhy představují jen zlomek potenciálních směrů pro budoucí výzkum, které by mohly dále prohloubit mé porozumění a zlepšit praktické aplikace optimalizačních algoritmů s různými číselnými reprezentacemi.

ZÁVĚR

Mým cílem v této práci bylo prozkoumat a porovnat vliv různých přístupů k reprezentaci čísel – konkrétně Fixed-Point, Gray kódování (jako specifické kódovací schéma pro binární reprezentace), IEEE 754 a přímé reálné hodnoty – na výkonnost genetických algoritmů při řešení optimalizačních problémů. Provedl jsem experimenty s rozsáhlou sadou 24 benchmarkových funkcí, a získané výsledky jsem podrobil detailní statistické analýze pomocí Friedmannova testu a Nemenyiho post-hoc testu, přičemž jsem se zaměřil především na funkce, u kterých jsem zjistil statisticky významné rozdíly mezi reprezentacemi.

Klíčovým zjištěním pro mě je, že volba číselné reprezentace má statisticky významný dopad na efektivitu optimalizace u většiny testovaných funkcí (p -hodnota < 0.05). Zejména u unimodálních funkcí jako f_1 , f_2 , f_4 , f_5 a f_6 jsem zjistil, že reprezentace reálnou hodnotou (real-value) dosahovala konzistentně nejlepších výsledků s průměrným rankem 1.0000. U těchto funkcí statisticky významně překonávala ostatní reprezentace (Fixed-Point, Gray kódování a IEEE 754). To pro mě znamená, že pro tyto méně komplexní optimalizační problémy může přímá práce s reálnými čísly vést k nejefektivnější konvergenci a dosažení optimálních řešení.

Zajímavý je případ funkce f_{22} , která je složitou multimodální funkcí. Zde jsem pozoroval statisticky významný rozdíl mezi reprezentacemi (p -hodnota < 0.05), avšak s odlišným pořadím. Pro f_{22} Fixed-Point a Gray kódování dosahovaly lepších průměrných ranků (2.0968 pro obě) než IEEE 754 (2.2258) a zejména real-value (3.5806). Rozdíl mezi IEEE 754 a real-value byl navíc statisticky významný. To podtrhuje, že i pro některé multimodální funkce mohou být binární reprezentace s promyšleným kódováním efektivnější než reprezentace s plovoucí desetinnou čárkou.

Na druhou stranu, pro menší skupinu funkcí, jako jsou f_{12} , f_{14} , f_{19} a f_{23} , jsem nezjistil statisticky významné rozdíly mezi zkoumanými reprezentacemi (p -hodnota ≥ 0.05). To naznačuje, že u těchto konkrétních problémů je vliv číselné reprezentace méně dominantní ve srovnání s jinými faktory genetického algoritmu nebo inherentní složitostí prohledávaného prostoru.

Z praktického hlediska práce doporučuje, abych preferoval reprezentaci reálnou hodnotou pro problémy s unimodálními nebo jednoduššími multimodálními funkcemi, kde tato

reprezentace prokázala nejlepší výkon. V případě složitých multimodálních funkcí, jako je f_{22} , se jeví jako vhodnější Fixed-Point a Gray kódování. Při absenci předchozích znalostí o optimalizované funkci doporučuji provést pilotní studie s několika reprezentacemi. Zároveň jsem zdůraznil, že je nutné při interpretaci výsledků vždy brát v úvahu kritickou diferencí (CD), aby se předešlo mylným závěrům z nevýznamných rozdílů. Implementační složitost (kde Fixed-Point a Gray mohou vyžadovat složitější převody, zatímco IEEE 754 a RV jsou obvykle nativně podporovány) je také důležitým praktickým aspektem k zvážení.

Tento výzkum mi otevřel dveře pro budoucí bádání, včetně rozšíření sady testovacích funkcí o reálné aplikační problémy, podrobnější analýzu vlivu parametrů genetického algoritmu a prozkoumání hybridních a adaptivních strategií pro volbu číselné reprezentace. Dále by pro mě bylo přínosné porovnání s jinými optimalizačními algoritmy a teoretická analýza dopadů různých reprezentací na topologii prohledávaného prostoru.

Závěrem, ačkoliv neexistuje univerzálně nejlepší číselná reprezentace pro všechny optimalizační problémy, já jsem v této práci jasně demonstroval, že pečlivá volba může významně ovlivnit výkonnost genetických algoritmů, a to zejména pro určité třídy funkcí. Pochopení silných a slabých stránek jednotlivých reprezentací je pro mě klíčové pro efektivní návrh a aplikaci evolučních algoritmů v praxi.

SEZNAM POUŽITÉ LITERATURY

- [1] FOGEL, D. B., 1995. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press. ISBN 978-0-471-13582-3.
- [2] EIBEN, A. E. a SMITH, J. E., 2003. *Introduction to Evolutionary Computing*. Springer. ISBN 978-3-540-24699-6.
- [3] BÄCK, T.; FOGEL, D. B. a MICHALEWICZ, Z., 1997. *Handbook of Evolutionary Computation*. IOP Publishing. ISBN 978-0-7503-1156-5.
- [4] ZELINKA, Ivan, 2002. *Umělá inteligence v problémech globální optimalizace*. BEN - technická literatura. ISBN 80-7300-069-5. Kniha se zabývá evolučními algoritmy pro globální optimalizaci.
- [5] GOLDBERG, D. E., 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley. ISBN 978-0-471-40656-2.
- [6] MITCHELL, M., 1998. *An Introduction to Genetic Algorithms*. MIT Press. ISBN 978-0-262-01193-0.
- [7] BÄCK, Thomas, 1996. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Online. Oxford Academic. Dostupné z: <https://doi.org/10.1093/oso/9780195099713.001.0001>. [cit. 2025-04-28].
- [8] DEB, K.; PRATAP, A.; AGARWAL, S. a MEYARIVAN, T., 2002. *A fast and elitist multi-objective genetic algorithm: NSGA-II*. Online. [cit. 2025-04-28].
- [9] YANG, Jihoon a HONAVAR, Vasant G., March 1998. *Feature Subset Selection Using a Genetic Algorithm*. Online. IEEE Educational Activities Department. Dostupné z: <https://doi.org/10.1109/5254.671091>. [cit. 2025-04-28].
- [10] BA VINDRA K. AHUJA, JAMES B. ORLIN T. L. M., 1993. *NETWORK FLOWS*. PRENTICE HALL, Upper Saddle River, New Jersey 07458. ISBN 0-13-617S49-X.

- [11] WALEED HILAL, JOHN YAWNEY S. A. G., 2023. *Cognitive Dynamic Systems: A Review of Theory, Applications, and Recent Advances*. Online. Dostupné z: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10125578>. [cit. 2025-04-29].
- [12] HOLLAND, John H., 1992. *Cognitive Dynamic Systems: A Review of Theory, Applications, and Recent Advances*. Online. Dostupné z: <https://doi.org/10.7551/mitpress/1090.001.0001>. [cit. 2025-04-29].
- [13] DE JONG, Kenneth Alan, 1975. *Analysis of the behavior of a class of genetic adaptive systems*. Online. Dostupné z: <https://deepblue.lib.umich.edu/handle/2027.42/4507>. [cit. 2025-04-29].
- [14] EIBEN, A.E.; HINTERDING, R. a MICHALEWICZ, Z., 1999. *Parameter control in evolutionary algorithms*. Online. [cit. 2025-04-28].
- [15] YAZDANI, D.; OMIDVAR, M. N.; YAZDANI, D.; DEB, K. a GANDOMI, A. H., 2023. *GNBG: A Generalized and Configurable Benchmark Generator for Continuous Numerical Optimization*. Online. Dostupné z: <https://arxiv.org/abs/2312.07083>. [cit. 2024-06-01]. arXiv preprint.
- [16] GANDOMI, A. H.; YAZDANI, D.; OMIDVAR, M. N. a DEB, K., 2023. *GNBG-Generated Test Suite for Box-Constrained Numerical Global Optimization*. Online. Dostupné z: <https://arxiv.org/abs/2312.07034>. [cit. 2024-06-01]. arXiv preprint.
- [17] FREDERICK, William; SEDLMEYER, Robert a WHITE, Curt, 01 1993. *The Hamming Metric in Genetic Algorithms and Its Application to Two Network Problems*. Online. [cit. 2025-04-28].
- [18] BROWNLEE, Jason, 2021. *Chapter 3: Mutation and Its Role*. Online. Algorithm afternoon. Dostupné z: https://algorithmafternoon.com/books/genetic_algorithm/chapter03/. [cit. 2025-05-13].
- [19] SHASTRI, Hrishee a FRACHTENBERG, Eitan, May 2021. *An analysis of the locality of binary representations in genetic and evolutionary algorithms*. Online. Dostupné z: <https://doi.org/10.7717/peerj-cs.561>. [cit. 2025-04-28].

SEZNAM OBRÁZKŮ

Obrázek 1.1	Populace_bin	19
Obrázek 1.2	Ruleta_selekce	20
Obrázek 1.3	Turnaj_selekce	21
Obrázek 1.4	Poradi_selekce	22
Obrázek 1.5	Krizeni_bin1	23
Obrázek 1.6	Krizeni_bin2	24
Obrázek 1.7	Krizeni_bin3	24
Obrázek 1.8	Mutace_bin1	25
Obrázek 1.9	Mutace_bin2	26
Obrázek 1.10	Mutace_bin3	26
Obrázek 6.1	Konvergence nejlepší fitness během počtu vyhodnocení.....	56
Obrázek 7.1	tabulka_vlastnosti_funkci.....	63
Obrázek 8.1	chyba_f1_fixed-point	66
Obrázek 8.2	chyba_f1_gray	66
Obrázek 8.3	chyba_f1_ieee754	67
Obrázek 8.4	chyba_f1_real-value.....	67
Obrázek 8.5	chyba_f7_fixed-point	67
Obrázek 8.6	chyba_f7_gray.....	67
Obrázek 8.7	chyba_f7_ieee754	67
Obrázek 8.8	chyba_f7_real-value.....	67
Obrázek 8.9	chyba_f16_fixed-point.....	68
Obrázek 8.10	chyba_f16_gray	68
Obrázek 8.11	chyba_f16_ieee754.....	68
Obrázek 8.12	chyba_f16_real-value	68
Obrázek 8.13	chyba_f1_average.....	69
Obrázek 8.14	chyba_f7_average.....	69
Obrázek 8.15	chyba_f16_average	69
Obrázek 8.16	chyba_f1_best_runs	70
Obrázek 8.17	chyba_f7_best_runs	70
Obrázek 8.18	chyba_f16_best_runs	70

Obrázek 8.19	casova_narocnost_f1	72
Obrázek 8.20	casova_narocnost_f7	72
Obrázek 8.21	casova_narocnost_f16	72
Obrázek 8.22	plots_f1	73
Obrázek 8.23	plots_f2	73
Obrázek 8.24	plots_f3	73
Obrázek 8.25	plots_f4	73
Obrázek 8.26	plots_f5	74
Obrázek 8.27	plots_f6	74
Obrázek 8.28	plots_f7	74
Obrázek 8.29	plots_f8	74
Obrázek 8.30	plots_f9	74
Obrázek 8.31	plots_f10	74
Obrázek 8.32	plots_f11	75
Obrázek 8.33	plots_f12	75
Obrázek 8.34	plots_f13	75
Obrázek 8.35	plots_f14	75
Obrázek 8.36	plots_f15	75
Obrázek 8.37	plots_f16	75
Obrázek 8.38	plots_f17	76
Obrázek 8.39	plots_f18	76
Obrázek 8.40	plots_f19	76
Obrázek 8.41	plots_f20	76
Obrázek 8.42	plots_f21	76
Obrázek 8.43	plots_f22	76
Obrázek 8.44	plots_f23	77
Obrázek 8.45	plots_f24	77

SEZNAM TABULEK

Tabulka 2.1	Převod mezi binárním a Grayovým kódem (pro 3 bity).....	34
Tabulka 8.1	Statistiky Konečné Chyby (Min, Max, Průměr, Směrodatná odchylka) ..	71
Tabulka 8.2	Statistiky Final Error (Min, Max, Průměr, Směrodatná odchylka) pro různé reprezentace pro unimodální funkce.....	78
Tabulka 8.3	Statistiky Final Error (Min, Max, Průměr, Směrodatná odchylka) pro různé reprezentace pro multimodální funkce s jedním komponentem	79
Tabulka 8.4	Statistiky Final Error (Min, Max, Průměr, Směrodatná odchylka) pro různé reprezentace pro multimodální funkce s více komponenty	80
Tabulka 8.5	Statistické porovnání mediánů final error pro různé reprezentace pro unimodální funkce	81
Tabulka 8.6	Statistické porovnání mediánů final error pro multimodální funkce s jedním komponentem	82
Tabulka 8.7	Statistické porovnání mediánů final error pro multimodální funkce s více komponenty	83
Tabulka 8.8	Výsledky Friedmanova a Nemenyiho testu pro unimodální funkce	84
Tabulka 8.9	Výsledky Friedmanova a Nemenyiho testu pro multimodální funkce s jedním komponentem	85
Tabulka 8.10	Výsledky Friedmanova a Nemenyiho testu pro multimodální funkce s více komponenty	86

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

GNBG	Generalized Numerical Benchmark Generator
GA	Genetické Algoritmy
BCD	Binary-coded Decimal
RGA	Real-valued Genetický Algoritmus
FES	Function Evaluations

SEZNAM PŘÍLOH

PŘÍLOHA P I: Archiv

PŘÍLOHA P I: ARCHIV

Obsah archivu Nedbal_BGA_final.zip:

- `Nedbal_BGA_final.ipynb` – Zdrojový kód pro načtení `.mat` souborů pro benchmarky a implementaci experimentu.
- `Nedbal_combined_gnbg_results.pkl` – Záznamy všech běhů pro všechny reprezentace použité v hlavním experimentu.
- `f1.mat` – `f24.mat` – Zdrojové soubory pro benchmarkové funkce.