VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

MOŽNOSTI AKCELERACE SYMBOLICKÉ REGRESE POMOCÍ KARTÉZSKÉHO GENETICKÉHO PROGRAMOVÁNÍ

ACCELERATION OF SYMBOLIC REGRESSION USING CARTESIAN GENETIC PROGRAMMING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. DAVID HODAŇ**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. ZDENĚK VAŠÍČEK, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Hodaň David, Bc.**
Program: Informační technologie Obor: Bioinformatika a biocomputing
Název: **Možnosti akcelerace symbolické regrese pomocí kartézského genetického programování**
Acceleration of Symbolic Regression Using Cartesian Genetic Programming
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s kartézským genetickým programováním (CGP) a jeho využitím v úloze symbolické regrese. Seznamte se s instrukcemi rozšíření SSE a AVX, kterými jsou vybaveny moderní procesory, a možnostmi jejich využití pro potřeby akcelerace výpočtů pracujících nad celými čísly.
2. Navrhněte optimalizaci využívající instrukce SSE/AVX umožňující akcelerovat CGP v úloze symbolické regrese.
3. Navrženou optimalizaci implementujte.
4. Zvolte vhodnou demonstrační aplikaci a proveďte sadu experimentů za účelem vyhodnocení výkonnosti navržené optimalizace.
5. Diskutujte parametry navrženého řešení a možnosti dalšího pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Vašíček Zdeněk, doc. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 26. října 2018

Abstrakt

Tato práce je zaměřena na hledání postupů, které by akcelerovaly symbolickou regresi v rámci kartézského genetického programování. Práce přibližuje kartézské genetické programování a jeho využití v úloze symbolické regrese. Zabývá se architekturou SIMD a instrukční sadou SSE a AVX. Práce představuje řadu optimalizačních metod, které vedou k výraznému urychlení evoluce v kartézském genetickém programování. Metoda bitově paralelní simulace používající vektory AVX2 umožňuje paralelně pracovat s 256 vstupními kombinacemi logického obvodu. Obdobně lze využít bajtově paralelní simulaci a pracovat se 32 bajty při evoluci obrazového filtru. Metoda akcelerace pomocí generování nativního kódu výrazně urychluje evaluaci kandidátních řešení. Nová metoda dávkové mutace může zrychlit evoluci kombinačních logických obvodů i tisíckrát v závislosti na velikosti problému. Kombinací zmíněných i dalších metod trvala například evoluce násobiček 5×5b v průměru 5,8 vteřin na procesoru Intel Core i5-4590.

Abstract

This thesis is focused on finding procedures that would accelerate symbolic regressions in Cartesian Genetic Programming. It describes Cartesian Genetic Programming and its use in the task of symbolic regression. It deals with the SIMD architecture and the SSE and AVX instruction set. Several optimizations that lead to a significant acceleration of evolution in Cartesian Genetic Programming are presented. A method of a bit-level parallel simulation that uses AVX2 vectors allows to process 256 input combinations of a logic circuit in paralell. Similarly it is possible to use a byte-level parallel simulation and work with 32 bytes when evolving an image filter. A new method of batch mutation can accelerate the evolution of combinational logic circuits thousand times depending on the problem size. For example, using a combination of these and other methods the evolution of 5×5b multipliers took 5.8 seconds on average on an Intel Core i5-4590 processor.

Klíčová slova

Evoluční algoritmy, kartézské genetické programování, symbolická regrese, filtrace obrazu, optimalizace, akcelerace, rychlost

Keywords

Evolutionary algorithms, cartesian genetic programming, symbolic regression, image filtering, optimization, acceleration, speed

Citace

HODAŇ, David. *Možnosti akcelerace symbolické regrese pomocí kartézského genetického programování*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Zdeněk Vašíček, Ph.D.

Možnosti akcelerace symbolické regrese pomocí kartézského genetického programování

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Doc. Ing. Zdeňka Vašíčka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
David Hodaň
21. května 2019

Poděkování

Děkuji tímto vedoucímu diplomové práce panu Doc. Ing. Zdeňku Vašíčkovi, Ph.D. za cenné připomínky a rady, které mi poskytl při vypracování diplomové práce.

Obsah

Kapitola 1

Úvod

Po dlouhou dobu Moorův zákon úspěšně předpovídal růst počtu tranzistorů v procesoru. Podle tohoto zákona by se počet tranzistorů v procesoru měl zdvojnásobit zhruba každé dva roky. V posledních letech ovšem růst počtu tranzistorů značně zpomalil. Malá velikost tranzistorů již naráží na fyzikální limity. Výrobci počítačových čipů stále nachází nové způsoby, jak hustotu tranzistorů na čipu zvyšovat, ale dnešní trend již Moorovu zákonu neodpovídá. Zvyšování výkonu čipů lze ale dosáhnout i vylepšováním jejich architektury.

V dnešní době jsou obvody značně komplexní a jejich návrh je složitý. Konvenční způsob navrhování číslicových obvodů je značně omezen tím, že návrhář musí danému obvodu či jeho abstrakci sám rozumět. Obvody navržené lidmi jsou přehledné a logicky uspořádané, ale často nevyužívají plný potenciál dostupných hradel. Tento problém se snaží řešit evoluční algoritmy.

Existuje celá řada evolučních algoritmů. Pro všechny je ale společné to, že navrhují řešení, která se značně liší od těch navržených lidmi [9]. Lze jimi vytvářet číslicové obvody, samotné počítačové programy, obrazové filtry, antény i třeba umělecké obrazy. Je jen málo oblastí, ve kterých by se evoluční algoritmy dosud neuplatnily. Jejich síla je ve schopnosti dobře optimalizovat a nalézat řešení, která by konvenčním návrhem nešla nalézt.

Tato práce je zaměřena na hledání postupů, které by akcelerovaly symbolickou regresi v rámci kartézského genetického programování (CGP). CGP je druhem evolučního algoritmu a je typem genetického programování (GP). CGP na rozdíl od GP reprezentuje kandidátní řešení pomocí obecného orientovaného acyklického grafu. V této práci je CGP použito zejména v úloze symbolické regrese, přičemž symbolická regrese je zde chápána ve své obecné podobě. Dále se práce zaměřuje na architekturu SIMD a instrukční sadu SSE a AVX. S využitím zmíněných instrukčních sad lze dosáhnout urychlení symbolické regrese.

Technická zpráva je členěna do osmi kapitol. Kapitola 2 nabízí teoretický úvod do problematiky evolučních algoritmů. Představuje genetické programování a kartézské genetické programování. V kapitole 3 je představen problém segmentace krevního řečiště v obrázku sítnice lidského oka. Kapitola 4 se věnuje paralelnímu počítání. Je představeno programování s více vlákny a architektura SIMD umožňující vektorizaci. V 5. kapitole jsou představeny různé optimalizace akcelerující kartézské genetické programování. V této kapitole jsou navrženy metody jako predikce fitness, paralelizace vyhodnocování jedinců v populaci, paralelní simulace a akcelerace za použití strojového kódu. V kapitole 6 je ukázáno, jakým způsobem byly různé metody implementovány. V Kapitole 7 jsou vyhodnoceny všechny implementované metody. Pro vyhodnocení jsou navrženy experimenty a představeny jejich výsledky. V závěrečné kapitole 8 je práce vyhodnocena.

Kapitola 2

Evoluční Algoritmy

Evoluční algoritmy (EA) je souhrnné pojmenování pro množinu prohledávacích algoritmů, jejichž vznik byl inspirován teorií biologické evoluce. Biologická evoluce je proces vývoje druhů živých organismů. Při evoluci se druhy adaptují na své prostředí, přičemž důvodem je přirozený výběr a náhodná mutace.

V polovině dvacátého století byla snaha využít různých biologických jevů při tvorbě algoritmů. Evoluční algoritmy se ukázaly jako dobrý nástroj pro řešení mnoha výpočetně těžkých problémů. Nezávisle na sobě vzniklo několik variant evolučních algoritmů a jejich členění se používá dodnes:

- Genetické algoritmy
- Genetické programování
- Evoluční strategie
- Evoluční programování

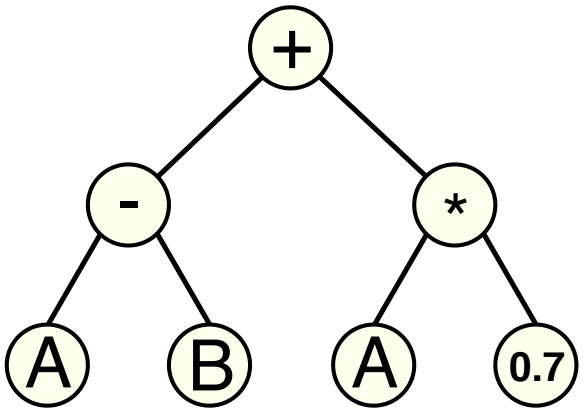
Tyto varianty se od sebe liší zejména ve způsobu přístupu ke kandidátním řešením a prohledávání stavového prostoru. Přes jejich odlišnosti existuje řada termínů, které je nutné znát pro pochopení problematiky této práce. Evoluční algoritmy v sobě zahrnují následující základní pojmy:

- **Fenotyp** či **jedinec** je kandidátním řešením daného problému, které je odvozeno z genotypu.
- **Genotyp** či **chromozom** je kód jednoho stavu uvnitř prohledávaného prostoru řešení. Z genotypu se po jeho interpretaci stává fenotyp.
- **Gen** je základní jednotkou chromozomu. Většinou jde o bit, nebo celé číslo. Konkrétní hodnota genu se nazývá **alela**.
- **Populace** je multimnožina obsahující chromozomy.
- **Fitness** je hodnota **fitness funkce** udávající schopnost daného jedince přežít.
- **Generace** je iterace evolučního algoritmu, při které dochází ke změnám populace.

2.1 Genetické programování

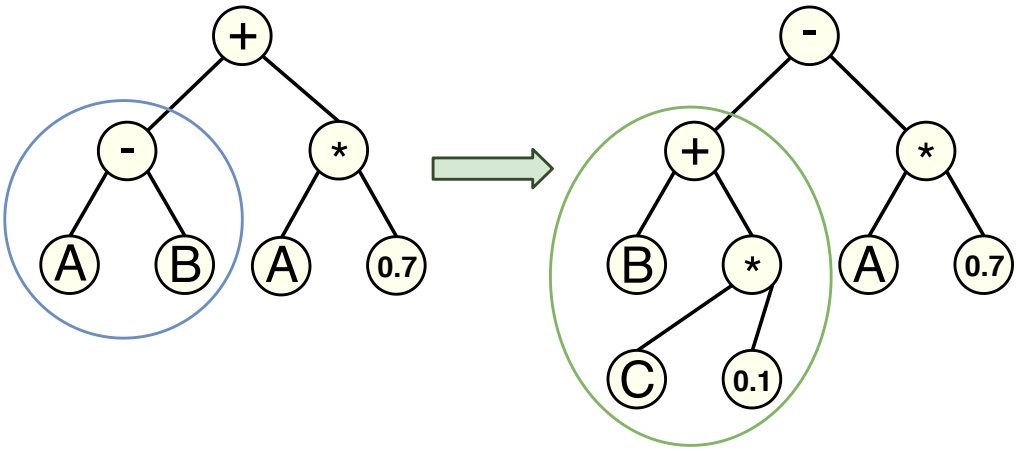
Genetické programování (GP) je jednou z variant evolučních algoritmů. Vzniklo na konci 80. let 20. století. Za jeho vznikem a popularizací stojí John Koza. Genetické programování se odlišuje od ostatních evolučních algoritmů tím, že vytváří přímo celé spustitelné programy. Tyto programy mohou nabývat libovolné délky. Jako první jazyk programu byl použit LISP, ale dnes se používají i ostatní programovací jazyky. Programy jsou často reprezentovány syntaktickými stromy 2.1, nebo souborem instrukcí. Fitness jedinců se zjišťuje provedením jejich kódu pro požadovanou množinu vstupů a porovnáním výstupů.

Velmi intuitivní je právě reprezentace programu pomocí syntaktického stromu. Strom se skládá z množiny terminálů (terminálních symbolů) a množiny funkcí (neterminálních symbolů). Za terminály považujeme vstupy programu, konstanty a funkce bez argumentů. Množina funkcí je volena uživatelem dle řešeného problému. V GP by všechny funkce měly být chráněné, tedy definované pro všechny možné hodnoty.



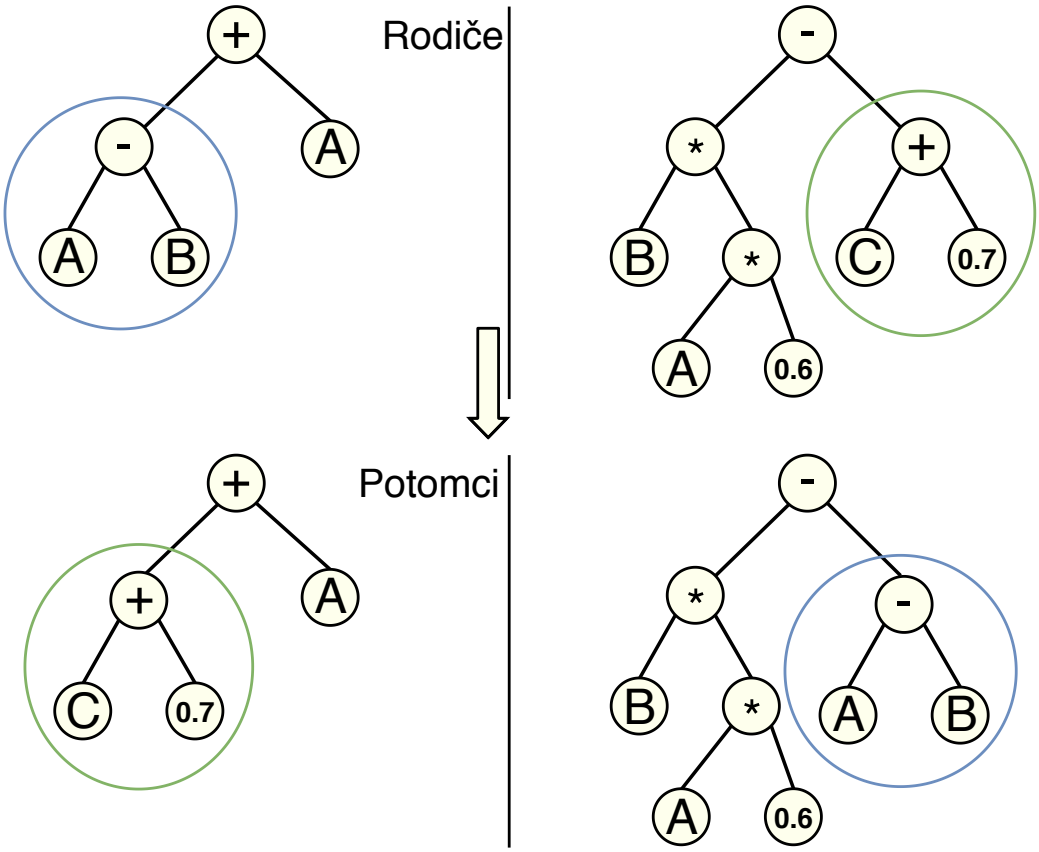
Obrázek 2.1: Program reprezentovaný syntaktickým stromem.

V průběhu evoluce se programy postupně mění. Ke změnám dochází aplikací operátorů křížení a mutace. Při implementaci GP lze použít i dalších pokročilých operátorů, které umožňují vytváření modulů a podprogramů. Ilustrace použití operátorů mutace a křížení je na obrázcích 2.2, 2.3.



Obrázek 2.2: Znázornění použití genetického operátoru mutace

Při mutaci se u jedince náhodně vybere podstrom, který se kompletně nahradí nově vygenerovaným podstromem. Nový podstrom může být menší i větší než původní podstrom. Při křížení se vymění dva náhodně vybrané podstromy dvou rodičů. Tím vzniknou dva noví jedinci, kteří jsou potomky těchto rodičů. Postupnými mutacemi a křížením může vzniknout problém zvaný "bloat", kdy má jedinec příliš velký chromozom. Tento problém lze omezit penalizací velikosti chromozomu (výšky stromu) uvnitř fitness funkce.

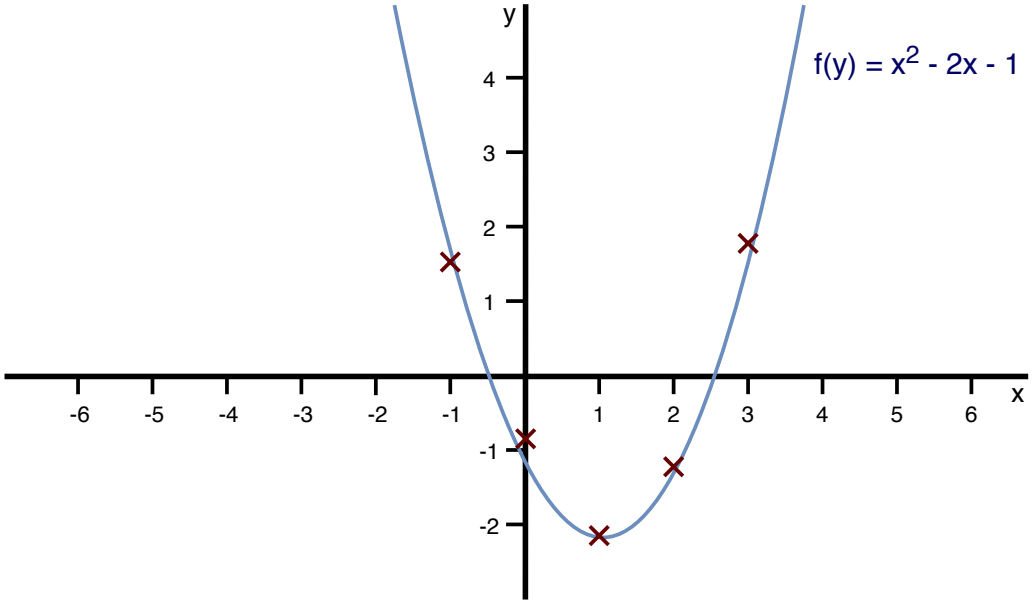


Obrázek 2.3: Znázornění použití genetických operátorů křížení

Jedním z ukázkových problémů, které se dají řešit pomocí genetického programování, je symbolická regrese. Jsou dány dvě datové množiny a úkolem je nalézt funkci (program), která by nejlépe namapovala korespondující hodnoty těchto množin. Příklad symbolické regrese lze vidět na obrázku 2.4. Fitness funkcí může být například součet absolutních hodnot rozdílů vypočítaných a očekávaných výstupů.

2.2 Kartézské genetické programování

Kartézské genetické programování (CGP) je variantou genetického programování [12]. CGP na rozdíl od GP reprezentuje řešení problému obecným orientovaným acyklickým grafem. GP reprezentuje problém pomocí syntaktického stromu. Název obsahuje slovo "Kartézské", neboť program je reprezentován dvourozměrnou mřížkou obsahující programovatelné elementy – uzly. Dalším rozdílem CGP a GP je to, že CGP chromozom má omezenou délku. Ta je omezena velikostí mřížky. CGP tedy netrpí problémem bloat, tak jako tomu je u GP.



Obrázek 2.4: Ukázka problému symbolické regrese na hledání funkce (*modrá*) procházející body (*červená*). Nalezená funkce je v tomto případě pouze aproximací.

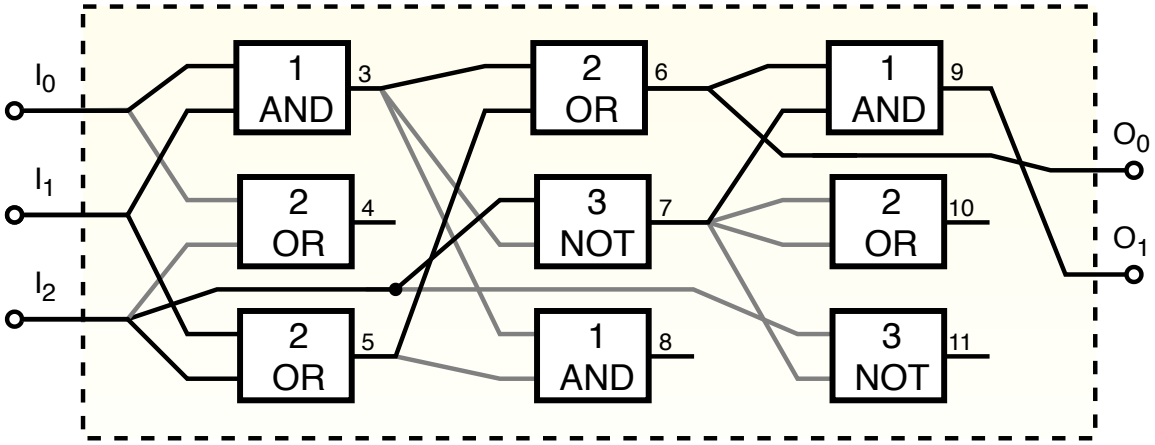
2.2.1 Definice CGP

CGP reprezentuje problém pomocí grafu, jenž je zakódován řetězcem celých čísel. Tento řetězec uchovává informaci o propojení všech uzlů, jejich funkcí a zapojení výstupů. Tento řetězec je chromozomem v CGP. Mřížka programovatelných elementů má předem danou velikost $n_c \times n_r$, kde n_c značí počet sloupců mřížky a n_r značí počet řádků mřížky. Celkový počet vstupů je n_i a celkový počet výstupů je n_o. Uzly mohou provádět jednu z funkcí z množiny funkcí Γ. Všechny uzly mají předem určený počet vstupů n_a. Ten je volen jako hodnota maximální arity ze všech funkcí v Γ. Některé funkce tedy nemusí vždy potřebovat všechny vstupy uzlu. Obvyklým příkladem je funkce NOT, která používá pouze jeden vstup.

Uzly mohou připojit své vstupy na výstupy uzlů předchozích. Připojení uzlů ovlivňuje parametr l-back. Ten udává maximální počet předchozích sloupců od určitého uzlu, ke kterým se tento uzel může připojit. Daný uzel se tedy může připojit pouze na výstupy uzlů, jež jsou až v l-back sloupcích před ním. Nehledě na parametr l-back se uzly vždy mohou připojit na vstupy i_0 až i_{n_i-1}. Pro l-back=1 je tedy propojitelnost maximálně omezena a propojit se mohou pouze uzly sousedních sloupců. Pokud je l-back=*max*, tak není žádné omezení na propojení uzlů ve sloupcích.

Chromozom lze zapsat jako posloupnost celých čísel, kdy postupně procházíme uzly po sloupcích a zapisujeme si jednotlivá připojení. Předtím než přejdeme na další uzel, zapíšeme číslo funkce daného uzlu. Například pokud mají uzly dva vstupy, vznikne nám posloupnost trojic čísel. Na konec posloupnosti přidáme zapojení celkových výstupů o_0 až o_{n_o-1}.

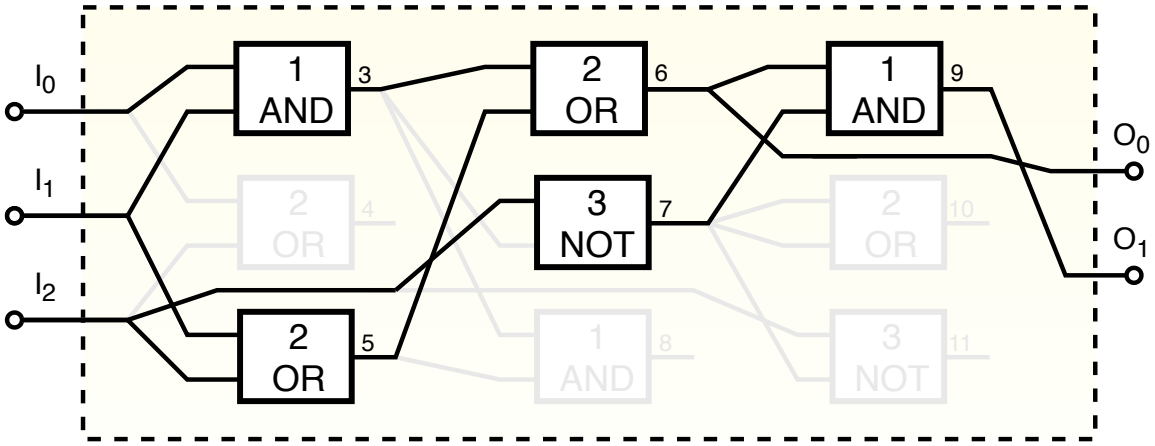
Jako příklad nám poslouží genotyp na obrázku 2.5. Uvedený chromozom se zapisuje: 0,1,1; 0,2,2; 1,2,2; 3,5,2; 2,3,3; 3,5,1; 6,7,1; 7,7,2; 7,2,3; 6,9. Toto kandidátní řešení vzniklo s použitím následujících parametrů: $n_c = 3$, $n_r = 3$, $n_i = 3$, $n_o = 2$, l-back $= 1$, $\Gamma = \{$XOR(0), AND(1), OR(2), NOT(3)$\}$.



Obrázek 2.5: Příklad fenotypu CGP. Chromozom: 0,1,1; 0,2,2; 1,2,2; 3,5,2; 2,3,3; 3,5,1; 6,7,1; 7,7,2; 7,2,3; 6,9

2.2.2 Převod genotypu na fenotyp u CGP

Na základě propojení se může stát, že některé uzly nemají žádný vliv na výstupy. Tyto uzly se nazývají nekódující. Na obrázku je lze poznat tak, že z nich nevede žádné propojení. Pokud bychom takovéto kandidátní řešení vyhodnocovali, bylo by zbytečné počítat výstupy těchto uzlů. Pro převod genotypu na fenotyp je tedy vhodné použít algoritmus, který tyto uzly vynechá. Příklad fenotypu lze vidět na obrázku 2.6.



Obrázek 2.6: V procesu vytváření fenotypu z genotypu jsou vyřazeny uzly, které nemají žádný vliv na výstupy.

Algoritmus pro odstranění nekódujících uzlů je jednoduchý. Na začátku označí všechny uzly za nekódující. Projde výstupy a všechny uzly, které jsou na ně připojené, označí jako kódující. Postupně prochází odzadu množinu kódujících uzlů a dle arity jejich funkcí označuje připojené uzly za kódující. Jakmile algoritmus nemá další kódující uzly, tak končí.

Z výše uvedeného plyne, že CGP genotyp má neměnnou velikost, zatímco CGP fenotyp může mít jakoukoli velikost od nuly až po velikost genotypu. Je důležité dodat, že určitý počet nekódujících uzlů je při evoluci prospěšný. Chromozomům s nekódujícími geny je umožněn takzvaný genetický drift [10]. Takto je označen proces změn genotypu, jenž vede na

stejný fenotyp. S tímto procesem také souvisejí neutrální mutace, tedy mutace, jež nevedou ke zhoršení fitness daného jedince. V CGP je podstatné neutrální mutace akceptovat a nahrazovat rodiče potomky se stejnou fitness.

2.2.3 Průběh evoluce u CGP

Tato podkapitola se bude věnovat samotné evoluci u CGP. Předpokládejme, že uživatel nastavil všechny parametry CGP a vybral vstupní i výstupní trénovací data.

Na počátku se inicializují jedinci celé populace. Alely genů jsou značně omezené. Funkční geny musí odkazovat na platné funkce z Γ. Propojovací geny musí odkazovat na výstupy uzlů z předchozích sloupců, nebo na celkové vstupy. Tato omezení je nutné dodržet i při mutacích genů. Na konci inicializace je počáteční populace tvořena náhodnými jedinci.

Dalším krokem je provedení ohodnocení populace. Jedince v populaci lze ohodnotit metodou simulace. Pro všechna vstupní trénovací data jsou porovnány odsimulované výsledky kandidátních řešení a je vypočtena fitness jedinců pomocí uživatelem stanovené fitness funkce. Vzhledem k frekvenci výpočtu fitness funkce je žádoucí, aby tato funkce byla jednoduše vypočítatelná. Většinou postačí spočítat množství bitů, ve kterých se výstupy liší, nebo použít sumu absolutních vzdáleností. Mimo simulaci existují i další metody vyhodnocení fitness populace. Jednou z těchto metod je například použití algoritmů řešících problém splnitelnosti (SAT) [15].

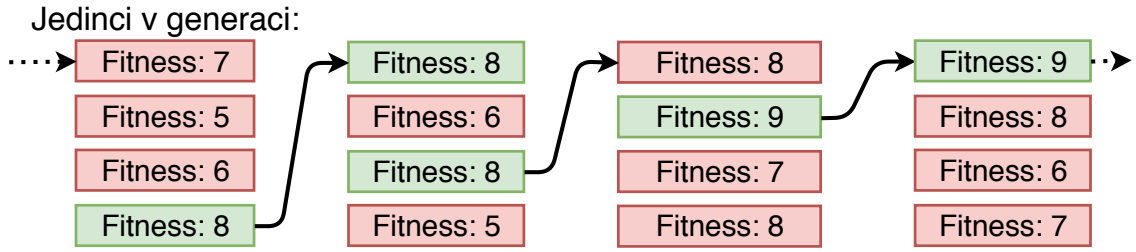
Moderní SAT algoritmy jsou vhodné pro řešení velkých instancí problémů, ve kterých jsou použity logické funkce. Pomocí SAT algoritmu lze rychle rozhodnout, zda je kandidátní obvod funkčně ekvivalentní s referenčním obvodem. K tomuto vyhodnocení stačí spojit výstupy kandidátního a referenčního obvodu a vytvořit tím nový obvod. Následně je tento obvod převeden do konjunktivní normální formy, která zachycuje správnost mapování obou původních obvodů. SAT algoritmus vyhodnotí, zda je daná CNF formule splněna, čímž také určí, zda jsou si obvody funkčně ekvivalentní. Při vyhodnocování kandidátních řešení lze také zkombinovat metodu SAT s metodou simulační [21].

Běh evoluce je vhodné podmínit určitým ukončovacím kritériem. Jako vhodné kritérium se obvykle volí dosažení určité fitness nejlepším jedincem v populaci. Proces evoluce je náhodný, a proto by se mohlo stát, že algoritmus nikdy neskončí. Dalším kritériem by tedy mohl být maximální počet generací, které algoritmus před ukončením vytvoří.

Nedílnou součástí každého EA je proces selekce a reprodukce rodičů do nové generace. V CGP se prakticky nepoužívá operátor křížení, neboť při křížení dochází k příliš výrazným změnám. Používá se ale operátor bodové mutace. Počet mutovaných genů se řídí mírou mutace, která se udává v procentech [11]. Například pro 100 genů a míru mutace 2% budou mutovány dva geny. Pro různé problémy jsou vhodné různé míry mutací, ale většinou se tato hodnota stanovuje v řádu jednotek procent. Mutují se funkční i propojovací geny, a to tak, aby nové hodnoty vyhovovaly omezením.

Evoluční strategie v CGP se označuje jako $(1 + \lambda)$. Populaci tvoří $(1 + \lambda)$ jedinců a nová populace vzniká výběrem nejlepšího jedince a jeho dosazením do nové populace. Tento jedinec se stává rodičem a zbytek populace je vytvořen jeho mutováním. V případě, že existuje více jedinců s nejlepší fitness, vybere se ten, který nebyl rodičem v předešlé generaci. Příklad evoluční strategie $(1 + 3)$ je na obrázku 2.7.

Takto vzniklá populace je další generací. Evoluce následně iterativně opakuje zmíněné kroky až do naplnění kritéria. Fitness hodnota nejlepších jedinců nebude nikdy klesat.

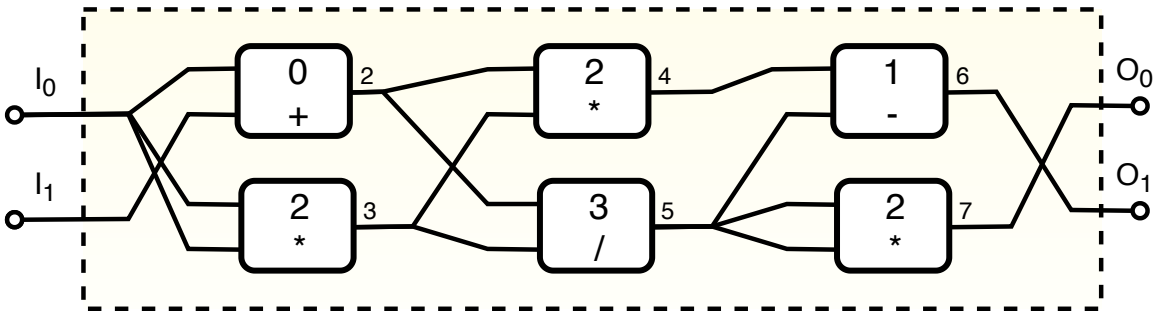


Obrázek 2.7: Ukázka výběru nejlepších jedinců v populaci při evoluční strategii $(1 + 3)$. Zeleně jsou označeni jedinci s nejlepší fitness v dané generaci. Výběr jedince do nové generace je znázorněn šipkou.

2.2.4 Použití CGP

Kartézské genetické programování se běžně používá pro evoluci číslicových obvodů. CGP dokáže navrhovat kompaktní a efektivní řešení [17]. Změnou fitness funkce se mění optimalizované kritérium. Lze tak dosáhnout minimalizace počtu hradel v obvodu, minimalizace zpoždění obvodu, nebo minimalizace jeho příkonu. Navržené obvody pak mohou být vůči zmíněným kritériím pareto-optimální.

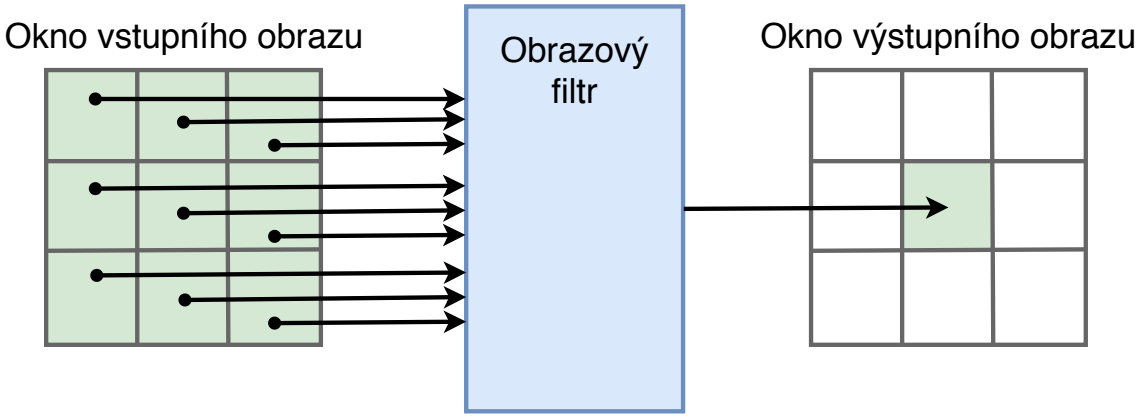
Přestože bylo CGP původně navrženo pro evoluci hradel, lze je použít pro evoluci různých výpočetních struktur. Pro změnu použití stačí změnit množinu funkcí Γ. Například pro $\Gamma = \{\text{plus}(0), \text{mínus}(1), \text{krát}(2), \text{děleno}(3)\}$ bude evolučně vytvářena matematická funkce, která nejlépe mapuje vstupy na výstupy. Pro více výstupů lze systém chápat jako soubor více rovnic. Příklad takové funkce v CGP je na obrázku 2.8.



Obrázek 2.8: CGP genotyp tvořící dvě matematické rovnice. $\Gamma = \{\text{plus}(0), \text{mínus}(1), \text{krát}(2), \text{děleno}(3)\}$. Chromozom: 0,1,0; 0,0,2; 2,3,2; 2,3,3; 4,5,1; 5,5,2; 7,6

Dalším problémem, který CGP úspěšně řeší, je evoluce obrazových filtrů [17]. Jako příklad lze uvést filtry na odstranění šumu, dilataci, erozi, rozmazání a detekci hran. Takovéto filtry používají klouzavé okno, jež je přikládáno na pixely vstupního obrázku. Filtry jsou pak funkce, které mají na vstupu 8-bitové hodnoty vstupních pixelů a jeden 8-bitový výstup. Koncept je ilustrován na obrázku 2.9. Výstupní hodnotu považujeme za hodnotu pixelu korespondujícím s pozicí středu okna ve vstupním obrázku. Postupným přikládáním okna na všechny pozice v obrázku vzniká výstupní obrázek.

Fitness funkce se počítá jako součet všech hodnot v rozdílovém obrazu. Porovnává se obraz vytvořený kandidátním filtrem s očekávaným obrazem. Účelem je tento rozdíl minimalizovat, a tím snížit chybu filtru. Je třeba vhodně zvolit trénovací data. Nesmí jich být



Obrázek 2.9: Příklad obrazového filtru využívajícího okno o velikosti 3×3.

kód	funkce	kód	funkce
0	255	8	x » 1
1	x	9	x » 2
2	255 − x	10	SwapNibbles(x, y)
3	x OR y	11	x + y
4	!x OR y	12	x − y
5	x AND y	13	Average(x, y)
6	x NAND y	14	Maximum(x, y)
7	x XOR y	15	Minimum(x, y)

Tabulka 2.1: Množina možných funkcí použitých v obrazových filtrech.

příliš mnoho, protože by evoluce byla pomalá. Nesmí jich být ani příliš málo, protože by výsledný filtr špatně generalizoval.

Jako množinu funkcí Γ můžeme použít například funkce z tabulky 2.1 [17]. Tyto funkce byly vybrány tak, aby se daly rychle vypočítat. Při evoluci dochází k neustálému vyhodnocování těchto funkcí, proto je vhodné do Γ nevkládat složité funkce, jako jsou násobení či dělení. Funkce, které tabulka obsahuje, byli uzpůsobeny tak, aby je bylo možné jednoduše generovat v nativním kódu pomocí instrukcí procesoru.

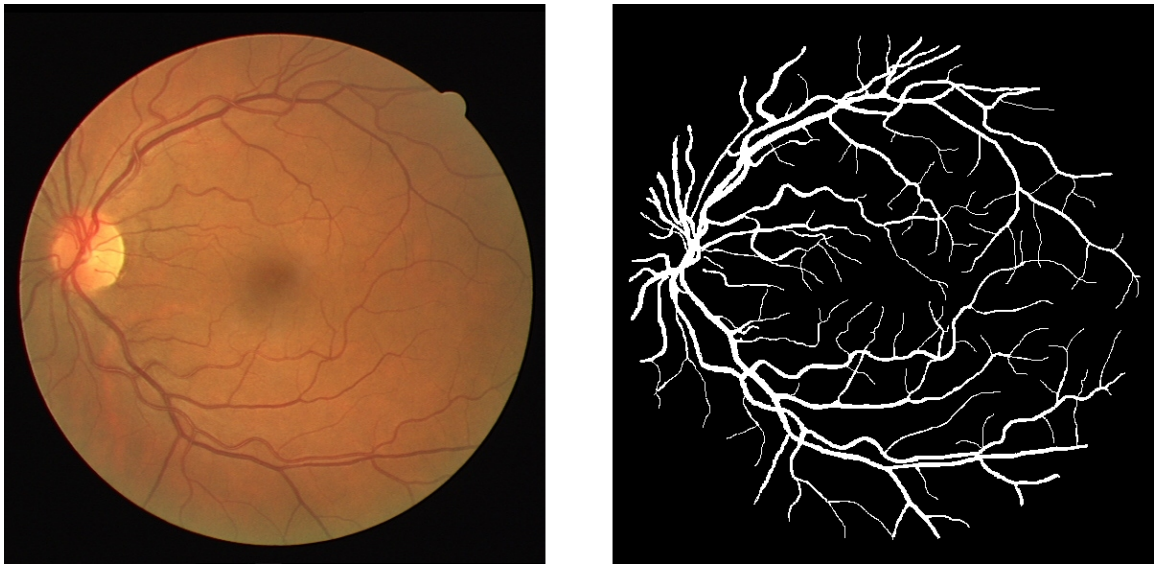
Zajímavým příkladem využití CGP je generování uměleckých obrazů [1]. Na vstup genotypu jsou přivedeny hodnoty souřadnic x a y pro konkrétní pixel. Genotyp má tři výstupy odpovídající barevným složkám RGB. Výstupy mohou nabývat pouze hodnot 0 až 255. Aby bylo rozmezí hodnot dodrženo i po aplikaci jednotlivých funkčních prvků, je potřeba dobře vybrat množinu funkcí Γ. Lze použít funkce jako sinus a cosinus, ale jejich výstupy jsou v rozmezí -1 a 1. Místo čisté funkce sinus lze tedy například použít funkci $abs(sin(input_1) * 255)$.

Evaluace kandidátních řešení je obtížná. V ideálním případě by estetičnost obrázků vyhodnocoval člověk, ale to je z časových důvodů nemožné. Existují však různé metody, které se snaží estetičnost obrázků vyhodnotit automaticky. Příkladem je použití Houghovy transformace pro hledání útvarů v obrázku. Předpokladem je, že obrázky obsahující určitý počet a rozvržení objektů jsou estetické.

Kapitola 3

Segmentace krevního řečiště ve snímku sítnice oka

Jednou z vybraných úloh, jež je v práci implementována, je automatický návrh algoritmu pro vyznačení krevního řečiště v obrázku sítnice oka pomocí kartézského genetického programování (CGP). Tuto úlohu lze zařadit do skupiny úloh, které řeší evoluční návrh nelineárních obrazových filtrů. Zástupcem úloh v této skupině je například často používaná detekce hran v obraze [3]. Algoritmy pro vyznačení krevního řečiště jsou ale složitější, neboť často samy obsahují fázi detekce hran a k tomu ještě další fáze zpracování obrazu [18]. Složitost segmentační úlohy dokládá také postupný vývoj různých metod [25]. Výsledkem segmentace je obrázek, ve kterém jsou pixely odpovídající cévám vyznačeny bíle a pixely pozadí černě. Příklad fotografie sítnice s odpovídající vyznačenou segmentací je na obrázku 3.1.



Obrázek 3.1: Vlevo obrázek sítnice oka. Vpravo obrázek manuálně provedené segmentace. (součástí databáze DRIVE [13])

Automatická segmentace cévního řečiště na snímcích sítnice je důležitá ze dvou důvodů. Prvním důvodem je její použití v medicíně. Různá kardiovaskulární onemocnění lze detekovat právě pohledem na sítnici lidského oka. Obrázky sítnic jsou využívány oftalmology, kteří používají automatickou segmentaci sítnic k tomu, aby jim usnadnila diagnostiku. Cí-

lem segmentace je přesně vyznačit krevní řečiště. Pokud je krevní řečiště vyznačeno, lze je z obrázku odstranit a tím zjednodušit náhled na viditelné vrstvy sítnice. Samotnou stromovou strukturu cév lze analyzovat a tím nalézat abnormality a kontrolovat průběh různých onemocnění [4].

Další využití nachází automatická segmentace krevního řečiště v biometrii [8]. Cévní systém v oku je unikátní pro každého jedince a s časem se téměř nemění. Lze jej proto použít pro autentizaci osob. Ve stromové struktuře cév sítnice se vyskytují místa, která lze snadno identifikovat. V biometrii jsou tato místa označována jako markanty. Mezi markanty patří například místa větvení cév, křížení cév či konce cév. Pro vyznačení těchto markantů je zapotřebí nejdříve znát strukturu cév, což umožňuje právě automatická segmentace krevního řečiště.

3.1 Databáze DRIVE

Všechny snímky sítnic obsažené v této práci pocházejí z databáze DRIVE (Digital Retinal Images for Vessel Extraction) [13]. Databáze DRIVE vznikla roku 2004 za účelem poskytnutí sady obrázků, které by usnadnily porovnávání výsledků mezi výzkumníky. Databáze obsahuje celkově 40 fotografií sítnic. Fotografie byly vytvořeny v projektu kontroly pacientů s diabetem a náhodně vybrány z větší sady. Sedm obrázků vykazuje známky mírné diabetické retinopatie.

Fotografie mají rozměr 565 na 584 pixelů. Ke všem fotografiím sítnic existují také dvě různé manuální segmentace provedené odborníky a maska určující oblast zájmu (ROI). Manuální segmentace slouží jako zlaté standardy pro úlohu segmentace. Sada obrázků je rozdělena na dvě části. Dvacet obrázků je trénovacích a dvacet testovacích. Příklad fotografie sítnice a odpovídající manuální segmentace z databáze DRIVE je na obrázku 3.1.

3.2 Postupy při provedení segmentace

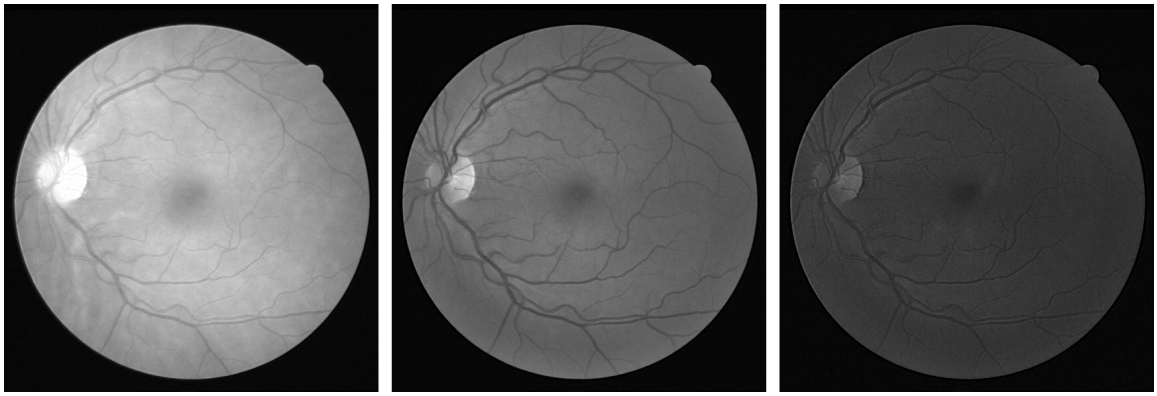
Množství metod bylo historicky aplikováno na problém segmentace krevního řečiště v obrázku sítnice oka [16]. Mezi tyto metody patří například: rozpoznávání vzorů pomocí učení s učitelem, souhlasné filtrování (matched filtering), detekce přímek, tzv. scale-space analýza, morfologické zpracování obrazu, nebo hybridní metody kombinující více přístupů.

Vyznačení krevního řečiště pomocí CGP je metodou rozpoznávání vzorů, která využívá trénovací fázi učení s učitelem. Do stejné kategorie se řadí také neuronové sítě, ale lze použít i jiné klasifikátory. Metody využívající učení s učitelem potřebují trénovací vzory, které jsou předem označené. Jako zdroj trénovacích a testovacích dat lze použít například databáze DRIVE[1] či STARE[2], které byly za tímto účelem vytvořeny.

Předtím než je možné obrázek segmentovat, je důležité provést fázi předzpracování obrazu. Obrázky sítnic jsou v databázích barevné, ale při zpracování se často používá pouze jejich zelený barevný kanál [16]. Je to z důvodu, že právě zelený kanál nejlépe kontrastuje mezi pozadím a cévním systémem. Kontrast cév a pozadí u jednotlivých barevných kanálů je vidět na obrázku 3.2, který ukazuje jednotlivé složky RGB samostatně.

Po extrakci zelené složky je vhodné v obrázcích srovnat lokální světlosti. Na obrázky je aplikován kruhový mediánový filtr, kterým je získána úroveň šedi. Takto získaný obrázek

[1] https://www.isi.uu.nl/Research/Databases/DRIVE/
[2] http://cecas.clemson.edu/~ahoover/stare/



Obrázek 3.2: Rozklad obrázku sítnice oka na jednotlivé barevné kanály – složky RGB. Červená složka je vlevo, zelená uprostřed a modrá vpravo.

lokálních úrovní šedi pozadí je odečten od originálního obrázku. Následně je třeba hodnoty pixelů ustřednit a normalizovat do rozsahu vhodného pro další zpracování.

3.3 Vyhodnocení algoritmů v úloze segmentace

Úspěšnost segmentace se hodnotí podle těchto čtyř ukazatelů:

- Korektně detekované pixely cévy – **TP** (True Positive)
- Nesprávně detekované pixely cévy – **FP** (False Positive)
- Korektně detekované pixely pozadí – **TN** (True Negative)
- Nesprávně detekované pixely pozadí – **FN** (False Negative)

Cílem trénování segmentačního algoritmu je maximalizovat TP a TN. Poměr žádaných výsledků vůči všem pixelům zachycuje veličina přesnost (accuracy).

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{3.1}$$

Pro vyhodnocení fungování segmentačního algoritmu se dále používají preciznost (precision), citlivost (sensitivity) a specifičnost (specificity):

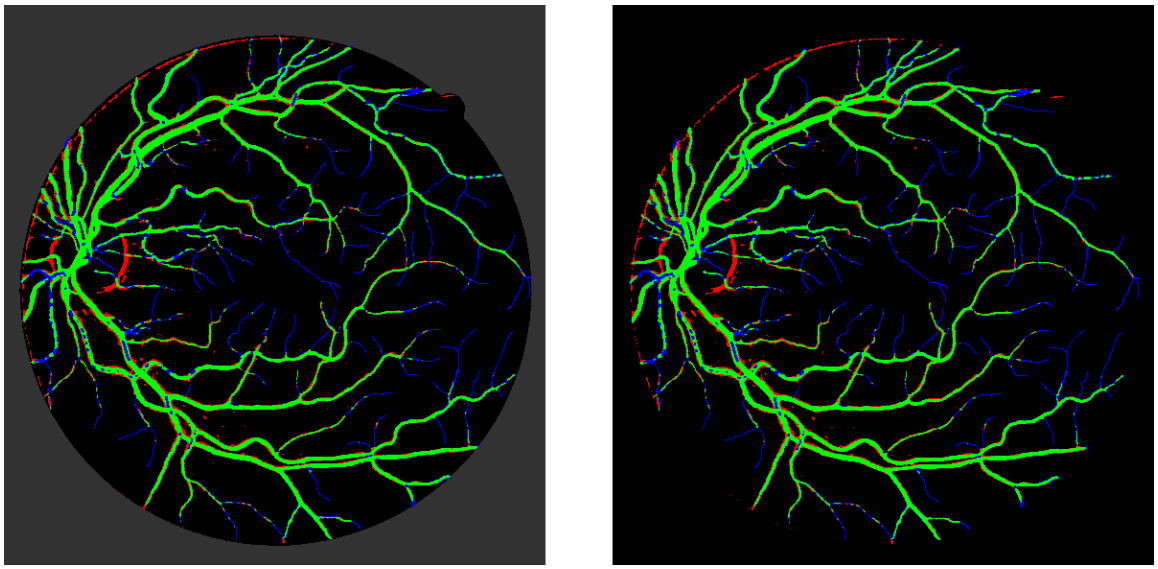
$$Precision = \frac{TP}{(TP + FP)} \tag{3.2}$$

$$Sensitivity = \frac{TP}{(TP + FN)} \tag{3.3}$$

$$Specificity = \frac{TN}{(TN + FP)} \tag{3.4}$$

Mimo použití těchto veličin lze fungování algoritmu ilustrovat obrázkem, ve kterém jsou barevně indikovány pixely TP, FP, TN a FN. Příkladem je obrázek 3.3, který přesně ukazuje vztah mezi manuálně provedenou segmentací a segmentací provedenou algoritmem. Je důležité upozornit na fakt, že existují dva způsoby vyhodnocení dosažených výsledků [26].

Tyto dva způsoby se odlišují v započítání masky do správně detekovaných pixelů. Maska je také často označována jako oblast zájmu (ROI). Odlišnost v náhledu na výpočet výsledků je též ukázána na obrázku 3.3. Pokud jsou pixely masky vnímány jako korektně detekované, je hodnota přesnosti 0,9406 pro daný snímek a algoritmus. Pokud jsou ale pixely masky ignorovány, vzroste hodnota přesnosti na 0,9598. Při srovnávání výsledků algoritmů z různých publikací je důležité tento rozdíl vnímat.



Obrázek 3.3: Obrázky ilustrující zobrazení přesnosti segmentačních metod. Barevné schéma označuje: korektně detekované pixely cév zeleně (TP), nesprávně detekované pixely cév červeně (TP), korektně detekované pixely pozadí černě (TN) a nesprávně detekované pixely pozadí modře (FN). Levý obrázek využívá oblast zájmu (ROI), která je dána maskou snímku. Pravý obrázek oblast zájmu nevyužívá a obsahuje proto mnohem více černých pixelů s korektně detekovaným pozadím.

Kapitola 4

Paralelní počítání na moderních CPU

Paralelní výpočty jsou dnes velmi důležitou součástí vytváření programů. Historicky se neustále zvyšovala rychlost procesorů, a proto nebyl kladen velký důraz na vytváření paralelních programů. Tento trend se ale v posledních zhruba patnácti letech obrátil a moderní procesory mají často nižší frekvenci, ale zato obsahují více jader schopných paralelně provádět nezávislé výpočty. Použitím paralelizace tedy odemykáme výpočetní potenciál, který se skrývá v našich procesorech.

Další možností, jak dosáhnout paralelizace, je spojení několika počítačů do výpočetních clusterů. Tímto způsobem lze levně zvýšit dostupný výpočetní výkon. Podobným způsobem vznikají i superpočítače. V clusterech je důležité správně implementovat komunikaci mezi obsaženými výpočetními systémy.

Paralelní systémy lze klasifikovat dle Flynnovy taxonomie. Ta dělí systémy dle počtu proudů instrukcí a toku dat. Systémy lze klasifikovat do čtyř základních skupin:

- **SISD** (Single Instruction Single Data) je klasický starý sekvenční počítač.
- **SIMD** (Single Instruction Multiple Data) jsou vektorové (SSE, AVX) a maticové procesory.
- **MIMD** (Multiple Instruction Multiple Data) jsou víceprocesorové/vícejaderné počítače.
- **MISD** (Multiple Instruction Single Data) jsou systémy uzpůsobené na opravy chyb.

Vzhledem k tématu této práce bude v dalších podkapitolách věnována pozornost zejména paralelizaci pomocí vláken a paralelizaci pomocí vektorizace.

4.1 Programování s více vlákny

Vytváření programů využívajících více vláken je velmi komplikované. Pro dosažení co největšího výkonu je důležité znát hardware, na kterém bude software spuštěn. Algoritmy, se kterými se setkáváme, mají často sekvenční formu a je na programátorovi, aby daný algoritmus paralelizoval. Jsou dvě hlavní možnosti, jak paralelizace dosáhnout.

První možností je přímo navrhnout rozdělení úlohy mezi procesy, jejich synchronizaci a komunikaci mezi nimi [14]. Za tímto účelem existuje knihovna a specifikace Message Passing

Interface (MPI). Použití samotné MPI je složité, a proto existují knihovny, jako je například Open MPI[1], které jsou mnohem jednodušší na použití.

Knihovna Open MPI je založena na modelu distribuované paměti. V tomto modelu je komunikace mezi procesy vedena zasíláním zpráv. Systém zasílání zpráv je vhodné použít například při výpočtech v clusteru, kde jednotlivé procesy běží samostatně na různých fyzických zařízeních.

Druhou možností je použití knihovny, která se o paralelizaci z velké části sama postará. Příkladem je knihovna a standard OpenMP[2]. Tato knihovna pracuje nad sdílenou pamětí [2], což značí, že jednotlivá vlákna mají přístup do všech míst paměti a primárně spolu komunikují právě pomocí společné paměti. Knihovnu lze využít v programovacích jazycích Fortran, C a C++. Knihovna OpenMP obsahuje direktivy pro překladač, kterými programátor označí oblasti kódu, jež se mají vykonávat paralelně. Pokud překladač danou direktivu nerozezná, jednoduše ji ignoruje a vykoná program sekvenčně.

Příklady direktiv OpenMP jsou direktivy *"parallel"* a *"for"*. V bloku za direktivou *"parallel"* je kód zduplikován a prováděn všemi vlákny. Vlákna jsou očíslována, a je tedy možné každému z nich přidělit určitou část kódu k provedení. Direktiva *"for"* rozdělí cyklus *for* mezi vlákna tím způsobem, aby na každé iteraci pracovalo právě jedno vlákno. Cykly, které nemají závislosti mezi iteracemi, lze tímto způsobem značně urychlit.

4.2 Architektura SIMD

Dnešní procesory umožňují provádět instrukce nad celými vektory dat. Tyto instrukce řadíme do kategorie SIMD (Single Instruction Multiple Data). Pokud je potřeba nad množstvím dat provést stejné operace, je vhodné tato data uspořádat do vektoru a použít právě instrukcí SIMD. Dosažené urychlení tak závisí na tom, kolik různých dat lze do vektoru paralelně naskládat.

4.2.1 Instrukční sada SSE

SSE (Streaming SIMD Extensions) [6] je rozšíření instrukční sady MMX (Multi Media Extension). Instrukční sada MMX byla vytvořena společností Intel a uvedena na trh roku 1997. Technologie MMX rozšířila cache a vytvořila nové 64bitové registry MM0 až MM7. Stejné registry byly ale použity i pro čísla s pohyblivou řádovou čárkou, a proto v původní MMX nešlo kombinovat operace celočíselné s operacemi floating point.

Instrukční sada SSE byla uvedena na trh roku 1999. Za jejím vznikem stojí opět společnost Intel. Postupně vznikaly nové revize SSE2, SSE3, SSE4, SSE4.1 a SSE4.2. Byly přidány stovky instrukcí a vzniklo šestnáct 128bitových registrů XMM0 až XMM15. Tyto registry jsou přístupné pouze v 64bitovém módu.

Silou této technologie je způsob, kterým se nahlíží na data v registrech. Různé instrukce nahlíží na uspořádání dat v registrech odlišně. Data lze reinterpretovat jako vektor 8bitových, 16bitových, 32bitových, 64bitových celých čísel, nebo jako vektor 32bitových a 64bitových čísel s pohyblivou řádovou čárkou. Pokud tedy například sečteme dva vektory 8bitových celých čísel, výsledky nikdy nepřesahují do okolních čísel. Tímto způsobem lze dosáhnout vektorové paralelizace.

[1] https://www.open-mpi.org/
[2] https://www.openmp.org/

4.2.2 Instrukční sada AVX

AVX (Advanced Vector Extensions) [6] je instrukční sada rozšiřující SSE. Tato technologie byla uvedena na trh roku 2011 společností Intel. Existují k ní dvě revize AVX2 a AVX-512.

AVX2 rozšiřuje šestnáct 128bitových registrů XMM na 256bitové registry YMM. Zavádí také řadu instrukcí pro práci s nimi. V AVX2 lze najednou pracovat například s 32 8bitovými celými čísly. Obdobně je tomu tak i pro 16bitová, 32bitová a 64bitová celá čísla, nebo pro čísla s plovoucí řádovou čárkou.

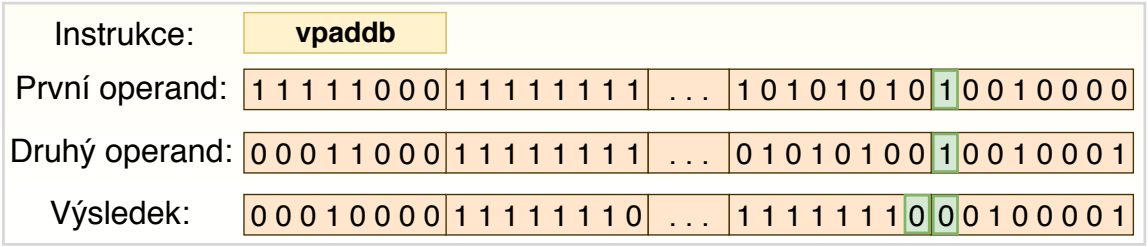
V tomto projektu je k AVX2 instrukcím přistupováno prostřednictvím strojového kódu. Jednotlivé instrukce mají přesně definovaný kód, kterým procesor danou instrukci identifikuje [7]. V kódu je také obsažena informace o použitých operandech. Instrukce tedy mohou přistupovat k paměti a k registrům.

Logické operace pracují s celým 256bitovým vektorem. Příklad takových operací je ukázán v tabulce na obrázku 4.1. Hexadecimální reprezentaci instrukcí je kód, který lze nahrát do paměti počítače a tím utvořit spustitelný kód. Zápis instrukcí v assembleru je nejčastěji používaná reprezentace instrukcí.

Hexadecimální kód instrukcí	Zápis instrukcí v assembleru
C5 FD 6F 04 25 30 12 BC 0A	**vmovdqa** ymm0, YMMWORD PTR ds:0xabc1230
C5 FD 6F 0C 25 30 12 BC 0A	**vmovdqa** ymm1, YMMWORD PTR ds:0xabc1230
C5 FD 7F 04 25 00 6F 30 00	**vmovdqa** YMMWORD PTR ds:0x306f00, ymm0
C5 FD DB 04 25 00 A4 31 00	**vpand** ymm0, ymm0, YMMWORD PTR ds:0x31a400
C5 ED EB 0C 25 00 BC 12 00	**vpor** ymm1,ymm2,YMMWORD PTR ds:0x12bc00

Obrázek 4.1: Ukázka AVX2 instrukcí pro manipulaci s pamětí a logické operace ve strojovém kódu a v assembleru. Instrukce vmovdqa přesouvá 256bitový vektor z pozice určené druhým operandem na pozici určenou prvním operandem. Obdobně instrukce vpand a vpor provedou logický AND resp. OR nad vektory určenými druhým a třetím operandem a výsledek uloží na pozici prvního operandu.

Aritmetické operace pracují s čísly, které jsou obsaženy ve 256bitovém vektoru. Příklad takových instrukcí je ukázán v tabulce na obrázku 4.3. Vybrané instrukce pracují s 8bitovými čísly. Těchto čísel je ve vektoru 32.



Obrázek 4.2: Obrázek znázorňuje práci s pamětí u instrukce *vpaddb*, která sčítá 8bitová celá čísla obsažená ve dvou 256bitových vektorech.

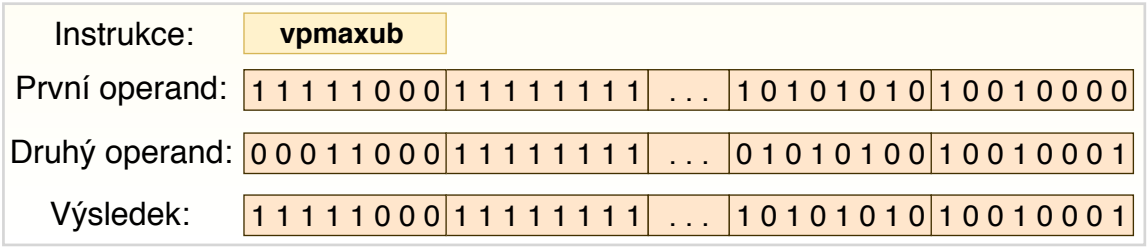
Jednotlivá 8bitová čísla jsou při operacích vzájemně chráněna. Tedy například při použití instrukce *vpaddb*, která sčítá dva 256bitové vektory 8bitových celých čísel, jsou jednotlivá

celá čísla chráněna před přetečením od ostatních 8bitových čísel. Jak konkrétně instrukce *vpaddb* zachází se vstupními vektory je přiblíženo na obrázku 4.2.

Hexadecimální kód instrukcí	Zápis instrukcí v assembleru
C5 FD FC 04 25 00 A4 31 00	**vpaddb** ymm0,ymm0,YMMWORD PTR ds:0x31a400
C5 FD F8 04 25 00 A4 31 00	**vpsubb** ymm0,ymm0,YMMWORD PTR ds:0x31a400
C5 7D E0 04 25 00 A4 31 00	**vpavgb** ymm8,ymm0,YMMWORD PTR ds:0x31a400
C5 BD DE 04 25 00 A4 31 00	**vpmaxub** ymm0,ymm8,YMMWORD PTR ds:0x31a400
C5 3D DA 04 25 00 A4 31 00	**vpminub** ymm8,ymm8,YMMWORD PTR ds:0x31a400

Obrázek 4.3: Ukázka AVX2 instrukcí pro aritmetické operace po osmibitových úsecích vektorů ve strojovém kódu a v assembleru. Instrukce vpaddb, vpsubb, vpavgb, vpmaxub a vpminub provedou příslušnou operaci nad registrem určeným druhým operandem a vektorem uloženým v paměti určeném třetím operandem. Výsledek je uložen do registru určeného prvním operandem.

Schopnost procesoru zpracovat odděleně jednotlivá celá čísla obsažená ve vektorech dobře demonstruje například instrukce *vpmaxub*. Instrukce porovnává navzájem celá čísla ze dvou vektorů a jako výsledek ukládá do registru procesoru vektor obsahující maxima. Výsledný vektor tedy obsahuje kombinaci 8bitových čísel, které byly v porovnání mezi operandy maximální. Příklad práce instrukce *vpmaxub* je ukázán na obrázku 4.4.



Obrázek 4.4: Obrázek znázorňuje práci s pamětí u instrukce *vpmaxub*, která vrací maxima pro 8bitová celá čísla obsažená ve dvou 256bitových vektorech.

Kapitola 5

Optimalizace akcelerující kartézské genetické programování

V předešlých kapitolách byl čtenář seznámen s kartézským genetickým programováním (CGP) a množstvím technologií, které lze použít při akceleraci programů. V této kapitole bude navržena řada vylepšení, jež by měly CGP výrazně urychlit.

CGP lze použít při řešení velkého množství problémů. Hlavním omezením jeho použití je časová složitost, která stoupá s komplexností řešeného problému. K tomu, aby se daly řešit i komplexní problémy, je potřeba CGP algoritmus optimalizovat a urychlit. Ve druhé kapitole byla již zmíněna jedna ze základních optimalizací CGP, a to odstranění nekódujících uzlů při převodu genotypu na fenotyp.

5.1 Predikce fitness

Aby bylo možné vypočítat fitness každého jedince, je třeba vypočítat výstupy daného jedince pro všechny vstupní hodnoty. Pokud například vytváříme obrazový filtr a trénujeme jej na obrázku 128×128, tak iterujeme 15876krát přes všechna data. Tolik iterací evoluci výrazně zpomaluje. Řešením je fitness daného jedince přibližně predikovat.

Každý jedinec je vytvořen mutací svého rodiče. Některé mutace způsobí, že se fitness jedince dramaticky sníží. Tyto mutace lze snadno detekovat, neboť fitness je nízká i pro malý počet vstupních dat. Lze předpokládat, že se fitness hodnota po několika stovkách iterací ustálí na určité hodnotě a poté už roste a klesá pouze málo.

Algoritmus pro predikci fitness pracuje následovně: v intervalech (například každých 1000 iterací) vypočítává fitness jedinců a ukládá si výsledky. Tyto výsledky jsou srovnávány s fitness jedinců (rodičů) z předchozích generací, které odpovídají danému intervalu. Je zaveden parametr η, který určuje maximální povolenou odchylku fitness v každém intervalu. Pokud pro určitý interval byla nejnižší hodnota fitness předchozích generací rodičů větší než η+fitness nového jedince, je evaluace nového jedince předčasně ukončena. Dalším parametrem algoritmu je počet uložených hodnot intervalových fitness předešlých generací. Pokud by byl tento parametr příliš malý, hrozí, že některá dobrá kandidátní řešení budou předčasně odstraněna. Pokud je evaluace jedince předčasně ukončena, místo něj se vygeneruje jedinec nový.

5.2 Paralelizace v populaci

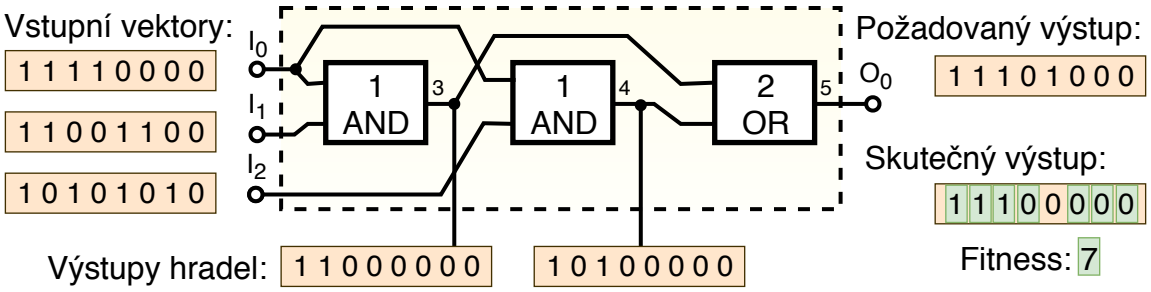
Při evaluaci každého jedince musí být vyhodnoceny jeho výstupy na sadě vstupů. Evaluace jednoho jedince nijak nezávisí na evaluaci jiného jedince. Tohoto faktu lze využít při paralelizaci.

Lze využít knihovny OpenMP a použít její direktivu *"parallel for"* k tomu, aby různé jedince v populaci vyhodnocovala různá vlákna. Přes jedince je iterováno cyklem *for*. Vlákna spolu nemusí až do plného vyhodnocení jedinců komunikovat, a proto lze předpokládat, že takto vytvořená paralelizace bude mít nízkou režii. Pokud je počet jedinců stejný jako počet jader procesoru, je velikost očekávaného zrychlení o něco nižší než je počet jader.

5.3 Paralelní simulace

Při vyhodnocování kandidátních řešení jsou na sadu vstupních hodnot postupně aplikovány různé operace (určené fenotypem daného jedince). Lze využít toho, že je tato sada operací stejná (jedinec se při vyhodnocování nemění) a data vektorizovat [23]. Většina dnes používaných počítačů již umožňuje využít instrukční sady AVX2, a proto bude právě s touto sadou počítáno.

Základní variantou paralelní simulace je bitově paralelní simulace. U této simulace se využívají vektory pro uložení všech kombinací hodnot bitů na vstupech fenotypu. Demonstrační příklad je uveden na obrázku 5.1. Zobrazené kandidátní řešení má tři vstupy. Pro zakódování všech vstupních kombinací je proto potřeba využít vektory o velikosti 2^3. S vektory lze dále pracovat stejně jako se samotnými bity. Simulace tedy může dané kandidátní řešení vyhodnotit jediným průchodem. Všechny postupně vypočtené výstupy uzlů jsou opět vektory a na primárním výstupu fenotypu je vytvořen vektor, který je odezvou fenotypu na vstupní vektory. Při výpočtu fitness pak často stačí skutečný výstupní vektor porovnat s požadovaným výstupním vektorem a spočítat množství stejných hodnot odpovídajících si bitů.



Obrázek 5.1: Příklad ilustrující použití datové vektorizace. Při paralelní simulaci jsou vstupní kombinace zakódovány do vstupních vektorů. Nad vektory jsou prováděny bitové operace, čímž je jedním průchodem kandidátního obvodu vyhodnoceno všech osm možných vstupních kombinací.

Abychom dosáhli maximálního zrychlení, je důležité zvolit množinu funkcí Γ tím způsobem, aby se jednotlivé funkce daly implementovat pomocí sekvence dostupných vektorových instrukcí. Pokud bude velikost dat 8 bitů, tak lze dosáhnout zrychlení 32× při použití 256bitových registrů. 8bitové hodnoty jsou časté například pro obrazové filtry.

5.4 Přímé použití strojového kódu

Při evaluaci kandidátního řešení je neustále nutné vyhodnocovat, kterou z funkcí z Γ daný uzel vykonává. K tomuto účelu se v kódu nachází *switch*, který na základě indexu funkce danou funkci provede. Jedná se tedy o interpretaci chromozomu, která musí být provedena pro každou zpracovávanou hodnotu. Tento krok lze ale obejít tím, že se vygeneruje strojový kód představující daného jedince[3]. Strojovým kódem se vytvoří spustitelná funkce, která v sobě obsahuje přesnou posloupnost všech uzlů jedince i s jeho funkcemi a závislostmi [24].

Vytvořený kód lze spouštět při každém vyhodnocení kandidátního řešení s tím, že se pouze mění vstupní hodnoty. Tímto krokem je evaluace kandidátních řešení zrychlena. Vyhodnocování kandidátních řešení je u CGP časově nejnáročnější, a proto je touto metodou výrazně urychlen běh celé evoluce.

5.5 Metoda dávkové mutace

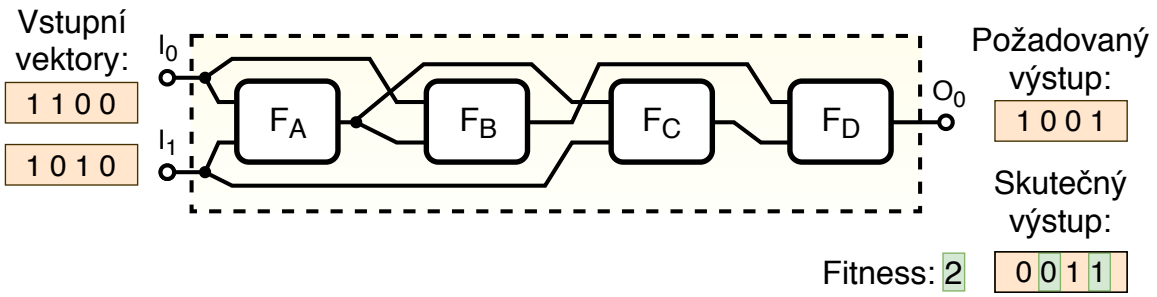
Tato kapitola je věnována návrhu metody dávkové mutace, což je nová metoda, která je vytvořena v této práci s cílem výrazně urychlit evoluční návrh logických obvodů pomocí CGP. Metodu lze teoreticky použít i při návrhu obrazových filtrů a dalších úloh, které nevyužívají pouze logické funkce, avšak její užitečnost výrazně upadá při použití jiných datových typů, než je boolean. Metoda je totiž založena na výpočtu odezev na všechny přípustné hodnoty jednoho vstupu náhodně vybraného uzlu. Pokud je kandidátní řešení složeno pouze z logických funkcí, mohou vstupy uzlů nabývat pouze dvou hodnot – 0 a 1. V takovém případě jsou nutné pouze dva průchody grafem kandidátního řešení pro výpočet potřebných odezev. Pokud by fenotyp pracoval nad datovými typy o velikosti jednoho bajtu, tak množství potřebných průchodů grafem vzroste na 255, a tím klesne efektivita této metody. Z tohoto důvodu bude metoda dávkové mutace dále vysvětlována pouze na úloze evolučního návrhu logických obvodů.

Obrázek 5.2 ukazuje příklad vyhodnocení fitness hodnoty jednoho kandidátního řešení. V příkladu je evolučně navrhován obvod, který je uživatelem definován pomocí tabulky požadovaných výstupů na konkrétní vstupy. Přestože není nezbytné definovat odezvu pro všechny kombinace vstupních hodnot, tak je v praxi evolučnímu algoritmu většinou poskytnuta tabulka všech možných vstupních kombinací a jim příslušejících požadovaných výstupních hodnot. Cílem evolučního algoritmu je najít mapování zmíněných tabulek za použití logických funkcí z množiny funkcí Γ. Pokud jsou výstupy plně definovány, je pro vyhodnocení každého kandidátního řešení potřeba simulovat obvod pro všech 2^{n_i} vstupních kombinací, kde n_i značí počet vstupů obvodu.

V příkladu na obrázku 5.2 jsou vstupní kombinace zakódovány ve vstupních vektorech. Obdobně požadovaný výstup je vektor, ve kterém jsou zakódovány odpovídající výstupní hodnoty, jichž se evoluce snaží dosáhnout. Skutečný výstup je odezva fenotypu na vstupní vektory. Ve stejném duchu budou dále i ostatní hodnoty označovány jako vektory. Porovnáním všech bitů požadovaného výstupu a skutečného výstupu se vypočítá fitness hodnota kandidátního řešení. Konkrétně fitness nabývá hodnoty rovné množství souhlasných bitů mezi oběma vektory. V obrázku jsou zelenou barvou označeny oba souhlasné bity a výsledná fitness je tedy rovna dvěma.

Použité parametry nejsou náhodné, ale byly zvoleny k demonstraci metody dávkové mutace. Přestože samotná metoda nevyžaduje konkrétní nastavení parametrů, je z imple-

[3] http://www.fit.vutbr.cz/ vasicek/cgp/?pg=accel



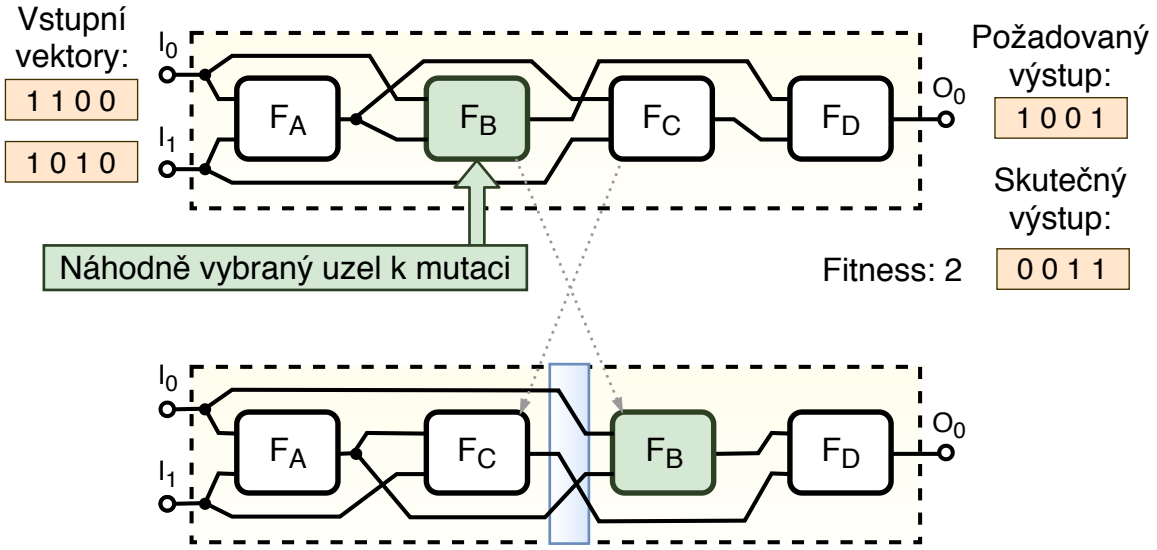
Obrázek 5.2: Reprezentace jednoduchého fenotypu CGP, na kterém bude postupně vysvětlena metoda dávkové mutace. Chromozom: 0,1,F_A; 0,2,F_B; 2,1,F_C; 3,4,F_D; 5. Obecné funkce bloků jsou označeny F_A - F_D. Parametry: $n_i = 2$, $n_0 = 1$, $n_r = 1$, $n_c = 4$, $n_a = 2$, l-back=max.

mentačního hlediska vhodné použít následující nastavení. Hodnota l-back by měla být na maximu, a tím dovolit plnou propojitelnost funkčních bloků fenotypu. Omezení propojitelnosti má u metody dávkové mutace přímý dopad na výkonnost a do značné míry i funkčnost této metody. Počet řádků je vhodné nastavit na jedna, neboť více řádků je implementačně složitější a nemá kladný dopad na výkon. Použití více řádků pro hardwarovou implementaci CGP je možné. Je vhodné zvolit relativně velký počet sloupců pole uzlů fenotypu. Počet sloupců v příkladu na obrázku 5.2 je malý pouze proto, aby bylo možné celé kandidátní řešení jednoduše zobrazit na obrázku. Ve skutečnosti může mít fenotyp řádově tisíce funkčních bloků a tedy i tisíce sloupců. Konkrétní optimální hodnotu n_c lze stanovit statistickým vyhodnocením pro danou evoluční úlohu. Počty primárních vstupů a výstupů mohou být z hlediska fungování metody libovolné. Stejně tak nezáleží na zvolené množině logických funkcí Γ.

Obdobně jako u základní implementace CGP je prvním krokem při použití metody dávkové mutace odstranění všech nekódujících uzlů z chromozomu. Tuto i další fáze lze provádět na úrovni genotypu, ale pro ilustraci bude používána reprezentace na úrovni fenotypu. V obrázku 5.2 jsou již všechny nekódující uzly odstraněny.

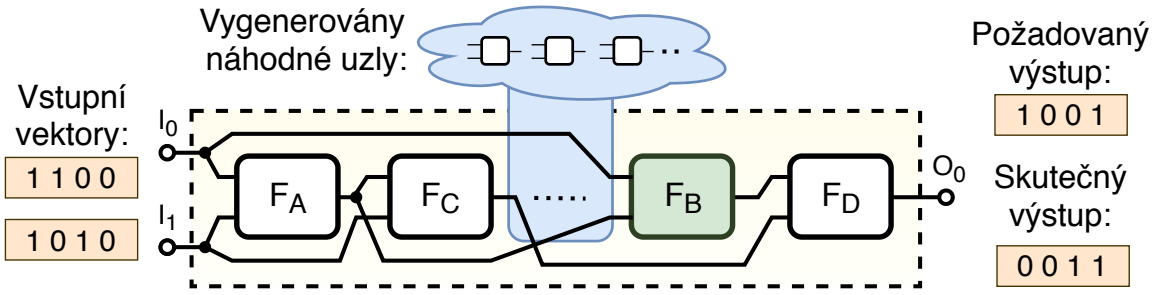
Dalším krokem je výběr náhodného místa ve chromozomu, kde dojde k mutaci. Na obrázku 5.3 je náhodně vybraný uzel ilustrován zelenou barvou. Tomuto uzlu bude mutován jeden z jeho vstupů. Výběr uzlu nemusí být proveden podle rovnoměrného rozložení. V souvislosti s dalšími úpravami je vhodné při náhodném výběru lehce preferovat uzly blíže ke vstupu. Odůvodnění je uvedeno v kapitole o experimentech.

Na obrázku 5.3 je také naznačena následná fáze transformace chromozomu. Poté co je vybrán uzel, jehož jedno vstupní zapojení se bude mutovat, jsou uzly v chromozomu rozděleny na dvě části. Jsou rozděleny na zadní část, která závisí na výstupu daného mutovaného uzlu, a na přední část, která není na mutovaný uzel napojena přímo ani nepřímo. Uzly před mutovaným uzlem zůstanou na svém místě, neboť zpětné vazby nejsou povoleny a tedy tyto uzly nemůžou záviset na výstupu mutovaného uzlu. Jsou tedy automaticky v přední části chromozomu. Uzly za mutovaným uzlem, které nejsou na uzel nijak vázány, jsou přesunuty ve stejném pořadí před mutovaný uzel a za uzly, které byly původně před mutovaným uzlem. Mutovaný uzel a všechny uzly, které jsou na něm závislé, jsou v zadní části chromozomu. V obrázku 5.3 je takto přesunut uzel s funkcí F_C, neboť tento není připojen na uzel s funkcí F_B. Uzel označený F_C je tedy přesunut do přední části chromozomu.



Obrázek 5.3: Diagram přibližující fázi výběru místa mutace ve chromozomu a následnou transformaci chromozomu spojenou s přeskládáním uzlů. Tyto kroky jsou součástí základních úprav chromozomu v metodě dávkové mutace. Uzly, jejichž výsledek nezávisí na vybraném uzlu k mutaci, jsou přesunuty před mutovaný uzel.

V další fázi jsou mezi obě části vygenerovány nekódující uzly. V obrázku 5.4 jsou tyto uzly znázorněny v modrém oblaku. Tyto nově vytvořené uzly jsou náhodně připojeny na primární vstupy, na uzly první části chromozomu, nebo na předchozí vygenerované uzly. Takto je vytvořená třetí část (prostřední) chromozomu obsahující pouze nekódující uzly. Uzly jsou nekódující, neboť na ně nebyly připojeny žádné stávající kódující uzly.

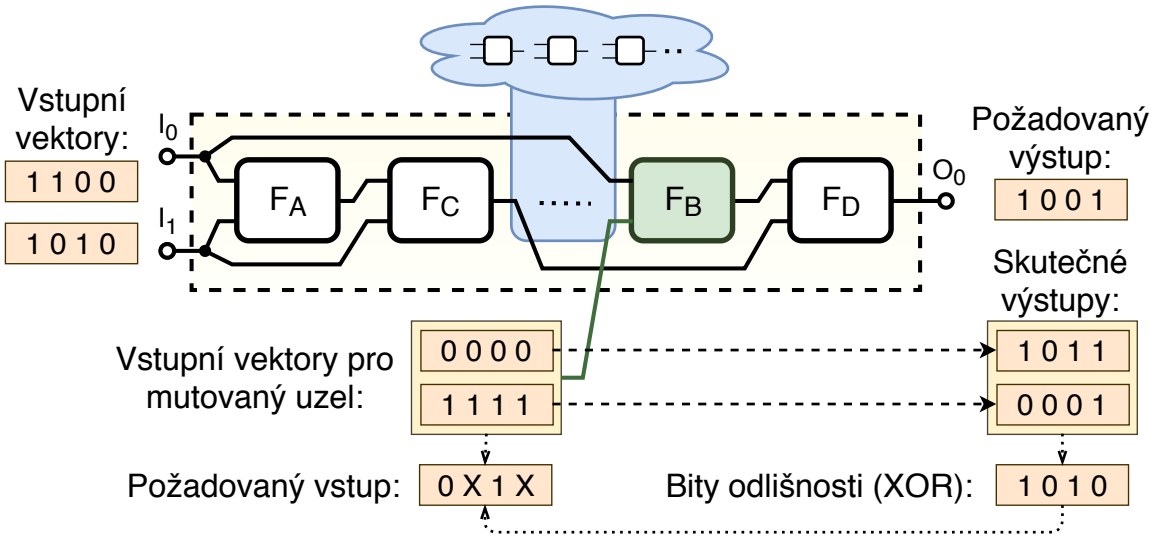


Obrázek 5.4: Diagram přibližující fázi generování náhodných uzlů mezi přední (uzly nenavazující na mutovaný uzel označený F_B) a zadní část (uzly závislé na mutovaném uzlu označeném F_B) chromozomu. Vygenerované uzly jsou naznačeny v modrém oblaku a tvoří prostřední část chromozomu.

Přestože jsou na diagramu 5.4 vygenerované uzly v prostřední části chromozomu pouze naznačeny, tvoří důležitý základ pro fungování metody dávkové mutace. Prostřední část chromozomu je několikrát větší než její zbylé dvě části. Vygenerované uzly vnáší do metody potřebnou rozmanitost, neboť jsou zdrojem nových funkcí a propojení, jež se v chromozomu dosud nemusely vyskytovat. Prosté CGP používá mutace, které mění jak propojení uzlů, tak také funkce uzlů. Při použití metody dávkové mutace je od mutování funkcí uzlů upuštěno a všechny mutace probíhají na úrovni propojení. Je proto nutné generovat sadu náhodných

uzlů k tomu, aby se různé nové funkce mohly do chromozomu připojit. Je důležité upozornit na skutečnost, že se mezi vygenerovanými uzly mohou nacházet uzly se stejným zapojením, jako mají uzly v přední části chromozomu. Pokud se tyto uzly liší pouze ve své funkci, tak je mutace propojení efektivně ekvivalentní s mutací funkcí u daných uzlů. Metoda dávkové mutace tedy v principu kopíruje fungování základního CGP.

Zásadní fáze pro vysvětlení metody dávkové mutace je zobrazena na obrázku 5.5. V této fázi již dochází k samotnému vyhodnocení fenotypu, který byl vytvářen v předcházejících krocích. Obvod je částečně simulován pro vstupní vektory. Částečnou simulací je míněna simulace přední a prostřední části chromozomu včetně všech nekódujících uzlů. Výstupy všech takto dosažených uzlů jsou uloženy v paměti. Simulace obvodem proběhne až do místa mutovaného uzlu, kde se zastaví. Odezva samotného mutovaného uzlu zatím není počítána.



Obrázek 5.5: Diagram přibližující výpočet vektoru požadovaného vstupu pro mutovaný uzel. Na obrázku je mutovaný vstup u mutovaného uzlu označen zeleně a je spojen s nulovým a jedničkovým vektorem. Vztah vstupních vektorů uzlu a skutečných výstupů je naznačen čárkovanou čarou a znamená, že skutečné výstupy vznikly odezvou na dané vektory. Bity odlišnosti označují vektor odlišnosti obou skutečných výstupů.

U mutovaného uzlu dochází k rozdělení simulace na dva běhy. V prvním běhu je na mutovaný vstup mutovaného uzlu (v obrázku 5.5 spodní vstup označen zeleně) přiveden vektor obsahující samé nuly. Simulace obvodu pokračuje. Pro mutovaný uzel je tedy vypočtena jeho odezva na nulový vektor a na vektor opatřený standardním způsobem přes jeho nemutovaný vstup. Simulace končí výpočtem primárních výstupů celého obvodu. V obrázku 5.5 je takto vypočtený vektor prvním vektorem označeným jako skutečný výstup. Výstup prvního běhu simulace je odezvou obvodu na vstupní vektory a na nulový vektor, který je na vstupu mutovaného uzlu.

Ve druhém běhu simulace je na mutovaný vstup mutovaného uzlu přiveden vektor obsahující samé jedničky. Simulace probíhá znovu od mutovaného uzlu, neboť všechny předešlé výstupy uzlů přední a prostřední části jsou již známé a neměnné. Simulace opět doběhne až do konce s tím, že výsledek je odezvou obvodu na vstupní vektory a na jedničkový vektor,

jenž je na vstupu mutovaného uzlu. Na obrázku 5.5 je takto vypočtený vektor druhým vektorem označeným jako skutečný výstup.

Vzniklá dvojice výstupních vektorů obou běhů simulace v sobě obsahuje informaci o ideálním vstupu pro mutovaný uzel. Výstupní vektory mají sobě příslušející bity na stejné pozici buďto shodné (0-0 a 1-1) nebo rozdílné (0-1 a 1-0). Pokud jsou shodné, tak ať bude na pozici těchto bitů na vstupu mutovaného uzlu cokoli, nemůže to ovlivnit výsledek. Pokud jsou bity rozdílné, hodnoty bitů na odpovídajících pozicích na vstupu mutovaného uzlu určují hodnoty daných bitů u primárního výstupu.

Aplikací logické operace XOR na dvojici výstupních vektorů se vypočítá vektor odlišnosti – maska. V tomto vektoru jsou jedničky na pozicích, kde se výstupní vektory odlišují. Vektor odlišnosti je dále používán jako maska určující důležité bity na vstupu mutovaného uzlu, které mají vliv na výstup a tím také na fitness chromozomu. V obrázku 5.5 je tento vektor označen popiskem „Bity odlišnosti“.

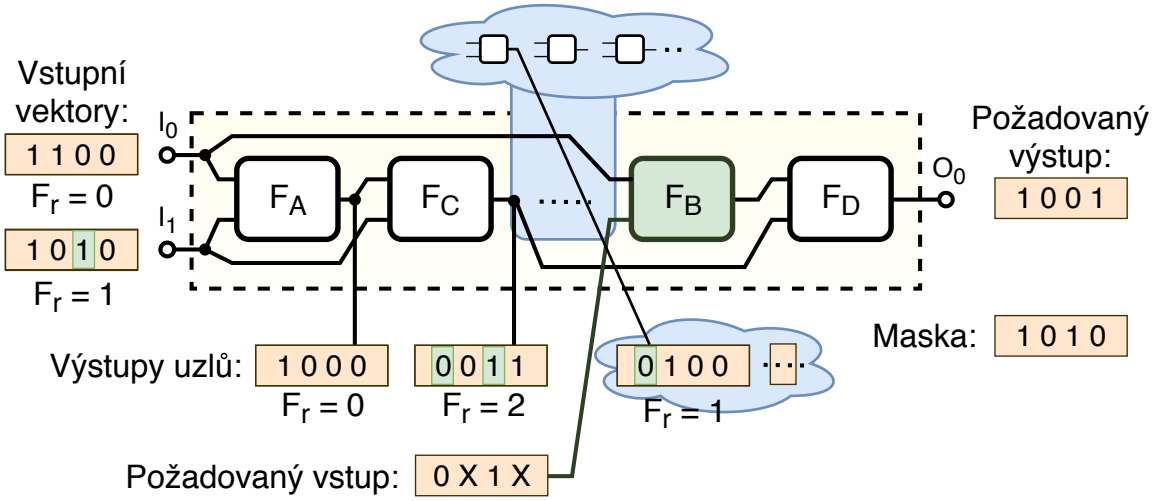
Na základě vektoru odlišnosti, skutečných výstupních vektorů obou běhů simulace, požadovaného výstupního vektoru a znalosti pořadí nulového a jedničkového vektoru, lze vypočítat požadovaný vstup pro mutovaný uzel. Požadovaný vstup je vektor, který obsahuje takové hodnoty bitů, aby odezva obvodu na tento vektor měla maximální fitness. V obrázku 5.5 požadovaný vstup mutovaného uzlu obsahuje mimo hodnoty 0 a 1 také znak X označující, že pro dané bity nezáleží na jejich hodnotě. Ve skutečnosti budou na místech X hodnoty 0 nebo 1, ale při výpočtech budou ignorovány na základě masky vektorů odlišnosti.

Výpočet požadovaného vstupního vektoru má následující základ. Pokud je u bitu požadovaného na primárním výstupu fenotypu stejná hodnota, jako u skutečného výstupu, který vznikl odezvou na nulový vektor, bude na pozici tohoto bitu uvnitř požadovaného vstupního vektoru mutovaného uzlu dosazena hodnota 0. Pokud naopak odpovídá hodnota bitu požadovaného celkového výstupu danému bitu ve výstupním vektoru odpovídajícímu odezvě na jedničkový vektor, bude do požadovaného vstupního vektoru dosazena hodnota 1. Pro takto vzniklý vektor požadovaného vstupu mutovaného uzlu platí, že odezva fenotypu na něj by obsahovala na výstupu správné hodnoty bitů na pozicích, kde vektor odlišnosti nabývá hodnot 1. Na těchto pozicích by tedy byl požadovaný výstup shodný se skutečným výstupem.

Cílem evolučního algoritmu je maximalizovat fitness hodnotu mutacemi na chromozomu. Při evoluci logických obvodů s využitím CGP je fitness vypočtena jako počet shodných bitů reálného a požadovaného výstupu. Výpočtem požadovaného vstupního vektoru mutovaného uzlu a vektoru odlišnosti lze výpočet fitness u CGP transformovat, neboť tyto vektory v sobě již přeneseně obsahují informaci o celkové fitness. Novou úlohou u metody dávkové mutace je hledání výstupů uzlů před mutovaným uzlem, které nejvíce odpovídají požadovanému vstupu mutovaného uzlu. Počet shodných bitů náhodného vektoru a požadovaného vstupu udává minimální celkovou fitness fenotypu s odezvou na daný náhodný vektor. Jelikož neurčené hodnoty (označené znakem X) požadovaného vstupního vektoru neobsahují informaci o celkové fitness, bude počet shodných bitů určitého vektoru a požadovaného vstupu označen jako relativní fitness – F_r.

Úloha hledání výstupních vektorů uzlů, které se nacházejí ve fenotypu před mutovaným uzlem, je ukázána na obrázku 5.6. V této fázi metody dávkové mutace se hledá optimální zapojení mutovaného vstupu tak, aby byla relativní fitness maximální, a tím se také přeneseně maximalizovala celková fitness nově vznikajícího obvodu.

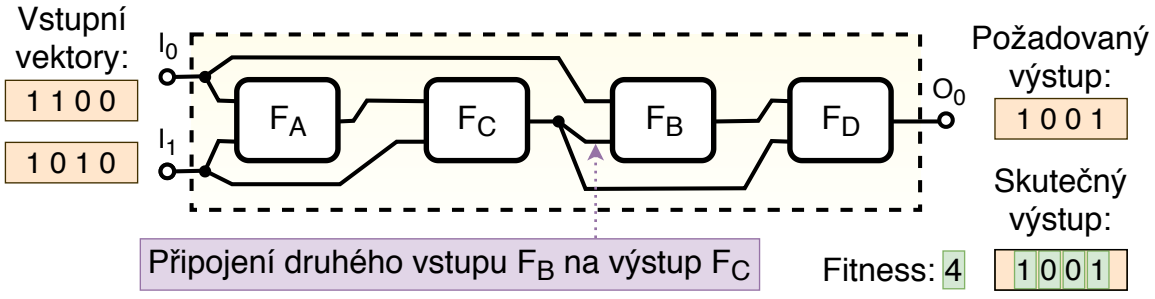
Vstup mutovaného uzlu lze připojit na kterýkoli výstup předchozích uzlů, nebo na primární vstupy fenotypu. Na obrázku 5.6 jsou ukázány vektory výstupů uzlů i vstupní vektory a zelenou barvou jsou označeny bity, ve kterých se vektory shodují s požadovaným vstu-



Obrázek 5.6: Diagram přibližující hledání ideálního zapojení pro mutovaný uzel. V obrázku je zobrazena řada vektorů, které vznikly při simulaci fenotypu. Jedná se o výstupy všech uzlů, jež se nacházejí před mutovaným uzlem, a primární vstupní vektory. Vektory jsou na místech, kam se může mutovaný vstup uzlu připojit. Porovnáním požadovaného vstupu s těmito vektory je vypočtena relativní fitness – F_r. V diagramu jsou zanedbány některé výstupní vektory uzlů prostřední části chromozomu.

pem. Vektory výstupů uzlů i vstupní vektory byly vypočteny v prvním běhu simulace. Na základě jejich porovnání s požadovaným vstupem je vypočtena relativní fitness každého připojení. Ze všech možných potenciálních připojení mutovaného uzlu je vybráno to, které má maximální relativní fitness. Pokud je takových připojení více, je jedno vybráno náhodně.

V příkladu na obrázku 5.6 je maximální relativní fitness přiřazena výstupu uzlu s označením F_C. V dalším kroku je proto druhý vstup mutovaného uzlu připojen právě na výstup uzlu s označením F_C. Tuto operaci znázorňuje obrázek 5.7. V příkladu je tímto dosaženo maximální celkové fitness fenotypu. Při použití metody dávkové mutace není celková fitness kandidátních řešení nikdy počítána, a je proto nutné ji kontrolovat externě. Celkovou fitness lze vypočítat z relativní fitness, nebo lze při simulaci provést třetí běh, ve kterém je u mutovaného uzlu místo nulového či jedničkového vektoru použit korektní vstupní vektor.



Obrázek 5.7: Diagram znázorňující konečné propojení všech uzlů chromozomu, které bylo nalezeno metodou dávkové mutace. Mutovaný vstup uzlu se připojil na výstup uzlu označeného F_C. Z chromozomu byly odstraněny nekódující uzly. Celková fitness chromozomu vzrostla na 4, což je maximální možná fitness pro obvod se dvěma vstupy.

Na rozdíl od standardního CGP není při použití metody dávkové mutace nutné ukládat nastavení dosud nejlepšího chromozomu. Nalezené zapojení vstupu mutovaného uzlu musí mít minimálně tak dobrou fitness, jako chromozom v předchozí generaci. To proto, že v nejhorším případě bude opět nalezeno stejné propojení. Fitness jedinců proto nemůže nikdy klesat.

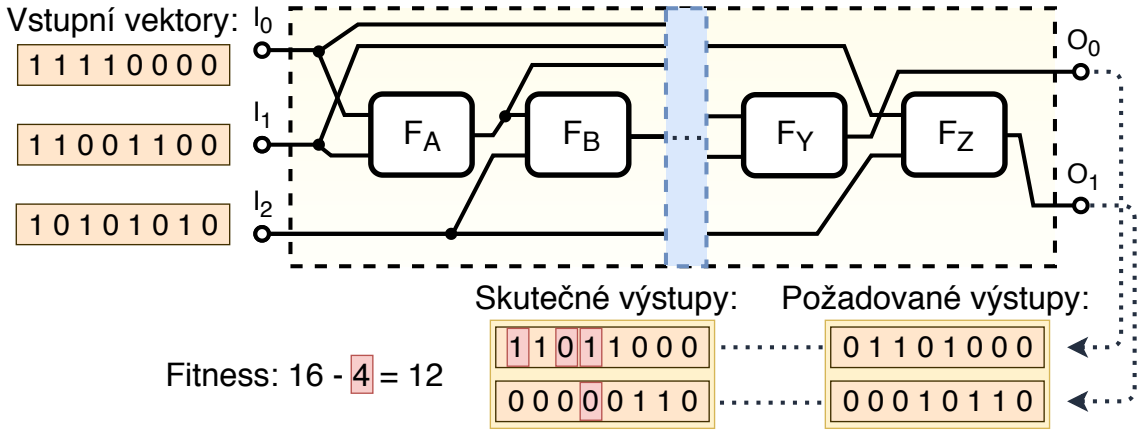
Síla metody dávkové mutace spočívá v tom, že každé porovnání požadovaného vstupu mutovaného uzlu s výstupem předchozího uzlu, nebo s primárním vstupem, je ekvivalentní s evaluací jedné mutace u standardního CGP. Z teoretického hlediska je v každé iteraci metody dávkové mutace provedeno tolik porovnání, kolik je uzlů před mutovaným uzlem. Tato porovnání jsou jednoduchá a časově nenáročná. Ve srovnání s evaluací mutací u standardního CGP dochází použitím metody dávkové mutace ke značnému urychlení. Pro standardní CGP, které vyhodnocuje kandidátní řešení simulací, trvá fáze simulace nejdéle. Stejnou fázi obsahuje i CGP používající metodu dávkové mutace, kdy je dokonce simulační fáze delší, neboť se větví na dva běhy a simuluje se velké množství vygenerovaných nekódujících uzlů. Ke zrychlení dochází u metody dávkové mutace až u fáze porovnávání, neboť srovnávání vektorů je rychlé a přitom odpovídá celým evaluacím u standardního CGP. Přestože je tedy jedna iterace u standardního CGP rychlejší, je třeba takovýchto iterací provést velké množství, aby bylo dosaženo stejného výsledku jako u jedné iterace využívající metodu dávkové mutace.

Sadu všech porovnání lze označit jako dávku mutací, kde jsou všechny mutace vyhodnoceny pomocí relativní fitness. Na základě tohoto poznatku byla metoda nazvána metodou dávkové mutace. Jednou dávkou mutací jsou míněna všechna možná propojení jednoho ze vstupů u mutovaného uzlu. Z tohoto hlediska se standardní CGP odlišuje od CGP používající metodu dávkové mutace, neboť standardní CGP mutuje propojení uzlů náhodně. Pokud by se standardní CGP zaměřovalo vždy pouze na náhodný vstup náhodného uzlu a postupně testovalo všechny možné varianty zapojení, tak by bylo ryze ekvivalentní s přístupem používajícím dávkovou mutaci. Ve skutečnosti i standardní CGP musí nalézat optimální propojení uzlů, avšak optimalizace uzlů jsou postupné a rozložené v čase.

5.5.1 Komprese fitness pro metodu dávkové mutace

V podkapitole 5.5 byla metoda dávkových mutací vysvětlena na fenotypu, který měl jediný primární výstup. Na základě výstupního vektoru je vypočten požadovaný vstup mutovaného uzlu a jeho maska. Pokud by primárních výstupů bylo u fenotypu více, je nutné vytvořit i více vektorů požadovaného vstupu a více vektorů bitových masek. Základní verze metody dávkové mutace tedy pracuje s tolika vektory požadovaného vstupu, kolik má obvod výstupů. To představuje zátěž, neboť své rychlosti dosahuje metoda dávkové mutace ve fázi porovnávání vektorů požadovaného vstupu s vektory výstupů uzlů a vstupními vektory. S rostoucím množstvím primárních výstupů obvodu tedy lineárně stoupá časová náročnost porovnávací fáze. Tento problém řeší navržená metoda komprese fitness, která umožňuje převod všech vektorů požadovaného vstupu do jediného vektoru.

Pro ilustraci metody komprese fitness bude použit ukázkový fenotyp jehož reprezentace je na obrázku 5.8. Při evaluaci tohoto kandidátního řešení je vypočtena dvojice skutečných výstupů, kterou lze porovnat s dvojicí požadovaných výstupů a vypočítat fitness hodnotu. Maximální fitness pro obvod se dvěma výstupy je rovna šestnácti. Vstupní kombinace jsou zakódovány do podoby vektorů. Obdobně budou dále všechny hodnoty uváděny formou vektorů.



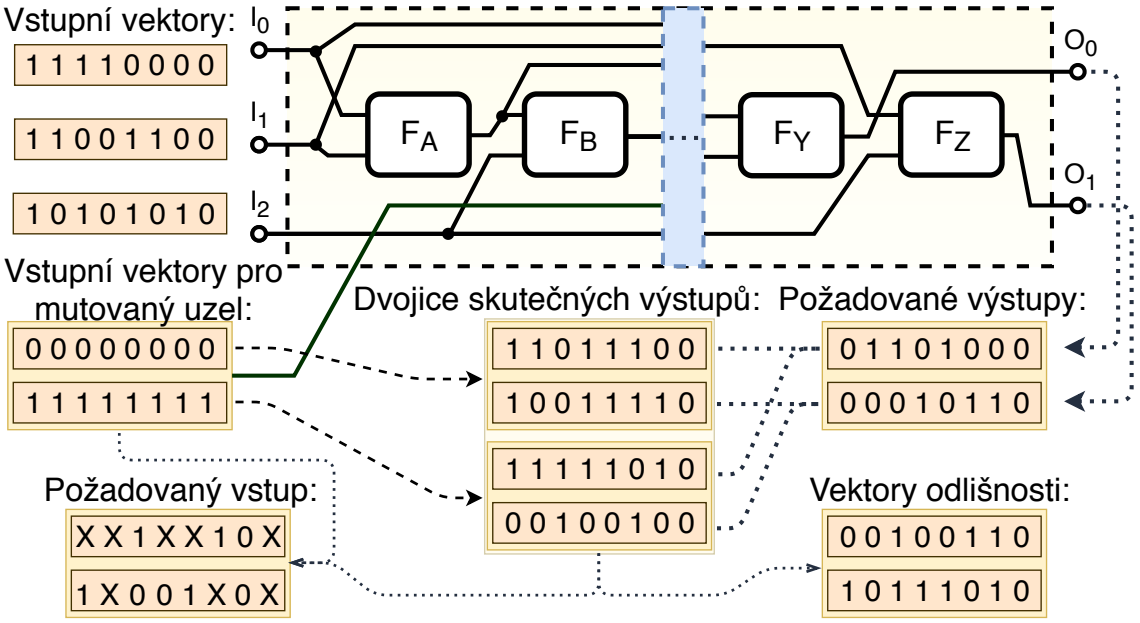
Obrázek 5.8: Diagram znázorňující reprezentaci fenotypu CGP, který má dva primární výstupy. Na obrázku jsou zobrazeny pouze první dva uzly a poslední dva uzly. Zbytek fenotypu je abstrahován a ukryt uvnitř modrého okna. Chromozom: 0,1,F_A; 3,2,F_B; ... ; 6,25,F_Y; 26,23,F_Z; 27,28. Obecné funkce bloků jsou označeny F_A - F_Z. Parametry: $n_i = 3$, $n_0 = 2$, $n_r = 1$, $n_c = 26$, $n_a = 2$, l-back=max.

Většina fází metody dávkové mutace zůstává při použití komprese fitness zachována. Z chromozomu jsou odstraněny nekódující uzly, je náhodně vybrán uzel určený k mutování. Chromozom je rozdělen na tři části, jsou vygenerovány náhodné nekódující uzly a jsou provedeny dva běhy simulace. Výsledkem každého běhu simulace je pro případ dvou primárních výstupů dvojice výstupních vektorů, jak je ukázáno na obrázku 5.9. První dvojice výstupních vektorů vzniká jako odezva fenotypu na nulový vektor a druhá dvojice jako odezva na jedničkový vektor. Z těchto dvojic vektorů je vypočtena dvojice vektorů odlišnosti, kdy je první vektor z dvojice vektorů odlišnosti vypočítán aplikací operace XOR na první vektor první dvojice skutečných výstupů a na první vektor druhé dvojice skutečných výstupů. Obdobně je vypočítán i druhý vektor odlišnosti a stejným způsobem je přistupováno i k výpočtu vektorů požadovaného vstupu.

Problém u evoluce obvodů s více výstupy je v tom, že pro každý primární výstup vzniká samostatný vektor požadovaného vstupu. Dle základní metody dávkové mutace jsou tyto vektory porovnávány s ostatními vektory výstupů uzlů a primárních vstupů. Při výpočtu relativní fitness je nejdříve porovnán určitý vektor s prvním požadovaným vstupem a množství stejných odpovídajících si bitů je zaznamenáno. Dále je stejný vektor porovnán s druhým požadovaným vstupem a množství shodných bitů je přičteno k první zaznamenané hodnotě. Stejným způsobem se pokračuje i pro další potenciální vektory požadovaných vstupů, kdy jsou všechny mezivýsledky přičítány do výsledné relativní fitness.

Při použití více vektorů požadovaného vstupu vypočtená relativní fitness přesně odpovídá celkové fitness daného kandidátního řešení, která byla vypočtena přes masku odlišnosti. Tedy počet jedničkových bitů ve vektorech odlišnosti udává maximální možnou relativní fitness. Pro nulové bity ve vektorech odlišnosti je informace o fitness ztracena, neboť tyto bity nelze přes mutovaný vstup ovlivnit. Úkolem relativní fitness je zachovávat informaci o tom, jaký dopad by mělo určité propojení uzlů na celkovou fitness fenotypu.

Problém vzrůstajícího počtu vektorů požadovaného vstupu by bylo možné vyřešit, pokud by existoval způsob, kterým lze všechny vektory požadovaného vstupu zkombinovat do jediného vektoru. Pro takovýto vektor ovšem nemůže relativní fitness odpovídat celkové fitness, neboť jednotlivé bity nemohou nést informaci o fitness všech jednotlivých



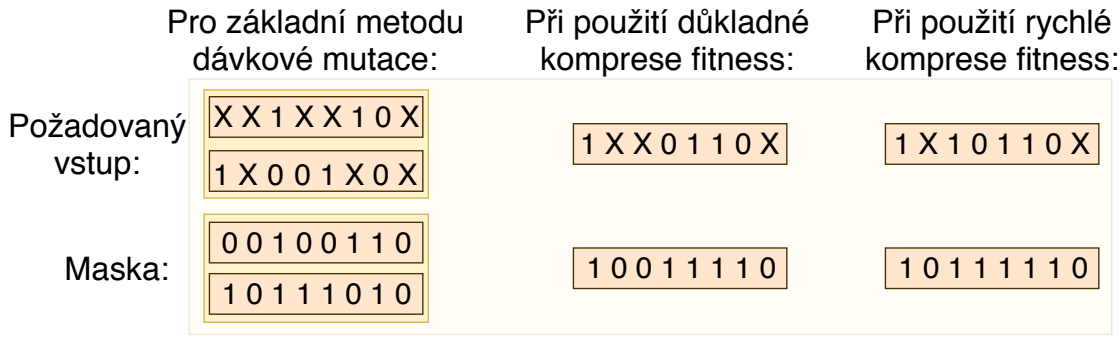
Obrázek 5.9: Diagram přibližující výpočet vektorů požadovaného vstupu pro mutovaný uzel. Na obrázku není samotný mutovaný uzel zobrazen, ale jeho vstup je obarven zeleně a spojen s nulovým a jedničkovým vektorem. Vztah vstupních vektorů uzlu a dvojic skutečných výstupů je naznačen přerušovanou čarou a znamená, že dvojice skutečných výstupů vznikly odezvou na dané vektory.

výstupů. Například si lze představit situaci, kdy by v odezvě na nulový bit mutovaného vstupu všechny výstupy fenotypu nabyly špatné hodnoty na daném bitu, ale v odezvě na jedničkový bit by všechny výstupy měly požadovanou hodnotu. V takovém případě by pro konkrétní bit byl rozdíl celkové fitness o velikosti počtu výstupů. Lze také konstatovat, že by daný bit měl velkou váhu při hledání správného zapojení mutovaného uzlu, neboť výrazně ovlivňuje celkovou fitness.

Metoda komprese fitness zanedbává zachování vah jednotlivých bitů při výpočtu celkové fitness a zaměřuje se pouze na to, která z hodnot bitu je lepší pro maximalizaci celkové fitness. Vektor požadovaného vstupu tedy obsahuje nulu tam, kde je odezva fenotypu na nulu blíže požadovanému výstupu, než je odezva na jedničku. Tímto způsobem pracuje metoda, která bude v práci označována jako důkladná komprese fitness. Pro příklad z obrázku 5.9 je přepočet požadovaného vstupu ukázán na obrázku 5.10.

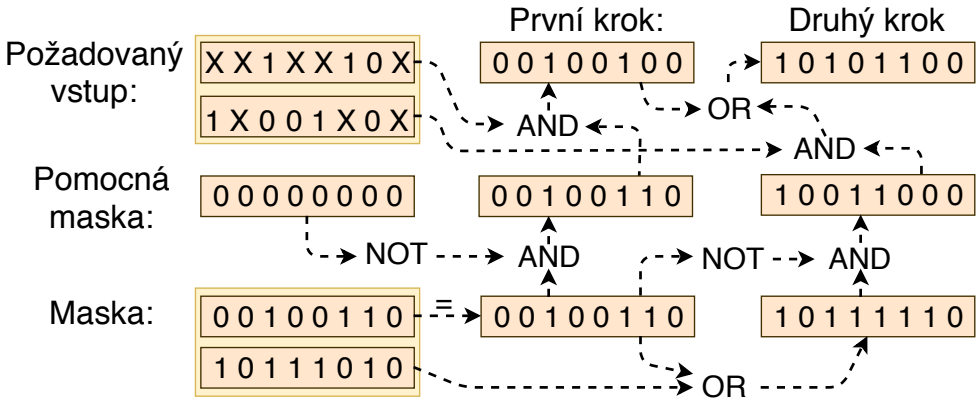
Problémem důkladné komprese fitness je to, že výpočet převládajících hodnot bitů ve vektorech požadovaného vstupu je velmi náročný. Pro reálné uplatnění komprese bude metoda zjednodušena tím, že bity komprimovaného požadovaného vstupu přijmou první hodnotu z vektorů požadovaného vstupu, kdy je bit masky daného vektoru nastaven na jedna. V kontextu srovnávajícím oba přístupy bude zjednodušená verze metody komprese fitness nadále označována jako rychlá komprese fitness.

Výpočet vektoru požadovaného vstupu a jeho masky v metodě rychlé komprese fitness vychází z vektorů požadovaného vstupu a masek, které jsou standardním způsobem vypočteny pro všechny primární výstupy. Výpočet je znázorněn na obrázku 5.11. Výpočet probíhá v krocích, kdy každý krok zpracovává jeden z vektorů požadovaného vstupu a jeho masku. Při výpočtu je použita pomocná maska. Pokud je hodnota určitého bitu v pomocné



Obrázek 5.10: Tabulka různých náhledů na požadovaný vstup a odpovídající masku.

masce 1, tak to znamená, že hodnota daného bitu v právě zpracovávané masce je také 1 a hodnota stejného bitu všech předešlých masek byla 0. Díky tomu jsou do vznikajícího vektoru požadovaného vstupu přidávány hodnoty pouze na místa, která nemají dosud určenou požadovanou hodnotu.



Obrázek 5.11: Diagram výpočtu vektoru požadovaného vstupu a jeho masky v metodě rychlé komprese fitness. Znakem X se rozumí nedefinovaná hodnota.

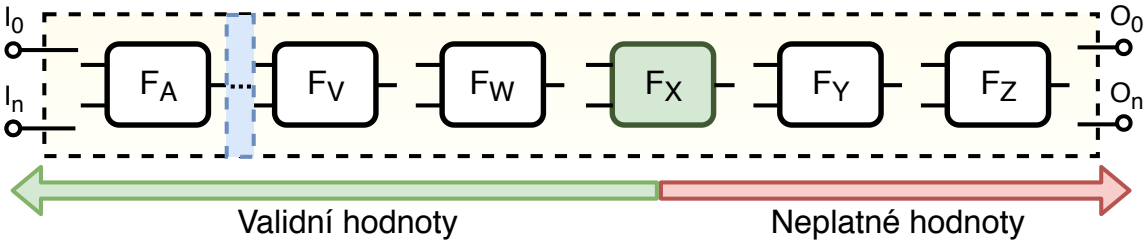
Při použití metody rychlé komprese fitness vzniká uměle priorita mezi jednotlivými výstupy fenotypu. To znamená, že evoluce bude upřednostňovat vyřešení určitých primárních výstupů před jinými. Dle implementace se může jednat o výstupy s nižším, nebo s vyšším indexem. Při řešení konkrétního problému by se měly vyzkoušet obě varianty a zvolit ta, která funguje lépe.

5.5.2 Metoda recyklace hodnot v metodě dávkové mutace

Zrychlení metody dávkové mutace vychází především z fáze výpočtů relativních fitness hodnot, tedy z porovnávání vektorů požadovaného vstupu s výstupními vektory uzlů a primárními vstupními vektory. Metodu naopak zpomaluje fáze simulace fenotypu, kdy je nutné vypočítat všechny výstupy uzlů. Metoda recyklace hodnot prodlužuje dobu, po kterou algoritmus pracuje ve fázi výpočtů relativních fitness, a nepřímo zkracuje dobu simulace.

Na konci simulační fáze jsou známy hodnoty výstupů všech uzlů fenotypu, které se nacházejí před mutovaným uzlem. Výstupy uzlů nacházejících se za mutovaným uzlem jsou odezvou na jedničkový vektor na vstupu mutovaného uzlu. Známé hodnoty výstupů uzlů

jsou označeny jako validní a hodnoty, které vznikly odezvou na jedničkový vektor na vstupu mutovaného uzlu, jsou označeny za neplatné. Tento koncept je ukázán na obrázku 5.12.

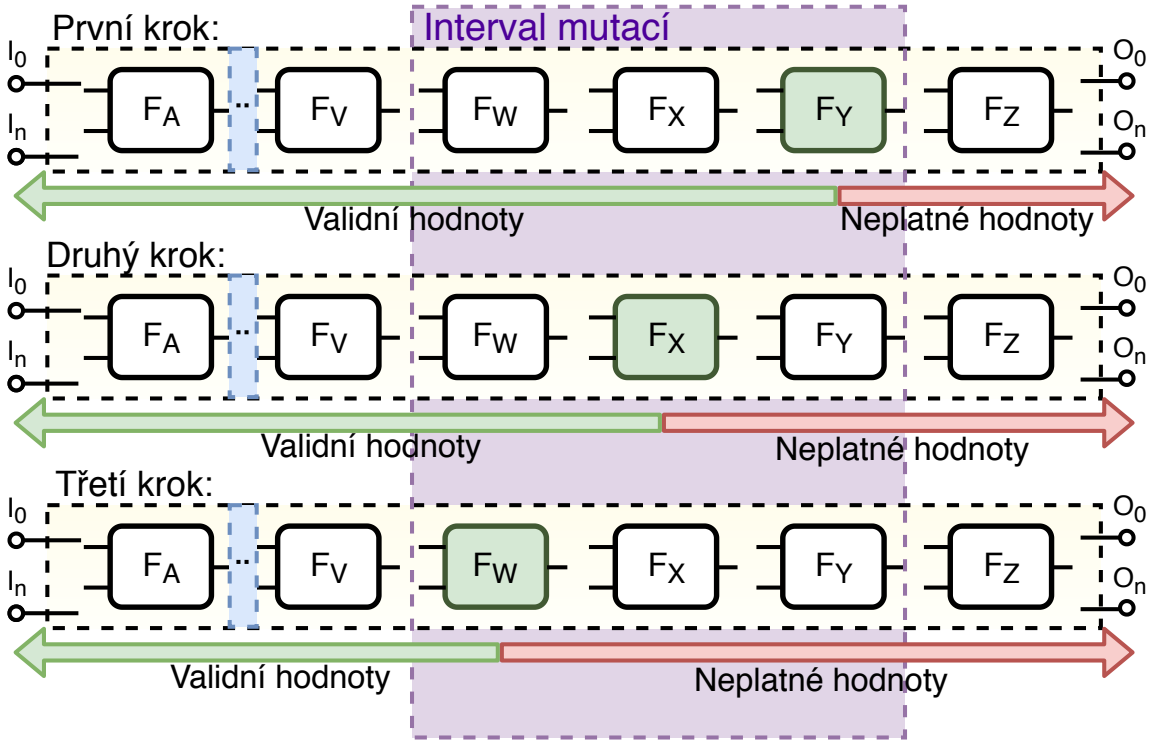


Obrázek 5.12: Diagram znázorňující validní a neplatné hodnoty výstupů uzlů fenotypu na konci druhého běhu simulace. Hodnoty, které nejsou validní, vznikly odezvou na jedničkový vektor. Mutovaný uzel je zabarven zeleně. Modrý obdélník abstrahuje další uzly ve fenotypu. Propojení uzlů není zobrazeno.

Metoda recyklace hodnot vychází z toho, že je většina výstupních hodnot uzlů již vypočtena, a po nalezení propojení mutovaného uzlu pokračuje v mutování dalších vstupů uzlů. Metoda tedy na konci vyhodnocení celé jedné dávky mutací nezahajuje novou iteraci, ale pokračuje ve vyhodnocení další dávky mutací. Metoda v mutacích uzlů postupuje odzadu a vždy po zmutování všech vstupů daného uzlu pokračuje na uzel předešlý. Tento postup vyplývá z toho, že hodnoty výstupů uzlů před mutovaným uzlem jsou stále validní, a lze proto zahájit mutaci dalšího uzlu bez nutnosti validní část fenotypu znovu simulovat. Vždy stačí pouze vypočítat odezvu mutovaného uzlu na nulový a jedničkový vektor, čímž jsou neplatné hodnoty přepočítány.

Postup jednotlivých kroků v metodě recyklace hodnot je na obrázku 5.13. Metoda recyklace zavádí nový atribut. Tím je délka intervalu mutací. Tato délka stanovuje počet uzlů, které budou postupně mutovány dávkovou mutací. Začátek intervalu je stanoven náhodně vybraným mutovaným uzlem. Tento uzel je tedy mutován jako poslední, neboť mutace postupují odzadu. Pokud interval přesahuje za poslední uzel ve fenotypu, je zkrácen a mutování je zahájeno na posledním uzlu. V každém kroku jsou provedeny dávkové mutace všech vstupů mutovaného uzlu. Po nalezení optimálního propojení daného uzlu je pokračováno uzlem předešlým.

Po dokončení mutace posledního uzlu dochází k nové iteraci, která znovu prochází všechny fáze základní metody dávkové mutace, jako je například vygenerování nové množiny nekódujících uzlů. Mohlo by se zdát, že stanovovat interval pro vykonávání dávkových mutací je zbytečné, ale tímto krokem se zamezuje tomu, aby byly neustále mutovány uzly v zadní části chromozomu. Je také třeba uzly pravidelně přeskládávat a generovat nové nekódující náhodné uzly. Oba tyto kroky ale vyžadují spuštění simulace a přepočítání všech výstupů. Délka intervalu mutací by měla být stanovena experimentálně pro daný problém.



Obrázek 5.13: Diagram přibližující postup mutování uzlů u metody recyklace hodnot. Velikost intervalu mutací je rovna třem. Pro mutaci byl náhodně zvolen uzel označený F_W. Zvolený interval mutací zahrnuje uzly označené F_W, F_X a F_Y a je v diagramu naznačen fialovým obdélníkem. V prvním kroku je provedena dávková mutace na uzlu označeném F_Y. V dalších krocích jsou mutovány uzly předchozí, čímž je využito validních hodnot. Validní hodnoty výstupů uzlů byly vypočteny v prvním běhu simulace. Neplatné hodnoty vznikly odezvou na jedničkový vektor. Mutovaný uzel je zabarven zeleně. Modrý obdélník abstrahuje další uzly ve fenotypu. Propojení uzlů není zobrazeno.

Kapitola 6

Implementace optimalizačních metod

Součástí diplomové práce jsou programy, které implementují metody popsané v předchozích kapitolách. Jelikož některé metody nelze kombinovat, byly vybrány dvě třídy problémů, na něž se práce zaměřuje. První třídou je automatický návrh obrazových filtrů. Druhou třídou je automatický návrh logických obvodů. V následujících dvou kapitolách budou popsány vytvořené programy včetně toho, které akcelerační metody implementují.

6.1 Použití CGP pro návrh obrazových filtrů

Programy byly implementovány v jazyce C++ za použití knihoven OpenCV (3.2.0) a OpenMP (4.0). Implementace byly testovány na operačním systému Windows 7 – 64bit. Programy využívají instrukčních sad AVX2 a SSE, a proto je lze spustit pouze na procesorech architektury x86-64, které příslušné sady podporují. Pro úspěšný překlad a spuštění je také nutné mít nainstalované potřebné knihovny.

Byly vytvořeny následující tři programy:

- `chrom_gen.cpp` - pro vytvoření a trénování chromozomů
- `chrom_tester_1.cpp` - pro otestování jednoho chromozomu
- `chrom_tester_2.cpp` - pro otestování celé sady chromozomů

Jako zdroj obrázků sítnic byla využita databáze DRIVE[1]. Ta obsahuje celkově 40 obrázků, k nimž jsou poskytnuty vždy dvě různé manuální segmentace provedené odborníky. Program byl trénován na obrázcích 1-20 a testován na obrázcích 21-40.

Programy využívají akcelerace CGP s využitím strojového kódu, kdy jsou kandidátní řešení testována přímým spuštěním sekvence instrukcí instrukční sady AVX2. Proces evoluce je dále akcelerován využitím paralelizace mezi všemi jádry procesoru. Paralelizace je docíleno použitím knihovny OpenMP a vložením direktivy *"parallel for"* před hlavní smyčku vyhodnocující jedince populace. V paralelní části není nutná komunikace mezi vlákny, proto je zrychlení téměř rovno počtu jader. V programech je implementována vektorizace hodnot s využitím instrukční sady AVX2. Každá instrukce tedy simultánně pracuje s 32 bajty.

Následující odstavec přibližuje základní nastavení programů. U chromozomů je dovoleno maximální propojení. Platí tedy l-back=max. Počet řádků chromozomu byl nastaven na

[1] https://www.isi.uu.nl/Research/Databases/DRIVE/

$n_r = 1$. Pro velikost chromozomu a celkový počet uzlů byla zvolena hodnota $n_c = 64$. Počet povolených funkcí pro uzly je 16. Byly implementovány funkce z tabulky 2.1. Tyto funkce mají maximální aritu $n_a = 2$. Počet vstupů chromozomu je $n_i = 25$ a počet výstupů je $n_o = 1$.

Obrázky sítnic jsou předzpracovány a je z nich extrahována zelená složka. V obrázcích jsou náhodně vybrány vstupní vektory tím způsobem, aby 50 % obsahovalo cévy a zbylých 50 % obsahovalo pozadí. Vstupy chromozomu představují oblast 5×5 pixelů. Pokud je pixel v centru této oblasti pixelem příslušícím cévě, tak je vektor označen jako obsahující cévu. V opačném případě jde o vektor pozadí.

Generátor chromozomů načte obrázky (`01_test.bmp` – `20_test.bmp` a `01_manual.bmp` – `20_manual.bmp`) a evolučně vygeneruje sadu chromozomů (`chromosome_out_01.txt` – `chromosome_out_64.txt`). Program lze nastavit změnou hodnot maker, která ovlivňují: počet vstupních obrázků, počet náhodně zvolených míst uvnitř obrázků, počet uzlů v chromozomu, velikost populace a počet generací. Jako fitness funkce slouží přesnost (accuracy):

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{6.1}$$

Testovací programy načítají chromozom, nebo sadu chromozomů a ukazují výsledek segmentace. Pokud je načtena sada chromozomů, tak výsledný obrázek vzniká kompozicí mezivýsledků těchto chromozomů. Hodnota každého pixelu ve výsledném obrázku je konkrétně spočtena jako vážený průměr 3x3 okolí daného pixelu napříč celou řadou mezivýsledků všech chromozomů. Tímto postupem je výsledek výrazně zpřesněn.

6.2 Použití CGP pro návrh číslicových obvodů

Programy byly implementovány v jazyce C++ a využívají knihovnu OpenMP(4.0). Implementace byly vytvořeny a testovány na operačním systému Windows 7 – 64bit. Programy využívají instrukčních sad AVX2 a SSE, a proto je lze spustit pouze na procesorech architektury x86-64, které příslušné sady podporují. Pro překlad a spuštění je také nutné mít nainstalovanou potřebnou knihovnu.

Byly vytvořeny následující tři programy:

- `CGP_circuit_accelerator_V1.cpp` - pro návrh logických obvodů (*V1*)
- `CGP_circuit_accelerator_V2.cpp` - pro návrh logických obvodů (*V2*)
- `CGP_circuit_tester.cpp` - pro otestování vytvořených obvodů

Verze *V1* a *V2* značí různé přístupy k ukládání vektorů do paměti. Program s verzí *V1* ukládá vektory za sebe v průběhu jednoho běhu vyhodnocujícího jednu sadu vstupních vektorů. Pokud je tedy sad více, nejsou výstupní vektory jednotlivých uzlů uloženy pohromadě. Naopak pro verzi *V2* platí, že jsou výstupní vektory uzlů z různých běhů, které postupně vyhodnocují všechny vstupní kombinace, uloženy pohromadě.

Pro automatický návrh obvodů je třeba mít tabulku všech vstupních vektorů a tabulku odpovídajících výstupů. Jde o úlohu symbolické regrese, kdy je cílem evoluce hodnoty obou tabulek na sebe namapovat. Vstupní i výstupní tabulka hodnot je v programech vygenerována automaticky.

Programy využívají akcelerátoru CGP, který strojovým kódem vytváří spustitelné funkce. Kandidátní řešení jsou proto vyhodnocována přímým spuštěním vygenerované sekvence instrukcí, která odpovídá danému fenotypu. Tyto instrukce pochází z instrukční sady AVX2 a

pracují s 256-bitovými vektory. V programech je tedy využívána bitově paralelní simulace. Dalšího zrychlení je dosaženo díky knihovně OpenMP, která umožňuje vytváření paralelních sekcí programu.

V programech jsou paralelně vyhodnocováni jedinci populace. Aby se zamezilo čekání při určování nejlepšího jedince v populaci, jsou mezi vlákny programu sdíleny údaje a fitness nejlepšího jedince. Pokud je po vyhodnocení jedince jeho fitness menší než sdílená nejlepší fitness, je do vlákna zkopírován nejlepší jedinec. Vlákna na sebe tedy nemusí čekat.

Následující odstavec přibližuje základní nastavení programů. U chromozomů je dovoleno maximální propojení. Platí tedy l-back=max. Počet řádků chromozomu byl nastaven na $n_r = 1$. Pro velikost chromozomu a celkový počet uzlů byla zvolena hodnota $n_c = 625$ v úloze návrhu násobiček 4×4b. Množina funkcí Γ obsahuje logický AND a NAND. Tyto funkce mají aritu $n_a = 2$. Počet vstupů chromozomu pro úlohu návrhu násobiček 4×4b je $n_i = 8$ a počet výstupů je $n_o = 8$.

V programech byla dále implementována metoda dávkové mutace společně se všemi jejími rozšířeními. Z tohoto důvodu v sobě programy obsahují další parametry, které nejsou typické pro běžné CGP. Mezi tyto parametry patří interval mutací a priorita komprese fitness. Pro správné fungování metody dávkové mutace je také nutné nastavit dostatečnou velikost chromozomu.

Program využívá zajímavé vlastnosti metody dávkové mutace, kdy je nutné opakovaně simulovat malou (zadní) část fenotypu. Akcelerátorem CGP je vytvářen strojový kód, který při spuštění postupně vyhodnocuje všechny uzly ve fenotypu. Pokud je následně potřeba vyhodnotit pouze zadní část fenotypu, tak je na potřebné místo uvnitř strojového kódu vložena hlavička obsahující kód instrukcí, které je potřeba vykonat. Díky tomu lze kód spustit přímo z jeho určité části, a vyhodnotit tak pouze výstupy určitých uzlů fenotypu. V průběhu recyklace hodnot proto není nutné sestavovat strojový kód a místo toho je pouze využíván kód již existující.

Kapitola 7

Experimentální vyhodnocení přínosu jednotlivých metod

Cílem této kapitoly je vyhodnotit implementace metod z předchozích kapitol. Zejména v páté kapitole této práce byla popsána a navržena řada metod, které by měly akcelerovat symbolickou regresi pomocí kartézského genetického programování (CGP).

Evoluce je pravděpodobnostního charakteru, a proto je třeba při vyhodnocení provést sady běhů a ty statisticky analyzovat. V této práci byl nejčastěji testován čas potřebný pro vytvoření funkčního jedince. Pokud do stavu plné funkčnosti nelze dojít, tak byla zkoumána rychlost programu pomocí počtu provedených generací za vteřinu a dalších kritérií.

Všechny experimenty byly vyhodnoceny na stolním počítači osazeném procesorem Intel Core i5-4590 s frekvencí 3,3 GHz a 16 GB RAM. Programy byly implementovány v jazyce C++ a testovány v prostředí operačního systému Windows 7.

7.1 Evoluční návrh obrazových filtrů

V předchozích kapitolách bylo uvedeno několik metod, jejichž použití v CGP by mělo akcelerovat evoluční návrh obrazových filtrů. Automaticky navržené obrazové filtry mohou mít řadu výhod oproti filtrům, které byly přímo navrženy programátorem. Při automatickém návrhu lze optimalizovat řadu vlastností filtru, jako je jeho velikost či rychlost. Nevýhodou je ale skutečnost, že samotný evoluční návrh je nesmírně časově náročný. Z tohoto důvodu implementovaný algoritmus využívá metod, které by měly návrh filtrů akcelerovat.

Experimenty budou zaměřeny na úlohu segmentace krevního řečiště ve snímcích sítnice lidského oka. Budou vytvářeny jednoduché filtry, které segmentaci provádí. Filtry jsou sestavovány z množiny 16 funkcí, které jsou zobrazeny v tabulce 2.1. Jde o logické funkce a funkce běžné pro práci s celými čísly.

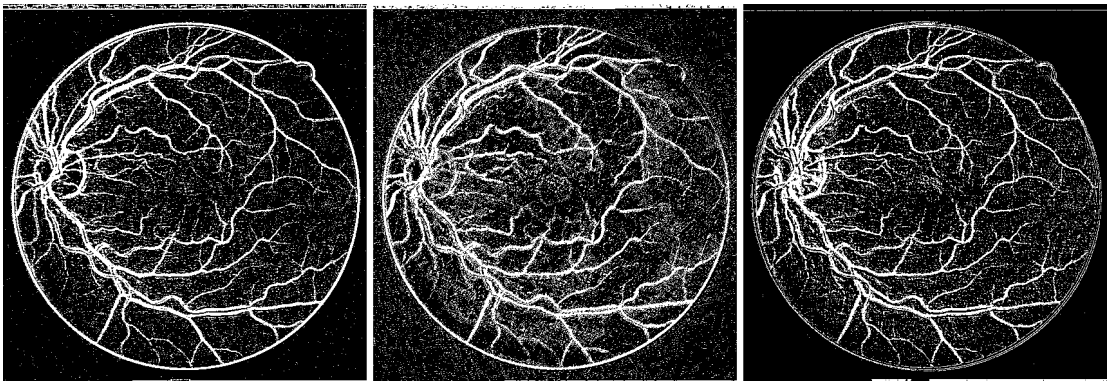
Všechny experimenty proběhly v programech, které byly implementovány v této práci. Program pro evoluční návrh obrazových filtrů využívá následujících akceleračních metod: V programu je použita paralelní simulace, která pracuje s instrukcemi AVX2 pro práci s vektory o velikosti 256 bitů. Do programu bylo zahrnuto přímé použití strojového kódu, tedy při převodu chromozomu na fenotyp vzniká spustitelný kód daného fenotypu. Při zpracování populace program využívá paralelizace, tedy evaluace různých jedinců může probíhat simultánně. Experimenty proběhly na procesoru Intel Core i5-4590, který má čtyři jádra. Paralelně byli současně vyhodnocováni 4 jedinci populace.

7.1.1 Analýza úlohy segmentace krevního řečiště

Při evolučním návrhu obrazových filtrů je nutné mít obrázky a odpovídající zlaté standardy, kterých se evoluce snaží dosáhnout. Pro návrh filtrů, jež provádí segmentaci krevního řečiště v obrázcích sítnic, bylo využito snímků a zlatých standardů z databáze DRIVE. Filtry byly evolučně vytvářeny s použitím trénovací sady dvaceti snímků.

Evolučně navržené filtry byly testovány za použití parametrů, které byly opakovanou analýzou shledány jako vhodné pro řešení daného problému. Velikost chromozomu byla 64 uzlů. Byla použita predikce fitness, která byla nastavena tak, aby předčasně ukončovala přibližně 10 % průběhů vyhodnocení kandidátních řešení.

Proběhlo 100000 generací v populaci 128 jedinců. V jednotlivých generacích spolu jedinci navzájem nesoutěžili, k tomu došlo až v poslední generaci, kdy byl jako výsledný filtr použit nejlepší jedinec populace. Experiment se skládal ze 40 takových evolučních běhů. Na obrázku 7.1 jsou tři příklady použití filtrů na segmentaci prvního testovacího snímku z databáze DRIVE.



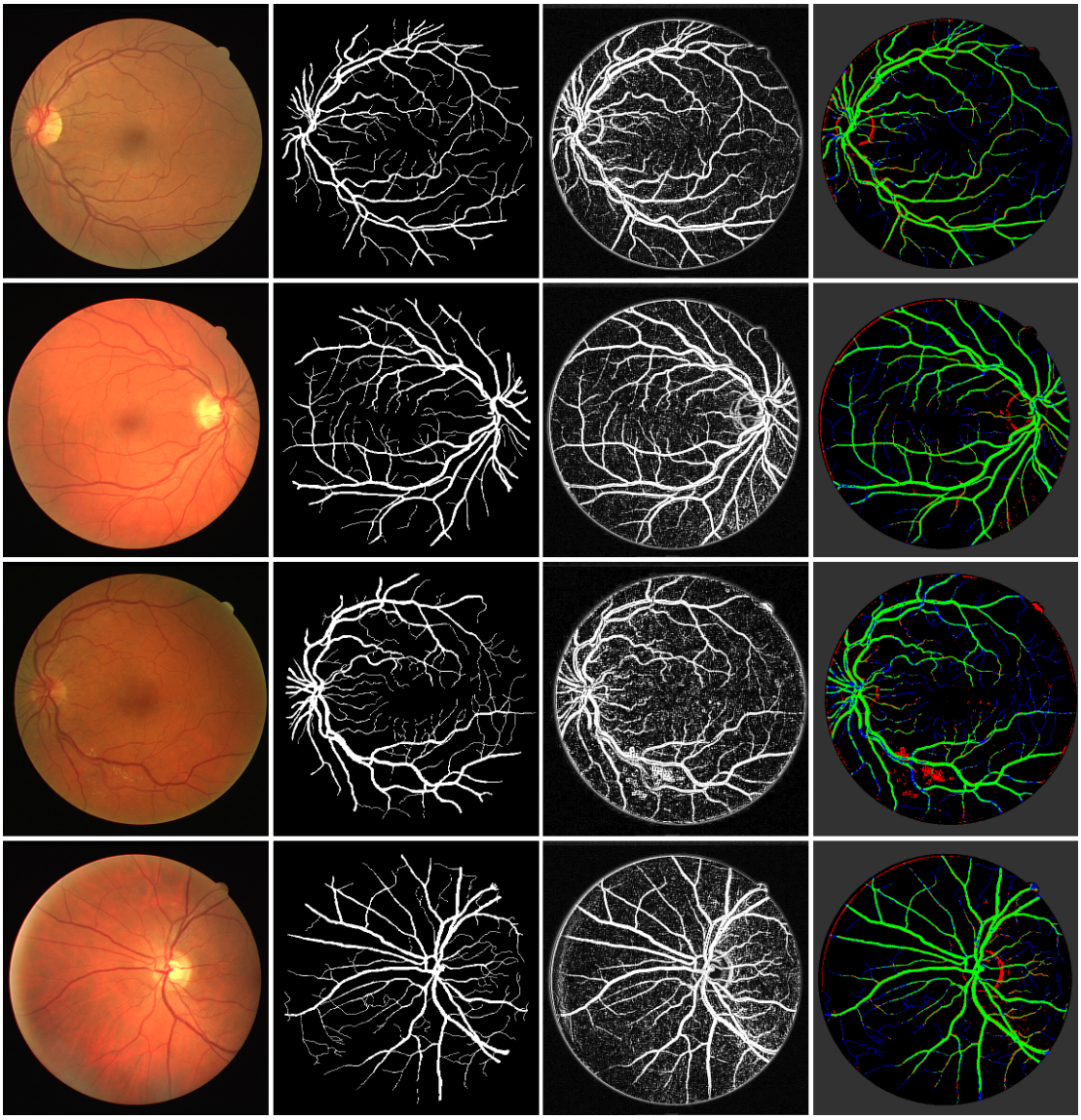
Obrázek 7.1: Vizuální srovnání výsledků tří náhodně vybraných obrazových filtrů. Filtry byly evolučně navrženy v úloze segmentace. Na obrázcích jsou znázorněny účinky jejich aplikace na první snímek databáze DRIVE.

Průměrná přesnost (accuracy) segmentačních filtrů na všech testovacích snímcích byla pouhých 0,859. Tento výsledek byl negativně ovlivněn několika filtry, které dosáhly přesnosti menší než 0,8. Přesnost nejlepšího filtru byla 0,917.

Z výsledků lze vyvodit, že samostatné filtry využívající maximálně 64 funkčních bloků nedokáží spolehlivě segmentovat snímky sítnice oka. Tento výsledek není neočekávaný, neboť úloha segmentace krevního řečiště je velmi složitá, zatímco vytvářené filtry jsou jednoduché. Z tohoto důvodu bylo navrženo vylepšení, které využívá více jednoduchých evolučně navržených filtrů pro vytvoření jediného filtru s lepšími vlastnostmi.

V novém experimentu o stejných parametrech bylo znovu evolučně zpracováno 128 jedinců populace a proběhlo 100000 generací. Tento experiment se od předchozího lišil v tom, že do výsledné sady filtrů bylo vybráno 64 nejlepších jedinců. Na konci všech 40 běhů tedy existovalo celkem 2560 různých filtrů, které byly rozděleny do 40 sad.

Navržené řešení kombinuje výstupy jednotlivých filtrů pro sadu 64 filtrů. Z výstupů všech filtrů v sadě je tedy vytvořen jediný obrázek kombinované segmentace, kde každý filtr mírně přispívá ke světlosti pixelů ve snímku. Výsledná segmentace je stanovena jako prahovaný vážený průměr 3×3 okolí zpracovávaného pixelu v obrázku kombinované segmentace. Výsledky tohoto postupu jsou znázorněny na obrázku 7.2.



Obrázek 7.2: Výsledky segmentace krevního řečiště ve snímku sítnice oka. Segmentace byla učiněna na základě 64 jednoduchých evolučně navržených filtrů. Na řádcích jsou obrázky vztahující se k prvním čtyřem snímkům sítnic databáze DRIVE. Vlevo jsou snímky sítnic. Ve sloupci vlevo uprostřed jsou vzorové segmentace. Ve sloupci vpravo uprostřed jsou obrázky kombinovaných výstupů filtrů. Vpravo jsou vizuálně zobrazené výsledky provedených segmentací.

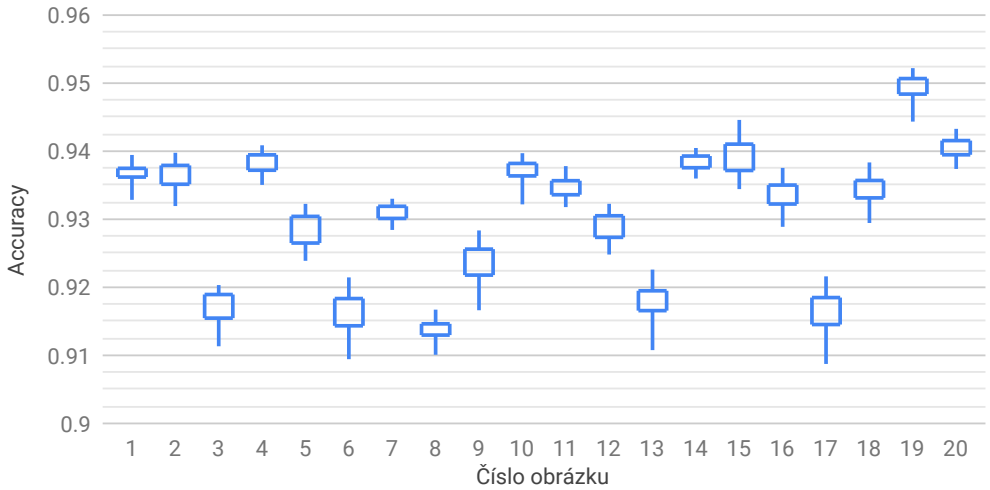
Výsledky experimentů jsou pro metodu kombinující navržené filtry výrazně lepší, než jsou výsledky segmentace samostatných filtrů. Kombinované řešení dosáhlo průměrné hodnoty přesnosti (accuracy) 0,952 pro celý snímek a 0,931 pro oblast zájmu (ROI). Pro jednotlivé testovací obrázky jsou odpovídající naměřené průměrné hodnoty důležitých veličin uvedeny v tabulce 7.1.

V grafu na obrázku 7.3 jsou výsledné hodnoty přesnosti pro testovací snímky srovnány z pohledu jednotlivých běhů. Z grafu lze vyčíst, že se výsledky mezi evolučními běhy příliš nelišily. Minimální průměrná hodnota přesnosti byla 0,95. Maximální průměrná hodnota

Č. Obr.	1	2	3	4	5	6	7	8	9	10
Acc (ROI)	0.937	0.936	0.917	0.938	0.929	0.916	0.931	0.914	0.924	0.94
Accuracy	0.957	0.957	0.943	0.957	0.951	0.942	0.952	0.941	0.947	0.96
Precision	0.86	0.917	0.864	0.883	0.914	0.923	0.851	0.789	0.955	0.86
Sensitivity	0.619	0.633	0.51	0.62	0.523	0.444	0.581	0.428	0.367	0.57
Sp (ROI)	0.985	0.99	0.986	0.987	0.992	0.994	0.984	0.984	0.998	0.99
Specificity	0.99	0.993	0.991	0.992	0.995	0.996	0.99	0.989	0.998	0.99
Č. Obr.	11	12	13	14	15	16	17	18	19	20
Acc (ROI)	0.935	0.929	0.918	0.938	0.939	0.934	0.916	0.934	0.949	0.94
Accuracy	0.955	0.951	0.943	0.958	0.958	0.954	0.943	0.955	0.965	0.96
Precision	0.858	0.853	0.917	0.815	0.714	0.92	0.929	0.861	0.865	0.84
Sensitivity	0.593	0.522	0.461	0.617	0.691	0.54	0.348	0.512	0.685	0.55
Sp (ROI)	0.985	0.987	0.993	0.981	0.968	0.993	0.996	0.989	0.985	0.99
Specificity	0.99	0.991	0.995	0.988	0.979	0.995	0.998	0.993	0.99	0.99

Tabulka 7.1: Srovnání průměrných výsledků segmentace jednotlivých obrázků metodou kombinující sadu filtrů. Vzhledem k zaužívanosti jsou v tabulce uvedeny anglické termíny.

přesnosti byla 0,954. Lze tedy konstatovat, že je uvedeným způsobem dosahováno dobrých výsledků [25]. Nejmodernější segmentační nástroje dosahují i přesností 0,98, jedná se ovšem o velmi propracované a často i komerční algoritmy.



Obrázek 7.3: Graf srovnávající výsledky segmentací testovacích obrázků. Segmentace byla učiněna na základě 64 jednoduchých evolučně navržených filtrů. Bylo provedeno 40 evolučních běhů, a v každém z nich se vygenerovalo 128 jednoduchých filtrů, ze kterých bylo vždy vybráno nejlepších 64. Evoluce trvala 100000 generací. Pro trénovaní bylo použito 1024 vstupních vektorů. Krabicový graf zachycuje maximum, minimum, 3. a 1. kvartil.

Na základě provedených experimentů lze usoudit, že evolucí navrhované filtry mohou být slučovány do větších celků, které tímto dosahují mnohem lepších výsledků než jednotlivé filtry samostatně. Důvodem je skutečnost, že různé evolučně navržené filtry využívají rozdílných závislostí vstupních hodnot při výpočtu svých výstupů. Jeden filtr může pro detekci cév například klást velký důraz na světlost vstupních pixelů, ale jiný filtr může

k detekci využívat spíše rozdílů hodnot pixelů. Výsledné filtry kombinují velké množství různých závislostí vstupů.

Pro zajímavost je na obrázku 7.4 ukázka aplikace sady segmentačních filtrů na obrázek, jenž není snímkem sítnice lidského oka. Výstup každého z filtrů přispívá ke světlosti pixelů ve výsledném snímku. Je vidět, že filtry do určité míry provádí detekci hran. Nabízí se využití takovýchto filtrů pro umělecké účely.

Obrázek 7.4: Ukázka aplikování sady segmentačních filtrů na obrázek, který není snímkem sítnice. Vlevo originální obrázek – Lena. Vpravo obrázek filtrovaný.

7.1.2 Vyhodnocení rychlosti implementace

Experimentálně bylo naměřeno, že program vyhodnocuje v průměru 38000 kandidátních řešení za vteřinu. Tohoto výsledku bylo dosaženo pro 15872 vstupních vektorů, což pro srovnání přibližně odpovídá vyhodnocování trénovacího obrázku o velikosti 128 × 128 pixelů. Počet vstupů byl 25, což odpovídá klouzavému oknu 5 × 5. Počet uzlů byl nastaven na 32. Experiment proběhl na jediném jádře procesoru Intel Core i5-4590 s frekvencí 3,3 GHz.

Implementace je 6krát rychlejší, než obdobný algoritmus, který byl akcelerován za použití FPGA o frekvenci 100 MHz [20], a 270krát rychlejší než obdobná implementace na PC využívající Celeron 2,4 GHz [22], jež ale nevyužívá akceleračních metod implementovaných v této práci.

7.2 Evoluční návrh číslicových obvodů

V předchozích kapitolách byly uvedeny metody, jejichž použití v CGP by mělo akcelerovat evoluci logických obvodů. Vytváření logických obvodů je velmi obtížný problém, neboť se stoupajícím počtem vstupů obvodu exponenciálně roste množství kombinací, které je potřeba simulačně vyhodnotit. Současně se zvětšuje počet uzlů potřebných pro sestavení daného obvodu, což vede na potřebu zvolit větší velikost chromozomu, a to dále snižuje rychlost evoluce.

Experimenty budou zaměřeny na jednu z nejtěžších úloh, kterou je evoluce násobiček [19]. Tato práce se věnuje pouze generování nových násobiček a nezabývá se již jejich další optimalizací. Násobičky jsou sestavovány z dvouvstupových hradel AND a NAND.

Všechny testy proběhly v programech, které byly implementovány v této práci. Programy pro evoluční návrh logických obvodů využívají následujících akceleračních metod. V programech je použita paralelní simulace, která využívá instrukcí AVX2 k práci s vektory o velikosti 256 bitů. Do programů bylo zahrnuto přímé použití strojového kódu, tedy při převodu chromozomu na fenotyp vzniká spustitelný kód daného fenotypu. Program využívá paralelizace v populaci, kdy evaluace různých jedinců může probíhat simultánně. Testy proběhly na procesoru Intel Core i5-4590, který má čtyři jádra. Paralelně tedy byli vyhodnocováni 4 jedinci současně. V programech byla implementována metoda dávkové mutace, která převádí problém hledání pozitivní mutace na problém hledání optimálního zapojení určitého uzlu. Metoda dávkové mutace byla rozšířena metodou komprese fitness a metodou recyklace hodnot.

7.2.1 Vliv výběru mutovaného uzlu na úspěšnost evoluce

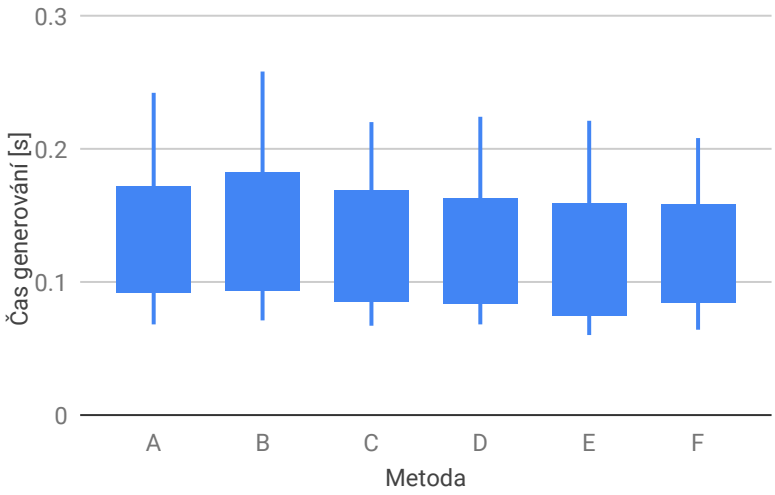
V metodě dávkové mutace je třeba náhodně vybírat uzly, které budou dále mutovány. Jelikož byla metoda dávkové mutace implementována i s rozšiřující metodou recyklace hodnot, výběrem uzlu se také stanovuje, které uzly se budou nacházet v intervalu mutací. Pro velký interval mutací by mohlo dojít k tomu, že budou neustále mutovány uzly, které se nacházejí v zadní části chromozomu, a uzly nacházející se v přední části budou do jisté míry ignorovány. Mohlo by tedy být výhodné při náhodném výběru lehce upřednostňovat uzly nacházející se v přední části. Lze ovšem dojít i k opačnému závěru, a to proto, že na uzly přední části je navázáno množství dalších uzlů, tudíž by nemusela jejich častá mutace vést k rychlejšímu nalezení řešení.

K otestování zmíněných hypotéz bylo vytvořeno šest metod výběru mutovaného uzlu. Metody byly označeny znaky *A*–*F*. Metoda *F* je standardní náhodný výběr uzlu, kdy může být vybrán kterýkoli uzel se stejnou pravděpodobností. Metody *A*–*E* při výběru upřednostňují určitou oblast v chromozomu. Metoda *A* zvyšuje pravděpodobnost výběru uzlů ze přední části a naopak metoda *E* z části zadní. Metoda *B* upřednostňuje uzly v okolí první čtvrtiny, *C* poloviny a *D* třetí čtvrtiny. Tohoto efektu je dosaženo aplikací normálního rozdělení.

Hypotézy byly testovány za použití parametrů, které byly opakovanou analýzou shledány jako optimální pro řešení daného problému. Velikost chromozomu byla 625 uzlů. Interval mutací byl zvolen 16 při použití fáze promíchání uzlů. Byla nastavena pravá priorita komprese fitness. Chromozom byl na začátku každého evolučního běhu prázdný. Byla použita verze přístupu k paměti *V1*.

Výsledky experimentů jsou vidět na obrázku 7.5. Metody *A*–*F* byly srovnány z hlediska doby potřebné pro vygenerování plně funkční násobičky 4×4b. Bylo provedeno 200 běhů pro každou variantu. Nejlepšího výsledku dosáhla metoda *E*, která generovala násobičky v průměru za 0,129 vteřin. Medián potřebného času byl 0,104 vteřin.

Výsledky experimentu jsou pro jednotlivé metody velmi vyrovnané. Nejrychlejší byla evoluce při výběru, který k mutování upřednostňoval uzly v zadní části chromozomu. Rychlá byla ovšem i metoda *F*, která vybírala uzly rovnoměrně. Vzhledem k nízkým rozdílům těchto výsledků lze konstatovat, že upřednostňování určitých uzlů při výběru uzlu k mutaci metodou dávkové mutace nevede ke zrychlení evolučního návrhu násobiček. Důvodem by mohly být náhodně generované uzly, které jsou vždy vytvořeny před mutovaným uzlem.



Obrázek 7.5: Graf srovnávající různé metody volby mutovaného uzlu v metodě dávkové mutace. Metody jsou v grafu srovnávány z hlediska doby potřebné pro vygenerování plně funkční násobičky 4×4b. Počet běhů byl 200 pro každou variantu. Krabicový diagram zachycuje 90. a 10. percentil a 3. a 1. kvartil.

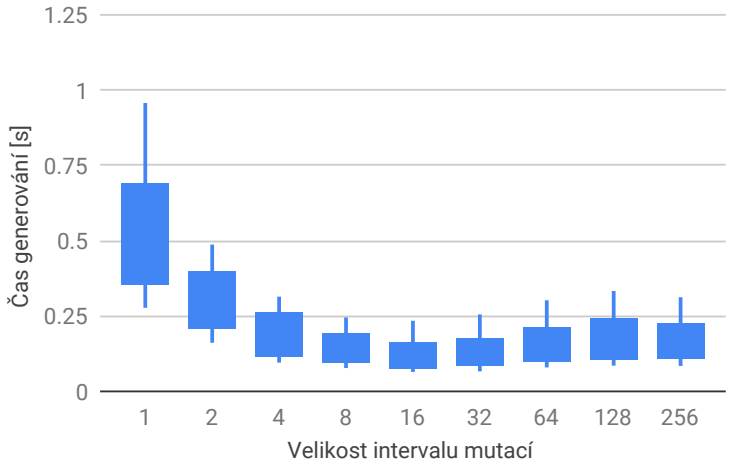
Tedy nehledě na to, který uzel je mutován, existuje vždy velké množství nových uzlů, na jejichž výstupy se může napojit. Tímto patrně dochází ke snížení rozdílů při výběru mutovaného uzlu.

7.2.2 Vliv velikosti intervalu mutací na úspěšnost evoluce

Metoda recyklace hodnot rozšiřující metodu dávkové mutace zavádí do algoritmu CGP nový parametr. Tímto parametrem je velikost intervalu mutací. Parametr udává počet po sobě jdoucích uzlů, které budou postupně mutovány. Při těchto mutacích není nutné simulovat celý fenotyp a také není prováděna fáze generování nových uzlů a fáze přeskládávání kódujících uzlů. Pokud by byla zvolena příliš malá velikost intervalu mutací, bude program často vykonávat simulační fázi a tím se sníží rychlost evoluce. Naopak pro příliš velkou velikost intervalu mutací hrozí to, že přílišná absence fáze generování nových uzlů a fáze přeskládávání kódujících uzlů povede ke zhoršení schopnosti nalézat optimální propojení uzlů.

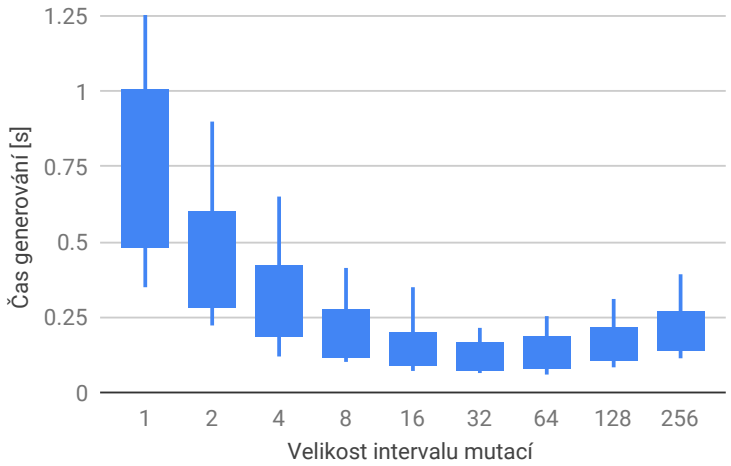
Správné nastavení parametru velikosti intervalu mutací bylo testováno za použití parametrů, které byly opakovanou analýzou shledány jako optimální pro řešení daného problému. Velikost chromozomu byla 625 uzlů. Výběr mutovaného uzlu byl čistě náhodný. Byla nastavena pravá priorita komprese fitness. Chromozom byl na začátku každého evolučního běhu prázdný. Byla použita verze přístupu k paměti *V1*. V každé iteraci evoluce byla uplatněna fáze přeskládávání kódujících uzlů.

Program byl testován s následujícími nastaveními parametru velikosti intervalu mutací: 1, 2, 4, 8, 16, 32, 64, 128 a 256. Výsledky testů jsou vidět na obrázku 7.6. Každé nastavení parametru bylo testováno ve 100 evolučních bězích. Pro každý běh byl změřen čas potřebný pro vygenerování plně funkční násobičky 4×4b. Nejlepších výsledků program dosahoval s nastavením parametru velikosti intervalu mutací na 16, kdy byly násobičky generovány v průměru za 0,1323 vteřin a medián byl 0,115 vteřin.



Obrázek 7.6: Graf srovnávající časy potřebné pro vygenerování plně funkčních násobiček 4×4b pro různé velikosti intervalu mutací při použití metody *T0*. Metoda *T0* značí využití promíchávání uzlů chromozomu. Počet běhů byl 100 pro každou variantu. Krabicový diagram zachycuje 90. a 10. percentil a 3. a 1. kvartil.

Optimální velikosti intervalu mutací závisí na tom, zda byla při evoluci uplatněna fáze přeskládávání kódujících uzlů. Při přeskládávání totiž dochází k úbytku uzlů zadní části chromozomu. Pro zjištění dopadu fáze přeskládávání kódujících uzlů na rychlost evoluce byl proto proveden test se stejnými parametry jako výše, avšak fáze přeskládávání kódujících uzlů byla z algoritmu odstraněna. Výsledky tohoto testu jsou na obrázku 7.7. Nejlepších výsledků v tomto testu bylo dosaženo při nastavení velikosti intervalu mutací na 32, kdy byl průměrný čas generování 0,1374 vteřin a medián 0,107 vteřin.

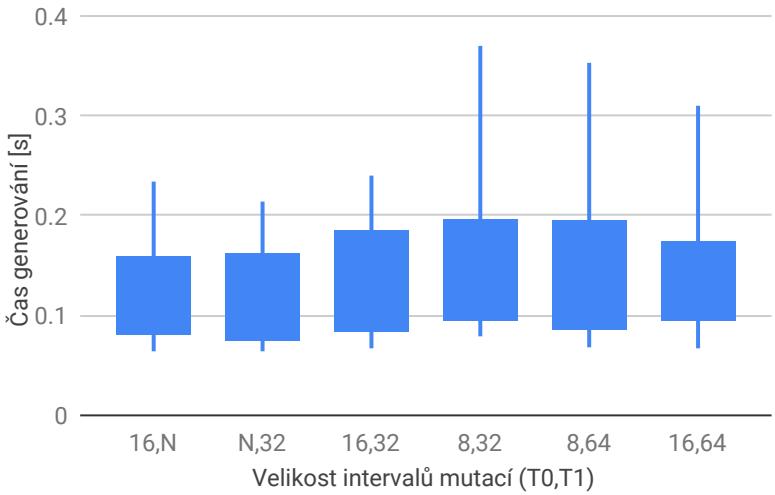


Obrázek 7.7: Graf srovnávající časy potřebné pro vygenerování plně funkčních násobiček 4×4b pro různé velikosti intervalu mutací při použití metody *T1*. Metoda *T1* přeskakuje fázi promíchávání uzlů chromozomu. Počet běhů byl 100 pro každou variantu. Krabicový diagram zachycuje 90. a 10. percentil a 3. a 1. kvartil.

Nejlepší výsledky z obou testů jsou velmi podobné, z čehož lze usoudit, že fáze přeskládávání kódujících uzlů nemá příliš velký vliv na rychlost evoluce v problému generování

funkčních násobiček 4×4b. Parametr velikosti intervalu mutací naopak má značný vliv na rychlost evoluce, což dokazují velké rozdíly naměřených časů pro jeho různá nastavení.

V posledním testu velikostí intervalu mutací byl proveden pokus spojit metodu používající fázi přeskládávání kódujících uzlů s metodou, která tuto fázi v programu přeskakuje. Metody byly označeny *T0* a *T1*, kde metoda *T0* využívá promíchávání a metoda *T1* ne. Každá z metod měla vlastní parametr velikosti intervalu mutací. Při experimentu byla v každé iteraci metoda *T0*, nebo *T1* vybrána náhodně. Hypotézou pro tento pokus byla možnost, že by spojením daných metod došlo ke zrychlení evoluce. Výsledky experimentu jsou na obrázku 7.8.



Obrázek 7.8: Graf srovnávající časy potřebné pro vygenerování plně funkčních násobiček 4×4b pro různé kombinace metod *T0* a *T1*. Metoda *T0* značí využití promíchávání uzlů chromozomu. Metoda *T1* přeskakuje fázi promíchávání uzlů chromozomu. V každé iteraci metody dávkové mutace je náhodně vybrána jedna z těchto metod. Metoda využívá uvedených intervalů mutací. Znak N značí nepoužití metody. Počet běhů byl 100 pro každou variantu. Krabicový diagram zachycuje 90. a 10. percentil a 3. a 1. kvartil.

Spojením metod *T0* a *T1* došlo ke zhoršení výsledků ve všech čtyřech různých nastaveních velikostí intervalů mutací. Při implementaci metody dávkové mutace je tedy vhodné zvolit pouze jednu z těchto metod a experimentálně pro ni nalézt optimální nastavení velikosti intervalu mutací. Jednodušší je tedy fázi přeskládávání vůbec neimplementovat. Dosažené výsledky lze ovšem aplikovat pouze na úlohu generování násobiček 4×4b a pro jiné problémy by tato fáze mohla vést k urychlení. Při porovnání grafů lze také shledat, že při použití fáze přeskládávání bylo dosaženo lepších výsledků pro nižší velikosti intervalu mutací.

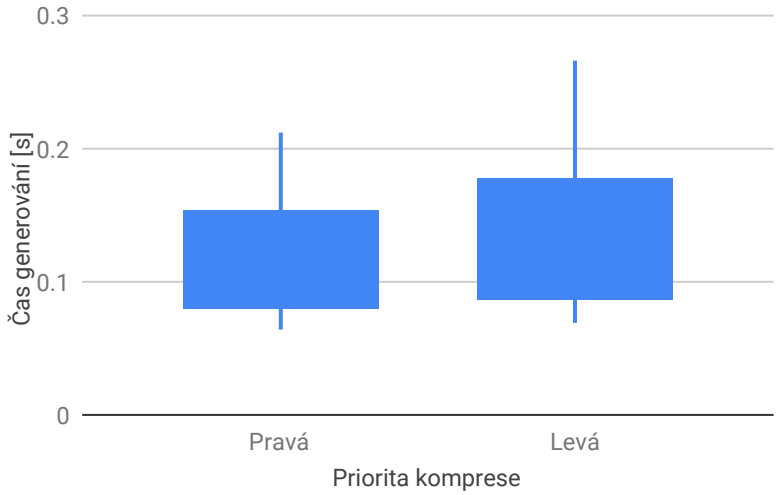
7.2.3 Vliv různých priorit výstupů při kompresi fitness na úspěšnost evoluce

Vedlejším efektem použití metody komprese fitness je vytvoření určité priority řešení některých primárních výstupů obvodu před jinými výstupy. Tato priorita vzniká z důvodu, že při výpočtu vektoru požadovaného vstupu mutovaného uzlu jsou zaznamenány potřebné bity výstupů v určitém pořadí a nalezené bity se dále nepřepisují.

Existují dva základní postupy při průchodu výstupy logického obvodu. Výstupy mohou být procházeny vzestupně, nebo sestupně dle svého indexu. Tyto postupy budou označeny pravá a levá priorita komprese, kde pravá priorita upřednostňuje vyřešení nejméně významných bitů výstupu obvodu a levá priorita upřednostňuje vyřešení nejvíce významných bitů výstupu.

Priority komprese byly testovány za použití parametrů, které byly opakovanou analýzou shledány jako optimální pro řešení daného problému. Velikost chromozomu byla 625 uzlů. Výběr mutovaného uzlu byl čistě náhodný. Interval mutací byl zvolen 16 při použití fáze promíchání uzlů. Chromozom byl na začátku každého evolučního běhu prázdný. Byla použita verze přístupu k paměti *V1*.

Výsledky testu vlivu pravé a levé priority komprese jsou vidět na obrázku 7.9. K otestování pravé i levé priority bylo provedeno 1000 běhů evolučního algoritmu s cílem generovat plně funkční násobičky 4×4b. Časy běhů byly zaznamenány. Při použití pravé priority program generoval nové násobičky s průměrnou rychlostí 0,1311 vteřiny. Medián byl 0,111 vteřiny. Pro levou prioritu byl průměrný čas 0,1507 vteřiny a medián 0,126 vteřiny.



Obrázek 7.9: Graf srovnávající dvě možné implementace priority výstupů komprese v úloze generovaní plně funkčních násobiček 4×4b. Pravá priorita komprese upřednostňuje vyřešení nejméně významných bitů výstupu násobičky. Levá priorita komprese upřednostňuje vyřešení nejvíce významných bitů výstupu. Počet běhů byl 1000 pro každou variantu. Krabicový diagram zachycuje 90. a 10. percentil a 3. a 1. kvartil.

Z výsledků je patrné, že v úloze automatického návrhu násobiček je rychlejší pravá priorita. U výstupů násobiček jsou hodnoty nejvíce významných bitů závislé na nejvíce vstupech. Je možné, že použití pravé priority, která upřednostňuje méně významné bity, je v úloze násobiček přirozené. Struktura uzlů vytvořená ve chromozomu pro řešení méně významných bitů muže být totiž využita při řešení bitů s vyšší významností.

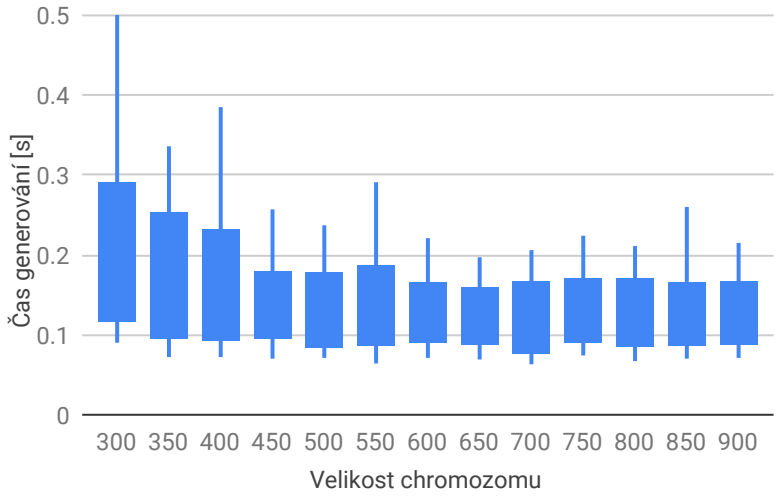
7.2.4 Vliv velikosti chromozomu na úspěšnost evoluce

Jedním z nejvýznamnějších parametrů CGP rozšířeného o metodu dávkové mutace je velikost chromozomu, která značí počet uzlů, které bude mít evoluční program při běhu k dispozici. Při použití dávkové mutace je důležité zvolit velikost chromozomu dostatečně velkou, neboť rozmanitost kandidátních řešení do určité míry obstarávají náhodně genero-

vané uzly, jejichž množství je určeno právě velikostí chromozomu. Větší chromozom také znamená, že v porovnávací fázi bude k dispozici více uzlů na propojení s mutovaným uzlem. Více porovnání zvyšuje šanci nalézt optimální propojení. Pokud by ale byla velikost chromozomu příliš velká, tak by vyhodnocení jedné dávkové mutace trvalo příliš dlouho.

Nejlepší nastavení velikosti chromozomu bylo testováno při použití parametrů, které byly opakovanou analýzou shledány jako optimální pro řešení daného problému. Výběr mutovaného uzlu byl čistě náhodný. Interval mutací byl zvolen 16 při použití fáze promíchání uzlů. Byla nastavena pravá priorita komprese fitness. Chromozom byl na začátku každého evolučního běhu prázdný. Byla použita verze přístupu k paměti *V1*.

Byla otestována nastavení velikosti chromozomu v rozmezí od 300 do 900 s krokem 50. Výsledky jsou zobrazeny na obrázku 7.10. Pro každou variantu proběhlo 100 běhů, ve kterých byly měřeny časy potřebné k vygenerování nových funkčních násobiček 4×4b. Nejlepšího výsledku bylo dosaženo při velikosti chromozomu 650. S tímto nastavením byla průměrná doba pro generování 0,1349 vteřiny a medián 0,11 vteřiny.



Obrázek 7.10: Graf závislosti velikosti chromozomu na času potřebném pro nalezení funkční násobičky 4×4b. Počet běhů byl 100 pro každou variantu. Krabicový diagram zachycuje 90. a 10. percentil a 3. a 1. kvartil.

Z výsledků lze vyčíst platnost hypotézy, jež říká, že v metodě dávkové mutace je důležité povolit dostatečně velký chromozom. Pokud je velikost chromozomu 625, tak mají nalezené násobičky průměrnou velikost 352 hradel. Přibližně polovina uzlů je tedy nekódujících. Optimální velikost chromozomu se z principu liší pro úlohy různé velikosti. S každým přidaným bitem na vstupu obvodu je vhodné velikost chromozomu přibližně zdvojnásobit.

7.2.5 Vliv různých přístupů při ukládání mezivýsledků na úspěšnost evoluce

Při simulaci kandidátního řešení jsou všechny mezivýsledky výstupů uzlů ukládány do paměti. Jelikož počty kombinací vstupních hodnot pro větší obvody přesahují velikosti AVX2 vektorů, je nutné spustit simulaci vícekrát, aby byly všechny vstupní kombinace vyzkoušeny. Je ale otázkou, zda je lepší ukládat vektory za sebe tak, jak simulace prochází fenotypem, nebo zda by měly být uloženy pohromadě odpovídající vektory z těchto různých spuštění.

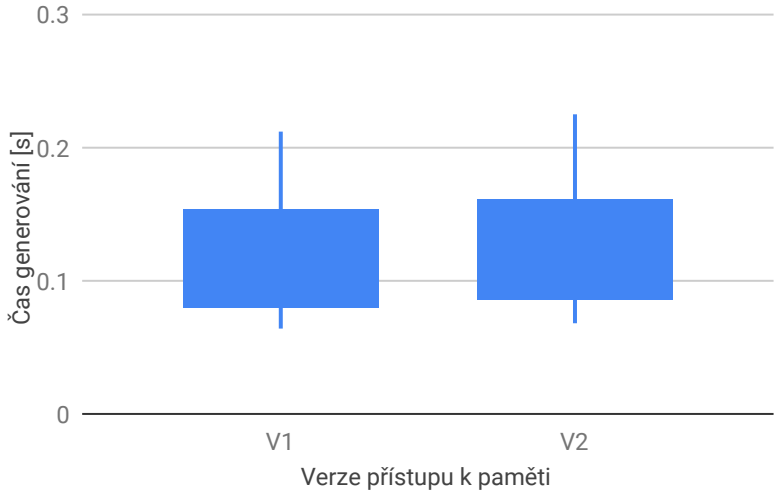
Zmíněné dvě verze přístupu uložení vektorů do paměti budou označeny jako *V1* a *V2*. Verze *V1* ukládá vektory za sebe v průběhu každého simulačního běhu vyhodnocujícího odezvu na skupinu vektorů vstupních kombinací. Tímto je zvýhodněna fáze simulace, ve které je užitečné mít vektory uloženy za sebou. Verze *V2* ukládá za sebe vektory pocházející ze stejných míst ve fenotypu z běhů vyhodnocujících všechny odezvy na vektory vstupních kombinací. Tento přístup zvýhodňuje porovnávací část, kdy je vhodné vyhodnotit jedno propojení uzlů najednou a až poté přejít na další.

Při implementaci verze *V2* lze také využít metody predikce fitness a vyhodnocování propojení ukončit v případě, že dosažená relativní fitness již na počátku nepřekročí určitý práh.

Lze předpokládat, že by metoda *V1* měla dosáhnout lepších výsledků pro logické obvody, které mají menší, nebo stejný počet vstupních kombinací, jako je velikost vektorů použitých pro bitovou paralelizaci. Naopak metoda *V2* by měla lépe škálovat, neboť pro větší počty vstupních kombinací umožňuje lépe přistupovat k vyhodnocovaní relativních fitness.

Obě verze přístupu k paměti při ukládání mezivýsledků byly otestovány za použití parametrů, jež byly opakovanou analýzou shledány jako optimální pro řešení daného problému. Velikost chromozomu byla 625 uzlů. Výběr mutovaného uzlu byl čistě náhodný. Interval mutací byl zvolen 16 při použití fáze promíchání uzlů. Byla nastavena pravá priorita komprese fitness. Chromozom byl na začátku každého evolučního běhu prázdný.

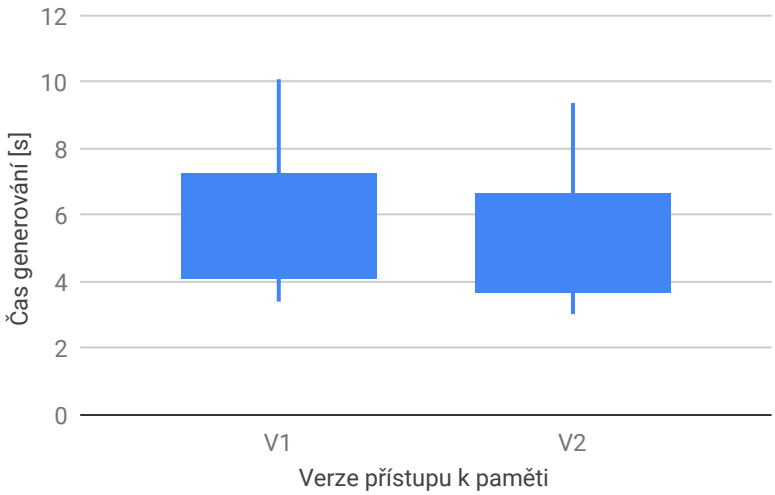
Výsledky prvního testu jsou k dispozici na obrázku 7.11. V prvním testu byly měřeny časy potřebné pro generování násobiček 4×4b. Pro vyhodnocení každé z verzí *V1* a *V2* bylo uskutečněno 1000 běhů. Dle předpokladu dosáhla verze *V1* lepšího výsledku.



Obrázek 7.11: Graf srovnávající dva rozdílné přístupy k ukládání mezivýsledků. Verze označená jako *V1* ukládá mezivýsledky probíhající simulace do paměti za sebe. Verze označená jako *V2* ukládá za sebe odpovídající vektory všech vyhodnocujících běhů. Metody jsou v grafu srovnávány z hlediska doby potřebné pro vygenerování plně funkční násobičky 4×4b. Počet běhů byl 1000 pro každou variantu. Krabicový diagram zachycuje 90. a 10. percentil a 3. a 1. kvartil.

Proběhl také druhý test, jehož výsledky jsou uvedeny na obrázku 7.12. Ve druhém testu byly měřeny časy potřebné pro generování násobiček 5×5b. Pro vyhodnocení každé z verzí *V1* a *V2* bylo uskutečněno 2000 běhů. Dle předpokladu dosáhla verze *V2* lepšího výsledku. Konkrétně vytvářel program, který využíval verzi *V2*, násobičky 5×5b s průměrnou rychlostí

5,801 vteřin. Medián byl 4,8 vteřiny. Program s verzí *V1* měl průměrnou rychlost 6,3051 vteřin a medián 5,292 vteřin.



Obrázek 7.12: Graf srovnávající dva rozdílné přístupy k ukládání mezivýsledků. Verze označená jako *V1* ukládá mezivýsledky probíhající simulace do paměti za sebe. Verze označená jako *V2* ukládá za sebe odpovídající vektory všech vyhodnocujících běhů. Metody jsou v grafu srovnávány z hlediska doby potřebné pro vygenerování plně funkční násobičky 5×5b. Počet běhů byl 2000 pro každou variantu. Krabicový diagram zachycuje 90. a 10. percentil a 3. a 1. kvartil.

Na základě výsledků uvedených experimentů lze usoudit, že způsob ukládání mezivýsledků do paměti označený jako *V2* je vhodný pro řešení velkých problémů. V prvním testu byly evolučně navrhovány násobičky 4×4b, a počet vstupních kombinací byl proto roven velikosti použitých vektorů AVX2. V tomto testu se tedy neuplatnila predikce relativní fitness, protože k jejímu výpočtu byl použit pouze jediný vektor. Ve druhém testu byly pro výpočet relativní fitness potřeba čtyři vektory, a proto mohl být výpočet předčasně zastaven již na základě výsledku prvního vektoru.

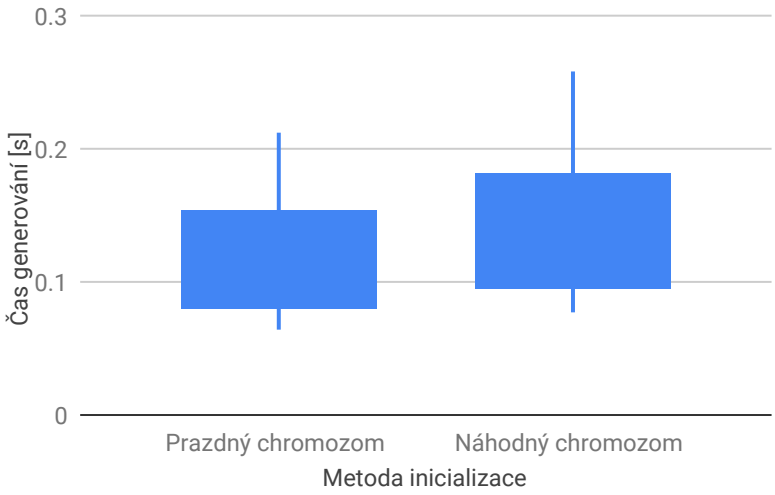
7.2.6 Vliv inicializace chromozomu na úspěšnost evoluce

Při rozšíření CGP o metodu dávkové mutace není nutné chromozom inicializovat, neboť na začátku evoluce jsou automaticky vygenerovány náhodné nekódující uzly, na které se mohou výstupy kandidátního řešení připojit. Otázkou je, zda je vhodnější náhodnou inicializaci přesto provést, nebo zda je lepší začínat vždy s prázdným chromozomem. Z tohoto důvodu byly vytvořeny experimenty, které možné způsoby inicializace vyhodnotí.

Oba způsoby inicializace chromozomu byly otestovány za použití parametrů, jež byly opakovanou analýzou shledány jako optimální pro řešení daného problému. Velikost chromozomu byla 625 uzlů. Výběr mutovaného uzlu byl čistě náhodný. Interval mutací byl zvolen 16 při použití fáze promíchání uzlů. Byla nastavena pravá priorita komprese fitness. Byla použita verze přístupu k paměti *V1*.

Výsledky experimentů jsou uvedeny na obrázku 7.13. Obě metody inicializace byly prověřeny tisíci běhy, ve kterých byly generovány plně funkční násobičky 4×4b. Nejlepšího výsledku dosáhla metoda inicializace prázdným chromozomem. Ta dosáhla průměrné doby

generování 0,1311 vteřiny a mediánu 0,111 vteřiny, zatímco druhá metoda dosáhla průměrné doby 0,1544 vteřiny a mediánu 0,129 vteřiny.



Obrázek 7.13: Graf srovnávající dvě různé metody inicializace chromozomu. Evoluce může začínat s prázdným chromozomem, kdy je počet kódujících uzlů na počátku nulový. Evoluce může také začínat s náhodným chromozomem, kdy jsou uzly náhodně propojeny a počet kódujících uzlů je kladný. Metody jsou v grafu srovnávány z hlediska doby potřebné pro vygenerování plně funkční násobičky 4×4b. Počet běhů byl 1000 pro každou variantu. Krabicový diagram zachycuje 90. a 10. percentil a 3. a 1. kvartil.

Výsledky experimentu ukazují, že při použití metody dávkové mutace je na počátku každého evolučního běhu lepší ponechat chromozom prázdný, než jej náhodně inicializovat. Důvodem je pravděpodobně fakt, že při náhodné inicializaci musí chromozom prvotní strukturu propojení uzlů přizpůsobovat dané úloze, zatímco pro prázdný chromozom je přímo vytvářena propojovací struktura nová.

7.2.7 Vliv různého počtu vstupů na evoluci násobiček

Jedním z cílů této práce bylo vytvořit program, který by byl schopen velmi rychle řešit problém evolučního návrhu logických obvodů pomocí CGP. Jedním z nejtěžších problémů této domény je navrhování násobiček, a proto byly právě násobičky použity při vyhodnocování implementovaných programů. Kombinací metod popsaných v této práci lze vytvořit program, jenž dokáže navrhovat i násobičky o rozměru 6×6b.

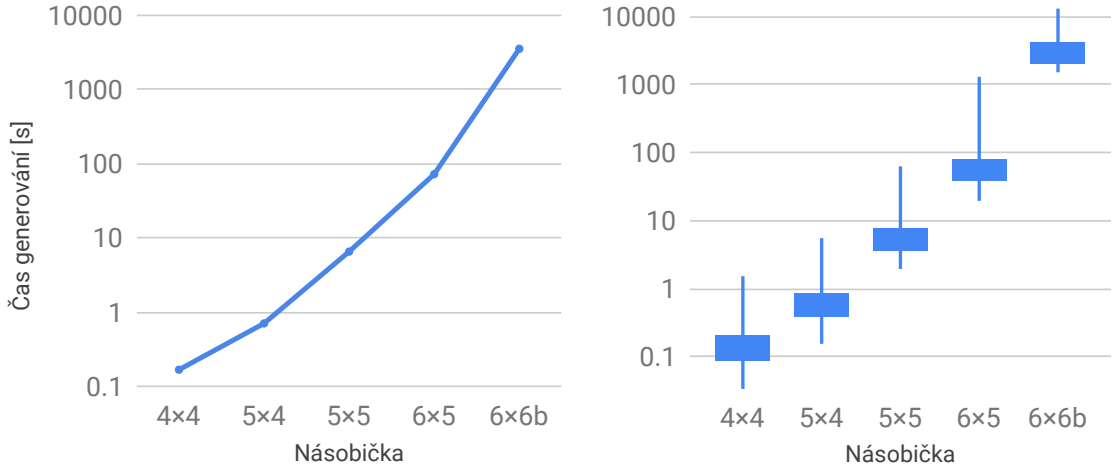
Návrh násobiček s různým počtem vstupů byl testován za použití parametrů, které byly opakovanou analýzou shledány jako dobré pro řešení daného problému. Velikost chromozomu byla 625 uzlů pro násobičky 4×4b. Při každém navýšení vstupu o jeden bit byla tato hodnota zdvojnásobena. Výběr mutovaného uzlu preferoval přední uzly. Interval mutací byl zvolen 10 při použití fáze promíchání uzlů a 100 bez použití promíchání. Metody volby uzlu byly náhodně vybírány. Byla nastavena pravá priorita komprese fitness. Chromozom byl na začátku každého evolučního běhu prázdný. Byla použita verze přístupu k paměti *V1*.

Výsledky experimentů jsou uvedeny na obrázku 7.14. Konkrétní hodnoty obsahuje tabulka 7.2.

Z výsledků lze usoudit, že program pro automatický návrh násobiček pracuje skutečně velmi rychle. Program nachází nové násobičky 6×6b v průměru za hodinu. Vzhledem k

[s]	4×4b	5×4b	5×5b	6×5b	6×6b
Průměr	0,169	0,711	6,582	72,64	3546,46
Medián	0,131	0,555	5,203	52,997	2924,99

Tabulka 7.2: Výsledky ukazující průměr a medián času potřebného pro vygenerování nových násobiček o uvedených velikostech. Všechny hodnoty jsou v sekundách.



Obrázek 7.14: Grafy srovnávající časy potřebné pro vygenerování násobiček s různými počty vstupů. Grafy jsou v logaritmickém měřítku. Vlevo je graf ukazující průměrnou dobu potřebnou pro nalezení nové násobičky. Vpravo je krabicový graf ukazující další statistické údaje generování nových násobiček. Grafy byly vytvořeny na základě 1000 běhů pro každou násobičku vyjma velikosti 6×6b, pro kterou byly k dispozici pouze výsledky 60 běhů. Krabicový diagram zachycuje maximum, minimum, 3. a 1. kvartil.

časové náročnosti byly uvedené hodnoty naměřeny již v průběhu tvorby této práce. Při dalším experimentování byly parametry dále drobně upravovány. Průměrná doba potřebná pro nalezení nové násobičky 5×5b byla 6,582 vteřiny. V podkapitole 7.2.5 se tento výsledek podařilo vylepšit na 5,801 vteřiny.

7.2.8 Srovnání rychlosti implementace

Experimentálně bylo naměřeno, že program evolučně navrhuje nové násobičky 4×4b s průměrnou rychlostí 0,131 vteřin. Výsledku bylo dosaženo na procesoru Intel Core i5-4590 s frekvencí 3,3 GHz. Tento výsledek je přibližně 6200krát rychlejší než výsledek 817 vteřin dosažený na procesoru E5-2670 s využitím běžného CGP s jednou populací [5]. V obou případech algoritmus proběhl na 4 vláknech.

Kapitola 8

Závěr

Cílem této diplomové práce bylo navrhnout a implementovat optimalizace, které by urychlily kartézské genetické programování (CGP) v úloze symbolické regrese. Výsledkem této práce jsou dva programy, které demonstrují účinnost použitých optimalizačních metod.

První program používá CGP pro tvorbu obrazových filtrů, a to konkrétně filtru pro segmentaci krevního řečiště v obrázku sítnice lidského oka. Program využívá instrukce AVX2 pro simultánní vyhodnocení 32 vstupních vektorů. Program byl také urychlen implementací akcelerátoru, který generuje nativní kód pro jednotlivé chromozomy. Vyhodnocování chromozomů je tak výrazně urychleno, neboť pro různé vstupní vektory nemusí být chromozom opětovně interpretován. Dalšími optimalizacemi jsou predikce fitness a případné předčasné ukončení vyhodnocování chromozomu, využití všech registrů procesoru pro ukládání mezivýsledků funkcí fenotypu, paralelizovaný přístup při vyhodnocování populace. Všechny uvedené optimalizace přispěly k výsledné rychlosti prvního programu, který evolučně navrhuje obrazové filtry, a vyhodnocuje pro 15872 vstupních vektorů 38000 kandidátních řešení za vteřinu.

Druhý program využívá CGP pro tvorbu logických obvodů. Tento program je také akcelerován s využitím vektorového paralelismu. Instrukce AVX2 umožňují bitově paralelní simulaci pracující s 256 vstupními vektory najednou. V programu je implementována nová optimalizační metoda navržená v této diplomové práci. Tuto optimalizační metodu jsem pojmenoval metoda dávkových mutací, neboť její implementací lze jedním průchodem chromozomu získat relativní fitness hodnoty odpovídající velkému množství mutací pro jedno propojení uvnitř daného chromozomu. Metoda dávkových mutací přibližně odpovídá základnímu CGP s populací v řádu tisíců, kdy jsou jedinci populace vygenerováni jako všechny možné varianty jedné bodové mutace rodiče. Implementace této metody, jež je rozšířená o metodu komprese fitness vektorů výstupu, urychluje evoluci obvodů i tisíckrát v závislosti na velikosti problému. Kombinací těchto metod se dosahuje doby návrhu jedné plně funkční pětibitové násobičky v průměru 5,8 vteřin na procesoru Intel Core i5-4590, což považuji za skvělý výsledek.

Cílem výkladové části této diplomové práce bylo seznámit čtenáře s kartézským genetickým programováním a jeho využitím v úloze symbolické regrese, dále vyhodnotit možnosti využití instrukčních sad SSE a AVX pro potřeby akcelerace výpočtů pracujících nad celými čísly. Tento cíl práce naplňuje v první, druhé, třetí a čtvrté kapitole. Cílem bylo také navrhnout různé optimalizace umožňující akcelerovat CGP v úloze symbolické regrese. Tímto se zabývá čtvrtá a pátá kapitola. Dalším cílem bylo popsat implementované programy a vyhodnotit jejich fungování řadou experimentů pro zvolenou aplikaci. Tento cíl je naplněn v šesté kapitole, která je zaměřena na implementaci programů, a v sedmé kapitole, která

je zaměřena na vyhodnocení experimentů. Sedmá kapitola také popisuje různá nastavení parametrů programů, jejich omezení a možné cesty dalšího vývoje.

V této práci jsem navrhl a implementoval řadu optimalizací CGP. Kombinací těchto optimalizací bylo dosaženo značného zrychlení CGP v úloze generování obrazových filtrů i v úloze generování logických obvodů. Za velký přínos této práce považuji vytvoření metody dávkových mutací. V práci byla tato metoda nasazena na úlohu hledání funkčních logických obvodů. Nové obvody byly rychle nalezeny, ale počet funkčních prvků potřebných k jejich sestavení je značný. Tato práce se tedy nezabývala fází zmenšování nalezených obvodů. Implementace této fáze by byla vhodným pokračováním. Z hlediska generování obrazových filtrů lze na práci navázat a pokračovat v oblasti agregace více slabších filtrů do jednoho výsledného silného filtru. Slabé filtry by mělo být možné poskládat do různých hierarchií a tím značně zvýšit přesnost výsledného filtru. Kombinace více filtrů byla v této práci vyzkoušena pouze okrajově, ale i přesto přinesla zřetelné zlepšení výsledků.

Literatura

[1] Ashmore, L.; Miller, J.: *Evolutionary Art with Cartesian Genetic Programming.* Online Report, 2004, [Online; navštíveno 11.01.2019].
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.571.9476&rep=rep1&type=pdf

[2] Chandra, R.; Dagum, L.; Kohr, D.; aj.: *Parallel Programming in OpenMP.* Morgan Kaufmann, 2001, ISBN 1-55860-671-8.

[3] Dvořáček, P.; Sekanina, L.: *Evolutionary Approximation of Edge Detection Circuits. Genetic Programming: 19th European Conference on Genetic Programming.* LNCS 9594, Springer, Cham, str.19-34, 2016, ISBN 978-3-319-30667-4.

[4] Hart, W. E.; Goldbaum, M.; Cote, B.; aj.: *Automated measurement of retinal vascular tortuosity.* Proceedings of the AMIA Fall Conference, 1997.

[5] Hrbáček, R.; Sekanina, L.: *Towards Highly Optimized Cartesian Genetic Programming: From Sequential via SIMD and Thread to Massive Parallel Implementation. GECCO'14 Proceedings of the 2014 conference on Genetic and evolutionary computation.* New York: Association for Computing Machinery, str.1015-1022, 2014, ISBN 978-1-4503-2662-9.

[6] Intel: *Intel® Architecture Instruction Set Extensions Programming Reference.* Intel Corporation, 2012.

[7] Intel: *Intel® 64 and IA-32 Architectures Software Developer's Manual.* Intel Corporation, Vol. 2, 2016.

[8] Mazumdar, J. B.; Nirmala, S. R.: *Retina Based Biometric Authentication System: A Review.* International Journal of Advanced Research in Computer Science, Vol. 9, 2018.

[9] Miller, J.; Job, D.; Vassilev, V. K.: *Principles in the Evolutionary Design of Digital Circuits - Part I.* Journal of Genetic Programming and Evolvable Machines, Vol. 1, No. 1, 2000.

[10] Miller, J.; Smith, S.: *Redundancy and computational efficiency in Cartesian genetic programming.* Evolutionary Computation 10(2), 2006.

[11] Miller, J.; Thomson, P.: *Cartesian genetic programming. European Conference on Genetic Programming.* LNCS 1802, Springer, Berlin, str.121-132, 2000, ISBN 978-3-540-67339-2.

[12] Miller, J. F.: *Cartesian Genetic Programming*. Springer-Verlag Berlin Heidelberg, 2011, ISBN 978-3-642-17309-7.

[13] Niemeijer, M.; Staal, J.; Ginneken, B.; aj.: *DRIVE: Digital Retinal Images for Vessel Extraction*. 2004, [Online; navštíveno 19.04.2019].
URL http://www.isi.uu.nl/Research/Databases/DRIVE

[14] Quinn, M. J.: *Parallel programming in C with MPI and openMP*. McGraw-Hill Higher Education, 2004, ISBN 007-282256-2.

[15] Reda, S.; Shafique, M.: *Approximate Circuits: Methodologies and CAD*. Springer International Publishing, 2019, ISBN 978-3-319-99321-8.

[16] Sanjani, S. S.; Boin, J.-B.; Bergen, K.: *Blood Vessel Segmentation in Retinal Fundus Images*. [Online; navštíveno 28.02.2019].
URL https://stacks.stanford.edu/file/druid:yt916dh6570/Bergen_Boin_Sanjani_Blood_Vessel_Segmentation.pdf

[17] Sekanina, L.: *Evolvable components: From Theory to Hardware Implementations*. Natural Computing. Springer-Verlag Berlin, 2004, ISBN 978-3-540-40377-7.

[18] Sukkaew, L.; Uyyanonvara, B.; Barman, S.: *Comparison of Edge Detection Techniques on Vessel Detection of Infant's Retinal Image*. International Conference on Computer and Industrial Management, Bangkok, Thailand, str.6.1-6.5, 2005.

[19] Vassilev, V. K.; Miller, J. F.: *Towards the Automatic Design of More Efficient Digital Circuits. 2nd NASA/DoD Workshop on Evolvable Hardware*. IEEE Computer Society, Los Alamitos, CA, USA, str.151-160, 2000.

[20] Vašíček, Z.: *Acceleration Methods for Evolutionary Design of Digital Circuits*. disertační práce, Ústav počítačových systémů, FIT VUT v Brně, Brno, CZ, 2012.

[21] Vašíček, Z.: *Cartesian GP in Optimization of Combinational Circuits with Hundreds of Inputs and Thousands of Gates. Genetic Programming: 18th European Conference on Genetic Programming*. LNCS 9025, Springer International Publishing, Berlin, str.230-241, 2015, ISBN 978-3-319-16500-4.

[22] Vašíček, Z.; Sekanina, L.: *Evaluation of a New Platform for Image Filter Evolution. 2007 NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE Computer Society, str.577-584, 2007.

[23] Vašíček, Z.; Sekanina, L.: *Hardware Accelerators for CartesianGenetic Programming. Genetic Programming: 11th European Conference on Genetic Programming*. LNCS 4971, Springer-Verlag, Berlin-Heidelberg, str.230-241, 2008, ISBN 978-3-540-78670-2.

[24] Vašíček, Z.; Slaný, K.: *Efficient phenotype evaluation in cartesian genetic programming. Genetic Programming: 15th European Conference on Genetic Programming*. LNCS 7244, Springer-Verlag, Berlin-Heidelberg, str.265-276, 2012, ISBN 978-3-642-29138-8.

[25] Vostatek, P.; Claridge, E.; Uusitalo, H.; aj.: *Performance comparison of publicly available retinal blood vessel segmentation methods*. Computerized Medical Imaging and Graphics, Vol. 55, 2017.

[26] Yavuz, Z.; Köse, C.: *Blood vessel extraction in color retinal fundus images with enhancement filtering and unsupervised classification*. Journal of Healthcare Engineering, [Online; navštíveno 02.04.2019].
URL https://www.hindawi.com/journals/jhe/2017/4897258/