



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

DEPARTMENT OF COMPUTER SYSTEMS

ŘÍZENÍ PARAMETRŮ KARTÉZSKÉHO GENETICKÉHO PROGRAMOVÁNÍ

PARAMETER CONTROL FOR CARTESIAN GENETIC PROGRAMMING

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. PETR DOLEŽAL

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. MARTIN HURTA

BRNO 2025

Zadání diplomové práce



162987

Ústav: Ústav počítačových systémů (UPSY)
Student: **Doležal Petr, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Kybernetická bezpečnost
Název: **Řízení parametrů kartézského genetického programování**
Kategorie: Umělá inteligence
Akademický rok: 2024/25

Zadání:

1. Seznamte se s kartézským genetickým programováním (CGP) a s možnostmi jeho využití pro řešení vybraných úloh.
2. Seznamte se s metodami pro řízení parametrů a jejich využitím pro automatický návrh řešení vybraných úloh.
3. Zpracujte rešerši na výše uvedená témata.
4. Navrhněte metodu pro automatické řízení parametrů kartézského genetického programování.
5. Navrženou metodu implementujte.
6. Navrženou metodu experimentálně ověřte na zvoleném problému a proveďte porovnání s referenčním přístupem.
7. Zhodnoťte dosažené výsledky a diskutujte možnosti dalšího pokračování.

Literatura:

- Miller, J. F. Cartesian genetic programming. Springer Berlin Heidelberg, 2011. Natural computing series. ISBN 978-3-642-17309-7.
- Miller, J. F. Cartesian genetic programming: its status and future. Genetic programming and evolvable machines. New York: Springer US, 2020, sv. 21, 1-2, s. 129–168. ISSN 1389-2576.
- Další dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 4 zadání.

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

Vedoucí práce: **Hurta Martin, Ing.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1.11.2024
Termín pro odevzdání: 21.5.2025
Datum schválení: 31.10.2024

Abstrakt

Cílem této práce je prozkoumat možnosti řízení parametrů u kartézského genetického programování (CGP). Při tomto přístupu dochází ke změně hodnot parametrů (například síly mutace) v průběhu evoluce na základě jejich vlivu na zlepšení fitness. Práce představuje jeden algoritmus pro dynamickou selekci, jeden algoritmus pro řízení parametrů mřížky a tři algoritmy pro adaptivní mutaci. Tyto metody a jejich kombinace byly experimentálně porovnány s referenčním přístupem, kde byly hodnoty parametrů určeny na základě manuálního ladění. Pouze u jedné metody řízení mutace byl prokázán přínos. Tento mechanismus eliminuje potřebu ladění parametru mutace, čímž zjednodušuje a urychluje proces evolučního návrhu.

Abstract

This thesis aims to explore the possibilities of parameter control in the area of cartesian genetic programming (CGP). With this approach, parameters (for instance, mutation probability) are changed on the run based on their impact on fitness improvements. This thesis introduces one method for selection control, one method for grid parameters control, and three algorithms for dynamic mutation. These mechanisms and their combinations were experimentally compared to the reference approach, where parameter values are determined based on manual parameter tuning. Only one algorithm for mutation control has proven to be useful. This method eliminates the necessity of mutation tuning, which simplifies and speeds up an evolutionary design.

Klíčová slova

Evoluční algoritmy, symbolická regrese, kartézské genetické programování, řízení parametrů

Keywords

Evolutionary algorithms, symbolic regression, cartesian genetic programming, parameter control

Citace

DOLEŽAL, Petr. *Řízení parametrů kartézského genetického programování*. Brno, 2025. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Hurta

Řízení parametrů kartézského genetického programování

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana Ing. Martina Hurty. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....

Petr Doležal
18. dubna 2025

Poděkování

Děkuji vedoucímu Ing. Martinu Hurtovi za jeho čas, ochotu, cenné rady a aktivní přístup při vedení mé práce.

Obsah

1	Úvod	6
2	Evoluční algoritmy	7
2.1	Jednoduchý evoluční algoritmus	7
2.2	Interní reprezentace	8
2.3	Operátory reprodukce	9
2.4	Výpočet hodnoty fitness	10
2.5	Selekční mechanismy	10
2.6	Velikost populace	12
2.7	Evoluční algoritmy s vícero populacemi	13
3	Genetické programování	14
3.1	Kartézské genetické programování	15
3.2	Symbolická regrese	18
4	Řízení parametrů	21
4.1	Automatizované nastavení hodnot parametrů	21
4.2	Ladění parametrů	21
4.3	Metody řízení parametrů	22
4.4	Existující řešení řízení parametrů CGP	24
5	Návrh metody pro automatické řízení parametrů kartézského genetického programování	26
5.1	Řízení selekčního mechanismu	27
5.2	Řízení mutace	28
5.3	Řízení parametrů mřížky CGP	31
5.4	Návrh experimentů	32
6	Implementace	34
6.1	Inicializace populace	36
6.2	Vyhodnocování fitness	36
6.3	Tvorba nové generace	37
6.4	Mutace	38
7	Experimentální vyhodnocení adaptivních metod	39
7.1	Parametry referenčního přístupu	41
7.2	Ladění statických parametrů adaptivních metod	42
7.3	Návrh nové metody řízení parametrů na základě výsledků experimentů	49

7.4	Adaptivní mutace jako potenciálně vhodná první volba mutace	51
8	Závěr	54
	Literatura	55

Seznam obrázků

2.1	Asexuální reprodukce. Kandidátní řešení zakódované genotypovou reprezentací o velikost 12 bitů bylo zmutováno tak, že došlo k bitové inverzi dvou náhodně zvolených hodnot.	9
2.2	Sexuální reprodukce. Dvě kandidátní řešení o velikosti 12 bitů jsou zkřížena. Bod křížení byl vybrán náhodně. Výsledkem jsou 2 noví potomci, kteří zdědili část genotypu od prvního rodiče a zbytek od druhého rodiče.	9
3.1	Základní koncept genetického programování, převzato z [12], upraveno. Počáteční populace nemusí být nutně vygenerována náhodně. Kvalita programu je určena podle toho, jak moc se výstupy programu liší od očekávaných výstupů. Vytvoření programu probíhá za pomoci genetických operátorů (mutace, křížení...). Výsledkem je spustitelný program.	15
3.2	GP reprezentace pomocí syntaktického stromu, převzato z [12], upraveno. Tento program vrátí jako výsledek vyšší číslo z obou podstromů ($x + x$, nebo $x + 3y$).	15
3.3	Křížení ve stromovém GP. U dvou rozdílných stromů se vybere náhodně jeden uzel, jakožto bod křížení. Potomek vznikne tak, že se zvolený podstrom jednoho jedince (stromu) nahradí zvoleným podstromem druhého kandidátního řešení. V tomto případě vznikl pouze jeden potomek, nicméně je možné vytvořit i druhého potomka nahrazením vybrané části červeného stromu podstromem z modrého jedince.	16
3.4	Mutace ve stromovém GP. Jedná se prakticky o křížení jednoho kandidátního řešení s novým jedincem, kde byl jako bod křížení zvolen kořenový uzel. Podstrom rodiče je nahrazen celým novým kandidátním řešením.	17
3.5	Kartézské genetické programování. Jedná se o mřížku uzlů, kde funkce jsou vybírány z množiny základních funkcí: $\{+, -, *, /, \sin, \cos\}$. Mřížka obsahuje $n_c = 3$ sloupců a $n_r = 2$ řádků. Parametr <i>lback</i> je nastaven na hodnotu nc , výstup tedy může být brán z libovolného uzlu v předchozích sloupcích nebo vstupu. Tato mřížka také obsahuje n vstupů a m výstupů. Každý uzel má tolik vstupů, jaká je maximální arita napříč funkcemi. Genotypová reprezentace (která se nachází pod mřížkou) obsahuje skupinu trojic čísel (vstupy uzlu a jeho funkce) a samostatná čísla (zdroje výstupů). Uzly 8 a 9 jsou neaktivní, protože je nevyužívá žádný výstup.	18

3.6	Symbolická regrese. Nejprve je vytvořena počáteční populace kandidátních řešení (například náhodně). Tito jedinci (spustitelné programy) jsou vyvíjeni pomocí křížení či mutací. Selektční tlak upřednostňuje přesnější programy před méně přesnými, kdy méně vhodní jedinci jsou z populace odstraněni. Jakmile dojde k nalezení dostatečně přesného programu (nebo například vypršení časového limitu či přesáhnutí maximálního povoleného počtu generací), je běh symbolické regrese ukončen a dojde k navrácení nejvhodnějšího nalezeného jedince.	20
4.1	Klasifikace nastavování parametrů, převzato z [3], upraveno.	25
5.1	Adaptivní řízení selekčního mechanismu. Na začátku jsou jedinci vyvíjeni pomocí proporční selekce založené na fitness. Po několika úspěšných vylepšení této hodnoty začne selekce pracovat proporčně podle pořadí, kde dochází k postupnému zvyšování selekčního tlaku na základě parametru η^+ (pokud se fitness zvyšuje). Jakmile tento parametr dosáhne maximální hodnoty, selekce začne pracovat na základě turnajového výběru. S dalším vylepšením fitness se zvyšuje počet turnajů a nakonec dojde k přepnutí do algoritmu $(\mu + \lambda)$. Při stagnaci fitness ve fázi $(\mu + \lambda)$ může dojít k snížení selekčního tlaku a zpětnému nasazení turnajového výběru. Při stagnaci u turnajové selekce dochází ke snižování počtu turnajů, avšak do stádia proporční selekce se evoluční algoritmus už zpět nedostane.	28
6.1	Diagram tříd volitelně řízených komponent CGP, zjednodušeno. Konstruktor CGP vyžaduje instance potomků tří abstraktních tříd (Selection , Grid , Mutation). V každé abstraktní třídě je deklarována (i implementována) metoda update , která zodpovídá za řízení parametrů a která se pravidelně spouští během evolučního výpočtu. Tato metoda má v rodičovských třídách chování definováno tak, že nedělá nic, přičemž statické komponenty (mutace, selekce, mřížka) tuto metodu přímo dědí. Dynamické mechanismy však zmíněnou metodu přepisují vlastním chováním (například třída AdaptiveMutation mění pravděpodobnost mutace podle zlepšování nebo stagnace průměrné fitness v populaci). Ostatní metody na diagramu (konkrétně create_new_generation() , create() a mutate()) mají zpravidla vlastní chování (jedinou výjimkou je metoda mutate() , která je společná jak pro statickou mutaci, tak pro mutaci adaptivní).	35
6.2	Diagram vztahů mezi entitami Candidate , Grid a Node	36
7.1	Symbolická regrese na úlohách ze sbírky samostatně.	40
7.2	Porovnání vybraného nastavení symbolické regrese za použití dvou stejným způsobem vytvořených sad bodů. Graf obsahuje výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	41
7.3	Srovnání nastavení pravděpodobnosti mutace napříč čtyřmi úlohami ze sbírky. Selektčním algoritmem byl vždy mechanismus (1+4) a mřížka byla pokaždé nastavena na $n_r = 1, n_c = 20, lback = n_c$. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	42
7.4	Ladění statických parametrů mřížky CGP. Selektčním algoritmem byl mechanismus (1+4) a mutace byla $p = 0, 1$ (ve všech zkoumaných případech). Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	42

7.5	Hledání vhodného nastavení parametru P , viz sekce 5.3. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	43
7.6	Hledání vhodného nastavení parametru G a porovnání s referenčním přístupem. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	43
7.7	Porovnávání různých nastavení parametru <code>stddev</code> samo-adaptivní mutace a referenčního přístupu ($p=0,1$). Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	44
7.8	Hledání vhodného parametru G , který u adaptivní mutace značí, po kolika generacích stagnace fitness došlo k uvážnutí v lokálním optimu. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	45
7.9	Hledání vhodného parametru m , který u adaptivní mutace udává, jak moc se má změnit fitness, pokud došlo ke zlepšení průměrné fitness nebo uvážnutí v lokálním optimu. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	45
7.10	Porovnání adaptivní mutace a referenčního přístupu. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	46
7.11	Ladění parametru <code>mod</code> u adaptivní selekce. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	46
7.12	Ladění parametru G u adaptivní selekce a porovnání s referenčním přístupem. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	47
7.13	Porovnání referenčního přístupu a vyladěné adaptivní selekce se dvěma druhy řízených mutací. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	47
7.14	Porovnávání výkonů adaptivní selekce, která byla nastavena tak, aby se podobala selekčnímu algoritmu $(\mu + \lambda)$. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	48
7.15	Porovnání referenčního přístupu a různě nastavené samo-adaptivní mutace za využití turnajového výběru jakožto selekčního algoritmu. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	48
7.16	Ladění parametrů nové adaptivní mutace. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	51
7.17	Porovnávání referenčního přístupu (za využití dalších mutací, nejen nejlepší $p = 0, 1$) a nové adaptivní mutace, která byla vytvořena jako komponenta k selekčnímu algoritmu $(\mu + \lambda)$. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.	51
7.18	Porovnání adaptivní mutace a statických mutací na úlohách z PMLB . Mřížka byla nastavena na $n_r = 1$, $n_c = 20$, $lback = nc$. Graf neobsahuje odlehle hodnoty.	52
7.19	Porovnání adaptivní mutace a statických mutací na úlohách z PMLB . Mřížka byla nastavena na $n_r = 1$, $n_c = 150$ a $lback = 150$. Graf neobsahuje odlehle hodnoty.	53
7.20	Porovnání adaptivní mutace a statických mutací na úlohách z PMLB . Mřížka byla nastavena na $n_r = 1$, $n_c = 40$ a $lback = 1$. Graf neobsahuje odlehle hodnoty.	53

Kapitola 1

Úvod

Evoluční algoritmy spadají do oblasti strojového učení. Nabízejí alternativní přístup k v současné době nejčastěji používaným neuronovým sítím. Evoluční algoritmy se dlouhodobě ukazují jako skvělý prostředek pro řešení určitých komplikovaných úloh. Pomocí evolučních algoritmů lze řešit například: problém obchodního cestujícího, návrh obvodů (kde výsledky jsou minimálně srovnatelné s obvody navrženými profesionálními lidskými návrháři [12]), symbolickou regresi, tvorbu klasifikátorů nebo posilované učení.

Schopnost (nejen) evolučních algoritmů efektivně řešit dané úkoly velmi závisí na zvolených počátečních parametrech. Mezi takové parametry může patřit například velikost populace, pravděpodobnost mutace nebo selekční mechanismus. Nesprávné nastavení může vést k příliš dlouhému běhu algoritmu nebo neschopnosti nalézt požadované řešení.

Správné zvolení parametrů je složitý úkol, protože mimo znalosti z oblasti evolučních algoritmů vyžaduje hlubokou znalost řešeného problému, a i tak je mnohdy nemožné zvolit napoprvé vhodné nastavení. Neexistuje standardizovaný postup, jak tyto parametry obecně určit pouze na základě popisu vlastností úlohy a bez spouštění experimentálních běhů. Různé úlohy vyžadují velmi odlišné nastavení parametrů. To platí i v případě, že jsou tyto úlohy řešeny stejným algoritmem (například pomocí kartézského genetického programování (CGP)).

Existují nástroje na automatické a vhodné stanovení takového nastavení. Tyto nástroje nicméně porovnávají velké množství kombinací parametrů, což je časově náročné. Uživatel tak může čekat na řešení poměrně dlouho. Tato práce si klade za cíl prozkoumat možnosti automatického určování parametrů přímo za běhu algoritmu na základě průběhu evoluce, bez nutnosti spouštět program vícekrát.

Měnit parametry v průběhu evoluce nad určitou populací může být užitečné. Například velké mutace na začátku běhu mohou přispět k rychlému nalezení poměrně kvalitního řešení, následné snížení agresivity mutací pak může zajistit, že se kandidátní řešení budou lépe přibližovat nejbližšímu lokálnímu optimu. Při ponechání velkých mutací by se populace pohybovala směrem k lokálnímu optimu obtížněji. Naopak, pokud by na začátku evoluce byly povoleny pouze malé mutace, řešení by mohlo snadněji uváznout v nejbližším lokálním maximu.

V kapitole 2 se nachází základní popis evolučních algoritmů a jejich terminologie. Kapitola 3 obsahuje informace o genetickém programování, jakožto důležité instance evolučních algoritmů, která je podstatná z hlediska této práce. V kapitole 4 je detailněji popsáno využití řízení parametrů pro evoluční algoritmy. V následující kapitole 5 je popsán návrh několika metod pro řízení parametrů CGP. Kapitola 6 obsahuje popis implementace řešení a v kapitole 7 se nachází závěrečné experimentální vyhodnocení.

Kapitola 2

Evoluční algoritmy

Evoluční algoritmy jsou dlouho známým konceptem. Poprvé se v literatuře objevily v 50. letech minulého století. V následujících letech byly představeny tři důležité třídy algoritmů, které významně ovlivnily tuto oblast. Jednalo se o evoluční strategii (ES), genetické algoritmy (GA) a evoluční programování (EP). ES byla představena jako prostředek k řešení složitých problémů založených na optimalizaci hodnot z oboru reálných čísel [2]. EP byla vytvořena pro automatizovaný návrh konečných automatů, které určovaly chování inteligentních agentů [2]. GA využívaly genotypu (viz sekce 2.2), který byl dekodován a jehož kvalita byla až poté vyhodnocována. V prvních aplikacích těchto algoritmů se jednalo zpravidla o bitové řetězce. Tento postup se ukázal jako efektivní například při problémech plánování přístupů k omezeným prostředkům [19].

Tyto třídy algoritmů byly zkoumány až do 90. let minulého století relativně nezávisle na ostatních, což vedlo k nejednotnému názvosloví. Jakmile začalo docházet ke stále častějším interakcím mezi těmito skupinami, začalo být jasné, že bude třeba tuto terminologii sjednotit. Zmíněné třídy byly označeny jako důležité instance „Evolučních algoritmů“, přičemž ustálenou terminologií popisuje tato kapitola. [2]

Mimo tyto 3 koncepty existuje například i genetické programování (GP), které pracuje s populací spustitelných programů, viz kapitola 3.

2.1 Jednoduchý evoluční algoritmus

Evolučním algoritmem se obecně myslí evoluční proces, který zahrnuje populaci jedinců, kteří se postupně mění a na které je vyvíjen selekční tlak (ve smyslu přežití nejsilnějších). Je důležité zmínit, že evoluční algoritmy se nesnaží o vytvoření co nejvěrnější kopie nějakého známého evolučního systému, ale jsou evolucí pouze inspirovány. Omezení skutečných evolučních systémů se tedy nemusí nutně proplat do aplikovaných evolučních algoritmů. Tyto algoritmy pracují následovně a vážou se k nim následující otázky: [2]

- Obsahují populaci jedinců (kandidátních řešení).
 - Jak velká bude populace?
 - Jak přesně bude reprezentace jedinců vypadat?
- Každý takový jedinec má hodnotu „fitness“, která udává jeho kvalitu (viz sekce 2.4).
 - Jak vhodně navrhnout fitness funkci?

- Určití jedinci jsou zvoleni, aby vytvořili potomky, kteří jsou podobní, ale nikoli stejní, jako rodiče.
 - Jakým způsobem jsou rodiče zvoleni?
 - Jak jsou potomci vytvořeni z rodičů?
 - Kolik potomků by mělo být vytvořeno?
- Jedinci jsou určeni k zahynutí (tedy k odstranění z populace).
 - Na základě čeho se rozhodne o přežití jedince?

Obecnou šablonu evolučního optimalizačního algoritmu popisuje algoritmus 1. Pokud je takový evoluční algoritmus spuštěn po dobu několika generací (cyklus života a smrti), selekční mechanismus na základě volby jedinců s vyšší fitness zajistí postupné zvyšování průměrné fitness hodnoty, čímž dochází k optimalizaci řešení. [2]

Algorithm 1: Šablona jednoduchého evolučního optimalizačního algoritmu [2]

```

Vytvoř náhodně základní populaci o velikosti  $M$ ;
while Nejsou splněna ukončovací kritéria do
    | Vyber rodiče a vytvoř z nich potomky;
    | Odstraň určité jedince z populace;
return Nejlepší jedinec;

```

2.2 Interní reprezentace

Existují dvě interní reprezentace, které byly inspirovány v biologii: genotypová a fenotypová. Genotypová reprezentace kóduje problém v podobném duchu jako genetický kód, tedy jako řetězce obsahující srozumitelné znaky. Například u tradičních GA se jedná o binární řetězce. Operátory reprodukce jsou vysoce přenositelné napříč různými úlohami (například mutace, která je implementována binární negací určitých bitů, může být použita ve stejném duchu napříč úlohami). Aplikování již vytvořeného evolučního algoritmu na jiný problém tak zahrnuje pouze vytvoření nového systému kódování a dekódování kandidátních řešení. [2]

Oproti tomu fenotypová reprezentace pracuje na přímo s řešeným problémem bez mezikódu (například vektory reálných čísel u ES). Reprodukční operátory (viz 2.3) jsou definovány takovým způsobem, aby přímo manipulovaly s jedincem (kandidátním řešením). Tímto způsobem je možné evoluční algoritmus lépe přizpůsobit řešenému problému, nicméně daný algoritmus je jen velmi obtížně přenositelný. [2]

Níže se nachází příklad stejného jedince, který je kandidátním řešením problému o třech hodnotách, reprezentovaného genotypovou i fenotypovou reprezentací: [19]

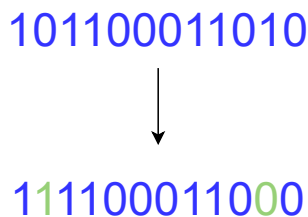
Fenotypová reprezentace: $\langle 32, 56; 18, 21; 9, 83 \rangle$

Genotypová reprezentace: $\langle 000111010100, 110100101101, 001001101011 \rangle$

2.3 Operátory reprodukce

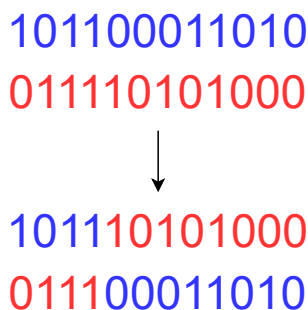
Operátory reprodukce slouží k tomu, aby bylo možné z rodičů (vybraných jedinců v populaci) vytvořit jim podobné potomky. Z biologického hlediska k tomu může dojít dvěma způsoby: sexuálně a asexuálně. [2]

Asexuální reprodukce spočívá v naklonování jednoho rodiče a poté jeho mutaci, aby došlo k variabilitě (mutace může zlepšit kvalitu jedince, zhoršit, nebo nemusí mít vůbec žádný vliv). Tato strategie bývá hojně využívána u ES a EP. Tento typ reprodukce znázorňuje obrázek 2.1. [2]



Obrázek 2.1: Asexuální reprodukce. Kandidátní řešení zakódované genotypovou reprezentací o velikost 12 bitů bylo zmutováno tak, že došlo k bitové inverzi dvou náhodně zvolených hodnot.

Oproti tomu sexuální reprodukce zahrnuje kombinace více než jednoho rodiče, potomek poté nese určité znaky každého ze svých rodičů. Tato strategie bývá využívána hojně u GA, kde potomci jsou vytvořeni na základě kombinace dvou rodičů, viz obrázek 2.2. Z hlediska evolučních algoritmů není však nutné se omezit na právě dva rodiče, rodičů může být i víc (není nutné se omezovat reálnou přírodou, jak již bylo zmíněno v sekci 2.1). [2]



Obrázek 2.2: Sexuální reprodukce. Dvě kandidátní řešení o velikosti 12 bitů jsou zkřížena. Bod křížení byl vybrán náhodně. Výsledkem jsou 2 noví potomci, kteří zdělili část genotypu od prvního rodiče a zbytek od druhého rodiče.

Z hlediska prohledávání se tyto dvě taktiky výrazně liší. Zatímco asexuální reprodukce slouží spíše k hledání lokálního maxima, sexuální reprodukce je více užitečná ke globálnímu prohledávání, protože se potomci často výrazně odlišují od svých rodičů. Vzhledem k tomu, že se tyto dva přístupy efektivně doplňují, velké množství dnešních evolučních algoritmů používá oba dva. [2]

2.4 Výpočet hodnoty fitness

Každému kandidátnímu řešení v populaci je přiřazena hodnota fitness, která určuje, do jaké míry správně řeší zkoumaný problém. Funkce, pomocí které je možné fitness určit, musí být nadefinována. Existuje vícero přístupů k fitness, mezi nejčastější patří „hrubá fitness“ a „standardizovaná fitness“. [16]

Hrubá fitness je přirozeným zhodnocením výsledků kandidátního řešení. Může se například jednat o počet metrů čtverečních, které uklízeční robot úspěšně vysál. Často, ale ne vždy, je tato fitness vypočítána na základě konečné množiny testovacích případů. Ty mohou mít formu například trénovacích vektorů, které obsahují vstupy pro vyvíjenou matematickou rovnici a očekávané výstupy. Hrubá fitness pak může být určena jako suma rozdílů očekávaných hodnot od skutečných [16]:

$$f_R(i) = \sum_{j=1}^N |S(i, j) - C(j)|^k, \quad (2.1)$$

kde $S(i, j)$ je hodnota vrácená jedincem i na základě vstupu j , $C(j)$ je správná a očekávaná hodnota uvedená v trénovacím vektoru. Hodnota k je přirozené číslo, často $k = 1$, nebo $k = 2$. Testovací případy zpravidla reprezentují malou podmnožinu všech možných vstupů a výstupů. [16]

Standardizovaná fitness je jinak formulována hrubá fitness, kde hodnoty blíže k nule značí kvalitnější jedince. V některých případech, kde je menší hrubá fitness lepší, odpovídá standardizovaná fitness té hrubé. Viz například rovnice 2.1. Pokud tomu tak není, standardizovaná fitness může být určena pomocí přičítání konstanty k hrubé fitness (nebo odečítání hrubé fitness od konstanty). Pokud pro nějaký problém platí, že čím je hrubá fitness větší, tím lepší je kandidátní řešení, je standardizovaná fitness takového jedince f_S definována jako $f_S = f_R^{max} - f_R$, kde f_R je hrubá fitness jedince a f_R^{max} je maximální možná hodnota hrubé fitness. [16]

2.5 Selekcční mechanismy

Mechanismus selekce určuje, kteří jedinci v populaci budou určeni jako rodiče, kolik potomků z nich bude vytvořeno a jací jedinci naopak budou z populace odstraněni. Selekcce se řídí primárně hodnotou fitness u kandidátních řešení, viz 2.1. Na první pohled se může zdát, že nejvhodnějšími rodiči budou jedinci s nejvyšší fitness, nicméně touto strategií dochází do jisté míry k hladovému prohledávání (anglicky greedy search). Takovéto algoritmy většinou vykazují znaky rychlé konvergence, nicméně zpravidla uváznou v lokálním optimu a jen někdy narazí na globálně nejlepší řešení. [2]

V některých případech může být lokální maximum dostatečné a přijatelné řešení, v jiných kvalita takového jedince nemusí stačit. Z toho plyne, že neexistuje jeden ideální selekcční mechanismus pro evoluční algoritmy. Určení správné míry selekcčního tlaku pro konkrétní problém je důležitou součástí návrhu evolučního algoritmu. Evoluční algoritmy zpravidla využívají fitness buď pro volbu rodičů, nebo pro určení přeživších. Zakládat obojí na hodnotě fitness může způsobit, že algoritmus bude nacházet pouze částečně optimální řešení. [2]

Další otázkou u EA je, jestli přeživší volit pouze z potomků, nebo z potomků a rodičů. Výběr z potomků i rodičů má opět tendenci více konvergovat do lokálního optima. Volení pouze z potomků může být pomalejší, ale má větší potenciál najít lepší řešení. Algoritmy

typu ES jsou rozděleny do dvou kategorií právě na základě toho, jestli je možné, aby rodič přežil do další generace, viz sekce 2.5. [2]

Selekční mechanismus nemusí být vždy snadné nastavit. Je třeba najít rovnováhu mezi globálním a lokálním prohledáváním. Velký důraz na lokální prohledávání povede k tomu, že dojde zpravidla k nalezení částečně optimálního řešení. Oproti tomu příliš velký důraz na globální prohledávání může vést k tomu, že nebude nalezeno ani lokální maximum. [2]

Například tradiční ES typicky mají elitářskou selekci (výběr N nejlepších jedinců, rodič se může dostat do další generace), která je vyrovnána velkou agresivitou mutací (během mutace dochází k velkým změnám na fenotypu). Oproti tomu tradiční GA mají menší agresivitu selekce. Rodiče nemohou být součástí nové generace a jsou určeni proporčně na základě fitness, nicméně potomci se od předků neliší tak výrazně, jako u ES. [2]

V následujícím seznamu jsou uvedeny nejčastěji používané selekční mechanismy, které jsou seřazeny od algoritmu s nejnižším selekčním tlakem po nejvyšší [1]:

- Proporční selekce podle fitness.
- Proporční selekce podle pořadí.
- Turnajový výběr.
- Selekcce (μ, λ) nebo $(\mu + \lambda)$.

Selekční tlak udává, jak moc se dbá na fitness jedinců při výběru potomků a přeživších. Při výběru nejlepšího jedince z celé populace jako rodiče je selekční tlak vyšší, než pokud by došlo k výběru 3 náhodných jedinců a z nich byl zvolen ten nejlepší. V rámci EA je selekční algoritmus více obecný než operátory reprodukce, je tedy mnohem snazší jej přenést z jednoho problému na druhý. [1]

Proporční selekce podle fitness

Jak již název napovídá, pravděpodobnost výběru je dána podle toho, jakou fitness má jedinec oproti ostatním kandidátním řešením v populaci, viz následující výraz [5]:

$$p_i = \frac{f_i}{\sum_{j=1}^M f_j}, \quad (2.2)$$

kde f_i značí fitness kandidátního řešení a M je velikost populace. Tato funkce předpokládá, že se snažíme fitness maximalizovat. Rovnici 2.2 je možné škálovat, například následujícím způsobem:

$$p'_i = \frac{f_i - c}{\sum_{j=1}^M f_j - M * c}, \quad (2.3)$$

kde M je opět počet jedinců v populaci, f_i fitness jedince a c je konstanta ($c > 0$). Oproti neškálované verzi této funkce zde dochází k větší pravděpodobnosti výběru nadprůměrných kandidátních řešení na úkor podprůměrných jedinců. [1]

Proporční selekce podle pořadí

Při využití tohoto selekčního mechanismu dochází k seřazení jedinců v populaci podle fitness. Pravděpodobnost výběru jedince pak může být určena například následujícím způsobem:

$$p_i = (\eta^+ - (\eta^+ - \eta^-) * \frac{i-1}{M-1})/M, \quad (2.4)$$

kde M značí velikost populace, η^+ a η^- jsou konstanty. Jedinec s nejlepší fitness má index 1 a nejhorší M . Aby platila podmínka $\sum_{i=1}^M p_i = 1$, je nutné, aby platilo $1 \leq \eta^+ \leq 2$ a současně $\eta^- = 2 - \eta^+$. Obecně se doporučuje nastavit $\eta^+ = 1,1$. Zvyšování této hodnoty zvýší i selekční tlak. [1]

Turnajový výběr

Selekční metoda založená na turnajovém výběru vybere nejlepšího jedince z $q \geq 1$ kandidátních řešení, která jsou náhodně vybrána z populace. Tento proces je možné opakovat (pro volbu rodičů, kteří vytvoří potomky, či určování přeživších). Často dochází k nastavení $q = 2$ [4]. Speciálním případem je náhodný průchod, kde platí $q = 1$. Zatímco u běžného turnaje ($q = 2$) dochází k většímu selekčnímu tlaku, než je tomu u proporcionální selekce podle fitness, při takto speciálně nastavené hodnotě $q = 1$ nehraje fitness vůbec roli. Při nastavení $q = 10$ dochází k obrovskému navýšení selekčního tlaku (vůči běžné variantě). [1]

Selekční mechanismy (μ, λ) a $(\mu + \lambda)$

Tyto typy selekcí jsou využívány v ES. Tyto metody byly původně navrženy k tomu, aby snížily velikost populace (která byla rozšířena tvorbou potomků pomocí křížení nebo mutací) z hodnoty $\lambda \geq \mu$ zpět na hodnotu počtu rodičů μ . Zatímco (μ, λ) volí μ nejlepších potomků jakožto rodičů nové generace, $(\mu + \lambda)$ vybírá jak z potomků, tak z rodičů. Obě metody jsou kompletně deterministické, vybírají totiž nejlepší jedince přímo, ne na základě pravděpodobnosti. [1]

2.6 Velikost populace

V reálných evolučních systémech se zpravidla velikost populace mění. Zvyšuje se a snižuje se na základě mnoha faktorů. Doposud se ale nepodařilo prokázat, že by nezafixovaná velikost populace v EA byla významně lepší, než zafixovaná. Vzhledem k tomu, že EA s pevnou velikostí populace jsou jednodušší na implementaci, využívá většina EA tento přístup, kdy je velikost populace pevně určena jako M . Při tvorbě nové generace nicméně dochází k vytvoření K nových potomků a populace tak dočasně přesáhne svůj limit. Je tedy třeba snížit velikost populace z $M + K$ zpět na M . [2]

Tvůrci EA musí navrhnout dvě hodnoty: velikost populace M a počet potomků K . Tyto hodnoty se napříč různými EA zásadně liší. Hodnota M určuje stupeň paralelismu, který je při prohledávání potřeba. Při prohledávání ve velkém množství dimenzí je zpravidla účinnější použít větší hodnotu M , než u dvou dimenzionálního prohledávání. Hodnota K určuje, kolik prohledávání bude provedeno na základě současné populace, než bude vytvořena nová. Opět platí, že vyšší hodnota K se hodí při komplexnějším prohledávání. U různých druhů EA se projeví jako relativně účinné jinak nastavené parametry, viz tabulka 2.1. [2]

Parametr	ES	EP	GA
M	1	10	50
K	1	10	50

Tabulka 2.1: Tabulka, která popisuje, jaké přibližné hodnoty parametrů (velikosti populace M a počtu potomků K) se obvykle používají u evoluční strategie (ES), evolučního programování (EP) a genetických algoritmů (GA) [2].

Při návrhu parametrů M a K je potřeba vycházet ze předešlých zkušeností a zpravidla různá nastavení i experimentálně ověřit [2].

2.7 Evoluční algoritmy s vícero populacemi

Existují různé přístupy k paralelizaci EA. Jedním z nich je tzv. „ostrovní model“, který se prokázal jako účinný při řešení velkých a složitých optimalizačních problémů. V tomto modelu je spuštěno několik EA, každý z nich pracuje odděleně, nicméně místy dochází k migraci jedinců z jednoho ostrova na jiný. Tímto způsobem může být využito současných hardwarových architektur pro řešení vysoce náročných paralelních výpočtů. [2]

Nad rámec ostrovního modelu existuje řada EA, které obsahují vícero populací. Tyto algoritmy jsou inspirovány biologickými principy, kdy spolu jedinci napříč populacemi soupeří, nebo kooperují. Typickým příkladem v přírodě je model dravec–kořist. Takovéto ko-evoluční algoritmy se prokázaly jako lepší v porovnání s klasickými EA například v úloze hraní počítačových her. [2]

Kapitola 3

Genetické programování

Genetické programování je souhrn technik založených na evolučním výpočtu, který umožňuje počítačům, aby řešily problémy automaticky. Tento přibližně 20 let starý koncept byl v minulosti úspěšně využit pro řešení problémů v oblastech jako návrh obvodů, tvorba celulárních automatů nebo tvorba inteligentních počítačových agentů hrajících hry, kde výsledky byly srovnatelné s lidskými výtvoři a dokonce byly pomocí něj navrženy patentovatelné vynálezy. [12]

V genetickém programování se pracuje s populací programů. S narůstajícím počtem generací dochází zpravidla k postupnému vylepšování kvality těchto programů (viz obrázek 3.1). GP je však náhodný proces, tudíž zlepšování fitness v čase není garantováno. Na druhou stranu se z důvodu této nahodilosti může vyhnout lokálním maximům, kde by jinak mohly uváznout deterministické metody. [12]

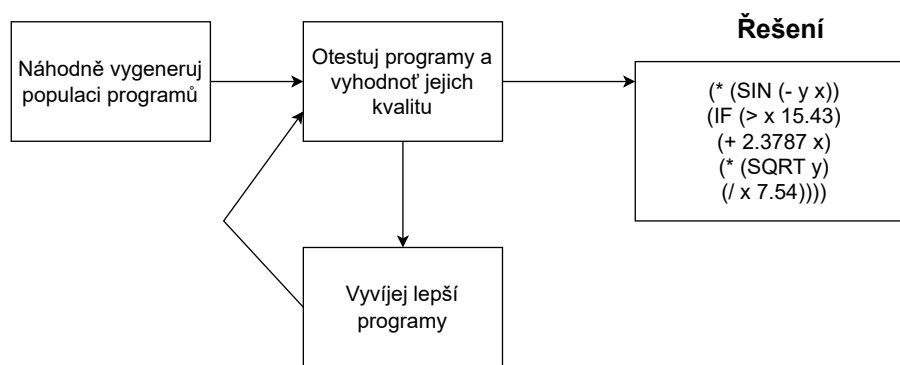
Kvalita kandidátních řešení (programů) může být posuzována vícero způsoby. Například na základě rozdílů mezi očekávaným výstupem a skutečným výstupem jedince, podle času (či třeba zdrojů) potřebného k dosažení určitého stavu nějakého systému (z nějakého počátečního stavu) nebo podle přesnosti klasifikátoru. [12]

V některých případech je tedy podstatný pouze výstup programu, jindy může být důležité sledovat akce, kterých se program za běhu dopustil. V každém případě je fitness určena podle toho, jak program reagoval na velkou množinu vstupů (potenciálně i za velmi různorodých okolností). Program může být testován například na všech možných kombinacích vstupu $x_1, x_2 \dots x_n$. Ohodnocení programu pro kontrolu robota může být zase založeno na základě jeho chování, přičemž při každém běhu startuje na jiné počáteční pozici. [12]

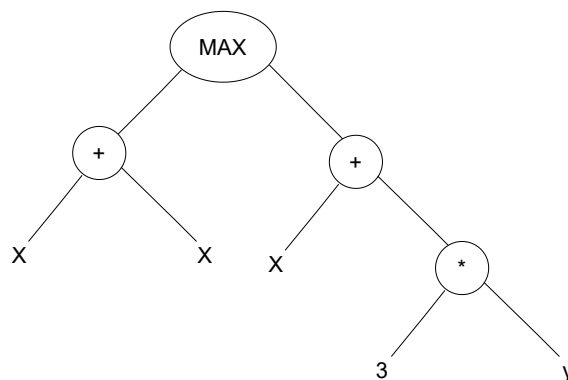
Hlavními genetickými operátory jsou křížení a mutace, viz sekce 2.3. Výhodou GP je, že je schopno automaticky řešit různé úkoly, aniž by uživatel musel takovým problémům sám rozumět. [12]

V GP jsou programy reprezentovány většinou pomocí syntaktických stromů, použití řádků kódu je méně časté. Na obrázku 3.2 se nachází stromová reprezentace programu $\max(x+x, x+3*y)$. Proměnné a konstanty v programu ($x, y, 3$) jsou uloženy v listových uzlech. V terminologii GP jsou označovány jako terminály, zatímco aritmetické operace ($+, *, \max$) jsou nelistové uzly zvané funkce. Každý GP systém obsahuje množinu povolených terminálů a funkcí, ze kterých jsou tyto syntaktické stromy tvořeny. [12]

GP se liší od ostatních EA v implementaci křížení a mutace. Nejčastější formou křížení je křížení na základě podstromů. U dvou rodičů je bod křížení nezávisle na sobě zvolen jako jeden uzel. Po zvolení takového bodu jsou vytvořeni potomci pomocí prohození podstromů rodičů. V případě potřeby tak může být vytvořen i pouze jeden potomek, a to zahazením druhé kombinace. Tento proces je znázorněn na obrázku 3.3. [12]



Obrázek 3.1: Základní koncept genetického programování, převzato z [12], upraveno. Počáteční populace nemusí být nutně vygenerována náhodně. Kvalita programu je určena podle toho, jak moc se výstupy programu liší od očekávaných výstupů. Vytvoření programů probíhá za pomoci genetických operátorů (mutace, křížení...). Výsledkem je spustitelný program.



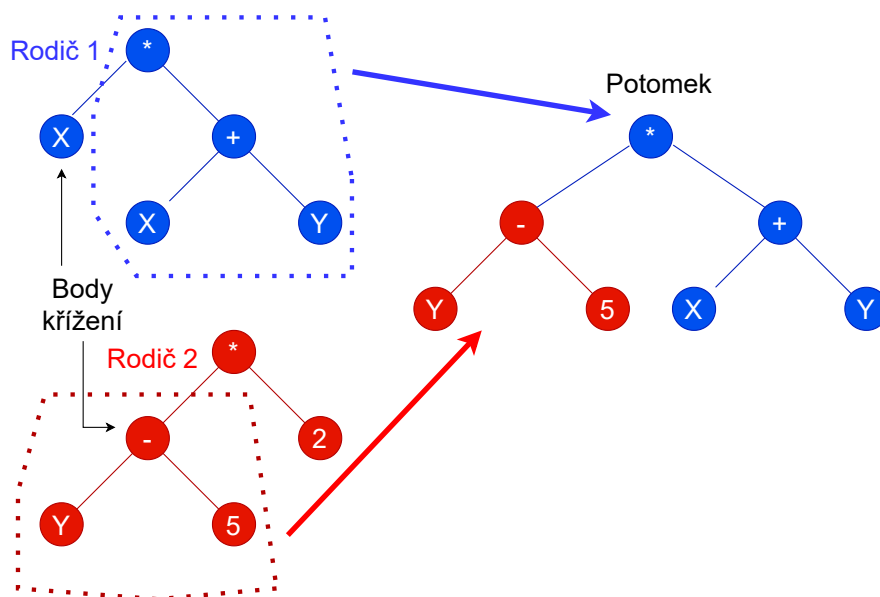
Obrázek 3.2: GP reprezentace pomocí syntaktického stromu, převzato z [12], upraveno. Tento program vrátí jako výsledek vyšší číslo z obou podstromů ($x + x$, nebo $x + 3y$).

Nejobvyklejší forma mutace (tzv. podstromová mutace) náhodně vybere uzel ve stromu a nahradí jej náhodně vygenerovaným stromem. Jedná se prakticky o křížení kandidátního řešení a nově vygenerovaného programu. Obrázek 3.4 ukazuje, jak probíhá taková mutace u stromového genetického programování. [12]

3.1 Kartézské genetické programování

Kartézské genetické programování (CGP) je zavedený koncept, který vznikl na základě metody používané k vyvíjení digitálních obvodů představené Julianem F. Millerem a spoluautory v roce 1997 [9]. Kandidátní řešení (jedinci) jsou zakódována pomocí grafů. Toto kódování obsahuje i geny, které jsou redundantní. Bylo prokázáno, že tyto nadbytečné geny vedou k zefektivnění prohledávání. CGP se brzy po svém představení začalo hojně používat jako nová forma genetického programování. [10]

Oproti stromovému genetickému programování, které bylo popsáno v sekci 3, jsou zde výpočetní struktury zakódovány (standardně) pomocí orientovaného acyklického grafu.

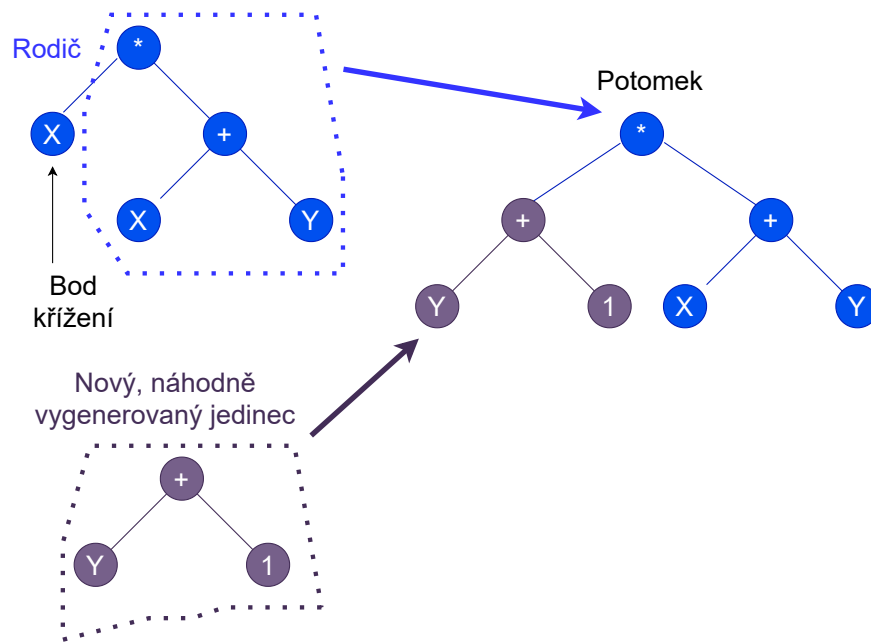


Obrázek 3.3: Křížení ve stromovém GP. U dvou rozdílných stromů se vybere náhodně jeden uzel, jakožto bod křížení. Potomek vznikne tak, že se zvolený podstrom jednoho jedince (stromu) nahradí zvoleným podstromem druhého kandidátního řešení. V tomto případě vznikl pouze jeden potomek, nicméně je možné vytvořit i druhého potomka nahrazením vybrané části červeného stromu podstromem z modrého jedince.

Tato podoba byla z velké části ovlivněna původním záměrem vytvořit evolvovatelnou strukturu pro obvody. Výhodou této reprezentace je, že určité uzly mohou být opakovaně použity a rovněž lze pracovat s vícero výstupy najednou. Kromě obvodů může tato reprezentace popisovat například rovnice nebo algoritmy. [10]

Standardní CGP obsahuje programy, které jsou reprezentovány pomocí dvou dimenzionálního acyklického orientovaného grafu. Odtud pochází název „kartézské“. Každý uzel v grafu obsahuje určitou funkci a je zakódován pomocí několika celých čísel. Uzly obsahují gen, co se odkazuje na jednu z funkcí, která byla předem nadefinována uživatelem. Zbývající geny uzlu určují, odkud je brán vstup. Takovýmto vstupem může být výstup jiného uzlu nebo vstup programu. Maximální počet genů, určujících vstup, je definován funkcí, která využívá nejvyššího množství vstupů. Pokud funkce v uzlu vyžaduje menší počet vstupů, než kolik je k dispozici, jsou zbývající vstupy ignorovány. Uzly v prvním sloupci mohou brát vstup pouze ze vstupů programu, zatímco následující uzly mohou kromě vstupů programu využívat i výstupy z uzlů v předchozích sloupcích. Jak již bylo zmíněno, program může mít vícero výstupů. Tyto výstupy jsou brány z uzlů (nebo rovnou vstupů). Obrázek 3.5 popisuje standardní reprezentaci CGP. [10]

CGP má 3 volně nastavitelné grafové parametry, které určují možnosti propojení a jeho velikost. Jedná se o počet řádků n_r , počet sloupců n_c a parametr l (*lback*). Součin prvních dvou parametrů (n_r a n_c) stanovuje počet uzlů. Poslední uvedený parametr l omezuje, z kolika předchozích sloupců může uzel brát svůj vstup (tato omezující podmínka se nevztahuje na vstup programu). Pokud je hodnota tohoto parametru nastavena jako $l = 1$, uzly mohou čerpat vstup pouze z předchozího sloupce nebo zmíněného vstupu programu. Při $l = n_c$ je možné brát vstup z libovolného uzlu předešlých sloupců (nebo vstupu). Speciálním a často



Obrázek 3.4: Mutace ve stromovém GP. Jedná se prakticky o křížení jednoho kandidátního řešení s novým jedincem, kde byl jako bod křížení zvolen kořenový uzel. Podstrom rodiče je nahrazen celým novým kandidátním řešením.

používaným případem je nastavení $n_r = 1$ a $l = nc$, kdy dochází k jedno dimenzionální topologií. [10]

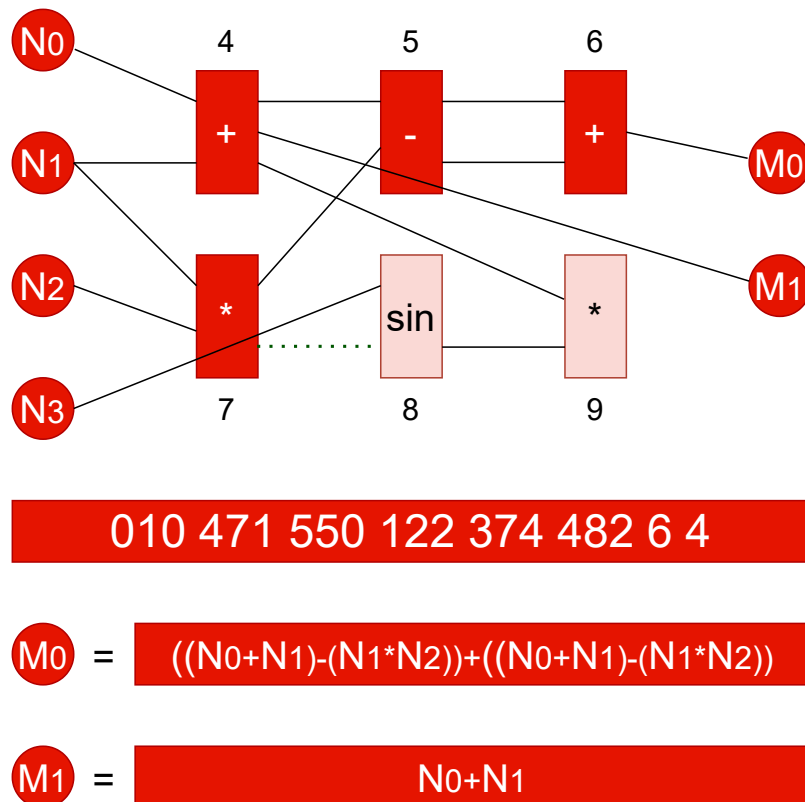
Pro získání výstupů na základě vstupů za účelem stanovení fitness (viz sekce 2.4) je potřeba nejprve určit aktivní uzly. Ne všechny uzly jsou využívány při vyhodnocování výstupů, není tedy nutné je zbytečně vyhodnocovat a tím zpomalovat běh algoritmu. Poté je třeba postupně určit hodnoty všech aktivních uzlů a nakonec stanovit i výstupní hodnoty. [9]

Určení aktivních uzlů probíhá rekurzivně. Vzhledem k tomu, že všechny výstupy jsou vždy využívány, je možné uzly, ze kterých výstupy berou data, označit rovněž za aktivní. Poté dochází k rekurzivnímu určení aktivních uzlů takovým způsobem, že pro všechny aktivní uzly se i jejich vstupy označí jako aktivní, a to až po vstupy. Pokud nějaký uzel obsahuje funkci s nižší aritou, než je maximální arita napříč funkcemi, jsou za aktivní označeny pouze ty uzly, které tento uzel s funkcí s nižší aritou skutečně využívá. [9]

Jakmile dojde k určení aktivních uzlů, je možné začít vyhodnocovat výstupní hodnoty. Postupně dochází k procházení aktivních uzlů od těch v prvním sloupci až po ty v posledním sloupci. Každý takovýto uzel si svoji vypočítanou hodnotu uloží. Jakmile dojde k stanovení všech výstupních hodnot, je vyhodnocování u konce. [9]

Co se prohledávání týče, CGP využívá přístup podobný ES. Jeden rodič se použije k vytvoření λ potomků. Následně se z celé populace vybere nejlepší jedinec. V případě, že se fitness rodiče shoduje s fitness některého z potomků, vždy se vybere potomek namísto rodiče. [10]

Tato poměrně jednoduchá strategie je vysoce efektivní mimo jiné z důvodů přítomnosti nevyužívaných genů v genotypu. Pro tyto geny platí, že výstup z uzlů, které reprezentují, se nevyužije při získávání výstupu programu na základě vstupu. Tyto geny mohou být postupně měněny neutrálními mutacemi. Aby k těmto užitečným neutrálním mutacím do-



Obrázek 3.5: Kartézské genetické programování. Jedná se o mřížku uzlů, kde funkce jsou vybírány z množiny základních funkcí: $\{+, -, *, /, \sin, \cos\}$. Mřížka obsahuje $n_c = 3$ sloupců a $n_r = 2$ řádků. Parametr *lback* je nastaven na hodnotu *nc*, výstup tedy může být brán z libovolného uzlu v předchozích sloupcích nebo vstupu. Tato mřížka také obsahuje *n* vstupů a *m* výstupů. Každý uzel má tolik vstupů, jaká je maximální arita napříč funkcemi. Genotypová reprezentace (která se nachází pod mřížkou) obsahuje skupinu trojic čísel (vstupy uzlu a jeho funkce) a samostatná čísla (zdroje výstupů). Uzly 8 a 9 jsou neaktivní, protože je nevyužívá žádný výstup.

cházel, je nutné vždy upřednostnit potomka před předkem, jak ukazuje algoritmu 2. Z toho plyne, že i když se může zdát, že algoritmus uvázl v lokálním optimu, ve skutečnosti stále dochází k prohledávání velkého množství různých potenciálních řešení. Jediným limitem je maximální velikost genotypu. [10]

CGP se na rozdíl od základního GP rovněž netýká problém označovaný jako „bloat“, při kterém dochází k neustálému růstu složitosti kandidátních řešení, které mají stejnou (nebo jen nepatrně vyšší) fitness. Zatím nebylo prokázáno, proč se CGP tento problém vyhýbá. [10]

3.2 Symbolická regrese

Symbolická regrese je zavedenou regresní metodou pracující například na základě evolučních výpočtů. Cílem této metody je prohledávání v rovině matematických výrazů. U tradičních regresních postupů je třeba najít vhodné parametry pro dosažení do rovnice v pevně stano-

Algorithm 2: Standardní CGP [9]

```
for  $i \leftarrow 0$  to 5 do
    └ Vytvoř náhodně jedince  $i$ ;
Zvol jedince s nejvyšší fitness jako rodiče;
while Není nalezeno řešení or nebyl dosažen limit generací do
    └ for  $i \leftarrow 0$  to 4 do
        └ Zmutuj rodiče pro zisk potomka  $i$ ;
        └ Zvol jedince s nejvyšší fitness jako rodiče, upřednostni potomka před rodičem;
```

veném tvaru. Symbolická regrese takto omezena není. V jejím průběhu dochází jak k hledání vhodného tvaru funkce, tak i vhodných parametrů. [14]

Původní rovnice (první generace jedinců, respektive kandidátních řešení) je zpravidla vygenerována náhodně. Tito jedinci se často skládají z algebraických operací (+, -, *, /), analytických funkcí (`sin`, `cos`), konstant (1, 20, π ...) a proměnných ($x_1, x_2 \dots x_n$). Nové rovnice jsou tvořeny křížením a mutacemi původních rovnic. Průběh symbolické regrese ukazuje obrázek 3.6. [14]

Selekční algoritmus upřednostňuje přesnější rovnice před méně přesnými a méně nadějnými (viz sekce 2.5). Kvalitu (přesnost) jedince lze určit například na základě fitness funkce s názvem střední kvadratická chyba (MSE):

$$\text{MSE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2, \quad (3.1)$$

kde n je počet pozorování, y_i je očekávaná hodnota a $\hat{y}_i$ je predikovaná hodnota. Jakmile je nalezeno takové kandidátní řešení, které splňuje požadovanou míru přesnosti, dojde k ukončení běhu algoritmu. Výsledkem je rovnice, která nejpresněji modeluje zkoumaný systém. [14, 18]

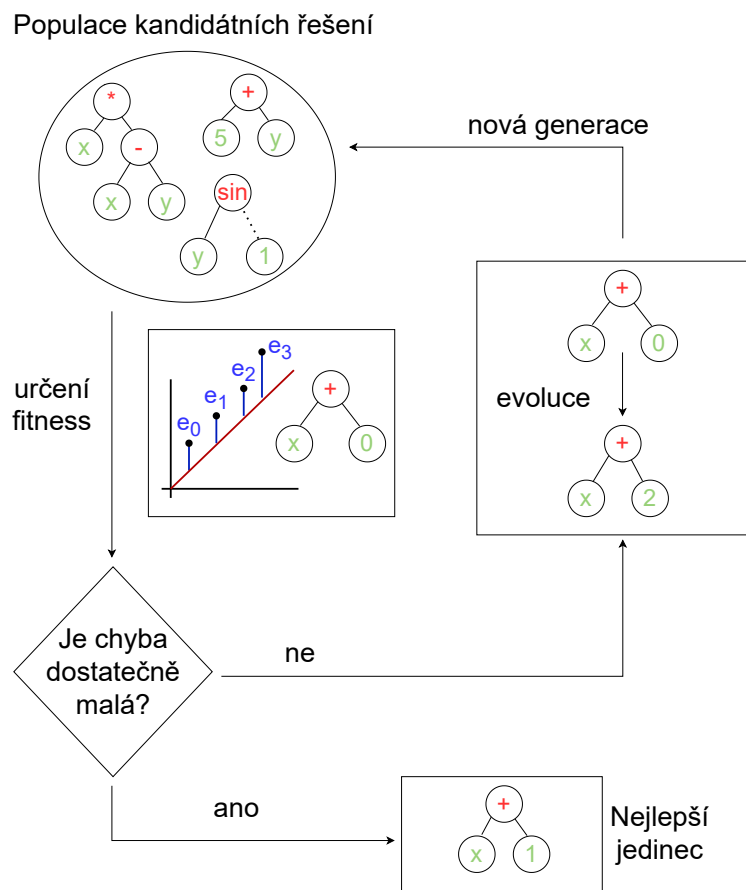
Symbolická regrese má i další výhody. Klasické modelování regresní křivky je proces, který vyžaduje čas a kvalifikované pracovníky. Při použití symbolické regrese je možné najít matematické vztahy mezi vstupy a výstupy ve zkoumané systému, aniž by uživatel měl jakékoli znalosti o takovém systému. [14, 18]

Regresní metody obecně poskytují výsledky (regresní křivky), které nemodelují vstupy na výstupy systému zcela přesně. To se týká i symbolické regrese. Naměřená data, tedy zmíněné výstupy systému, často obsahují šum. Sběr přesných dat může být téměř nemožný i při použití pokročilých a drahých zařízení. Regresní modely by z tohoto důvodu neměly být 100% přesné, ale je třeba, aby byly schopné do jisté míry generalizovat. [18]

Existuje spousta dalších modelovacích technik, například lineární regrese, nelineární regrese nebo neuronové sítě. Genetické programování a neuronové sítě jsou užitečné a lepší než klasické statistické metody v následujících případech [18]:

- Chování zkoumaného systému není známo a nelze jej odhadnout.
- Když jde o multidimenzionální regresi s velkým, nebo naopak malým množstvím dat.

Výhodou genetického programování oproti neuronovým sítím je schopnost vytvořit model, který je krátký, srozumitelný a interpretovatelný [18].



Obrázek 3.6: Symbolická regrese. Nejprve je vytvořena počáteční populace kandidátních řešení (například náhodně). Tito jedinci (spustitelné programy) jsou vyvíjeni pomocí křížení či mutací. Seleční tlak upřednostňuje přesnější programy před méně přesnými, kdy méně vhodné jedinci jsou z populace odstraněni. Jakmile dojde k nalezení dostatečně přesného programu (nebo například vypršení časového limitu či přesáhnutí maximálního povoleného počtu generací), je běh symbolické regrese ukončen a dojde k navrácení nejvhodnějšího nalezeného jedince.

Kapitola 4

Řízení parametrů

4.1 Automatizované nastavení hodnot parametrů

Při návrhu evolučního algoritmu je potřeba zvolit (navrhnout) určité komponenty, jako například funkce zodpovědné za mutace a křížení, selekční mechanismus nebo tvorbu inicializační populace. Každá z těchto komponent může být definována určitými parametry: pravděpodobnost mutace, velikost populace, počet turnajů během selekce apod. Hodnoty těchto parametrů silně ovlivňují průběh EA. Správně zvolit tyto parametry je časově náročný úkol. Rozlišovány jsou dva hlavní typy automatizovaného nastavení hodnot parametrů: *Řízení parametrů* (anglicky parameter control) a *Ladění parametrů* (anglicky parameter tuning). [3]

Ladění parametrů je pojem, který označuje tradiční přístup, kdy dochází k opakovanému spouštění EA a hledání nejvhodnějšího nastavení. Zkoušená nastavení se v průběhu algoritmu nemění. Výsledkem je statická množina hodnot, na které jsou nastaveny parametry. S takto nastavenými parametry je poté spuštěn algoritmus a hledá se pomocí něj vhodné řešení daného problému. Oproti tomu řízení parametrů označuje situaci, kdy dojde ke spuštění algoritmu s prvotním nastavením, které se v průběhu evolučního výpočtu mění. [3]

4.2 Ladění parametrů

V minulosti byl hojně používán standardizovaný GA (například pro řešení problémů plánování, viz kapitola 2), který byl založen na bitové reprezentaci, jednobodovém křížení, mutacích prováděných na základě bitové negace a ruletové selekce (s elitismem nebo bez). Při takovémto postupu bylo třeba zvolit pravděpodobnost mutace, křížení a velikost populace.

Tyto parametry byly obvykle nejprve nastaveny na předpokládané vhodné hodnoty, poté zpravidla docházelo k experimentování s těmito hodnotami, dokud nebylo nalezeno dostatečně kvalitní nastavení. [3]

V průběhu let bylo představeno vícero přístupů, jak určit správné hodnoty parametrů. Jedním z nich byla snaha o nalezení takového nastavení, které bude vhodné napříč větší sbírkou úloh. Jednalo se o snahy najít nejlepší obecné parametry, což v kontextu EA není možné. Z toho důvodu se začaly objevovat techniky pro vhodné nastavení parametrů šité na míru dané úloze. [3]

Jak již bylo zmíněno, ruční ladění parametrů (postupné experimentování s nastavením a hledání vhodných hodnot) je v oblasti EA běžnou praxí. Typicky se hledá vhodná hodnota pro parametry postupně (tedy nejprve se zkoumá první parametr, poté druhý, přičemž ty ostatní zůstávají stejné). Takovýto postup zpravidla nenalezne optimální řešení, protože spolu tyto parametry interagují a nejsou na sobě nezávislé. Souběžné hledání vhodných parametrů vede k obrovskému množství experimentů. [3]

Ladění parametrů má své nevýhody. Parametry nejsou nezávislé a vyzkoušet všechny možné kombinace je prakticky nemožné. Ladění parametrů je časově náročný proces i v případě, že jsou laděny po jednom. Dále platí, že hodnoty parametrů nejsou dokonalé, nehledě na čas a úsilí věnované jejich ladění. [3]

Existují automatizované nástroje, které určují hodnoty parametrů na míru dané úloze. Jejich použití nevyžaduje znalosti z oblasti ladění hodnot parametrů. Zmíněné nástroje pracují na základě různých metod. Základním a málo efektivním přístupem je zkoumání všech možných kombinací hodnot parametrů, přičemž pro každý parametr je předem určena konečná množina možností. Trochu více efektivní je zkoumat náhodně vybrané nastavení. Existují i pokročilejší metody založené na evolučních algoritmech nebo Bayesovské optimalizaci, které jsou využívány například nástroji jako *RayTune* nebo *Optuna*. [17]

Mimo ladění parametrů existují také další metody pro určování statických parametrů, založené na teoretické analýze daného problému či na volbě nastavení podle toho, jak se toto nastavení chovalo v podobných úlohách. Není nicméně jasné, jestli podobnost úloh nutně znamená, že vhodné parametry budou také podobné. Co se teoretické analýzy týče, tak ta je možná pouze po výrazném zjednodušení EA, tudíž možnosti jejího praktického využití zůstávají rovněž nejasné. [3]

Další potenciální nevýhodou ladění parametrů, bez ohledu na to, jak přesně probíhá, je to, že nepracuje adaptivně (na rozdíl od samotných evolučních algoritmů). Pevně stanovené nastavení, co se za běhu nemění, bude mít pravděpodobně rozdílnou účinnost v jednotlivých fázích evolučního výpočtu. Je intuitivní, že rozdílné hodnoty parametrů jsou účinné v různých fázích evoluce. Například velká pravděpodobnost mutace může být vhodná v počátcích evoluce, kdy je užitečné, pokud dochází k velkému prohledávání prostoru. Malá pravděpodobnost mutace v pozdějších fázích evoluce může přispět k tomu, že bude nakonec nalezeno řešení, které je (lokálním) optimem. [3]

Hledání vhodného nastavení je složitý problém. Nicméně je to takový typ problému, na který se evoluční algoritmy hodí. Je tedy poměrně přirozené použít EA nejenom na řešení problému, ale i k hledání vhodných hodnot parametrů pro používaný algoritmus za jeho vlastního běhu. [3]

4.3 Metody řízení parametrů

Mějme numerický optimalizační problém, který lze řešit pomocí ES, tedy vektoru reálných čísel. Každé kandidátní řešení je reprezentováno jako:

$$\vec{x} = \langle x_1, x_2 \dots x_n \rangle .$$

Dále mějme definován operátor mutace, který přičítá (odečítá) hodnoty na základě normálního rozdělení ke každému prvku vektoru, viz následující výraz:

$$x'_i = x_i + N(0, \sigma).$$

Střední hodnota je nastavena na 0 (standardní hodnota) a parametr σ určuje krok mutace. Nejjednodušší možná implementace tohoto algoritmu je pomocí stejného kroku mutace

pro všechny jedince v populaci, pro všechny proměnné každého z vektorů a po dobu celé evoluce (tedy bez řízení parametrů). Jak již bylo zmíněno v sekci 4.2, krok mutace bude pravděpodobně vhodné měnit již za běhu. [3].

Poměrně přímočarým postupem k takovému řízení mutace může být postupně snižovat pravděpodobnost mutace (či množství genů, na kterých mutace proběhne) na základě počtu generací. Nevýhodou takového postupu je to, že pracuje na základě generací (alternativně času) a nezohledňuje postup evoluce. Různé běhy algoritmu tak mohou vést k rozdílným výsledkům této taktiky. I tak se ale tento přístup ukázal jako užitečný. [3]

Existuje vícero možností, jak k řízení parametrů přistupovat. Metody řízení parametrů je možné klasifikovat do tří různých tříd (viz obrázek 4.1). Při klasifikaci je nutné brát v potaz několik aspektů [3]:

- Co se mění? Například pravděpodobnost mutace, selekční proces.
- Jak dochází k takové změně? Deterministicky/na základě zpětné vazby/podle vlastní adaptace.
- Rozsah změn (napříč celou populací, u jednoho jedince).
- Podle čeho dochází ke změnám? Například podle účinnosti operátorů.

Deterministické řízení parametrů

O tuto třídu se jedná pokud je hodnota parametru upravována na základě nějakého deterministického pravidla, například podle času nebo počtu generací. Nedochází ke zohledňování průběhu evoluce, zpětná vazba EA neexistuje. [3]

Pokud by měl být tento přístup aplikován na příklad ze sekce 4.3, mohl by být například parametr σ nahrazen dynamickým parametrem (funkcí) $\sigma(t)$. Tato funkce by byla závislá na počtu generací (alternativně době výpočtu). Krok mutace by pak mohl být určován následovně:

$$\sigma(t) = 1 - 0,9 \frac{t}{T},$$

kde t je číslo generace, přičemž je povoleno pouze T generací. V tomto případě je krok mutace $\sigma(t)$ použit pro všechny jedince a pro všechny proměnné ve vektoru. Tento krok mutace se v průběhu evoluce bude pomalu snižovat z hodnoty 1 na začátku po hodnotu 0,1 ke konci výpočtu. Jak již bylo zmíněno dříve v této sekci, parametr je řízen deterministicky a nedochází k zohlednění průběhu evoluce. [3]

Adaptivní řízení parametrů

Adaptivní řízení parametrů pracuje s určitou zpětnou vazbou od prohledávání. Tyto informace se využijí pro určení směru a síly změny hodnoty nějakého parametru (parametrů). Toto nastavení přetrvává, pokud se standardní populace EA chová podle očekávání. V opačném případě dojde opět k úpravě. [3]

Pokud by měl být aplikován tento přístup na příklad ze sekce 4.3, mohlo by být použito například Rechenbergovo pravidlo „1 z 5 úspěšných“, které se používá v ES založených na principu 1+1 (jeden rodič, jeden potomek) [13]. Jak název napovídá, toto pravidlo určuje, že pravděpodobnost úspěšné mutace by měla být 20 %. Pokud je úspěšnost vyšší, dojde ke zvýšení kroku mutace, pokud je naopak nižší, je krok mutace snížen. V tomto případě je vliv uživatele na hodnoty parametrů mnohem méně přímý, než tomu bylo v deterministické variantě. [3]

Samo-adaptivní řízení parametrů

Samo-adaptivní přístup (anglicky „self-adaptive“) popisuje princip vyvíjení parametrů souběžně s kandidátními řešeními. Nastavení je zakódováno do chromozomů jedinců a je upravováno genetickými operátory, jako třeba mutace nebo křížení. Lepší hodnoty parametrů vedou k lepším jedincům s vyšší fitness, u kterých je pravděpodobnější, že přežijí a budou se dále reprodukovat, přičemž při této reprodukci dále předají vhodné hodnoty těchto parametrů. [3]

Další varianta, jak upravit algoritmus v sekci 4.3, je tedy využít zmíněného samo-adaptivního přístupu a přiřadit individuální krok mutace pro každé kandidátní řešení zvlášť. To může být implementováno například pomocí rozšíření reprezentace jedince následujícím způsobem:

$$\langle x_1, x_2 \dots x_n, \sigma \rangle,$$

kde dochází k aplikování genetických operátorů nejen na reálná čísla x_i , ale i na parametr σ . Mutace pak mohou probíhat například následujícím způsobem:

$$\begin{aligned}\sigma' &= \sigma e^{N(0, \tau_0)}, \\ x'_i &= x_i + N(0, \sigma'),\end{aligned}$$

kde τ_0 je parametr metody volený uživatelem. Tento mechanismus spadá do třídy samo-adaptivních algoritmů. Při tomto přístupu odpadá nutnost definovat heuristickou funkci, která by měnila sílu mutace na základě zpětné vazby. [3]

V předchozím přístupu je hodnota σ individuální pro každého jedince v populaci. Toto může být rozšířeno pro všechny proměnné napříč jedinci. Níže uvedené výrazy popisují, jak by byla taková kandidátní řešení reprezentována, včetně výrazů pro jejich změnu na základě genetických operátorů [3]:

$$\begin{aligned}\vec{x} &= \langle x_1, x_2 \dots x_n, \sigma_1, \sigma_2, \dots \sigma_n \rangle \\ \sigma'_i &= \sigma_i e^{N(0, \tau_0)} \\ x'_i &= x_i + N(0, \sigma'_i)\end{aligned}$$

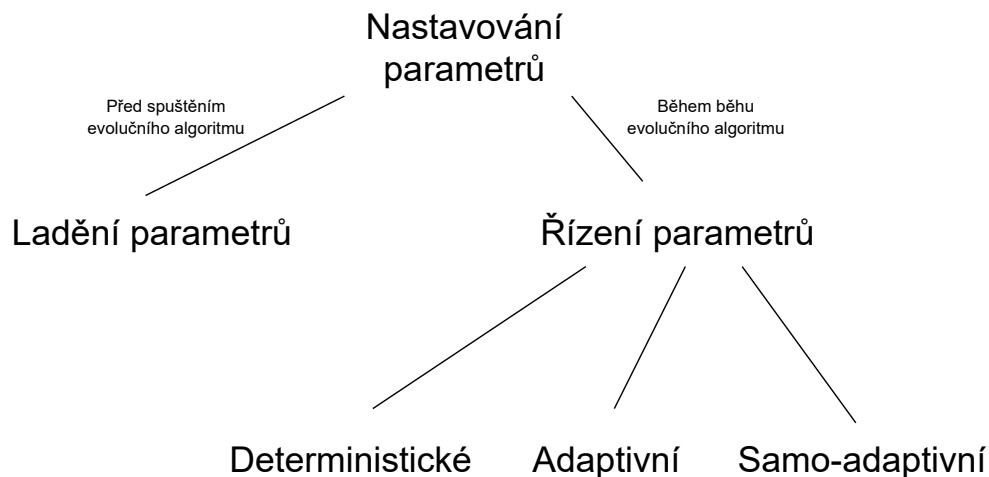
Takovýto mechanismus poskytuje velký prostor pro adaptování samotného prohledávání na řešený problém. [3]

4.4 Existující řešení řízení parametrů CGP

Řízení na základě diverzity populace

Kalkreuth a kol. [6] představili přístup, při kterém dochází k řízení parametrů u CGP takovým způsobem, aby bylo dosaženo pokud možno co nejvyšší diverzity populace. U populace s nízkou diverzitou hrozí, že bude náchylnější k uváznutí v lokálním optimu. Cílem diverzity populace je stav, kdy má co největší množství jedinců v populaci vysokou fitness a zároveň platí, že kandidátní řešení jsou co nejvíce rozdílná. Řešení počítá s křížením (nejen mutacemi) jedinců v populaci. Adaptivně je měněn zmíněný mechanismus křížení, poté mutace a k řízení dochází i u selekčního mechanismu.

Adaptivní křížení je řízeno na základě diverzity populace. Pokud je rozdílnost jedinců v populaci vysoká, dochází k zvýšení pravděpodobnosti křížení, ve snaze posunout populaci



Obrázek 4.1: Klasifikace nastavování parametrů, převzato z [3], upraveno.

blíže globálnímu optimu. Pokud je naopak populace málo diverzifikovaná, mohla by se populace při vysoké pravděpodobnosti křížení dostat do lokálního extrému. V takovém případě dochází k snížení pravděpodobnosti křížení, přičemž k zvýšení diverzity přispěje operátor mutace. Pravděpodobnost křížení je shora i zdola omezena. [6]

Oproti operátoru křížení je operátor mutace adaptován lokálně (viz dále). Mutace mají velký efekt na fenotyp v CGP, tudíž tento přístup nedoporučuje měnit pravděpodobnost mutace napříč celou populací, protože by slibná řešení mohla ztratit svoji kvalitu. Při málo agresivních mutacích dochází k příliš vysoké homogenitě populace. Mutace je zvyšována u těch jedinců, kteří mají malou fitness nebo jsou příliš podobní nějakému jinému kandidátnímu řešení. Jak již bylo zmíněno, každý jedinec má vlastní parametr mutace, který je určen na základě diverzity celé populace, nejedná se tedy o samo-adaptivní přístup. I zde existuje maximální a minimální pravděpodobnost mutace. [6]

Selekční tlak je zvyšován při vysoce diverzifikované populaci, naopak při homogenitě populace je tento tlak snížen. Typický přístup přežití nejsilnějších (u CGP $(\mu + \lambda)$) je neefektivní při vysoké homogenitě populace. Při zvolení kandidátních řešení s nižší fitness a s vyšší rozdílností dochází k nárůstu ukazatele diverzity. K úpravě selekčního tlaku dochází na základě velikosti turnajů, přičemž turnajový výběr je používán v průběhu celé evoluce. [6]

Řízení mutací funkcí podle úspěšnosti

Další přístup, který představili Manfrini a kol. [8], a který se prokázal jako účinný, spočívá v tom, že mutace na genech funkcí neprobíhají rovnoměrně. Pokud dojde při používání základního přístupu CGP k mutaci na genu, který udává funkci v nějakém uzlu, je nová funkce vybrána z množiny funkcí, přičemž každá funkce má stejnou pravděpodobnost výběru. [8]

V tomto alternativním přístupu dochází k zaznamenávání úspěšnosti mutací na funkčních blocích. Pro každý typ funkce (například log. AND) je vytvořena funkce rozdělení pravděpodobnosti. Pokud dojde k zvýšení fitness nějakého kandidátního řešení při změně funkce AND na XOR, je tato informace zaznamenána a při další mutaci funkce AND bude pravděpodobnost výběru XOR vyšší. [8]

Kapitola 5

Návrh metody pro automatické řízení parametrů kartézského genetického programování

Jak již bylo zmíněno v kapitole 2, u evolučních algoritmů je důležité zvolit vhodné nastavení. Různé problémy a úlohy vyžadují zpravidla velmi odlišné hodnoty parametrů. Tento problém platí i pro CGP, jehož řízení parametrů je tématem této práce. Mezi parametry, které je třeba vzít v potaz při implementaci CGP patří například:

- Počet řádků n_r .
- Počet sloupců n_c .
- Parametr l (*lback*).
- Počet jedinců v populaci.
- Selekcční mechanismus.
 - Typ selekcčního mechanismu (například proporční, turnajový, $(\mu + \lambda)$).
 - Síla selekcčního tlaku (například počet turnajů).
 - Počet rodičů a potomků.
- Mechanismus mutace.
- Případná přítomnost pro CGP neobvyklého křížení.

Tato kapitola obsahuje návrh metody řízení parametrů pro zmíněné CGP. Navrhovaná metoda obsahuje řízení následujícího nastavení: **mechanismu selekce**, **síly mutace** a **parametrů mřížky**. Selektce a mutace budou řízeny, protože agresivnější mutace a nižší selekcční tlak jsou zpravidla výhodnější na začátku evoluce, zatímco menší mutace a vyšší selekcční tlak je vhodné použít ke konci evolučního výpočtu (viz například kapitola 4). Řízení parametrů mřížky může vést k automatickému určení vhodného prohledávacího prostoru. Cílem navrhované metody je, aby v tomto automaticky určeném stavovém prostoru nejprve pomocí silných mutací a nízkého selekcčního tlaku došlo k prohledávání globálního charakteru, a poté k prohledávání lokálnějšího rázu naopak za využití vyššího selekcčního tlaku a méně agresivních mutací.

Pro tato nastavení (selekce, mutace, mřížka) budou vyzkoušeny různé přístupy k řízení (například adaptivní a samo-adaptivní), které budou porovnány s referenčním přístupem, kdy dochází k postupnému ladění hodnot parametrů samostatně. Nejlepší z těchto přístupů se propíše do výsledné metody. Pokud by se ukázalo, že si adaptivní parametry vzájemně nejen že nepomáhají, ale dokonce i škodí, bude výsledná metoda obsahovat řízení pouze některých parametrů. V případě mutací jsou představeny dva algoritmy, u selekce a mřížky se jedná o jeden přístup.

V algoritmech pro řízení zmíněných parametrů bude nezbytné stanovit určité statické nastavení, které bude například určovat, jak moc má být selekční tlak modifikován, když má k jeho změně dojít. Jakým způsobem budou hodnoty těchto parametrů voleny popisuje sekce 5.4.

5.1 Řízení selekčního mechanismu

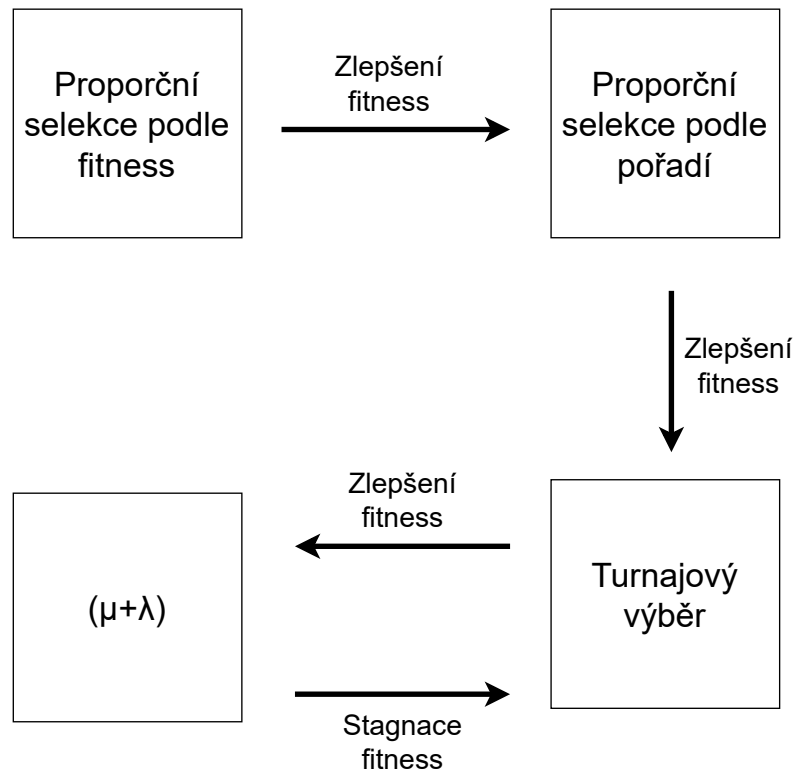
Selekční algoritmus a míra selekčního tlaku má (stejně jako mutace) velkou roli na „explore vs exploit“, tedy do jaké míry dochází k prohledávání globálního prostoru a do jaké míry evoluční výpočet konverguje do nejbližšího lokálního maxima (viz sekce 2.5 a 4.2). Jak již bylo popsáno ve zmíněné sekci 2.5, měnit sílu selekčního tlaku v průběhu evoluce se může projevit jako účinné.

Adaptivní selekce

V navrhované metodě bude docházet ke změně selekčního mechanismu nebo hodnot parametrů těchto algoritmů takovým způsobem, aby selekční tlak reflektoval průběh evoluce. Tedy aby na začátku došlo k prohledávání napříč stavovým prostorem (mírný selekční tlak) a na konci k pohybu směrem k (lokálnímu) maximu (vysoký selekční tlak).

Vzhledem k tomu, že EA nepracují deterministicky, nebude tolik efektivní selekční mechanismus měnit pouze na základě generací, ale vyplatí se zohlednit změny fitness, viz sekce 4.3. Selektce bude v průběhu evoluce měněna následovně: bude obsahovat proměnnou *pressure*, která udává sílu selekčního tlaku a která se mění podle toho, jestli se fitness zlepšuje, nebo stagnuje. Na základě této hodnoty pak bude vybrán příslušný selekční algoritmus (například turnajový výběr, nebo proporční selekce podle fitness). Mimo to budou i touto proměnnou určovány parametry zmíněných selekčních algoritmů, jako třeba velikost turnaje (viz sekce 2.5).

Na začátku bude selekce pracovat proporčně podle fitness, přičemž nebude docházet k žádnému škálování (tento algoritmus má nejnižší selekční tlak, viz sekce 2.5). Jakmile dojde ke změně nejlepší dosažené fitness, zvýší se *pressure* o hodnotu parametru *mod* (tento parametr bude určen na základě experimentů). Jakmile dojde k dostatečnému navýšení hodnoty proměnné *pressure*, změní se selekční mechanismus na proporční podle pořadí. Při tomto selekčním mechanismu bude konstanta η^+ postupně zvyšována od 1,1 do 2,0, kdy selekční tlak odpovídá binárnímu turnajovému výběru [1]. Dále bude docházet k postupnému navýšování velikosti turnajů a nakonec dojde k přepnutí na $(\mu + \lambda)$. V případě detekce uváznutí v lokálním optimu dojde naopak ke zmírnění selekčního tlaku (snížením proměnné *pressure*), přičemž tento selekční tlak se sníží maximálně na binární turnajový výběr, viz obrázek 5.1 a algoritmus 3. Za uváznutí v lokálním optimu bude označen stav, kdy se nejlepší fitness nezměnila po dobu G generací (tento parametr bude opět experimentálně určen). Zmírnění selekčního tlaku v tomto případě může přispět k úniku z lokálního extrému a k nalezení lepšího kandidátního řešení. Selektční tlak pak následně může být opět



Obrázek 5.1: Adaptivní řízení selekčního mechanismu. Na začátku jsou jedinci vyvíjeni pomocí proporční selekce založené na fitness. Po několika úspěšných vylepšení této hodnoty začne selekce pracovat proporčně podle pořadí, kde dochází k postupnému zvyšování selekčního tlaku na základě parametru η^+ (pokud se fitness zvyšuje). Jakmile tento parametr dosáhne maximální hodnoty, selekce začne pracovat na základě turnajového výběru. S dalším vylepšením fitness se zvyšuje počet turnajů a nakonec dojde k přepnutí do algoritmu $(\mu + \lambda)$. Při stagnaci fitness ve fázi $(\mu + \lambda)$ může dojít k snížení selekčního tlaku a zpětnému nasazení turnajového výběru. Při stagnaci u turnajové selekce dochází ke snižování počtu turnajů, avšak do stádia proporční selekce se evoluční algoritmus už zpět nedostane.

zvýšen v případě, že byla překonána fitness nejlepšího jedince z populace, u které byla stagnace detekována. Pokud by jako práh pro další zvyšování selekčního tlaku byla nastavena nejlepší dosažená fitness za celou dobu běhu evolučního výpočtu, mohl by selekční tlak klesnout na minimum na nepřiměřeně dlouhou dobu. Po stagnaci fitness z důvodu nalezení lokálního optima by totiž nalezení lepšího jedince zpravidla chvíli trvalo, přičemž by během tohoto čekání došlo k rapidnímu poklesu hodnoty proměnné *pressure* až na nejnížší možnou hodnotu.

5.2 Řízení mutace

Změna mutací za běhu programu bude probíhat na podobných principech, jako v příkladech v sekci 4.3. Postupně budou experimentálně ověřeny dva přístupy: adaptivní a samoadaptivní. Deterministický přístup, kdy dochází ke snižování velikostí mutací s přibývajícimi generacemi (či alternativně s časem) využit nebude, protože nezohledňuje průběh evoluce (viz sekce 4.3).

Algorithm 3: Adaptivní řízení selekce CGP

Input: mod, G – parametry určující rychlost změny selekce
pressure = 0; // určuje selekční tlak podle zlepšování/stagnace fitness
while *není splněna podmínka pro ukončení EA* **do**
 if *Nejlepší fitness se zlepšila* **then**
 | pressure += mod; // pressure $\in < 0, 30 >$
 if *Fitness G generací stagnuje* **and** *pressure > 20* **then**
 | pressure -= mod; // klesne maximálně na hodnotu 20
 if *pressure $\in < 0, 10 >$* **then**
 | Vyber 5 rodičů proporčně podle fitness;
 | Vytvoř pomocí mutace jednoho potomka z každého rodiče;
 | Vyber náhodně 5 přeživších;
 if *pressure $\in < 10, 19 >$* **then**
 | etaplus $\in < 1, 1; 2, 0 >$; // Roste s proměnnou *pressure*
 | Vyber 5 rodičů proporčně podle pořadí a proměnné etaplus;
 | Vytvoř pomocí mutace jednoho potomka z každého rodiče;
 | Vyber náhodně 5 přeživších;
 if *pressure $\in < 20, 29 >$* **then**
 | $T \in < 2, 5 >$; // Roste s proměnnou *pressure*
 | Vyber 5 rodičů na základě turnajů o velikosti T ;
 | Vytvoř pomocí mutace jednoho potomka z každého rodiče;
 | Vyber náhodně 5 přeživších;
 if *pressure == 30* **then**
 | Vyber nejlepšího jedince jako rodiče, upřednostni potomka před předkem;
 | Zmutuj rodiče pro zisk 4 potomků;
 | V populaci nech pouze rodiče a jeho potomky;

Přístup 1: Adaptivní řízení mutací

Tento postup bude pracovat se zpětnou vazbou evolučního algoritmu. Pokud bude CGP spuštěno s adaptivní mutací, dojde k počátečnímu nastavení pravděpodobnosti mutace na 80 %, přičemž program sníží tuto hodnotu pokaždé, když dojde ke zlepšení průměrné fitness v populaci, a to následujícím způsobem:

$$p_{i+1} = p_i - m * p_i, \quad (5.1)$$

kde p_i je současná pravděpodobnost mutace a m je parametr změny fitness. Hodnota p_{i+1} je shora i zdola omezena, musí spadat do intervalu $< 0,03; 0,8 >$. Naopak při detekci stagnace (tedy pokud uběhne G generací bez zlepšení průměrné fitness) bude mutace zvýšena za účelem usnadnění úniku z lokálního maxima, viz následující rovnice:

$$p_{i+1} = p_i + m * p_i * 5, \quad (5.2)$$

kde p_i je opět současná pravděpodobnost mutace a m je parametrem změny fitness. U hodnoty p_{i+1} platí stejná omezení jako v rovnici 5.1.

Při takovémto zvýšení pravděpodobnosti fitness nebude třeba pro její následovné snížení překonat nejlepší dosaženou průměrnou fitness po celou dobu běhu programu, ale průměrnou fitness v populaci, ve které byla stagnace detekována. Kdyby došlo k nalezení lokálního optima a všichni jedinci v populaci by byli zmíněným lokálním optimem, mohla by se při delší stagnaci fitness agresivita mutací postupně zvyšovat až na maximální hodnotu 80 %, bez možnosti snížení (nalezení lepšího průměru fitness by bylo velmi nepravděpodobné). V konečném důsledku by tedy docházelo ke generování téměř náhodných jedinců.

V případě elitářské selekce (kdy nejlepší jedinec vždy postupuje do další generace, například $(\mu + \lambda)$) může tento přístup postupným snižováním agresivity mutace přispět k tomu, že bude na počáteční prohledávání globálního charakteru navázáno lokálním prohledáváním. Pokud ale dojde ke stagnaci a uváznutí v lokálním optimu, bude docházet ke změnám fitness na základě toho, jakou fitness mají například čtyři potomci vytváření z neustále stejného (stejného fenotypem) předka. Což vyústí v mírnou oscilaci agresivity fitness, která sice pravděpodobně příliš neškodí, ale nejspíš nepřispěje k zvýšení pravděpodobnosti úniku z lokálního optima. Toto adaptivní řízení mechanismu mutace je tedy navrženo tak, aby spolupracovalo s adaptivní selekcí, viz sekce 5.1.

Obě rovnice 5.1 a 5.2 byly navrženy tak, aby docházelo k vhodným změnám v závislosti na aktuální pravděpodobnosti mutace. Při snižování pravděpodobnosti mutací bude nejspíš užitečné, aby docházelo k velkým změnám při agresivní mutaci a malým změnám při malé pravděpodobnosti mutace. Není žádoucí, aby se například 80 % mutace pomalu po jednom procentu snižovaly, stejně tak nejspíš nebude vhodné, aby se jednou změnou z 20 % pravděpodobnosti mutace stala 3 %. Rovněž při zvýšení agresivity mutace z 3 % na 4 % by nejspíš nedošlo k tíženému efektu a navyšování musí být výraznější. Parametry G a m budou nastaveny na základě experimentů.

Přístup 2: Samo-adaptivní řízení mutací

Obdobně jako u ES (sekce 4.3), může i CGP obsahovat gen, který bude udávat pravděpodobnost mutace. Poslední gen v genotypu CGP by tak namísto výstupu byl gen mutace. Ten by dosahoval hodnot z intervalu $< 0,03; 0,8 >$. Tato hodnota by pak udávala, kolik % genů se při mutaci daného kandidátního řešení změní. Například při hodnotě 0,5 by došlo

ke změně poloviny genů (nebo by každý gen měl 50% pravděpodobnost mutace). Kandidátní řešení s vhodněji nastavenou mutací budou pravděpodobněji přežívat a reprodukovat se, než ti jedinci, jejichž velikost mutace je v dané fázi evoluce méně účinná.

Nabízí se otázka, jakým způsobem bude mutován samotný gen mutace. Jednou z možností je měnit tento gen rozdílně podle aktivního selekčního mechanismu. Při mírném nebo naopak vysokém selekčním tlaku (proporční výběr podle fitness, strategie $(\mu + \lambda)$ a možná i proporční výběr podle pořadí) by mohlo být žádoucí, aby nedocházelo k příliš velkým změnám síly mutace. Na začátku by tak byly mutace velmi agresivní (aby se určila hrubá struktura vhodného kandidátního řešení) a ke konci mírné (lokální optimalizace), což by platilo napříč populací. Tento přístup však není plně samo-adaptivní, ale částečně splňuje definici adaptivního řízení parametrů, které již bude ověřováno jiným přístupem.

Vhodnějším přístupem změny této hodnoty z oboru reálných čísel by mohla být aplikace podobného přístupu, jako u příkladu řízení tradičních ES (viz sekce 4.3), kdy je daný gen měněn hodnotou z normálního rozdělení:

$$m_{i+1} = m_i + N(0, \sigma).$$

Při dosažení hodnoty z intervalu $(-\infty; 0,03 >$ by došlo ke změně mutace na 0,03, při přetečení (přesáhnutí hodnoty 0,8) by tento parametr byl snížen zpět na maximální hodnotu. Hodnota σ bude určena na základě experimentů (viz kapitola 5.4).

5.3 Řízení parametrů mřížky CGP

Při používání CGP je potřeba nastavit (mimo jiné) parametry mřížky, tedy n_c , n_r a *lback*, které udávají prohledávací prostor. Předem není jisté, jaké nastavení se ukáže jako nejvíce efektivní. Mějme například problém symbolické regrese, kde se v naměřených datech nachází padesát vstupních proměnných a jedna výstupní proměnná.

Pokud výstupní proměnnou ovlivňuje pouze několik málo vstupních proměnných (a to ještě na základě jednoduchých vztahů a matematických operací), mohou malé parametry mřížky vést k tomu, že bude rychleji a snadněji nalezeno řešení. Naopak, pokud jsou skutečné vztahy takového systému komplexní, nemusí nízké hodnoty parametrů n_c , n_r a *lback* stačit pro vyjádření dostatečně přesné rovnice.

Oproti tomu parametry mřížky nastavené na velkou hodnotu budou pravděpodobně poskytovat dostatečně velký stavový prostor pro prohledávání, nicméně výpočet může trvat zbytečně dlouho. Dále mohou takovéto hodnoty způsobit to, že se v populaci budou nacházet jedinci popisování nadbytečnými složitými vztahy.

Měnit parametry n_c , n_r a *lback* za běhu programu může být problém. Rozšiřování počtu řádků, sloupců nebo parametru *lback* by nemuselo být nijak problematické, jejich snižování ale ano. Při zvýšení parametru n_c o hodnotu 1 (implementovaného například přidáním nového sloupce za poslední sloupec) by nedošlo k přímé změně fitness kandidátních řešení, protože jejich výstup by se bral z předešlých uzlů. Rovněž změna počtu řádků či volnost volby vstupů z předešlých sloupců by nezpůsobila znehodnocení již vyvinutých jedinců.

To ale neplatí při snižování těchto hodnot. Pokud by byl výstup CGP brán z uzlu, který se nachází v posledním sloupci a během řízení parametrů by došlo ke snížení počtu sloupců, takový jedinec by poté nejspíš měl úplně jinou fitness. Odebírání uzlů je možné například za pomoci kontroly aktivních uzlů, kdy může být odebrán například sloupec, který není žádným kandidátním řešením využíván (pokud by vůbec nějaký takový k dispozici byl). Nicméně by i v tomto případě mohlo dojít ke snížení kvality jedinců, protože v neaktivních uzlech mohou probíhat užitečné neutrální mutace.

K takovýmto výrazným nebo nevýrazným změnám na fenotypu každopádně běžně dochází během žádoucích mutací, tudíž by manipulace s mřížkou mohla být nástavbou k operátoru mutace. Na začátku evoluce by výraznější změny mřížky a s tím i změny fenotypů (včetně jejich fitness) mohly být účinné, protože by se přibližně určil vhodný prohledávací prostor a zároveň by částečně došlo k prohledávání globálnějšího rázu.

Samo-adaptivní řízení parametrů mřížky

V této metodě bude mít každý jedinec vlastní parametry n_r , n_c a $lback$, které budou zakódované do genomu. Tyto parametry budou mít horní i spodní omezení. Parametr $lback$ bude z intervalu $< 1, n_c >$, hodnoty parametrů n_c a n_r budou náležet intervalu $< 3, 10 >$. Měnit hodnoty parametrů mřížky bude možné měnit pouze na začátku evoluce, protože by ke konci prohledávání mohly mít negativní vliv na kvalitu kandidátních řešení (z důvodů zmíněných dříve v této sekci). Na začátku evolučního výpočtu se však mohou projevit jako užitečné, protože mohou přibližně určit vhodný prohledávací prostor. Jedinci představující lepší stavový prostor pro danou úlohu budou mít větší pravděpodobnost přežití z důvodu selekčního tlaku na základě fitness, než ta kandidátní řešení, která mají nevhodné parametry n_r , n_c a $lback$.

Na začátku evoluce dojde k vytvoření počáteční populace, ve které budou mít jedinci rozdílné parametry mřížky, které budou přímo zakódované v genomu. S každou mutací jedince dojde na P % ke změně hodnot parametrů mřížky následujícím způsobem:

$$rsl = rsl \pm 1, \quad (5.3)$$

kde rls je jeden z parametrů $n_c, n_r, lback$. Jakmile dojde ke stagnaci fitness (nezlepší se po dobu G generací), jsou změny mřížky u konce. Hodnoty parametrů G a P budou stanoveny na základě experimentů, viz kapitola 5.

5.4 Návrh experimentů

Metoda, která byla popsána v této kapitole, bude porovnána na základě experimentů s referenčním přístupem (kdy dochází k postupnému ladění statických parametrů). Experimenty budou probíhat na úlohách z volně dostupných sbírek úloh symbolické regrese. Fitness funkcí bude pro oba případy průměrná absolutní odchylka (MAE):

$$MAE = \frac{1}{n} \sum_{i=1}^n (Y - \hat{Y}), \quad (5.4)$$

kde n je počet trénovacích vektorů, Y je očekávaná výstupní hodnota a $\hat{Y}$ je predikovaná výstupní hodnota.

Srovnávána bude kvalita výsledného řešení (nejlepší nalezený jedinec) při stanoveném časovém limitu. Zkoumána bude pouze zmíněná průměrná absolutní odchylka, bez ohledu na počet aktivních uzlů, výpočetní náročnost funkcí apod.

Statické parametry adaptivní metody

U řízených parametrů je nicméně stále nutné předem stanovit určité statické parametry, které mohou být pro různé úlohy rozdílně vhodné. Mezi takové parametry patří například m a G popisované v sekci 5.2. Vzhledem k tomu, že cílem navrhované metody je vyhnout se

nevýhodám ladění popsaných v sekci 4.2, proběhne ladění těchto parametrů na jiných úlohách, než na kterých pak bude tato metoda ověřována porovnáním se základním přístupem (pokud se řízení prokáže jako efektivní). Tento přístup, kdy dochází k ladění parametrů na základě sbírky úloh a jejich následné aplikaci na jiné problémy, není u statických parametrů vhodný, viz sekce 4.2.

U dynamických parametrů se ale tento postup může projevit jako použitelný. Už v minulosti se totiž prokázal jako užitečný, viz například v sekci 4.3 zmiňované Rechenbergovo pravidlo „1 z 5 úspěšných“, kdy je napříč úlohami v oblasti ES určena tato hodnota [13]. V rámci experimentů tak bude zkoumáno, jak moc jsou statické parametry u navrhované metody řízení parametrů přenositelné.

Kapitola 6

Implementace

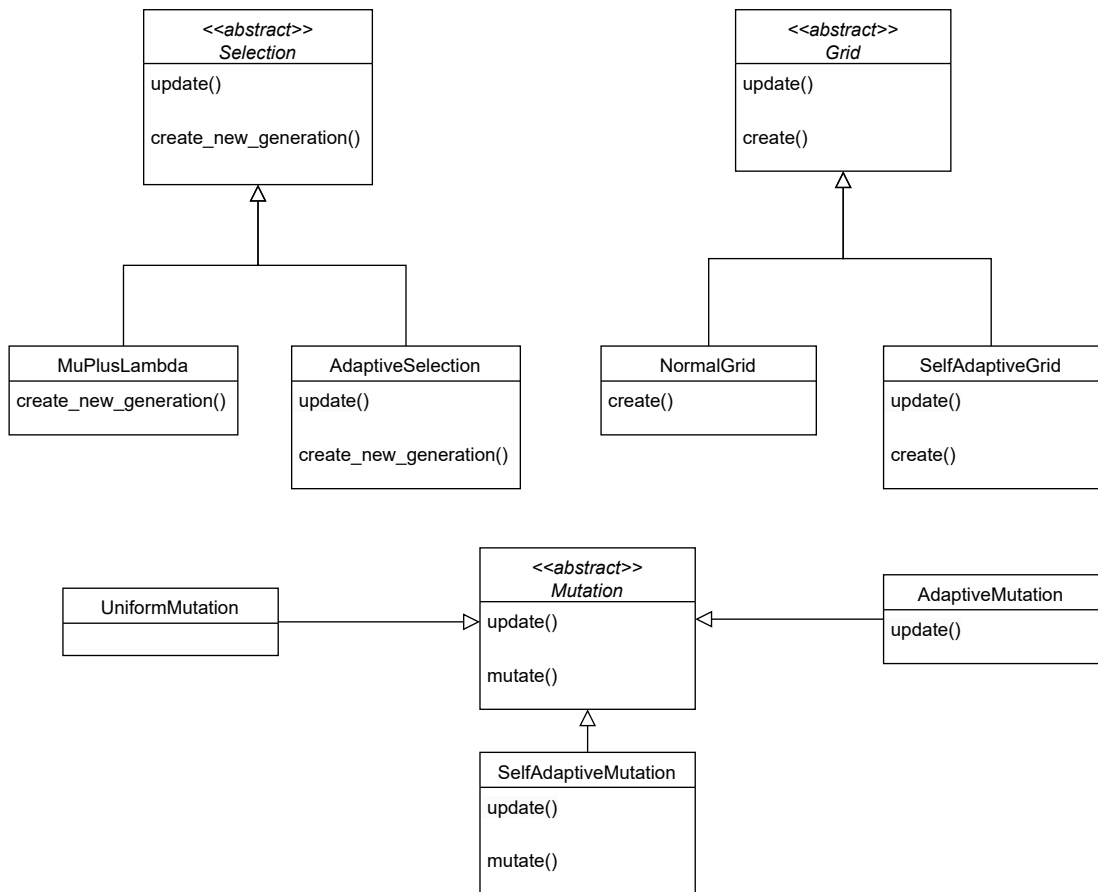
Implementace se skládá z několika Python skriptů. Hlavní komponentou je třída **CGP**, která se nachází ve stejnojmenném skriptu. Tato třída se používá k řešení úloh symbolické regrese, přičemž dochází k zaznamenávání statistických údajů jako například doba běhu programu nebo počet evaluací fitness. Při vytváření instance této třídy je potřeba specifikovat následující parametry konstruktoru: maximální povolený počet generací, vstupní soubor ve formátu csv (kde poslední sloupec je výstup a ostatní atributy jsou vstupy), maximální dobu výpočtu v procesorovém čase. Dále je třeba stanovit průměrnou absolutní odchylku, která stačí k tomu, aby jedinec byl označen za dostatečně kvalitní kandidátní řešení a nakonec mechanismy selekce, mutace a mřížky. Pro selekci, mutaci a mechanismus mřížky platí, že tyto komponenty musí být instancí potomků abstraktních tříd. Například parametr konstruktoru `mutation_mechanism`: `Mutation` neočekává, že bude předána přímo třída `Mutation`, ale nějaký její potomek, například `UniformMutation` nebo `AdaptiveMutation`. Na stejném principu jsou založeny parametry `grid_mechanism` a `selection_mechanism`, viz schéma na obrázku 6.1.

Po vytvoření instance třídy **CGP** se nejlepší nalezené řešení nastaví na hodnotu `None` a nejlepší dosažená fitness na `math.inf`, což při minimalizaci fitness značí nejhorší možnou hodnotu. Dále jsou načteny trénovací vektory ze specifikovaného vstupního csv souboru (kde poslední atribut značí výstup, jak již bylo zmíněno v této sekci). Evoluční výpočet je možné spustit pomocí metody této třídy s názvem `run()`, která spustí symbolickou regresi nad dříve specifikovanými daty, viz algoritmus 4.

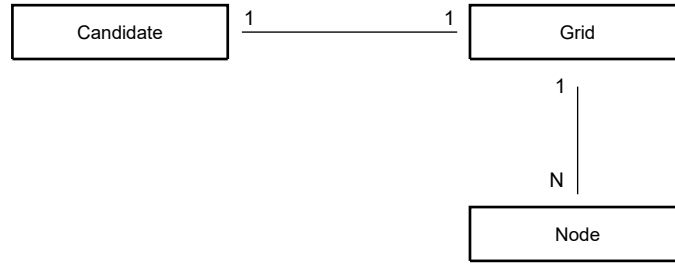
Algorithm 4: Metoda `run()` třídy **CGP**

```
self.init_population();
while not self.end_condition() do
    self.eval_fitness();
    self.next_generation();
    self.log_fitness();
return Informace o průběhu výpočtu
```

Běh programu je ukončen tehdy, pokud je nalezeno dostatečně kvalitní řešení, byl překročen maximální povolený procesorový čas nebo byl dosažen maximální počet generací. Program průběžně zaznamenává nejlepší dosaženou fitness (a odpovídajícího jedince), průběh fitness v čase (pomocí metody `log_fitness()`, viz algoritmus 4), počet vyhodnocení fitness a na konci běhu určí i dobu výpočtu. Tyto informace jsou nakonec navraceny zmíně-



Obrázek 6.1: Diagram tříd volitelně řízených komponent CGP, zjednodušeno. Konstruktor CGP vyžaduje instance potomků tří abstraktních tříd (**Selection**, **Grid**, **Mutation**). V každé abstraktní třídě je deklarována (i implementována) metoda **update**, která zodpovídá za řízení parametrů a která se pravidelně spouští během evolučního výpočtu. Tato metoda má v rodičovských třídách chování definováno tak, že nedělá nic, přičemž statické komponenty (mutace, selekce, mřížka) tuto metodu přímo dědí. Dynamické mechanismy však zmíněnou metodu přepisují vlastním chováním (například třída **AdaptiveMutation** mění pravděpodobnost mutace podle zlepšování nebo stagnace průměrné fitness v populaci). Ostatní metody na diagramu (konkrétně **create_new_generation()**, **create()** a **mutate()**) mají zpravidla vlastní chování (jedinou výjimkou je metoda **mutate()**, která je společná jak pro statickou mutaci, tak pro mutaci adaptivní).



Obrázek 6.2: Diagram vztahů mezi entitami **Candidate**, **Grid** a **Node**.

nou metodou `run()`. Nejdůležitější částí evolučního výpočtu jsou v cyklu opakovaně volané metody `eval_fitness()` a `next_generation()`, viz opět algoritmus 4.

6.1 Inicializace populace

Během vytváření počáteční populace bez řízení parametrů mřížky dojde k vytvoření pěti kandidátních řešení, přičemž každému je přiřazena mřížka o velikosti $n_r * n_c$ uzlů. Počet vstupů se určí z výstupního souboru, množství výstupů je zafixováno na hodnotu 1. Poté dojde ke kompletnímu zmutování všech uzlů (pro každý uzel jsou náhodně vybrány dva vstupy a funkce) a vybrání náhodného uzlu (nebo vstupu) jako výstupu jedince.

Uzly jsou ukládány do lineárního seznamu, přičemž každý uzel má při vytváření instance specifikovanou polohu ve mřížce (řádek a sloupec). Informace o sloupci se využijí při mutování některého ze vstupů uzlu, kdy kromě vstupů programů může být vstupem i uzel, jehož sloupec je maximálně *lback* sloupců před uzlem, který je mutován a pro který se určuje vstup. Zatímco uzly jsou instance třídy **Node**, vstupy programu jsou typu `int`. Vstupy uzlů tedy mohou nabývat vícero datových typů.

Pokud je využívána adaptivní mřížka, parametry n_r , n_c , a *lback* se určí pro každé kandidátní řešení náhodně. Počet řádků a sloupců je stanoven jako náhodné číslo z intervalu $< 3, 10 >$, parametr *lback* nabývá hodnot $< 1, n_c >$.

6.2 Vyhodnocování fitness

Pro určení fitness jedince slouží metoda `get_fitness()` třídy **Candidate**. Každý jedinec, který je instancí této třídy, obsahuje jednu mřížku (tedy instanci třídy **Grid**), která poté obsahuje vícero uzlů. Uzly v CGP jsou reprezentovány třídou **Node**. Vztahy těchto tříd zobrazuje diagram na obrázku 6.2.

Při vyhodnocování fitness se počítá průměrná absolutní odchylka napříč trénovacími vektory. Vyhodnocování probíhá takovým způsobem, jak popisuje sekce 3.1. Pokud je výstupem nějaký vstupní uzel, výstup se bere z něj. V opačném případě se nejprve rekurzivně určí aktivní uzly, přičemž je zohledňována arita funkcí (například funkce `sin` používá jen jeden ze dvou vstupů). Poté dochází k seřazení uzlů podle sloupců a následnému dopřednému vyhodnocování aktivních uzlů až po výstup programu.

Při vyhodnocování uzlů se nejprve ověří, jestli jeho hodnota už nebyla vypočítána. Pokud ano, je rovnou tento výsledek navrácen. K okamžitému navrácení hodnoty dochází i v případě, že funkčním blokem uzlu je konstanta. Uzel nejprve načte jeden, nebo dva operandy ze svých vstupů (podle toho, jaká je jeho arita). Poté dojde k vyhodnocení výrazu daného operandem (operandy) a funkcí uzlu. Tento výpočet proběhne v rámci `try-catch`

bloku, kterým jsou zachyceny chyby typu dělení nulou. Při zachycení takovéto chyby je výpočet fitness předčasně ukončen a kandidátnímu řešení se nastaví nejhorší možná fitness. K předčasnému ukončení dojde i v případě, že je hodnota některého z operandů příliš velká, nebo naopak příliš malá. Konkrétně jsou akceptována pouze čísla v intervalu $< -1e10, 1e10 >$ (velké hodnoty málokdy vedou ke kvalitnímu jedinci a operace s nimi často způsobí přetečení). Po určení hodnoty výstupního uzlu pro každý trénovací vektor se vypočte průměrná absolutní odchylka, která je uložena do jedince a i navráćena metodou `get_fitness()`.

6.3 Tvorba nové generace

Tvorba nových generací je významným způsobem určena zvoleným selekčním algoritmem ($\mu + \lambda$, nebo adaptivní selekce). Tento selekční algoritmus je třeba specifikovat při inicializaci CGP, jak již bylo zmíněno dříve v této kapitole. V metodě třídy `CGP next_generation()` nejprve dojde ke spuštění metod `update(population)` selekčního a mutačního mechanismu (viz obrázek 6.1), které v případě statických parametrů nedělají nic. U dynamických (řízených) parametrů se tyto metody starají o aktualizaci selekčního tlaku a síly mutace. Rovněž je zde zkoumáno, zda má být případně zastaveno řízení parametrů mřížky, viz sekce 5.3.

V případě samo-adaptivní mutace dojde k aktualizaci pravděpodobností mutací pro každé kandidátní řešení. Konkrétně je ke každé pravděpodobnosti přičtena hodnota z normálního rozdělení se střední hodnotou 0 a směrodatnou odchylkou danou parametrem. U adaptivní mutace může dojít ke změně pravděpodobnosti mutace (tato hodnota je stejná pro všechny jedince v populaci), pokud bylo detekováno zlepšení průměrné fitness, nebo byla zaznamenána stagnace (průměrná fitness se po dobu G generací nezlepšila), viz rovnice 5.1 a 5.2.

Obdobně jako u adaptivní mutace i u adaptivní selekce může dojít ke snížení, nebo zvýšení selekčního tlaku. Metoda `update(population)` selekčního algoritmu určuje, jaký selekční mechanismus bude následně použit. V této metodě dochází ke změně hodnoty atributu `self.pressure`. Při nalezení nové nejlepší fitness se tento atribut zvýší o hodnotu `self.mod` (parametr), přičemž nepřesáhne hodnotu 30. Pokud se fitness po dobu G generací nezlepšila, atribut `self.pressure` je naopak o stejnou hodnotu snížen, přičemž ke snižování dochází pouze v případě, že má tato proměnná hodnotu větší než 20, přičemž pod tuto hodnotu neklesne, viz algoritmus 3.

Po volitelné aktualizaci parametrů dojde ke spuštění metody `create_new_generation(population, mutation_mechanism)` selekčního algoritmu. Pokud selekce není nastavena na adaptivní, spustí se algoritmus ($\mu + \lambda$), který je definován v rodičovské abstraktní třídě `Selection` a statická selekce ji přímo dědí.

Pokud je selekce adaptivní, metoda `create_new_generation` na základě atributu třídy `self.pressure` vybere příslušný selekční algoritmus. Tím dojde ke spuštění jedné z metod `proportional_fitness`, `proportional_order`, `tournament`, nebo ke spuštění metody `create_new_generation` rodičovské třídy `Selection`, viz opět algoritmus 3. Všechny tyto zmíněné metody vycházejí ze sekce 2.5 a mají jako parametr populaci a mechanismus mutace.

6.4 Mutace

Nezávisle na selekčním algoritmu pracují mutace, které jsou využívány pro tvorbu potomků z předků (a to ve všech mechanismech selekce). Ve všech případech je využívána metoda `mutate(candidate)` abstraktní třídy `Mutation`. U neadaptivní mutace je tato metoda přímo děděna, adaptivní mutace ji rovněž využívá, přičemž průběžně aktualizuje pravděpodobnost (mutace). U samo-adaptivní mutace dojde vždy těsně před zavoláním této metody k nastavení pravděpodobnosti mutace na tu hodnotu, jaká je uložena do mutovaného kandidátního řešení. Ve zmíněné metodě `mutate(candidate)` dojde s procentuální pravděpodobností danou parametrem p ke změně libovolného genu (atributů `first`, `second`, `fn`) u každého uzlu v mřížce. Se stejnou pravděpodobností se změní i výstup. Poté dojde ke spuštění metody `update()` mřížky kandidátního řešení. Tato metoda v případě zafixovaných parametrů n_r , n_c , a `lback` nedělá nic, u samo-adaptivního řízení mřížky dojde s pravděpodobností stanovenou parametrem ke zvýšení, nebo snížení některého z parametrů mřížky. Při zvýšení n_r nebo n_c jsou nové řádky/sloupce doplněny o nově náhodně vygenerované uzly, přičemž výstup a fenotyp kandidátního řešení zůstává stejný. Při snížení některého z těchto parametrů jsou nevalidní vstupy uzlů (nebo výstup) znovu náhodně vybrány na nějakou validní hodnotu. Totéž platí i u snížení parametru `lback`, kdy může dojít ke změně některých vstupů. Při zvýšení tohoto parametru (`lback`) nedojde k okamžitým změnám na fenotypu, pouze se zvýší počet možných vstupů pro uzly u dalších mutací.

Kapitola 7

Experimentální vyhodnocení adaptivních metod

Experimenty byly provedeny nad sbírkou úloh z oblasti symbolické regrese. Tyto úlohy byly v minulosti v literatuře použity pro porovnávání variant genetického programování [15]:

$$f_1(x, y) = x^4 - x^3 + y^2/2 - y \quad (7.1)$$

$$f_2(x, y) = 6\sin(x)\cos(y) \quad (7.2)$$

$$f_3(x, y) = 8/(2 + x^2 + y^2) \quad (7.3)$$

$$f_4(x, y) = x^3/5 + y^3/2 - y - x \quad (7.4)$$

Pro každou úlohu byl vygenerován vstupní soubor formátu csv obsahující nezávislé proměnné i závislou proměnnou. Ve všech případech byly proměnné x a y voleny náhodně z intervalu $< -3, 3 >$. Tento rozsah byl zvolen na základě Keijzer M. [7].

Pro každou úlohu bylo vygenerováno 150 náhodných bodů, které byly použity pro porovnávání statických a adaptivních parametrů. Ověřování (porovnání slibného řízení parametrů s referenčním přístupem) pak probíhalo na datových sadách z **PMLB** [11]. Množina funkcí byla stanovena (napříč úlohami) na $\{a + b, a - b, a * b, a/b, \sin(a), \cos(a), e^a, \log(a), \text{const}\}$.

V rámci porovnávání došlo pro každou variantu ke spuštění 48 nezávislých běhů, a to pro každou ze čtyř úloh. Celkově se tedy jednalo o $48 * 4 = 192$ běhů na jedno nastavení (což při například šesti zkoumaných variantách během jednoho experimentu dělá $192 * 6 = 1152$ běhů). Výsledkem pak byl krabicový graf nejlepších dosažených hodnot fitness napříč čtyřmi funkcemi (jeden box byl vytvořen ze 192 běhů pro jednu variantu) a konvergenční křivky popisující vývoj těchto hodnot v čase.

Při hledání například nejlepší statické mutace (za použití stejného selekčního algoritmu a mechanismu mřížky) došlo ke spuštění zmíněných 48 běhů pro každou ze čtyř úloh, přičemž byly zkoumány například mutace s pravděpodobnostmi $\{0, 05; 0, 1; 0, 15; 0, 2; 0, 3; 0, 4\}$. Každý z těchto 1152 běhů pro celkově šest variant mutace ($192 * 6 = 1152$) trval 60 sekund procesorového času. Ve výsledku by tak jeden výpočet bez paralelizace trval více než 19 hodin (přičemž tato hodnota roste s vytížením procesoru).

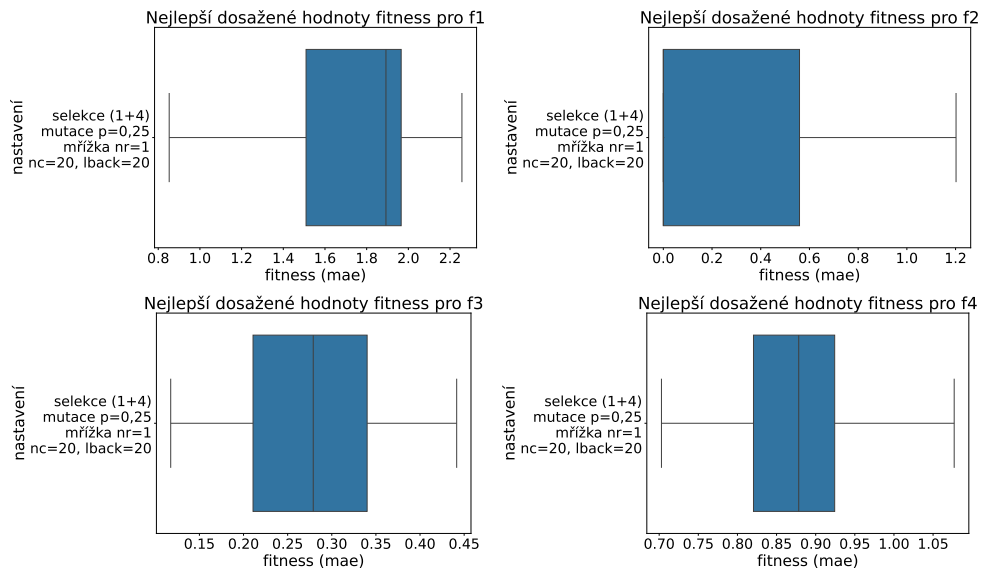
Statistické vyhodnocování bylo provedeno na zařízení s procesorem AMD Ryzen 7 2700X. Dvanáct z šestnácti dostupných logických procesorů bylo vyhrazeno pro zmíněné výpočty. Každému běhu byl přiřazen právě jeden logický procesor a 60 sekund jeho času. Tento přístup byl zvolen proto, aby došlo k výraznému zrychlení výpočtu (zpravidla do dvou

hodin skutečného času na jeden experiment nad zmíněnými čtyřmi úlohami) a zároveň aby měly porovnávané varianty co nejvyrovnanější podmínky.

V rámci experimentů byly použity statistické testy pro určení, zda je rozdíl v nejlepších dosažených hodnotách fitness mezi různými přístupy statisticky významný, nebo ne. Jako první byl vždy proveden Shapiro-Wilkův test na normalitu. Pokud normalita nebyla zamítnuta ($\alpha = 0,05$), byl použit T-test, v opačném případě U-test. Pokud je některé nastavení v této kapitole prohlášeno za nejlepší ze zkoumaných, znamená to, že se žádné další varianty neprokázaly jako statisticky významně lepší.

Analýza průběhu symbolické regrese u úloh ze sbírky samostatně

Při kombinaci výsledků nad vícero funkcemi do jednoho grafu by mohlo dojít k určitým problémům. Krabicový graf by mohl poskytovat zavádějící výsledky, pokud by například u tří ze čtyř funkcí došlo k nalezení řešení či lokálního optima hned na začátku evoluce, přičemž by ve většině případů nedocházelo u těchto tří úloh k dalšímu zlepšování hodnoty fitness. Ve skutečnosti by tak bylo porovnáváno, jak dobře si vede dané nastavení na jediné úloze. U tří nevhodných úloh by pro velmi rozdílná nastavení mohlo dojít při stejném časovém limitu k prakticky stejným výsledkům. Na obrázku 7.1 je vidět, že pro zvolené nastavení byly výsledky pro všechny úlohy ze sbírky dostatečně rozmanité (nedošlo například u 95 % běhů k uvážnutí v lokálním optimu).

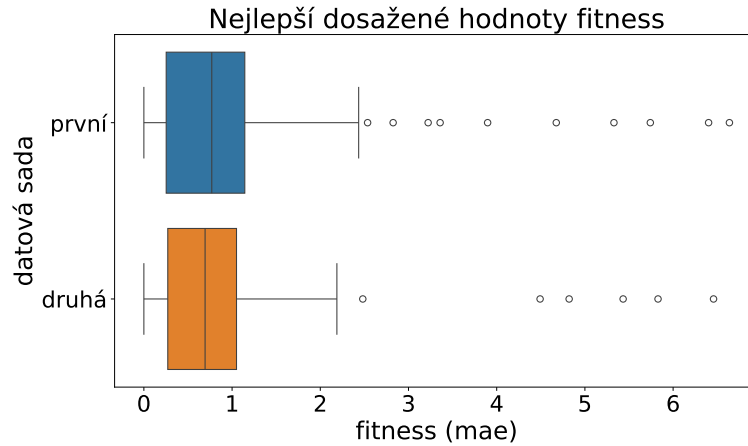


Obrázek 7.1: Symbolická regrese na úlohách ze sbírky samostatně.

Porovnání stejného nastavení na odlišných trénovacích vektorech

Obrázek 7.2 ukazuje, jak moc rozdílné byly výsledky vybraného nastavení (selekcce (1+4), mřížka n_r , n_c , $lback = (1, 20, 20)$ a statická mutace $p = 0,25$), pokud došlo k použití dvou sad náhodně vygenerovaných trénovacích vektorů pro sbírku úloh uvedenou na začátku této sekce. Pro tento experiment byla vytvořena druhá datová sada (došlo k novému náhodnému vygenerování) vstupních dat, přičemž trénovací vektory byly vytvořeny stejným způsobem, jako první sada.

Toto srovnání bylo provedeno, protože k případnému ověřování jsou použity datové sady z **PMLB**, nikoli jiné trénovací vektory pro čtyři úlohy ze sbírky uvedené na začátku této sekce. Mohlo by se tedy teoreticky stát, že na jedné podmnožině trénovacích vektorů je referenční přístup lepší, než řízení parametry, zatímco zmíněné řízení dosahuje výrazně lepších výsledků na jiné podmnožině. Ze zmíněného obrázku 7.2 je zřejmé, že mezi dvěma různými sadami trénovacích vektorů nebyly výrazné rozdíly. Ani na základě statistických testů nebyl zjištěn (statisticky) významný rozdíl.



Obrázek 7.2: Porovnání vybraného nastavení symbolické regrese za použití dvou stejným způsobem vytvořených sad bodů. Graf obsahuje výsledky z funkcí f_1 , f_2 , f_3 a f_4 dohromady.

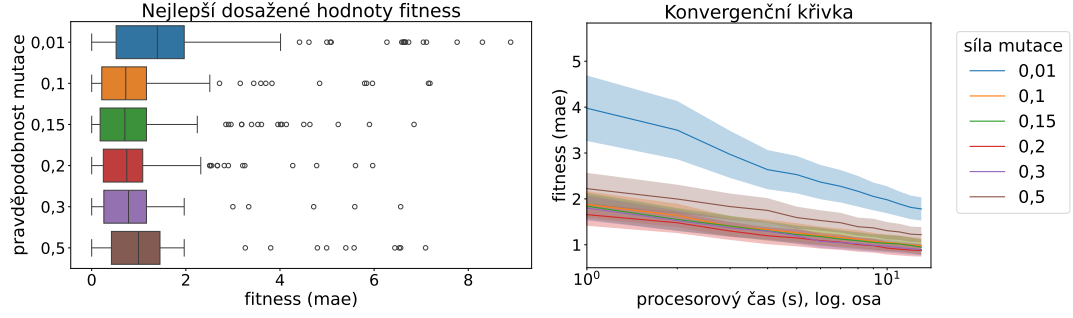
7.1 Parametry referenčního přístupu

Jako referenční přístup byla zvolena kombinace selekčního algoritmu ($\mu + \lambda$), konkrétně (1+4), společně se statickou mutací (u které je síla určena na základě ladění) a se zafixovanými parametry mřížky (opět určenými na základě ladění).

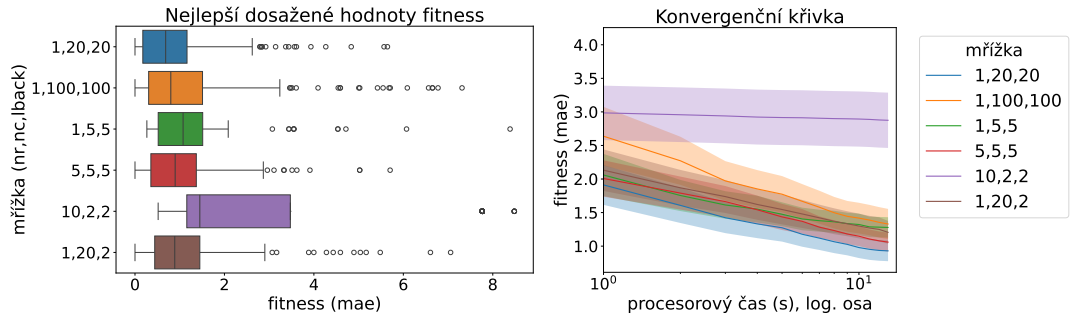
Nejprve došlo k hledání vhodné mutace napříč čtyřmi úlohami ze sbírky. Porovnávána byla nejlepší dosažená fitness za dobu běhu 60 sekund procesorového času. Selekčním algoritmem byl vždy mechanismus (1+4) a parametry mřížky byly pokaždé nastaveny na $n_r = 1$, $n_c = 20$, $lback = 20$. Nejlepších výsledků dosáhla mutace s $p = 0, 1$. Výsledky (krabicový graf nejlepších dosažených fitness a odpovídající konvergenční křivky) jsou zobrazeny na obrázku 7.3.

Jakmile byla nalezena vhodná pravděpodobnost mutace pro referenční přístup, došlo k ladění parametrů mřížky. Tato vyladěná mutace byla opět doplněna selekčním algoritmem (1+4). Srovnání různých nastavení parametrů mřížky je vidět na obrázku 7.4. Jako nejvhodnější se projevil přístup, kdy byly parametry zvoleny jako $n_r = 1$, $n_c = 20$, $lback = 20$.

Referenčním přístupem, se kterým byly adaptivní metody porovnávány, byl tedy určen algoritmus (1+4), neadaptivní mutace s pravděpodobností 0,1 a pevné parametry mřížky, konkrétně $n_r = 1$, $n_c = 20$ a $lback = n_c$.



Obrázek 7.3: Srovnání nastavení pravděpodobnosti mutace napříč čtyřmi úlohami ze sbírky. Selektčním algoritmem byl vždy mechanismus (1+4) a mřížka byla pokaždé nastavena na $n_r = 1, n_c = 20, lback = n_c$. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.



Obrázek 7.4: Ladění statických parametrů mřížky CGP. Selektčním algoritmem byl mechanismus (1+4) a mutace byla $p = 0, 1$ (ve všech zkoumaných případech). Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.

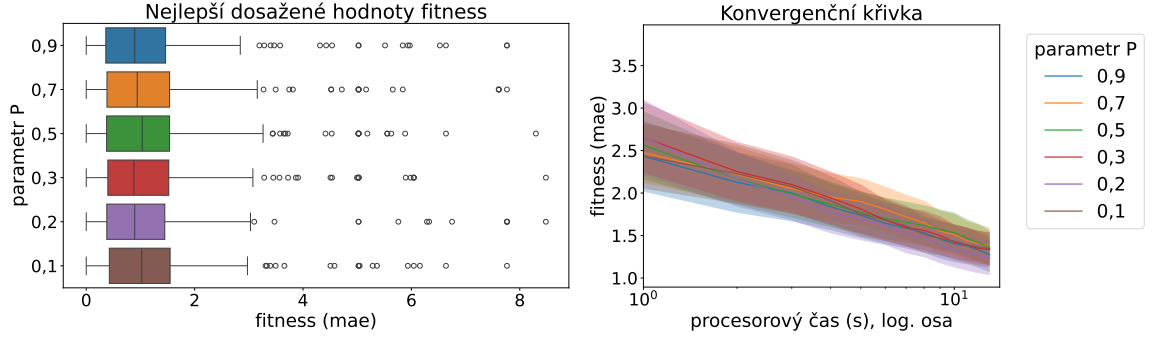
7.2 Ladění statických parametrů adaptivních metod

I adaptivní metody mají své statické parametry, které mohou být pro rozdílné úlohy odlišně vhodné. V rámci experimentů bylo ověřováno, jestli jsou vyladěné adaptivní metody lepší, než vyladěné statické přístupy, a jaká je jejich (adaptivních metod) případná přenositelnost na úlohy, na kterých nebyly laděny. V této sekci došlo vždy ke kombinaci jednoho řízeného mechanismu (selektce, mutace, parametry mřížky) a statického (vyladěného) nastavení.

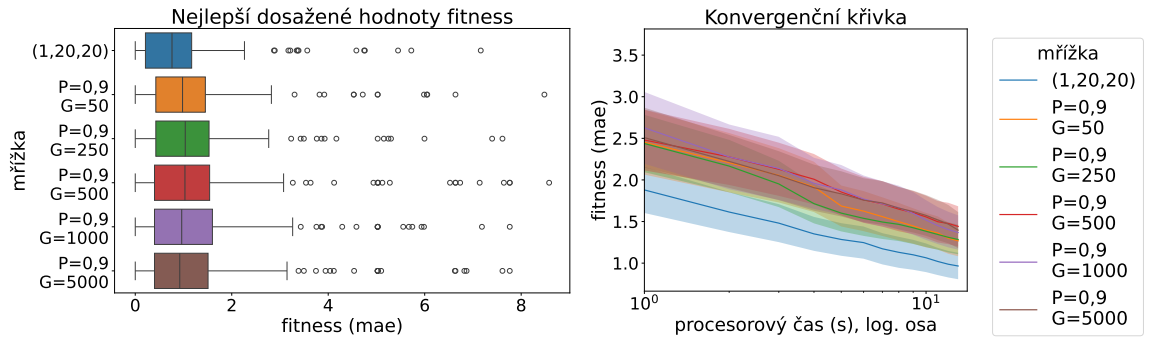
Samo-adaptivní řízení mřížky

Tento přístup byl zkombinován se základním selektčním algoritmem a mutací dříve určenou na základě ladění ($p=0,1$). V prvním experimentu (obrázek 7.5) se jako nejlepší prokázalo nastavení $P = 0,9$, přičemž parametr G byl během tohoto vyhodnocování zafixovaný na hodnotu 500. V druhém experimentu (viz obrázek 7.6) byl stanoven jako nejlepší parametr $G = 50$. Výsledkem je tedy nastavení $P = 0,9; G = 50$. Mezi zkoumanými variantami byly minimální rozdíly, zpravidla statisticky nevýznamné. Rovněž během druhého experimentu došlo k porovnání s referenčním přístupem.

Samo-adaptivní řízení mřížky se neprokázalo v kombinaci s $(\mu + \lambda)$ jako lepší, než zafixované parametry. A to ani v případě, kdy k řízení docházelo pouze na začátku evoluce (řízení bylo ukončeno při stagnaci nejlepší dosažené fitness po dobu $G = 50$ generací). Zároveň při velmi rozdílném nastavení parametrů tohoto řízení docházelo k velmi podobným



Obrázek 7.5: Hledání vhodného nastavení parametru P , viz sekce 5.3. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.



Obrázek 7.6: Hledání vhodného nastavení parametru G a porovnání s referenčním přístupem. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.

výsledkům. To mohlo souviset s tím, že se při využití selekčního algoritmu $(1 + \lambda)$ stal rodičem vždy jedinec s nejvyšší fitness.

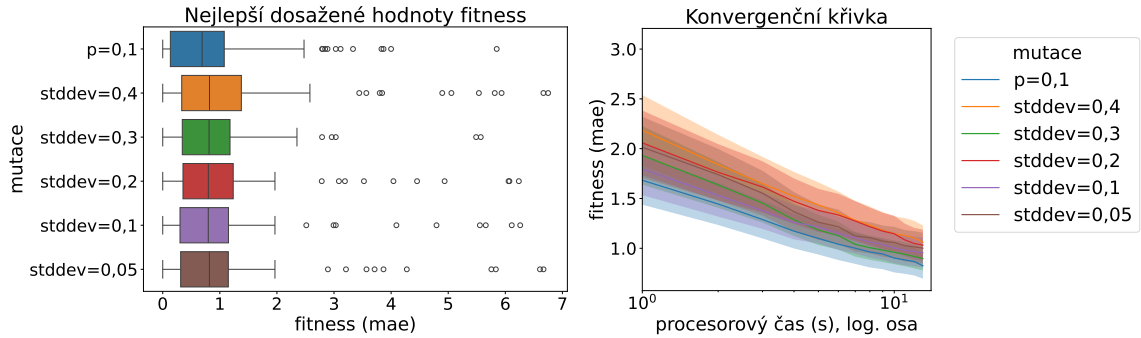
V takovém případě tak mohlo dojít k situaci, kdy se při prvotním náhodném vygenerování vybral jakožto rodič nejvhodnější jedinec (ostatní mřížky byly přitom zapomenuty), který sám svoji mřížku neměnil (řízení mřížky je nástavbou nad mutací) a ke změně mřížky docházelo pouze občas (pokud došlo k vytvoření lepšího nebo alespoň stejně dobrého potomka). Dále při provádění neutrálních mutací zvětšení parametrů k ničemu nejspíš nepřispělo, zatímco snížení například n_c mohlo slibného potomka zdeformovat.

Samo-adaptivní řízení mutace

Mechanismus řízení mutace tímto způsobem (popsaný v sekci 5.2) využívá pouze jeden parametr - `stddev`. Obdobně jako u samo-adaptivního řízení mřížky, i při kombinaci tohoto typu mutace s vyladěnými ostatními komponentami nedošlo ke zlepšení oproti statické mutaci, viz obrázek 7.7.

Hlavní myšlenkou tohoto řízení bylo, že selekční tlak přispěje ke vhodnému určení síly mutace pro danou fázi evoluce. Jenže při použití selekčního mechanismu $(1+4)$ je jako předek určen vždy nejlepší jedinec, ze kterého jsou vytvořeni všichni potomci. Takto vytvořeným jedincům v nové generaci je upravena jejich vlastní mutace náhodně, aniž by byla zohledněna užitečnost dané mutace při jednotlivých fázích evoluce. Pokud by byl využit například turnajový výběr, mohli by jedinci s vhodnější pravděpodobností mutace (v dané fázi

evoluce) častěji tvořit potomky, než ta kandidátní řešení, co mají tento parametr nastaven nevhodně. Při selekčním algoritmu $(\mu + \lambda)$ nicméně tento přístup vede k náhodné oscilaci pravděpodobnosti mutace u předka, který mnohdy zůstává po generaci stejný. V případě využití potomka (jakožto rodiče nové generace) je mutace zmíněného potomka upravena také náhodně, aniž by byla jakkoli zohledněna její vhodnost či nevhodnost. Samo-adaptivní přístup s turnajovým výběrem je zkoumán později v této sekci.



Obrázek 7.7: Porovnávání různých nastavení parametru `stddev` samo-adaptivní mutace a referenčního přístupu ($p=0,1$). Grafy obsahují výsledky z funkcí f_1 , f_2 , f_3 a f_4 dohromady.

Adaptivní řízení mutace

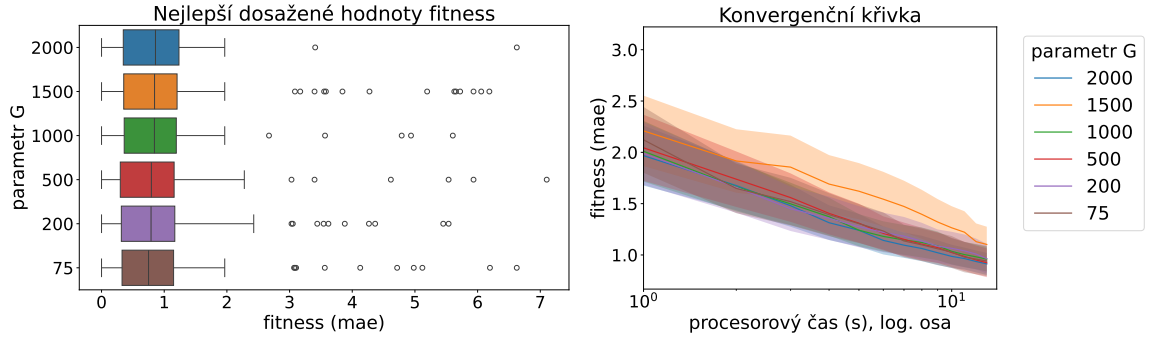
Obdobně jako u samo-adaptivní mutace, i tento přístup byl popsán v sekci 5.2. I zde byl při experimentech použit selekční algoritmus $(1+4)$ a statické nastavení mřížky, které vzešlo z experimentů. V případě této metody, která využívá dva parametry, došlo k ladění zmíněných parametrů postupně. Nejprve byl experimentálně určen parametr G , přičemž parametr m byl nastaven na hodnotu 0,05, viz obrázek 7.8. V kombinaci s $(1+4)$ byly mezi různými variantami malé rozdíly, zpravidla opět statisticky nevýznamné. Zvoleno bylo nastavení $G = 2000$.

Jakmile byl nalezen vhodný parametr G , došlo k ladění druhého parametru m , viz graf 7.9. Jako nejlepší se při zvoleném ostatním nastavení projevil parametr $m = 0,1$. Nakonec byla nejlepší kombinace porovnána s referenčním přístupem (graf 7.10). Adaptivní mutace sice na začátku dosahovala lepších výsledků, po 60 s procesorového času se však jejich rozdíl srovnal a mezi nejlepšími nalezenými hodnotami fitness není statisticky významný rozdíl.

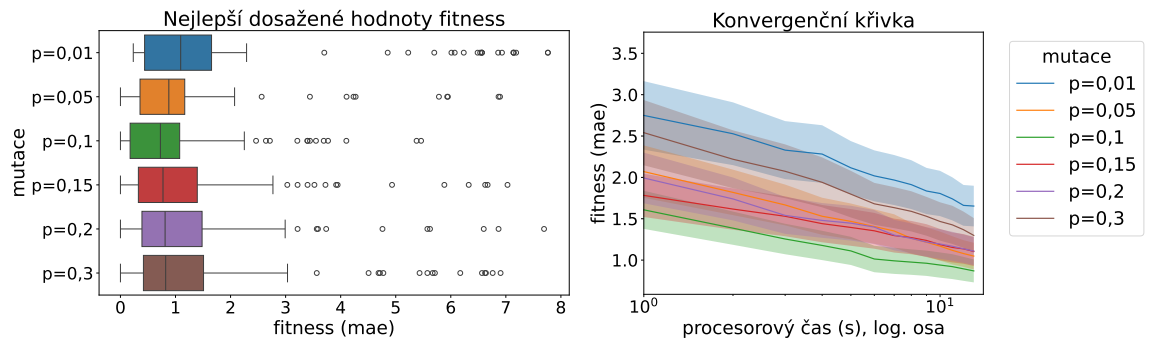
Adaptivní selekce

Poslední komponenta, která je v této práci řízena, je selekce. Popis metody se nachází v sekci 5.1. Při tomto přístupu dochází ke zvýšení selekčního tlaku při zvyšování fitness a ke snižování tohoto tlaku při stagnaci fitness. Selektivní tlak udává, jak moc jsou jedinci s lepší fitness upřednostňováni před těmi, co mají horší fitness. V navržené metodě na řízení selekce jsou používány dva parametry: mod a G . První zmíněný určuje, jak moc se má změnit selekční tlak při zlepšení či stagnaci fitness. Druhý stanovuje, kolik generací bez nalezení nové nejlepší fitness hodnoty značí uvážnutí v lokálním extrému. Obdobně jako u adaptivní selekce došlo i zde na postupné ladění těchto parametrů a závěrečné porovnání s referenčním přístupem.

Ladění prvního parametru při stanovení $G = 500$ je zdokumentováno na obrázku 7.11. Jako nejlepší se prokázala hodnota $mod = 7$. Se zvyšující se touto hodnotou bylo dosaženo



Obrázek 7.8: Hledání vhodného parametru G , který u adaptivní mutace značí, po kolika generacích stagnace fitness došlo k uváznutí v lokálním optimu. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.



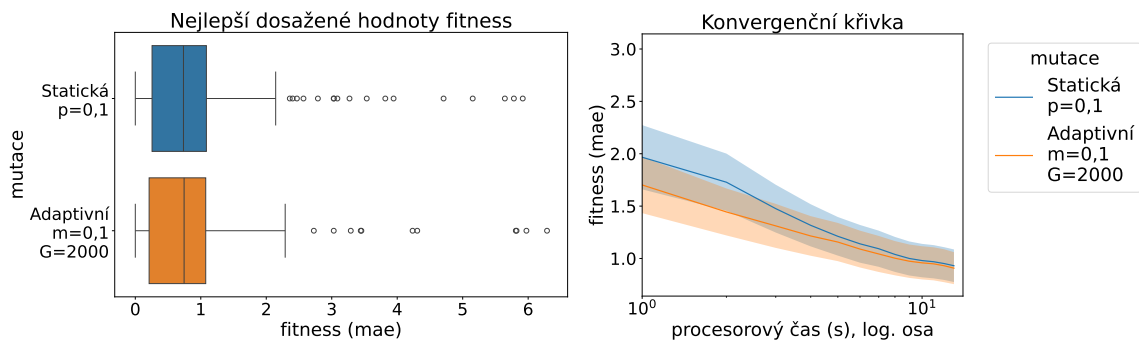
Obrázek 7.9: Hledání vhodného parametru m , který u adaptivní mutace udává, jak moc se má změnit fitness, pokud došlo ke zlepšení průměrné fitness nebo uváznutí v lokálním optimu. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.

lepších výsledků. Pokud by tento parametr byl nastaven na příliš vysokou hodnotu, došlo by k rychlému přepnutí do selekce (1+4), která by oscilovala s binárním turnajovým výběrem.

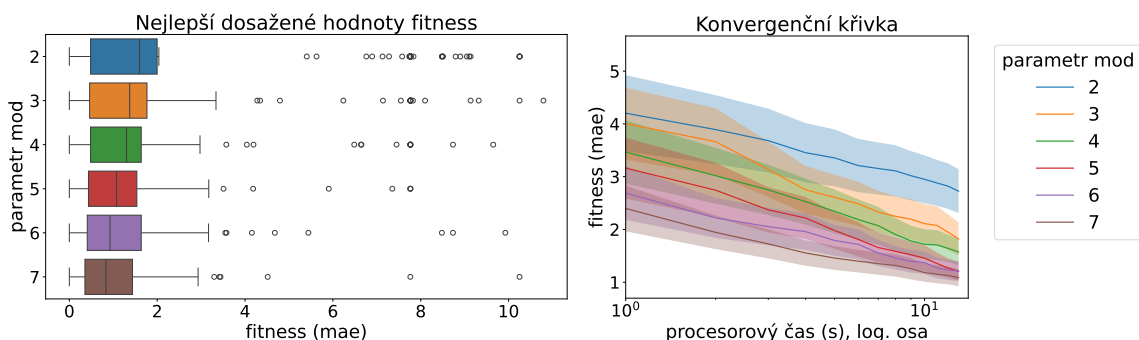
Hledání vhodné hodnoty druhého parametru je vidět na obrázku 7.12. Jako nejlepší se ukázala hodnota $G = 2000$. Obdobně jako při ladění parametru mod (viz obrázek 7.11) platí, že čím více se (zvyšováním hodnoty) blíží nastavení selekčnímu algoritmu (1+4), tím dosahuje adaptivní selekce lepších výsledků. Rovněž zde došlo k porovnání s referenčním přístupem. Statický mechanismus selekce se prokázal jako statisticky významně lepší, než řízená selekce.

Zkoumané varianty řízení parametrů bohužel nedosáhly v žádném z experimentů statisticky významně lepších výsledků než referenční přístup. Adaptivní selekce se nejspíš neprojevila jako účinná, protože při tomto přístupu nedochází tolik k neutrálním mutacím, ze kterých CGP může potenciálně velmi těžit. Při nalezení jedince z lokálního optima je tento jedinec z populace zpravidla odstraněn (to je způsobeno nasazením selekčního algoritmu s nižším selekčním tlakem, například binárním turnajem), přičemž nemá šanci doladit detaily svého fenotypu pomocí neutrálních mutací.

Stejně tak samo-adaptivní řízení mutací a mřížky se neprojevilo jako účinné. Jak již bylo zmíněno dříve v této sekci, pokud se nejlepší jedinec vždy dostane do další generace a je rodičem všech potomků, vhodné parametry mřížky a vhodná mutace se nemohou vyvíjet (minimálně u navržených metod) souběžně s kandidátními řešeními. Všichni potomci



Obrázek 7.10: Porovnání adaptivní mutace a referenčního přístupu. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.



Obrázek 7.11: Ladění parametru mod u adaptivní selekce. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.

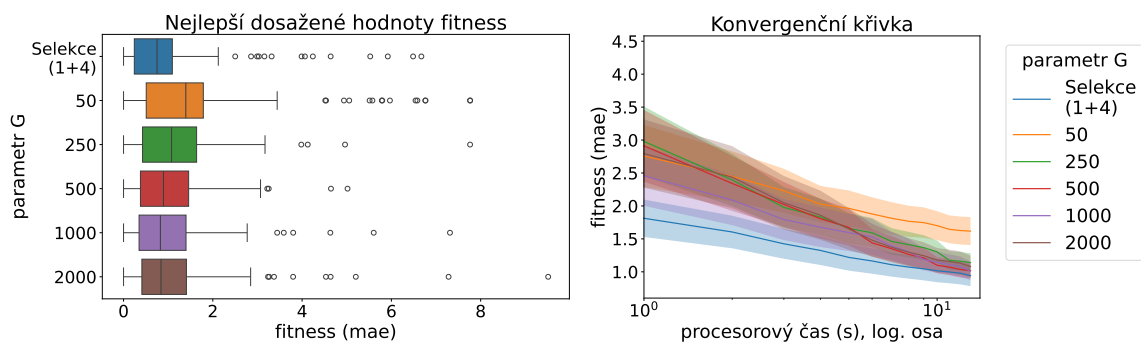
v horizontu několika generací totiž berou oba parametry od stejného kandidátního řešení, aniž by jim byl dán prostor tyto mnohdy velmi odlišené parametry v dané fázi vyzkoušet. Dokonce ani kombinace adaptivní selekce a (samo) adaptivní mutace za použití referenční mřížky nepřinesla žádoucí efekt, viz obrázek 7.13.

Shrnutí sekce

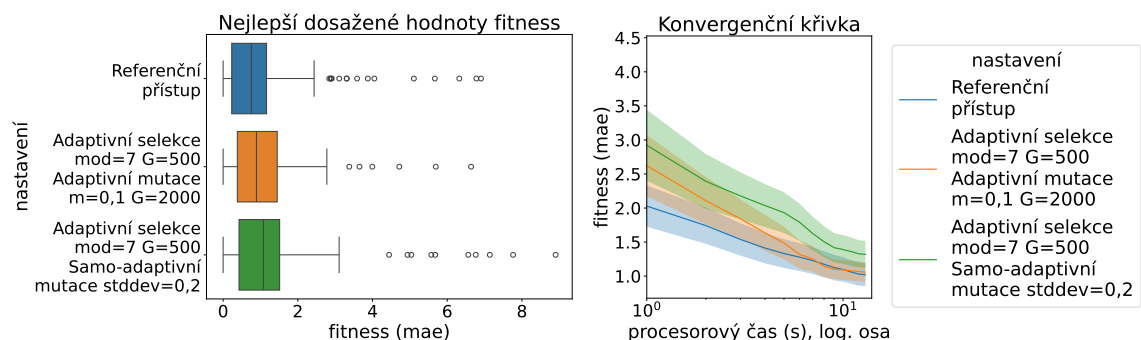
Řízení parametrů mřížky a síly mutace bylo navrženo tak, aby kooperovalo s adaptivní selekcí. Hlavním cílem adaptivní selekce bylo, aby na začátku populace neuvázla v lokálním optimu a pokud později v průběhu výpočtu k tomuto uváznutí dojde, mělo dojít snížením selekčního tlaku k dočasné snížení fitness v populaci za účelem úniku ze zmíněného lokálního extrému, a to bez restartu celé evoluce.

Při tomto přístupu ale u CGP nedochází tak často k neutrálním mutacím, jako u algoritmu (1+4). Kdy i přesto, že to vypadá, že došlo k uváznutí v lokálním optimu, dochází k užitečným neutrálním mutacím a tím k masivnímu prohledávání. Na obrázcích 7.11 a 7.12 je vidět, že čím více jsou parametry adaptivní selekce nastaveny tak, aby se selekční mechanismus podobal algoritmu (1+4) (tedy při zvyšování hodnot obou parametrů mod i G), tím lepších výsledků dané CGP dosahuje.

Otázkou zůstává, jestli vždy a jednoznačně platí, že čím více se adaptivní selekce podobá algoritmu (1+ λ), tím lepších výsledků dosahuje. Obrázek 7.14 ukazuje, jakých výsledků adaptivní selekce dosáhla, když byla nastavena tak, aby se velmi podobala selekci (1+4),



Obrázek 7.12: Ladění parametru G u adaptivní selekce a porovnání s referenčním přístupem. Grafy obsahují výsledky z funkcí f_1 , f_2 , f_3 a f_4 dohromady.

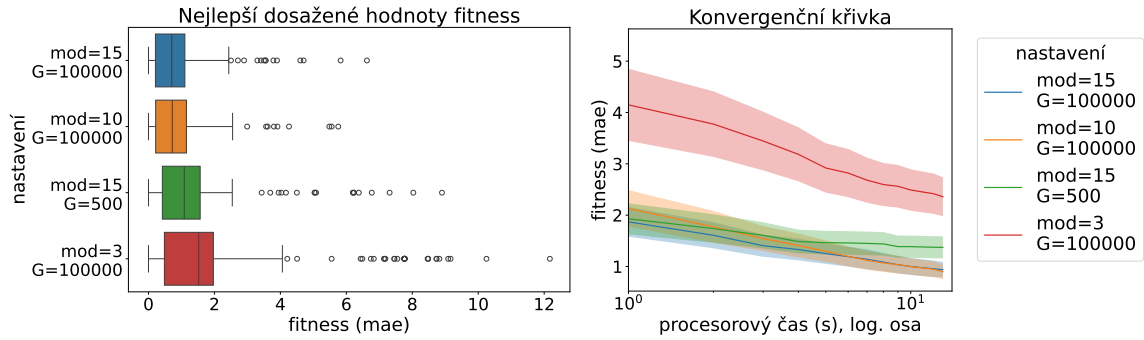


Obrázek 7.13: Porovnání referenčního přístupu a vyladěné adaptivní selekce se dvěma druhy řízených mutací. Grafy obsahují výsledky z funkcí f_1 , f_2 , f_3 a f_4 dohromady.

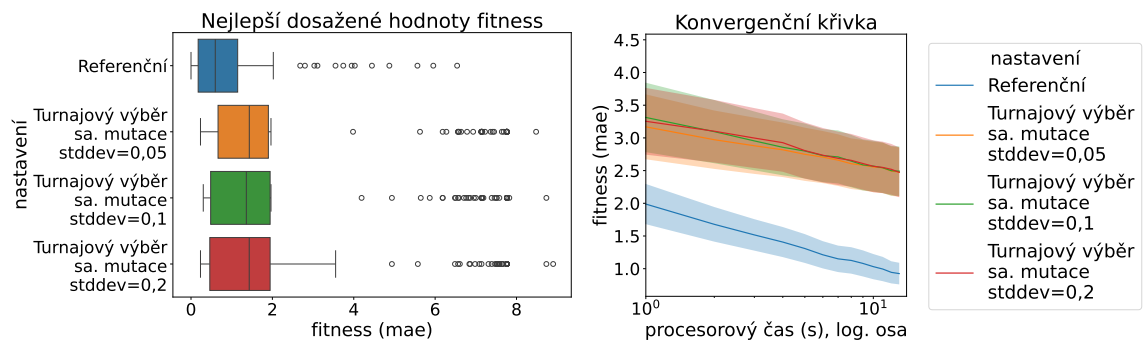
přičemž mutace i mřížka byly referenční. V případě parametru *mod* nastaveného na vysokou hodnotu (10 a víc) je po pár zlepšení nejlepší nalezené fitness selekční mechanismus přepnut na (1+4). Pokud je parametr G příliš vysoké číslo, není při daném časovém limitu selekční tlak nikdy snížen.

U prvních dvou nastavení (shora) na obrázku 7.14 je po dvou (respektive třech) zlepšení fitness selekční mechanismus nastaven na (1+4), přičemž na tomto selekčním algoritmu CGP zůstane. U třetího nastavení se sice algoritmus do stejného selekčního mechanismu také rychle dostane, nicméně po krátkém uvážnutí v lokálním optimu začne používat turnajový výběr. U posledního nastavení selekční tlak roste velmi pozvolna, ale prakticky nikdy nedojde k jeho snížení. Obrázek 7.14 ukazuje, že nastavení adaptivní selekce, které více podobá algoritmu (1+4), dosahuje lepších výsledků, než řízení, které se zmíněnému statickému algoritmu podobá méně.

Jenže čím více se selekce podobá zmíněnému selekčnímu mechanismu, tím menší smysl dávají další adaptivní metody zkoumané v této sekci. Například samo-adaptivní mutace a parametry mřížky nemohou na základě selekčního tlaku napříč populací prosazovat nastavení vhodné pro daný čas evoluce, pokud jsou všichni potomci tvořeni z jednoho předka, který se navíc napříč několika generacemi nemusí změnit. Byl vyzkoušen i turnajový výběr (společně s řízenými parametry), viz obrázek 7.15. Ani při použití selekce, která není příliš elitářská, to při zkoumání vybraného řízení nevypadá, že by navržené řízené parametry byly přínosem. Turnajový výběr, u kterého samo-adaptivní parametry dávají větší smysl, nedosahuje (minimálně na zkoumané sbírce úloh) tak dobrých výsledků, jako (1+4).



Obrázek 7.14: Porovnávání výkonů adaptivní selekce, která byla nastavena tak, aby se podobala selekčnímu algoritmu ($\mu + \lambda$). Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.



Obrázek 7.15: Porovnání referenčního přístupu a různě nastavené samo-adaptivní mutace za využití turnajového výběru jakožto selekčního algoritmu. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.

Navržené řízení parametrů bylo zkoumáno i za použití následujících úprav:

- Řídit mutaci a selekci jak na základě změny fitness, tak na základě změny **průměrné** fitness.
- Povolit či nepovolit adaptivní selekci snižovat selekční tlak před dosažením turnajového výběru (viz sekce 5.1).
- Snížit, nebo nesnížit selekční tlak pod turnajový výběr, pokud je turnajového výběru dosaženo (viz sekce 5.1).
- Měnit velikost populace, u proporčních selekčních algoritmů by velikost populace nastavená na hodnotu 5 nemusela být tolik účinná. Každý selekční algoritmus měl vlastní velikost populace.
- U adaptivní mřížky mutovat i více parametrů ($n_r, n_c, lback$) při jedné mutaci jedince.
- Řídit parametry za využití selekce (μ, λ) namísto ($\mu + \lambda$).
- U samo-adaptivní mutace měnit pravděpodobnost mutace pouze u potomků a neaktualizovat ji neustále u předka.

- Měnit pravděpodobnost mutace u samo-adaptivního přístupu před mutováním ostatních genů jedince.
- Řídit mutaci nebo mřížku u turnajového výběru s rozdílným nastavením (velikost populace a turnaje).
- Zkoumat, jestli adaptivní parametry nejsou lepší až při delších bězích.

Ani při takovýchto změnách se řízené parametry neprokázaly jako lepší, než referenční přístup.

7.3 Návrh nové metody řízení parametrů na základě výsledků experimentů

Metody řízení parametrů mřížky a mutace popsané v sekci 5 nebyly navrženy jako vhodný doplněk k selekčnímu algoritmu (1+4). Při vyhodnocování těchto metod bylo zjištěno, že algoritmus ($\mu + \lambda$) je pro CGP klíčový a představené adaptivní metody není výhodné s tímto statickým selekčním mechanismem použít. Z tohoto důvodu byl vytvořen návrh ještě jednoho řízení parametrů. Konkrétně se jedná opět o adaptivní mutaci, kde dochází ke změnám agresivity mutací na základě zpětné vazby algoritmu, přičemž pravděpodobnost mutace je pro všechny jedince v populaci stejná. Tato nová adaptivní mutace byla určena jako komponenta spolupracující se zmíněnou selekcí (1+4).

Adaptivní mutace pro selekční algoritmus ($\mu + \lambda$)

Tento přístup k řízení parametrů CGP dynamicky upravuje pravděpodobnost mutace takovým způsobem, aby docházelo k co největšímu množství neutrálních mutací (nebo mutací vedoucích ke zlepšení fitness jedinců). Při příliš velké pravděpodobnosti mutace dochází při použití selekce ($\mu + \lambda$) k přepisování aktivních uzlů a potenciálně ke znehodnocení kandidátních řešení. Pokud je naopak mutace příliš malá, dochází k častému generování potomků, kteří se genotypem neliší od předka.

Otázkou zůstává, jak zjistit, jestli je například 20 % pravděpodobnost mutace v dané fázi evoluce vhodná. Může být jak příliš vysoká, tak příliš nízká. Rovněž dříve zmíněné závisí i na parametrech mřížky. Při velké mřížce bude nejspíše vhodná agresivnější mutace, než při velmi malé mřížce. Adaptivní mutace by tak částečně mohla i vyvážit nevhodně nastavené parametry n_c , n_r a $lback$. Pokud by byla mřížka nevhodně velká, mutace by u stejné úlohy při tomto přístupu rostla. Naopak, při velmi malých parametrech n_r , n_c , $lback$ by se mutace přizpůsobila zmenšením agresivity.

V navrhované metodě je důležitým ukazatelem, ke kolika neutrálním mutacím (nebo mutacím, které vedly ke zlepšení fitness) došlo v horizontu $freq$ generací ($freq$ je parametrem metody). Pro každého potomka, který má stejnou (nebo lepší) fitness než předek, se spočítá počet genů, které se změnily.

Na začátku je pravděpodobnost mutace nastavena na 20 %. Pokud dojde ke zvýšení počtu užitečných mutací, je tato pravděpodobnost snížena o m procent. Tato změna se neodvíjí od současné pravděpodobnosti mutace, tudíž z 20 % se změní přímo o například 3 % na 17 %. Stejně tak je třeba z 6 % snížena na 3 %. Nejedná se o změnu m procent z aktuální síly mutace.

Tímto způsobem dochází k postupnému snižování agresivity mutace, dokud nedojde ke zhoršení počtu užitečných mutací. V takovém případě se otočí znaménko parametru

m a pravděpodobnost mutace se naopak začne zvyšovat. Ke zvyšování dochází tak dlouho, dokud roste počet vhodných mutací. Jakmile je opět detekováno zhoršení, je znaménko znovu změněno. Pokud je vhodná mutace například 12 %, pravděpodobnost mírně osciluje kolem této hodnoty, viz algoritmus 5.

Algorithm 5: Adaptivní mutace pro selekční algoritmus ($\mu + \lambda$)

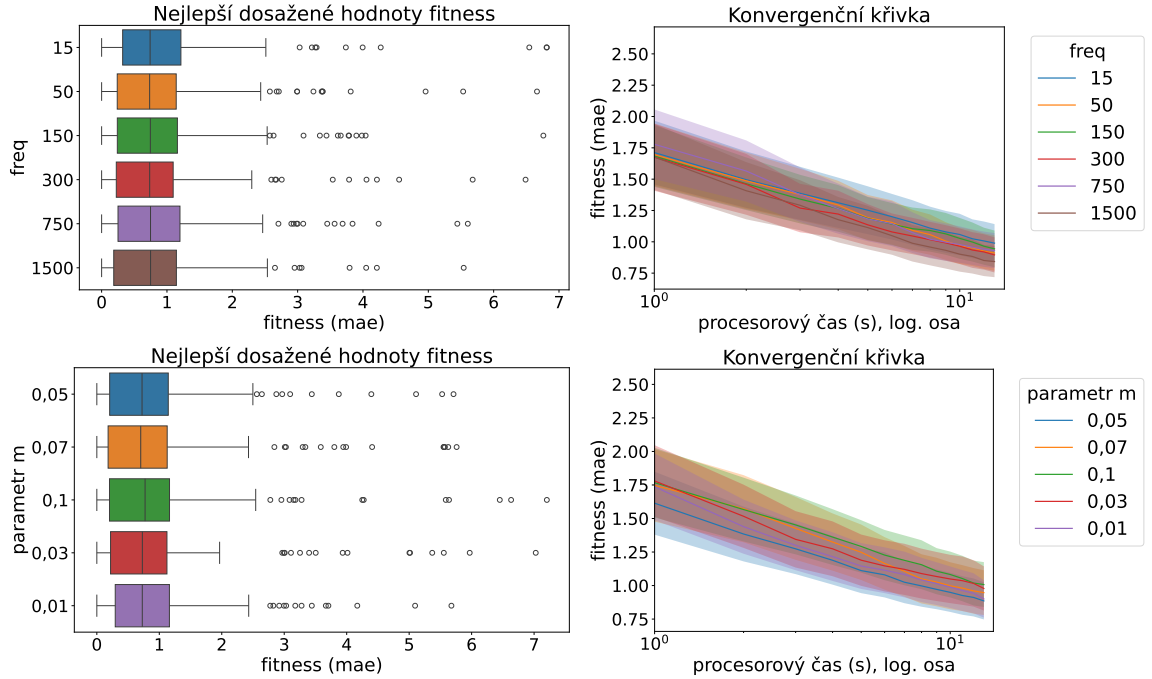
```

Input:  $m, freq$ 
sign = -1;
prevCount = 0;
count = 0;
gens = 0;
p = 0,2; // p je aktuální síla mutace
for each generation do
    gens += 1;
    for offspring in offsprings do
        if offspring.fitness <= parent.fitness then
            count += count_different_genes(offspring, parent);
    if gens > freq then
        gens = 0;
        if count < prevCount then
            sign = sign * -1;
        p = p + sign * m; //  $p \in [0,03; 0,8]$ 
        prevCount = count;
        count = 0;
```

V rámci experimentálního ověřování nejprve opět došlo ke ladění vhodných parametrů. Jak již bylo zmíněno, parametr m určuje, o kolik procent se má změnit síla fitness (nestanovuje, jakým směrem), pokud k takové změně má dojít. Parametr $freq$ určuje, jak často dochází ke změnám mutací (respektive k analýze zpětné vazby CGP). Ladění znázorňuje obrázek 7.16, nejprve bylo dočasně stanoveno nastavení $m = 0,03$, přičemž byl laděn parametr $freq$. Po určení vhodného parametru $freq$ došlo k hledání vhodné hodnoty pro m . Mezi zkoumanými variantami nebyly příliš velké rozdíly. Nejlepších výsledků dosáhlo nastavení $m = 0,03$, $freq = 300$.

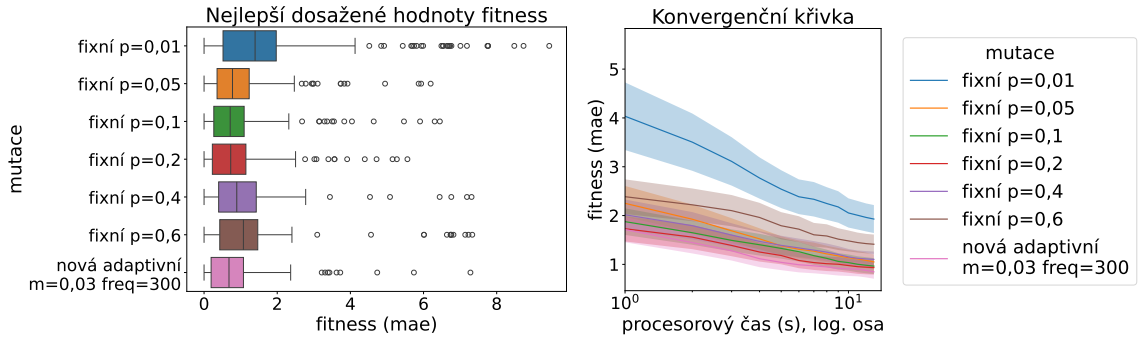
Následné porovnání s referenčním přístupem je zobrazeno na obrázku 7.17. Mezi adaptivní mutací a pevně stanovenými mutacemi s pravděpodobnostmi $p = 0,1$ a $p = 0,2$ nebyl statisticky významný rozdíl, ostatní zkoumané varianty dosahovaly horších výsledků (než zmíněné tři varianty).

V další sekci bylo zkoumáno, jestli je toto řízení vhodné jako „defaultní“ mutace na nových úlohách, tedy zda metoda s parametry vyladěnými na 4 úlohách ze sbírky poskytuje stejné (nebo lepší) výsledky, než vyladěné statické mutace (laděné na míru jiným, než 4 zmíněným úlohám). V takovém případě by pak odpadala nutnost věnovat čas i úsilí ladění mutace (vyladěná mutace by ve výsledku poskytovala stejné výsledky, jako adaptivní, u které ale nebylo třeba věnovat čas ladění). Rovněž tento přístup může vést k vyvážení nevhodně nastavených parametrů mřížky, kdy u příliš velké mřížky je mutace adaptivně zvýšena, při malé naopak snížena. Na čtyřech úlohách ze sbírky se sice nová adaptivní mutace neprokázala jako statisticky významně lepší, než všechny statické varianty, nicméně



Obrázek 7.16: Ladění parametrů nové adaptivní mutace. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.

dosahovala slibných výsledků a na rozmanitější sbírce úloh by mohla dosahovat statisticky významně lepších výsledků.



Obrázek 7.17: Porovnávání referenčního přístupu (za využití dalších mutací, nejen nejlepší $p = 0, 1$) a nové adaptivní mutace, která byla vytvořena jako komponenta k selekčnímu algoritmu $(\mu + \lambda)$. Grafy obsahují výsledky z funkcí f_1, f_2, f_3 a f_4 dohromady.

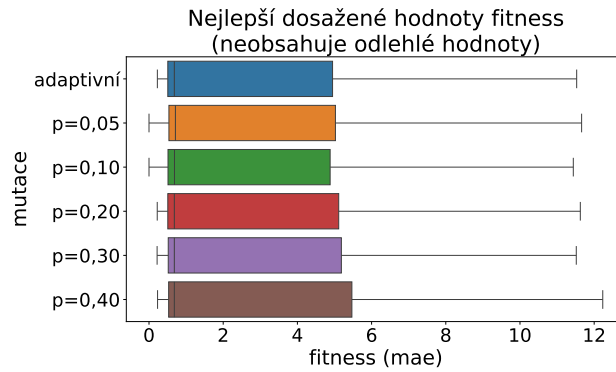
7.4 Adaptivní mutace jako potenciálně vhodná první volba mutace

Pro ověření výsledků nové adaptivní mutace byl využit repozitář **PMLB**. Použity byly všechny datové sady ze seznamu *regression_dataset_names*. Celkově se jednalo o 120 úloh. Z důvodu omezeného výpočetního výkonu byl počet trénovacích vektorů pro každou úlohu

omezen na maximálně 200 (bylo použito prvních 200 trénovacích vektorů). Pro každou datovou sadu bylo spuštěno 12 běhů (celkem $12 * 120 = 1440$ běhů na jedno nastavení), přičemž časový limit byl nastaven na 120 sekund procesorového času.

V rámci referenčního přístupu byla stanovena pravděpodobnost mutace na hodnotu $p = 0,1$. Tato hodnota nicméně nemusí být vhodná u datových sad ze sbírky **PMLB**. Z toho důvodu byla tato mutace opětovně laděna a následně porovnána s vyladěnými statickými parametry adaptivní metody. U adaptivních metod by potenciálně mohla být přenositelnost jejich statického nastavení vyšší, než u statických parametrů (byť pro každou úlohu je vhodné jiné nastavení, viz [2]). Během experimentů byla tedy porovnána nová adaptivní mutace s vyladěnými parametry na čtyřech úlohách ze sbírky (viz začátek této kapitoly) s různými variantami síly statické mutace laděnými na míru **PMLB**.

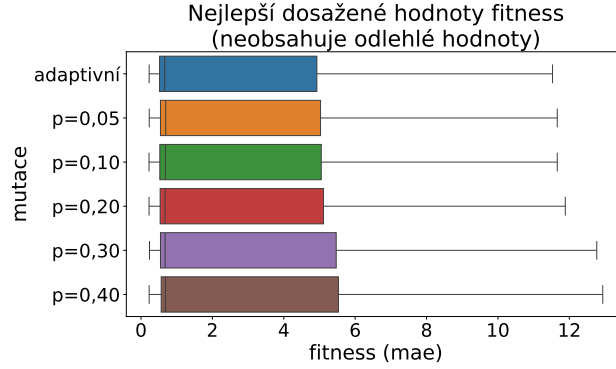
Při prvním experimentu (viz obrázek 7.18) byla použita mřížka, která vzešla z ladění na čtyřech úlohách uvedených na začátku této kapitoly, tedy $n_r = 1$, $n_c = 20$, $lback = nc$. Zkoumána byla adaptivní mutace pro selekční algoritmus ($\mu + \lambda$) a statické mutace s pravděpodobnostmi $\{0,05; 0,1; 0,2; 0,3; 0,4\}$. Varianty pravděpodobnosti mutace $p = 0,05$ a $p = 0,04$ dosáhly statisticky významně horších výsledků, než zbývající varianty, mezi kterými nebyl statisticky významný rozdíl. Pro specifikovanou mřížku tedy řízení mutace sice nepřineslo zlepšení, nicméně ani zhoršení (a to i přesto, že pro adaptivní metodu nebyly na **PMLB** statické parametry laděny).



Obrázek 7.18: Porovnání adaptivní mutace a statických mutací na úlohách z **PMLB**. Mřížka byla nastavena na $n_r = 1$, $n_c = 20$, $lback = nc$. Graf neobsahuje odlehlé hodnoty.

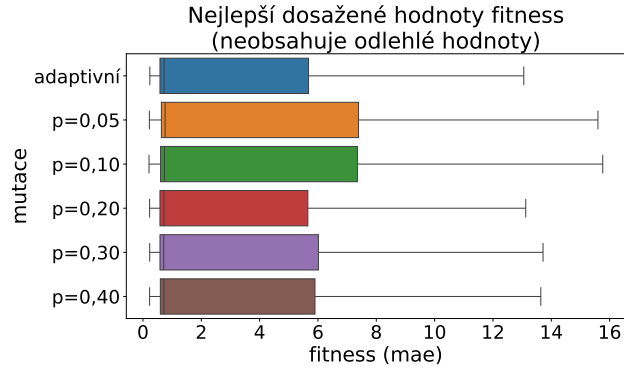
V následujícím experimentu (viz obrázek 7.19) bylo zkoumáno, jaké nastavení bude vhodné při mřížce, která je oproti té předchozí mnohem větší. Konkrétně se jednalo o nastavení, kde $n_r = 1$, $n_c = 150$ a $lback = 150$. V tomto případě mutace s pravděpodobnostmi z množiny $\{0,05; 0,3; 0,4\}$ dosahovaly horších výsledků než zbylá nastavení, mezi kterými nebyl statisticky významný rozdíl.

Poslední experiment ověřoval, jestli adaptivní mutace navržená pro selekční algoritmus ($\mu + \lambda$) vhodně zvýší svoji agresivitu v případě, že je použita mřížka, ve které jsou parametry nastaveny na $n_r = 1$, $n_c = 40$ a $lback = 1$. Tento přístup byl úmyslně zvolen, protože na něm mutace s pravděpodobností $p = 0,1$ na úlohách z **PMLB** nedosahovala (v rámci experimentů nad malou podmnožinou úloh) tak dobrých výsledků, jako statické silnější mutace. Výsledky (nad všemi 120 datovými sadami) jsou zobrazeny na obrázku 7.20. Nejlepších výsledků dosahovaly varianty $\{0,2; 0,3; 0,4; \text{adaptivní}\}$, mezi kterými nebyl statisticky významný rozdíl. Ve všech experimentech nad úlohami z **PMLB** nebyl prokázán



Obrázek 7.19: Porovnání adaptivní mutace a statických mutací na úlohách z **PMLB**. Mřížka byla nastavena na $n_r = 1$, $n_c = 150$ a $lback = 150$. Graf neobsahuje odlehlé hodnoty.

statisticky významný rozdíl mezi adaptivní mutací (se statickými parametry laděnými na jiných úlohách) a nejlepší vyladěnou statickou mutací (laděnou na míru sbírce **PMLB**). Zkoumané řízení mutací tak sice nevede k statisticky významně lepším výsledkům, než kterých dosahuje pevně stanovená vyladěná mutace, nicméně při použití adaptivní mutace odpadá nutnost věnovat čas a úsilí ladění. Napříč velmi odlišnými mřížkami nebyla nalezena laděním statická mutace, která by byla lepší, než adaptivní mutace (se statickými parametry určenými na základě jiných úloh). Přičemž pro různé mřížky byly vhodné odlišné pevně stanovené mutace. Adaptivní mutace se dané mřížce ve všech zkoumaných případech přizpůsobila.



Obrázek 7.20: Porovnání adaptivní mutace a statických mutací na úlohách z **PMLB**. Mřížka byla nastavena na $n_r = 1$, $n_c = 40$ a $lback = 1$. Graf neobsahuje odlehlé hodnoty.

Kapitola 8

Závěr

Cílem této práce bylo navrhnout a experimentálně ověřit metodu pro řízení parametrů CGP. Nejprve jsem prostudoval literaturu týkající se evolučních algoritmů, genetického programování a řízení parametrů v těchto oblastech. Na tato témata jsem zpracoval rešerši. Dále jsem navrhl vícero metod pro automatizované řízení parametrů u zmíněného CGP, přičemž mnohé z nich je možné kombinovat.

Ke změnám v průběhu evoluce docházelo u selekce, mutace a parametrů mřížky. V případě selekce byl vytvořen selekční algoritmus, kde je selekční tlak zvyšován se zlepšující se fitness a snižován při stagnaci fitness. U řízených parametrů mřížky má každý jedinec vlastní mřížku. Při tomto přístupu dochází k vývoji parametrů mřížky souběžně s populací. U mutací byly navrženy tři přístupy. Při prvním přístupu dochází při zlepšení průměrné fitness ke snížení agresivity mutace, při stagnaci průměrné fitness naopak ke zvýšení. V druhé variantě má každé kandidátní řešení vlastní mutaci, přičemž jedinci s vhodnější mutací mají větší pravděpodobnost přežití. Poslední přístup upravuje mutaci tak, aby docházelo k co nejvyššímu počtu neutrálních mutací. Tyto metody jsem poté implementoval a porovnal s referenčním přístupem, kterým byla určena kombinace selekce ($\mu + \lambda$) a vyladěných parametrů mřížky (*nr*, *nc* a *lback*) a vyladěné pravděpodobnosti mutace.

Většina navržených metod dosahovala statisticky významně horších výsledků, než referenční přístup. Téměř všechny metody řízení parametrů byly totiž navrženy jako doplněk k adaptivní selekci, přičemž bylo prokázáno, že statická selekce ($\mu + \lambda$) je pro CGP klíčová a řízení selekce pro toto genetické programování není vhodné.

Relativně dobrých výsledků dosáhl poslední ze tří jmenovaných způsobů řízení mutace. Při použití tohoto řízení navrženého jako doplněk k selekci ($\mu + \lambda$) nebyl zjištěn statisticky významný rozdíl mezi zmíněnou adaptivní metodou a vyladěnou statickou mutací. A to i pro velmi rozdílné parametry mřížky. Při využití tohoto přístupu tak odpadá nutnost věnovat čas a úsilí ladění parametru mutace.

Pro případné pokračování v práci doporučuji ponechat selekční algoritmus ($\mu + \lambda$) a soustředit se na řízení mutace a mřížky. Mutace by například mohla být řízena na základě užitečných mutací v uzlech před výstupním uzlem. Vhodná kombinace řízení mutace a mřížky by mohla vést k tomu, že pro většinu úloh nebude třeba tyto parametry ladit.

Téma této práce mi přišlo zajímavé. Dozvěděl jsem se spoustu nových informací z oblasti hledání vhodných hodnot parametrů a složitosti jejich vzájemných interakcí u (nejen) evolučních algoritmů. I přesto, že výsledky nenaplnily mé očekávání, můj zájem o evoluční algoritmy po této práci spíše vzrostl, než že by byl utlumen.

Literatura

- [1] BACK, T. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In: IEEE. *Proceedings of the first IEEE conference on evolutionary computation. IEEE World Congress on Computational Intelligence*. 1994, s. 57–62. ISBN 0-7803-1899-4.
- [2] DE JONG, K. Generalized Evolutionary Algorithms. In: ROZENBERG, G.; BÄCK, T. a KOK, J. N., ed. *Handbook of Natural Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 625–635. ISBN 978-3-540-92910-9. Dostupné z: https://doi.org/10.1007/978-3-540-92910-9_20.
- [3] EIBEN, A.; HINTERDING, R. a MICHALEWICZ, Z. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 1999, sv. 3, č. 2, s. 124–141.
- [4] GOLDBERG, D. E. a DEB, K. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In: RAWLINS, G. J., ed. *Foundations of genetic algorithms*. Elsevier, 1991, sv. 1, s. 69–93. Foundations of Genetic Algorithms. ISSN 1081-6593. Dostupné z: <https://www.sciencedirect.com/science/article/pii/B9780080506845500082>.
- [5] HOLAND, J. H. Adaptation in natural and artificial systems. *Ann Arbor: The University of Michigan Press*, 1975, sv. 32.
- [6] KALKREUTH, R.; RUDOLPH, G. a KRONE, J. Improving convergence in cartesian genetic programming using adaptive crossover, mutation and selection. In: IEEE. *2015 IEEE Symposium Series on Computational Intelligence*. 2015, s. 1415–1422. ISBN 978-1-4799-7560-0.
- [7] KEIJZER, M. Improving symbolic regression with interval arithmetic and linear scaling. In: Springer. *European Conference on Genetic Programming*. 2003, s. 70–82. ISBN 978-3-540-36599-0.
- [8] MANFRINI, F. A.; BERNARDINO, H. S. a BARBOSA, H. J. A novel efficient mutation for evolutionary design of combinational logic circuits. In: Springer. *Parallel Problem Solving from Nature–PPSN XIV: 14th International Conference, Edinburgh, UK, September 17–21, 2016, Proceedings 14*. 2016, s. 665–674. ISBN 978-3-319-45823-6.
- [9] MILLER, J. F. Cartesian Genetic Programming. In: MILLER, J. F., ed. *Cartesian Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, s. 17–34. ISBN 978-3-642-17310-3. Dostupné z: https://doi.org/10.1007/978-3-642-17310-3_2.

- [10] MILLER, J. F. Cartesian genetic programming: its status and future. *Genetic Programming and Evolvable Machines*. USA: Kluwer Academic Publishers, červen 2020, sv. 21, 1–2, s. 129–168. ISSN 1389-2576. Dostupné z: <https://doi.org/10.1007/s10710-019-09360-6>.
- [11] OLSON, R. S.; LA CAVA, W.; ORZECOWSKI, P.; URBANOWICZ, R. J. a MOORE, J. H. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, Dec 2017, sv. 10, č. 36, s. 1–13. ISSN 1756-0381. Dostupné z: <https://doi.org/10.1186/s13040-017-0154-4>.
- [12] POLI, R.; LANGDON, W. B. a MCPHEE, N. F. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008. ISBN 1409200736.
- [13] RECHENBERG, I. Evolutionsstrategie. *Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, 1973.
- [14] SCHMIDT, M. a LIPSON, H. Distilling Free-Form Natural Laws from Experimental Data. *Science*, 2009, sv. 324, č. 5923, s. 81–85. Dostupné z: <https://www.science.org/doi/abs/10.1126/science.1165893>.
- [15] TOPCHY, A. a PUNCH, W. F. Faster genetic programming based on local gradient search of numeric leaf values. In: Morgan Kaufmann San Francisco, CA. *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*. 2001, s. 155–162. GECCO'01. ISBN 1558607749.
- [16] VANNESCHI, L. a POLI, R. Genetic Programming — Introduction, Applications, Theory and Open Issues. In: ROZENBERG, G.; BÄCK, T. a KOK, J. N., ed. *Handbook of Natural Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 709–739. ISBN 978-3-540-92910-9. Dostupné z: https://doi.org/10.1007/978-3-540-92910-9_24.
- [17] VINCENT, A. M. a JIDESH, P. An improved hyperparameter optimization framework for AutoML systems using evolutionary algorithms. *Scientific Reports*. Nature Publishing Group UK London, Březen 2023, sv. 13, č. 1, s. 4737.
- [18] VLADISLAVLEVA, E. K.; SMITS, G. a HERTOOG, D. den. Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming. *IEEE Transactions on Evolutionary Computation*, Květen 2009, sv. 13, č. 2, s. 333 – 349.
- [19] WHITLEY, D. a SUTTON, A. M. Genetic Algorithms — A Survey of Models and Methods. In: ROZENBERG, G.; BÄCK, T. a KOK, J. N., ed. *Handbook of Natural Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 637–671. ISBN 978-3-540-92910-9. Dostupné z: https://doi.org/10.1007/978-3-540-92910-9_21.